Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, цикловой комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI»

Виконав: студент 2-го курсу, групи 2КН-22м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)

Поліщук А. А.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Денисюк В. О.
(прізвище та ініціали)
«07» 12 2023 р.

Опонент: к.т.н., доцент каф. САІТ
Козачко О. М.
(прізвище та ініціали)
«07» 12 2023 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А. А.
(прізвище та ініціали)
«08» 12 2023 р.

Вінниця ВНТУ - 2023 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизаціїКафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
д.т.н., проф. Яровий А. А.
29.08 **2023 року**

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Поліщуку Анатолію Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI
керівник роботи_ к.т.н., доцент кафедри КН Денисюк В. О
затверджені наказом ВНТУ від «18» вересня 2023 року № 247
2. Строк подання студентом роботи « 13 » листопада 2023 року
3. Вихідні дані до роботи:
Мова програмування – об'єктно-орієнтована; технологія реалізації - Open MPI; середовище розробки - Visual Studio Code; вимоги до обчислювальної машини - процесор 2.4 GHz і краще, відеокарта 64 Мб. і вище, оперативна пам'ять 256 Мб. і вище, процесор intel core i3; програма - складатиметься з модулів введення даних, створення потоків, виконання операцій в потоках, об'єднання даних з потоків та виведення результатів.
4. Зміст текстової частини:
Вступ, аналіз предметної області швидкого сортування масивів даних, розробка інформаційної технології швидкого сортування масивів з використанням Open MPI, програмна реалізація інформаційної технології швидкого сортування масивів даних з використанням Open MPI, економічна частина, висновки, перелік використаних джерел, додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
Алгоритм роботи програми швидкого сортування масивів даних з використанням Open MPI, структура програми швидкого сортування масивів з використанням Open MPI, UML діаграма класів програми швидкого сортування масивів з використанням Open MPI, робочі вікна програми, результати тестування програми швидкого сортування масивів з використанням Open MPI та програми-аналога.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-4	Денисюк В. О., к.т.н., доц. каф. КН	29.08.23	10.11.23
5	Ратушняк О.Г., к.т.н., доцент каф. ЕПВМ	10.11.23	28.10.23

7. Дата видачі завдання 29. 08 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного рівня інформаційних технологій швидкого сортування масивів даних . Постановка задач дослідження	01.09.23 – 08.09.23	
2	Побудова моделей швидкого сортування масивів даних з використанням Open MPI	9.09.23 – 19.09.23	
3	Практичне застосування та оцінка ефективності розроблених моделей	20.09.23 – 10.10.23	
4	Підготовка економічної частини	11.10-23 28.10.23	
5	Апробація та/або впровадження результатів дослідження	29.10.23 4.11.23	
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	5.11.23 10.11.23	

Студент _____ (підпис) Поліщук А.А.

Керівник роботи _____ (підпис) Денисюк В. О.

АНОТАЦІЯ

УДК 004.8

Поліщук А. А. Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI. Магістерська кваліфікаційна робота зі спеціальності 122 «Комп'ютерні науки», освітня програма «Системи штучного інтелекту».. Вінниця: ВНТУ, 2023. 103 с.

Укр. мовою. Бібліогр.: 43 назв; рис.: 42; табл. 10.

Дана магістерська кваліфікаційна робота присвячена розробці інформаційної технології швидкого сортування масивів даних з використанням Open MPI. Основна мета була досягнута шляхом реалізації алгоритма швидкого сортуванню масивів даних мовою C++. Для виконання паралельних дій було використано технологію Open MPI. Результатом розробки є готовий паралельний алгоритм швидкого сортування масивів даних з використанням Open MPI та реалізація його мовою C++. Тестування розробленої програми довело відповідність вимогам, якими керувалися при проектуванні та розробці, правильність результату для усіх можливих вхідних даних, виконання програмного додатку за прийнятний час, практичність використання, сумісність із програмним забезпеченням та операційними системами. У економічному розділі визначено, що згідно узагальненого коефіцієнту конкурентоспроможності, науково-технічна розробка переважає існуючі аналоги. Термін окупності становить менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок. Результати дослідження можуть мати практичне значення для застосування в різноманітних галузях використання інформаційних технологій та в учбовому процесі..

Ключові слова: алгоритм, прогрпмп, Open MPI, паралельний алгоритм, швидке сортування.

ABSTRACT

Polishchyk A.A. Information technology for data array quick sorting by Open MPI. Master's thesis in the specialty 122 «Computer sciences», educational program «Artificial intelligence systems». Vinnytsia: VNTU, 2023. 103 p. In Ukrainian language. Bibliogr .: 43 titles; fig. 42; table 10.

This master's thesis is devoted to the development of information technology for quick sorting of data arrays by Open MPI. The main goal was achieved by implementing the algorithm for fast sorting of data arrays by C++. Open MPI technology was used to perform parallel operations. The result of the development is a ready-made parallel algorithm for fast sorting of data arrays using Open MPI and its implementation by C++. Testing of the developed program proved compliance with the requirements that were guided during design and development, correctness of the result for all possible input data, execution of the software application in an acceptable time, practicality of use, compatibility with software and operating systems. In the economic section, it is determined that according to the generalized coefficient of competitiveness, scientific and technical development prevails over existing analogues. The payback period is less than 3 years, which indicates the commercial attractiveness of the scientific and technical development and may encourage a potential investor to finance the implementation of this development and its introduction to the market. The results of the research can be of practical importance for application in various fields of information technology use and in the educational process.

Keywords: algorithm, program, Open MPI, parallel algorithm, quick sorting

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Актуальність полягає в тому, що задача сортування масивів є однією із найважливіших у багатьох задачах інформаційних технологій, адже її метою є полегшення подальшої обробки певних даних, сортування та задач пошуку.

Розробка присвячена саме поліпшеню алгоритмів сортування масивів даних. Безліч інформаційних технологій пов’язані з використанням алгоритмвів сортування, які своїми параметрами та особливостями з швидкодії та використовуваної пам’яті впливають на інтегральні параметри інформаційних систем.

При обробці масивів даних, а особливо великих масивів даних, виникає необхідність підвищення ефективності сортування, тому що порядок розташування елементів масивів у заданій послідовності за обраними критеріями зменшує часові витрати на пошук та доступ до інформації із відповідністю до певних ознак елементів даної інформації.

Актуальність розробки за темою магістерської кваліфікаційної роботи підтверджується непереревною необхідністю розробки ефективних алгоритмів сортування, які можуть бути застосовані в різноманітних галузях використання інформаційних технологій.

Отже, використання паралельного швидкого сортування, як сучасного методу підвишення швидкодії та загальної ефективності інформаційних технологій, є актуальною темою дослідження нових методів використання паралельного швидкого сортування.

Зв'язок розробки з науковими темами, планами, програмами. Магістерська кваліфікаційна робота виконана у повній відповідності до напрямків наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки,

навчання та комунікацій», плану навчально-методичної і наукової робіт на кафедрі комп'ютерних наук університету.

Мета та завдання розробки. Метою розробки є підвищення швидкодії процесу сортування масивів даних за допомогою Open MPI

Завдання, які необхідно вирішити для досягнення мети розробки:

- проаналізувати особливості і параметри, що виникають в задачах швидкого сортування;
- розглянути існуючі методи вирішення задачі швидкого сортування, щоб обґрунтувати досягнення поставленої мети перед дослідженням
- розробити необхідну математичну модель для досліджуваного швидкого сортування та згідно з метою дослідження виконати її удосконалення;
- алгоритм програми і структуру продукту розробки на основі розроблених стадій дослідження інформаційної технології;
- реалізувати програму швидкого сортування масиву даних;
- проаналізувати результати, що отримані під час тестування розробленого програмного продукту.

Об'єктом дослідження є процес швидкого сортування масивів даних.

Предмет дослідження – алгоритми, методи та програмні засоби паралельного швидкого сортування масивів даних з Open MPI.

Методи дослідження. У розробці були використані методи наукових досліджень: метод системного аналізу, який дозволив проаналізувати структуру інформаційної системи; методи теорії алгоритмів; основи розподілених систем та паралельних обчислень; методи об'єктно-орієнтованого програмування для інформаційних систем та технологій.

Наукова новизна отриманих результатів розроблено метод сортування даних, який відрізняється від існуючих використанням інформаційної моделі Open MPI, що дозволяє підвищити швидкодію сортування

масиву даних.

Практичне значення одержаних результатів полягає у такому.

- розроблено паралельний алгоритм швидкого сортування масивів даних з використанням Open MPI;
- розроблено програмний засіб алгоритм швидкого сортування масивів даних з використанням Open MPI;
- тестування розробленого програмного засобу підтвердило правильність теоретичних досліджень;
- розроблений алгоритм та програмний засіб можуть бути впроваджені в начальний процес як лекції на тему «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI» дисципліни «Технології розподілених систем та паралельних обчислень».

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджена відповідністю постанови задачі до теми розробки, порівнянням результатів досліджень із відомими даними, коректним використанням математичних моделей і методів, відповідним використанням аналітичних співвідношень, відповідністю результатів математичного моделювання з результатами, які отримані під час тестування розробки та впровадження програмних засобів, що розроблені.

Особистий внесок магістранта. Усі результати, які наведені у магістерській кваліфікаційній роботі, самостійно отримані магістрантом.

Апробація результатів роботи. Результати роботи були апробовані участю магістранта у міжнародній науковій конференції «Міжнародна науково-практична інтернет-конференція Молодь в науці: дослідження, проблеми, перспективи» [1].

Публікації. За результатами дослідження у магістерській кваліфікаційній роботі було опубліковано тези доповіді на конференції [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ШВИДКОГО СОРТУВАННЯ МАСИВІВ ДАНИХ

1.1 Постановка задачі швидкого сортування масивів даних

Сортування даних є однією із важливих і основних задач, в якій відбувається розміщення елементів, що розташовані у довільному порядку, або у порядку, що нас не цікавить, у визначеному згідно цілей задачі порядку, наприклад, у зростаючому чи спадаючому порядку [1-4].

В деяких сучасних додатках жорстко стоять вимоги відповідної швидкодії, без якої система може бути взагалі непрацездатною.

Отже, використовуючи деякі ознаки елементів масивів даних (ключі), сортування розміщує їх у обраній послідовності.

Завдання, які стоять перед сучасними інформаційними технологіями, вимагають використання нових методів та алгоритмів сортування масивів даних. Ці завдання зумовлюють підвищення швидкодії та ефективності сортування масивів даних, наприклад, для систем реального часу

Режим реального часу та реалізація алгоритмів сортування здійснюється за допомогою конвеєризації процесів сортування, розпаралелювання обчислювальних процесів, що, в свою чергу, дозволяє використовувати інформаційну систему для обробки великого об’єму інформації.

На рисунку 1.1 представлена UML-діаграма класів алгоритмів сортування [5].

Велика кількість алгоритмів сортування зумовлена складністю та різноманітністю задач, в яких виникає необхідність сортування масивів даних.

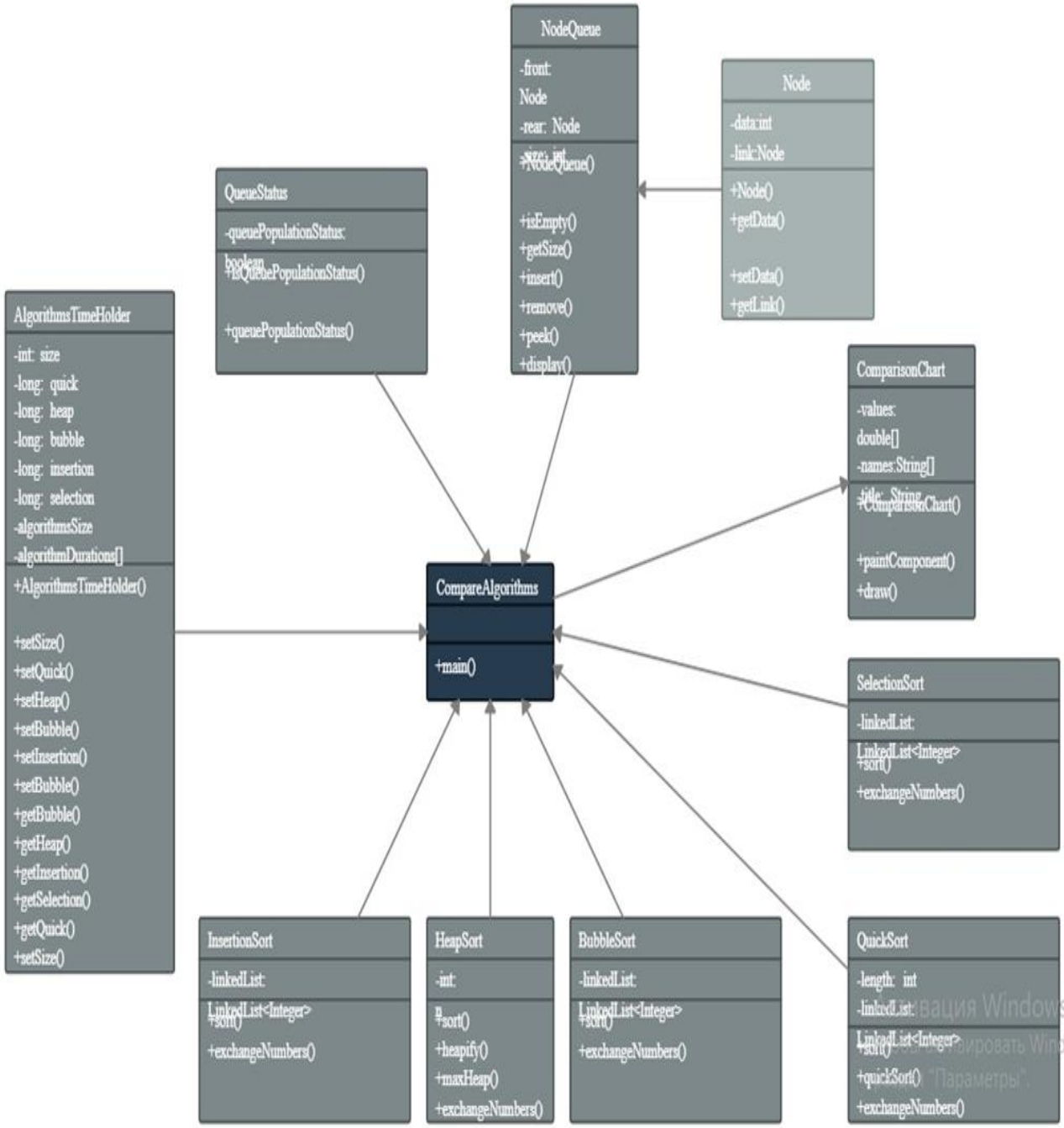

Рисунок 1.1 - UML-діаграма класів алгоритмів сортування

Використовувати швидке або паралельне швидке сортування елементів масивів даних краще за все в задачах, які пов’язані із значними об’ємами розрахунків. Наприклад, у наукових експериментах, при моделюванні складних систем, що містять велику кількість лінійних або нелінійних

рівнянь.

На рисунку 1.2 надано приклад UML-діаграми класів програмної системи сортування з використанням швидкого сортування [6].

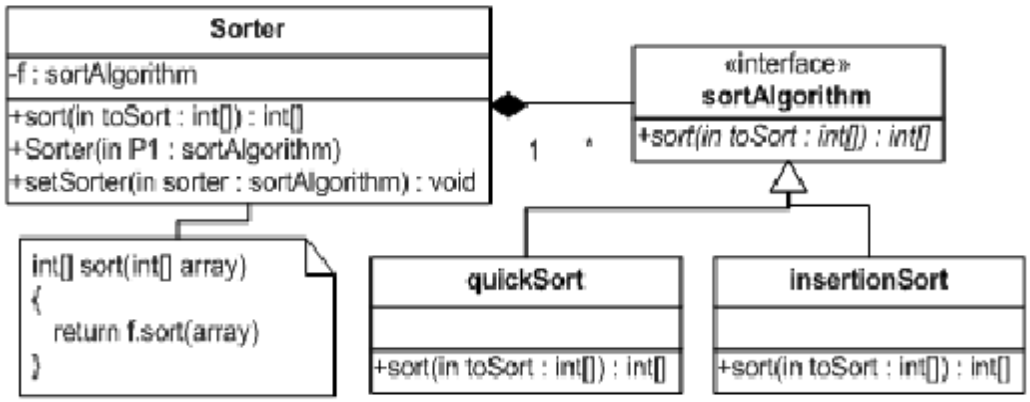

Рисунок 1.2 – UML-діаграма класів програмної системи сортування

Клієнти використовують цю програмну систему, викликаючи метод Sorter.sort(). Реалізація цього методу складається з виклику методу sortAlgorithm.sort() через атрибут f. Тут в залежності від значення атрибуту f, один із підкласів, а саме quickSort і insertionSort, інтерфейсу sortAlgorithm отримує виклик. Ці підкласи реалізують різні алгоритми сортування в методі sort(). Конструкція програмної системи сортувальника підтримує реконфігурацію зміни алгоритма сортування. Виклик f.sort() є поліморфним викликом (тобто одержувача виклику можна змінити), і залежно від значення змінної f під час виконання, різні підкласи інтерфейсу sortAlgorithm можуть отримати виклик. Поліморфізм RM змінює структуру та шаблон виклику, змінюючи отримувача виклику.

На рисунку 1.3 надано UML-діаграму використання різних методів сортування [7].

В залежності від особливостей задачі сортування користувач обирає оптимальний варіант її розв’язання.

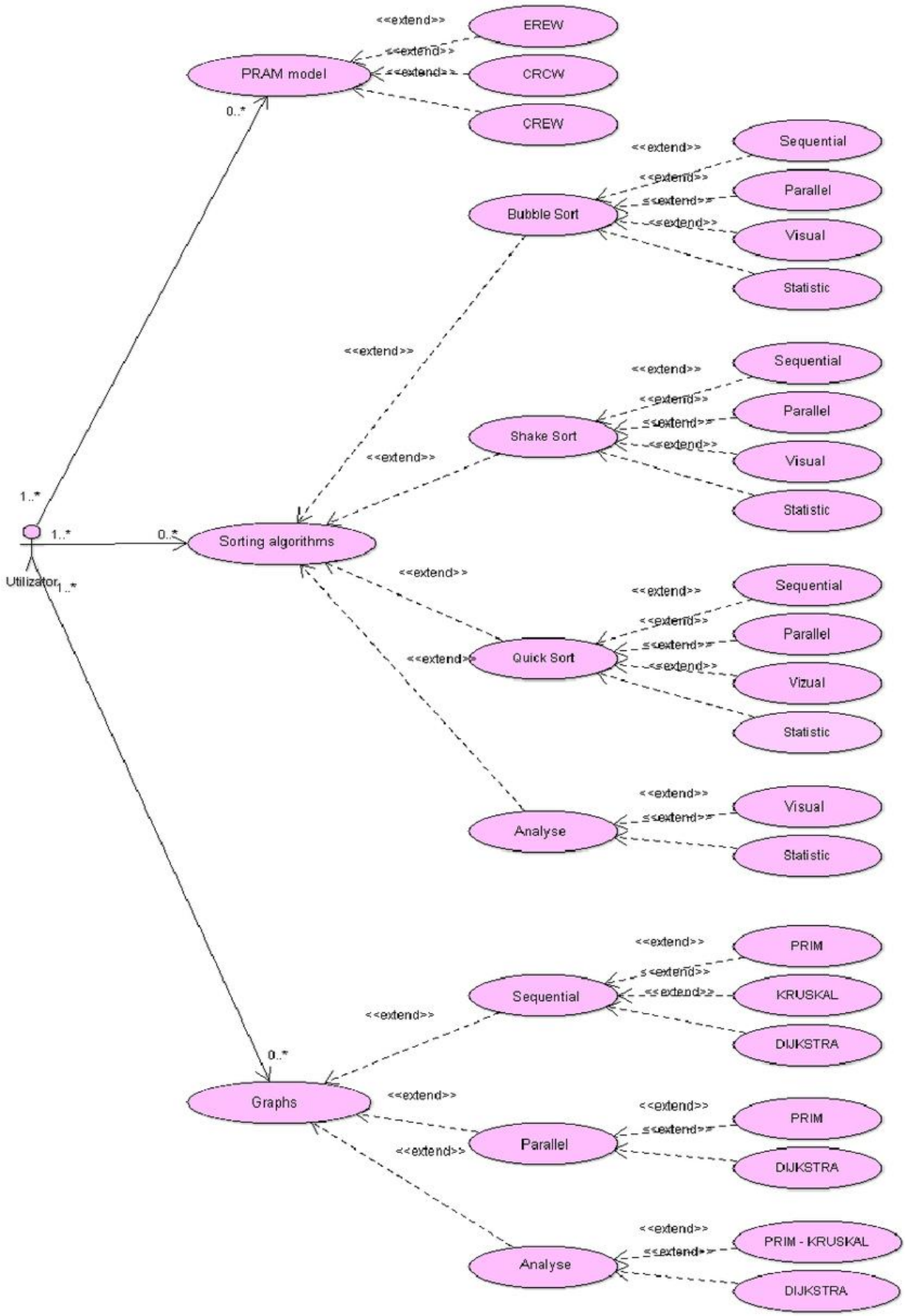

Рисунок 1.3 – UML-діаграма використання різних методів сортування

1.2 Актуальність та галузі застосування швидкого сортування масивів даних

Алгоритм швидкого сортування – це алгоритм ***Quicksort***, який був представлений науковцем Тоні Гоаром у 1962 році (Charles Antony Richard Hoare) [8-10].

Його особливості такі.

Додаткової пам'ті алгоритм не вимагає.

Середній час виконання алгоритма становить *O(nlog(n))* операцій.

Найгірший час виконання алгоритма становить $O(n^2)$ операцій.

Якщо порівнювати цей алгоритм з іншими алгоритмами, яким притаманна східна асимптотична складність, то він працює значно швидше та ще містить більш прості операції та цикли.

Алгоритм сортуванням злиттям, наприклад, в два рази повільніший за алгоритм Гоара.

Головне, що лежить в основі цього сортування, - це вибір такого елементу масиву (опорного елементу) аби отримати дві частини масиву такі, щоб будь-який елемент однієї частини не перевищував жодного елементу іншої частини.

Кожна з частина масиву впорядковується рекурсивно.

Хорошою рисою швидкого сортування є те, що його можливо використовувати як у двопов'язаних списках так і у масивах.

Класичний варіант, який запропонував Гоар, полягає у тому, що обирався деякий елемент із масиву даних, який розбивав вхідний масив на частини, причому, у одній частині повинно бути не більші за даний елемент, а у другій частині не менше за даний елемент.

Процедура ***Quicksort(A,p,q)*** виконує часткове впорядкування ***A***-масиву з першого індексу ***p*** до другого індексу ***q*** (рис.1.4) [8, 10].

```
Quicksort(A, p, q)
1  if p ≥ q return;
2  r ← A[p]
3  i ← p
4  j ← q + 1
5  while i < j do
6        repeat
7              i ← i + 1
8        until A[i] ≥ r
9        repeat
10             j ← j − 1
11       until A[j] ≤ r
12       if i < j
13          then Swap A[i] ↔ A[j]
14 Quicksort(A, p, j)
15 Quicksort(A, j + 1, q)
```

Рисунок 1.4 - Класичний алгоритм Гоара ***Quicksort***

Основна мета кожної інформаційно-пошукової системи – це надання користувачеві інформації, яка найбільше відповідає тій, яку користувач потребує.

Пошукова система містить роботів-пошуковців. Роботи-пошуковці переглядають сторінки Інтернету та використовуються для сортування даних та різних інтернет-ресурсів. Кожний з них виконує свої специфічні функції. Для великих об’ємів пошуку необхідно використовувати велику кількість роботів-пошуковців.

Після збирання інформації індексатор створює базу даних, яка використовується для надання інформації користувачеві.

Отже, наявна потреба збільшення швидкодії пошукових систем, що зумовлює необхідність використання швидших та ефективніших алгоритмів алгоритмів пошуку та сортування масивів даних.

Головним критерієм пошукових системи є швидкість пошуку інформації. Швидкість пошуку пошукової машини повинна бути достатньою аби задовільнити потреби користувачів у швидкому отриманні запитів.

1.3 Огляд відомих методів розв'язання задачі

Існує ряд необхідних вимог до алгоритмів сортування, як і до алгоритмів взагалі [2-5, 11-13]:

- детермінованість або визначеність – це означає те, що у кожний момент часу алгоритм надає слідуючий крок, а також те, що для одних і тих же вихідних даних алгоритм видає однакову відповідь;
- зрозумілість означає те, що виконавцю алгоритма відомі усі команди алгоритма;
- скінченність алгоритму означає правильну завершеність роботи алгоритма за відповідний час роботи;
- масовість означає можливість виконання алгоритму на достатньо великій множині даних;
- результативність означає те, що алгоритм завершується результатом.

Виконання алгоритмів має свої особливості.

Так для виконання алгоритмів необхідні початкові значення.

Початкові значення необхідні у роботі багатьох програм. Вони відповідають за перенесення значень у аргументи алгоритмів. Аргументи - це величини, що необхідні алгоритмам для виконання.

Нерезультативних алгоритмів не існує. Кожний алгоритм закінчується визначенням результатів.

Також в деяких випадках алгоритми використовують додаткові значення – проміжні значення.

Основні форми представлення алгоритмів такі.

1) Опис за допомогою вербальних означень.
2) Опис у вигляді таблиць.
3) Опис з використанням математичних формул, визначень предикатів.
4) Опис з використанням блок-схем за відповідним державним стандартом.
5) Опис з використанням псевдокоду, мови що наближена до алгоритмічної мови програмування
6) Опис за допомогою програмного коду однією із мов програмування.

Опис алгоритму за допомогою програмного коду однією із мов програмування є найбільш наближеним до виконання на комп'ютері варіантом представлення алгоритмів.

Основними параметрами кожного алгоритму є об'єм памяті, необхідної для роботи алгоритму та час, за який цей алгоритм надає результат.

Пам'ять на роботу алгоритма представляє собою частину загального простору пам'яті комп'ютера.

Час роботи алгоритма залежить від багатьох параметрів, наприклад, параметрів, пов'язаних з особливостями конкретної апаратно-програмної платформи, на якій алгоритм реалізується. Тому прийнято у якості часу роботи алгоритму обрати відносний параметр, наприклад, кількість кроків алгоритму, кількість операцій порівняння тощо.

У деяких алгоритмів виникає необхідність збереження проміжних значень. Тоді необхідно забезпечити додаткову пам'ять.

За організацією пам'яті для виконання алгоритмів виділяють таки типи пам'яті.

1) Спільна пам'ять.

2) Розподілена пам’ять.

3) Роздільна пам’ять.

4) Комбінована тощо.

Для характеристики алгоритма сортування у якості основного параметра, який характеризує час його виконання обрано умовний параметр, що визначається у кількості кроків на виконання алгоритму.

Таким чином, можна порівнювати різні варіанти алгоритмів за асимптотичною залежністю – зростанням часу виконання кроків.

По відношенню до використання пам’яті комп’ютера існує два основних типа сортування:

1) внутрішнє сортування,

2) зовнішнє сортування.

Якщо масив даних, який підлягає сортуванню, повністю вміщується в оперативну пам’ять комп’ютера, то мова іде про внутрішнє сортуваня. Таке можливо, коли масив данних невеликий, або маємо велику внутрішню пам’ять комп’ютера.

Якщо масив даних, який підлягає сортуванню, повністю не вміщується в оперативну пам’ять комп’ютера, то мова іде про зовнішнє сортуваня. Таке можливо, коли масив данних надвеликий, або маємо невелику внутрішню пам’ять комп’ютера.

В даній роботі будемо орієнтуватися на традиційний алгоритм внутрішнього сортування, а ефективність будемо досягати за рахунок розпаралелювання швидкого внутрішнього сортування з використанням розподіленої пам’яті.

Вважаємо, що всі достатньо великі масиви даних, які буде відсортовувати шуканий алгоритм, вміщуються в достатньо велику оперативну пам’ять комп’ютера.

Використання зовнішнього сортування може привести до втрати швидкодії, яка буде витрачена на передавання інформації між

периферійними запам’ятовуючими пристроями та оперативною пам’яттю.

Для ефективної роботи паралельного алгоритму швидкого сортування необхідно оптимально слідувати основним вимогам [14, 15]:

1) масштабованість,
2) прискорення обчислень,
3) синхронізація.

Масштабованість визначає у скільки разів прискорення обчислень перевищує витрати на обчислення. Паралельний спосіб розрахунків повинен обраховуватися з більшою швидкістю.

Масштабованість визначає у скільки разів прискорення обчислень перевищує витрати на обчислення.

Для синхронізації паралельних обчислень використовуються:

1) блокування,
2) передача повідомлень.

Блокування дозволяє призупинити виконання паралельних процесів для їх синхронізації.

При наявності швидкісних ліній зв’язку між вузлами паралельної обробки алгоритму ефективно для синхронізації використовувати передачу повідомлень.

1.4 Обгрунтування вибору аналогу до програми швидкого сортування масивів

Сортування - перегрупування заданої множини об’єктів в деякому певному порядку [2, 3, 11, 13]. Мета сортування - полегшити пошук елементів.

Термін сортування означає розділення елементів за певними правилами.

Паралельний алгоритм – це алгоритм, який може виконувати кілька команд одночасно на різних пристроях обробки, а потім поєднувати всі окремі вихідні дані для отримання остаточного результату [7, 14, 15].

Завдання ділиться на підзадачі і виконується паралельно отримання окремих вихідних даних. Пізніше ці окремі виходи поєднуються, щоб отримати кінцевий бажаний результат. Нелегко розділити велику проблему на підзадачі. Підзадачі можуть мати залежність даних між ними. Тому процесори повинні взаємодіяти один з одним, щоб вирішити проблему.

Було виявлено, що час, необхідний процесорам для зв'язку один з одним, більше, ніж фактичний час обробки. Таким чином, при розробці паралельного алгоритму необхідно враховувати правильне завантаження програмного забезпечення, щоб отримати ефективний алгоритм. Для розробки правильного алгоритму необхідно мати чіткі уявлення про основну модель обчислень на паралельному комп'ютері.

Як послідовні, так і паралельні комп'ютери працюють із набором (потоком) інструкцій, названих алгоритмами. Цей набір інструкцій (алгоритмів) інструктує комп'ютер у тому, що він повинен робити на кожному етапі.

Постановка задачі сортування така.

У нас є вхідна послідовність даних, масив даних із n елементів.

Наприклад, вхідна послідовність - (5, 6, 1, 8, 5, 8, 4, 3), тоді вихідна послідовність буде мати вигляд (1, 3, 4, 5, 5, 6, 7, 8, 8).

У якості основних характеристик алгоритмів розглядають час, необхідний на впорядкування n-елементного масиву, а також необхідність додаткової пам’яті для сортування [7, 11-13].

Широковідомі такі алгоритми сортування: сортування вибором, сортування злиттям, сортування вставкою, сортування обміном, сортування Шелла, сортування Гоара, сортування перестановкою, випадкове сортування.

У даній роботі розглядаємо алгоритм паралельного швидкого сортування.

Швидке сортування або сортування Гоара.

Цей алгоритм сортування використовує обмінне сортування [2-4, 8-11, 13].

На рисунку 1.5 надано приклад алгоритму швидкого сортування, який сортує послідовність розміром $n=8$ (pivot - опорна точка).

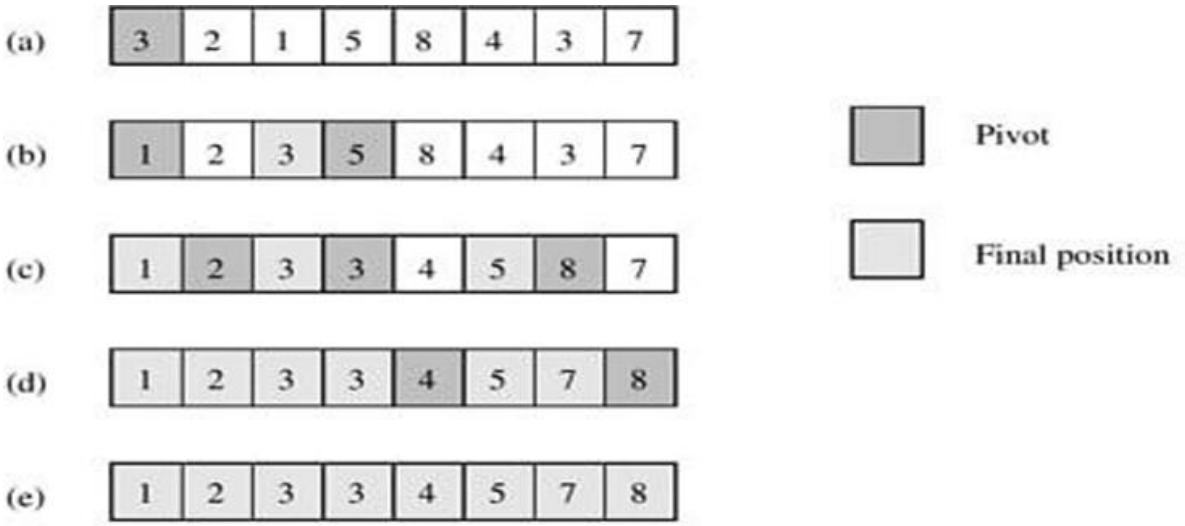

Рисунок 1.5 – Приклад алгоритму швидкого сортування

Складність швидкого сортування, в найкращому випадку становить $O(nlog(n))$, а в найгіршому – $O(n^2)$. Окрім низької обчислювальної складності алгоритму швидкого сортування властиві такі переваги:

1) із алгоритмів внутрішнього сортування, практично, - це самий швидкий алгоритм сортування;
2) простий для реалізації та розуміння;
3) не потребує додаткової пам'яті *(O(1))*, або *O (n);*
4) ефективно використовувати для сортування великих масивів даних;
5) можливо використовувати для сортування зв'язаних списків;

6) масиви можливо сортувати паралельно.

Швидке сортування має такі особливості.

По-перше, гірша часова складність у порівнянні із деякими іншими методами сортування. Зазвичай це відбувається, коли елемент, обраний як опорний, є найбільшим або найменшим елементом, або коли всі елементи однакові. Ці найгірші випадки різко впливають на продуктивність швидкого сортування. Вибір майже відсортованого масиву також може призвести до випадку коли розмір стека виконання перевищує розмір стека;

По-друге, нестабільність. Алгоритм вважається нестабільним, оскільки він не зберігає початковий порядок пари ключ-значення. У стабільних алгоритмах сортування, незважаючи на те, що елементи сортуються, оригінальний порядок пари ключ-значення запам’ятовується. Іншими словами, швидке сортування не збереже порядок елементів під час зміни порядку. Таким чином, існує ймовірність того, що два однакові елементи у невідсортованому списку можуть бути замінені під час сортування списку. Але ймовірність такого випадку достатньо мала та повинна компенсуватися загальною ефективністю паралельного швидкого сортування.

Проведемо аналіз швидкого сортування [16-21].

Час роботи швидкого сортування (Quicksort) у найгіршому та середньому випадку відрізняється. Розглянемо найгірший випадок тривалості роботи алгоритма. Припустимо, що нам дійсно не пощастило, і розміри підмасивів, що розділені, дійсно незбалансовані. Зокрема, припустимо, що опорна точка (pivot), обрана функцією розділення (partition), завжди є або найменшим, або найбільшим елементом у підмасиві із n елементів. Тоді один із підмасивів, що розділені, не міститиме елементів, а інший підмасив міститиме $n-1$ елементів — усі, крім опорної точки. Таким чином, рекурсивні виклики будуть на підмасивах розміром 0 елементів та розміром $n-1$ елементів.

Як і при сортуванні злиттям, час для сортування масиву із n елементів становить $\theta(n)$. У сортуванні злиттям це час для злиття, в швидкому сортуванні - час для розділення.

Час роботи в найгіршому випадку має місце, коли швидке сортування завжди має максимально незбалансовані розділи, тоді початковий виклик потребує *cn* часу для деякої константи *c*, рекурсивний виклик *n-1* елементів займає *c(n-1)* часу, рекурсивний виклик *n-2* елементів займає *c(n-2)* часу і так далі. На рисунку 1.6 надано дерево розмірів підмасивів та час їх розділення в найгіршому випадку.

На діаграмі найгіршої продуктивності для швидкого сортування (рис.1.6) зображено дерево ліворуч і часом розділення праворуч. Позначено "Subproblem sizes" – в нашому випадку розмір підмасива сортування, а праворуч позначено "Total partitioning time for all subproblems of this size" - загальний час розділення для всіх підмасивів цього розміру.

Перший рівень дерева показує окремий вузол *n* і відповідний час поділу на *c*, помножене на *n*.

Другий рівень дерева показує два вузли, *0* та *n-1*, і час розділення, що дорівнює *c* разів *n-1*.

Третій рівень дерева показує два вузли, *0* та *n-2*, і час розділення - *c* разів *n-2*.

Четвертий рівень дерева показує два вузли, *0* та *n-3*, і час розділення *c*, помножене на *n-3*. Під цим рівнем крапки вказують, що дерево продовжується таким же чином.

Передостанній рівень у дереві має один вузол *2* із часом розділення *2*, помноженим на *c*, а останній рівень має два вузли *0* та *1* із часом розділення *0*.

Час розділення для кожного рівня становить (1.1):

$$cn+c(n-1)+c(n-2)+\ldots+2c=c(n+(n-1)+(n-2)+\ldots+2)=c((n+1)(n/2)-1) \qquad (1.1)$$

Останній рядок є арифметичним рядом *1 + 2 + 3 + ... + n* як під час сортування вибору. Віднімається *1*, тому що для швидкого сортування підсумовування починається з *2*, а не з *1*). Є деякі члени молодшого порядку та постійні коефіцієнти, але при використанні θ , вони ігноруються. Для θ найгірший час роботи швидкого сортування становить $\theta(n^2)$.

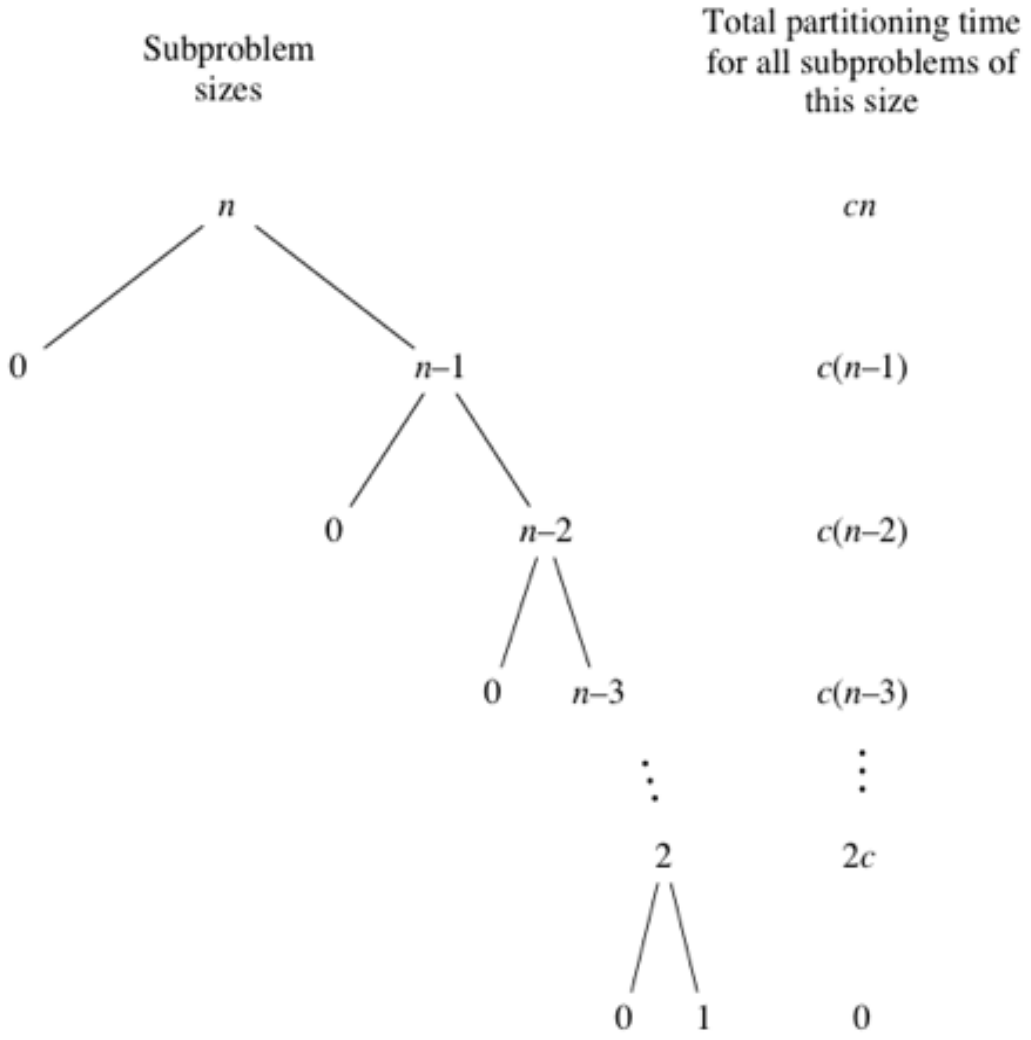

Рисунок 1.6 - Дерево розмірів підмасивів та час їх розділення в найгіршому випадку

Найкращий час роботи. Найкращий випадок швидкого сортування відбувається, коли розділи максимально рівномірно збалансовані: їхні розміри або однакові, або знаходяться в межах *1* один від одного. Перший випадок має місце, якщо підмасив має непарну кількість елементів, а опорна

точка знаходиться точно посередині після розділення, і кожен розділ має *(n-1)/2* елементів. Останній випадок має місце, якщо підмасив має парну кількість n елементів і один розділ має *n/2* елементів, а інший має *n/2-1*. У будь-якому з цих випадків кожен розділ має щонайбільше *n/2* елементів, і дерево розмірів підмасивів дуже схоже на дерево розмірів підмасивів для сортування злиттям, при цьому час розбиття виглядає як час об'єднання (рис.1.7).

Діаграма найкращої продуктивності для швидкого сортування з деревом ліворуч і часом розділення праворуч (рис.1.7). Дерево має позначку "Subproblem size" (розмір під проблеми – розмір масиву), а праворуч — "Total partitioning time for all subproblems of this size" (загальний час розділення для всіх підмасивів цього розміру).

Перший рівень дерева показує окремий вузол n і відповідний час поділу на *c*, помножене на *n (cn).*

Другий рівень дерева показує два вузли, кожен з яких менший або дорівнює *1/2 n*, а час поділу менший або дорівнює *2* помноженим на c, помноженим на *1/2 n*, як і c, помноженим на *n* ($\leq \frac{2cn}{2} = cn$).

Третій рівень дерева показує чотири вузли, кожен з яких менший або дорівнює *1/4 n*, а час розділення менше або дорівнює *4* помноженим на c помножене на 1/4 n, як і *c* помножене на *n* ($\leq \frac{4cn}{4} = cn$) .

Четвертий рівень дерева показує вісім вузлів, кожен з яких менш чи дорівнює *1/8 n*, а час розділення менше або дорівнює *8* помножених на c помножених на *1/8 n*, так само, як c помножених на *n* ($\leq \frac{8cn}{8} = cn$) . Під цим рівнем показано крапки, які вказують на те, що дерево продовжується таким чином.

Остаточний рівень показано з *n* вузлами по *1* і часом розділення, меншим або рівним *n*, помноженим на *c*, таким самим, як *c*, помноженим на *n* ($< nc = cn$).

Для оцінки θ найкращий час роботи швидкого сортування становить $\theta(nlog(n))$.

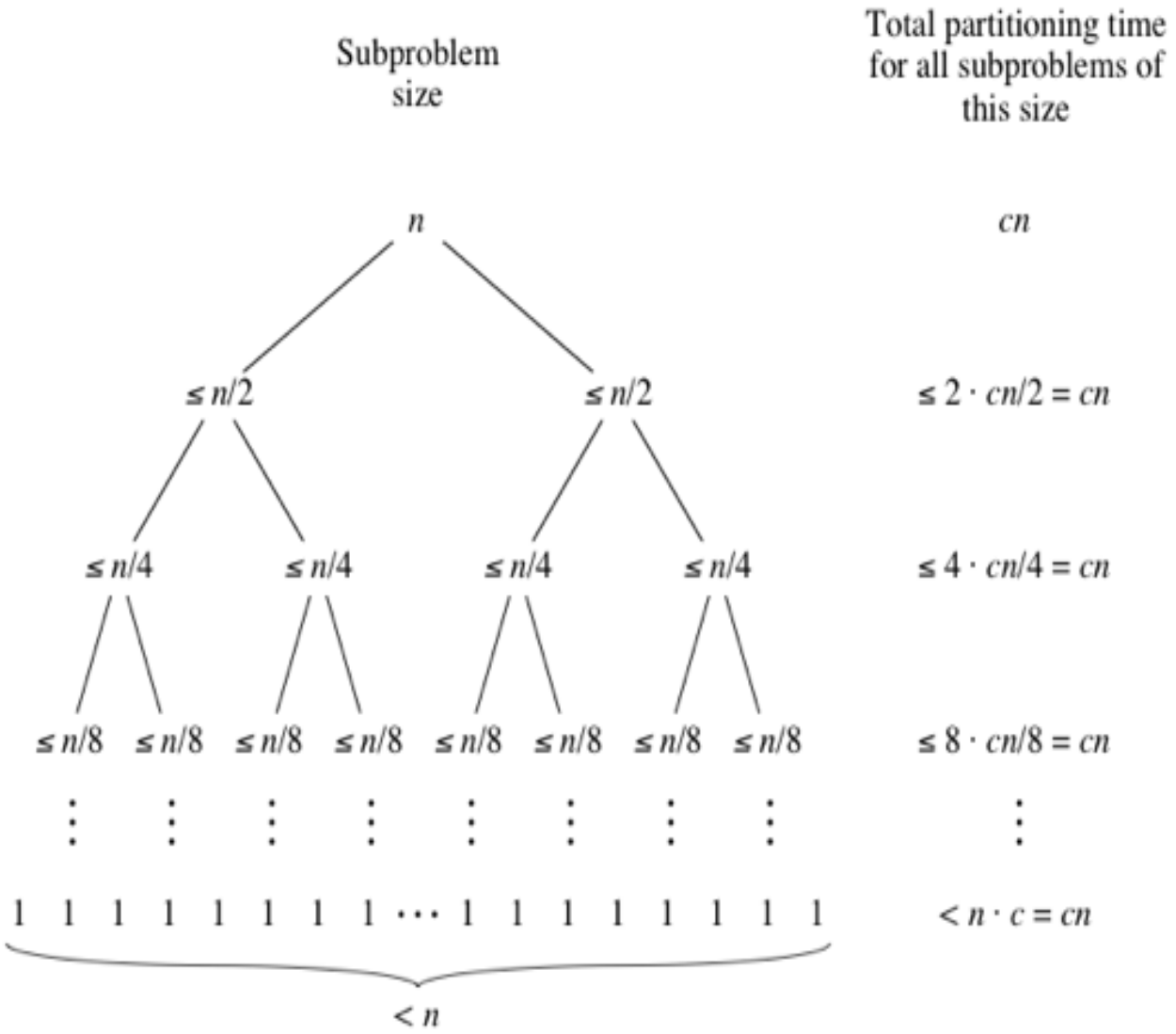

Рисунок 1.7 - Дерево розмірів підмасивів та час їх розділення в найкращому випадку

Середній час роботи.

Час виконання в середньому також дорівнює $\theta(nlog(n))$, Ця межа виникає оскільки середній час роботи не може бути кращим за найкращий час роботи. По-перше, уявімо, що не завжди отримуємо рівномірно

збалансовані розділи, але в гіршому випадку завжди отримуємо розподіл *3* до *1*.

Отже, кожного разу, коли розбиваємо, одна сторона отримує *3n/4* елементів, а інша сторона отримує *n/4*. Дерево розмірів підмасивів та часу розділення надано на рисунку1.8.

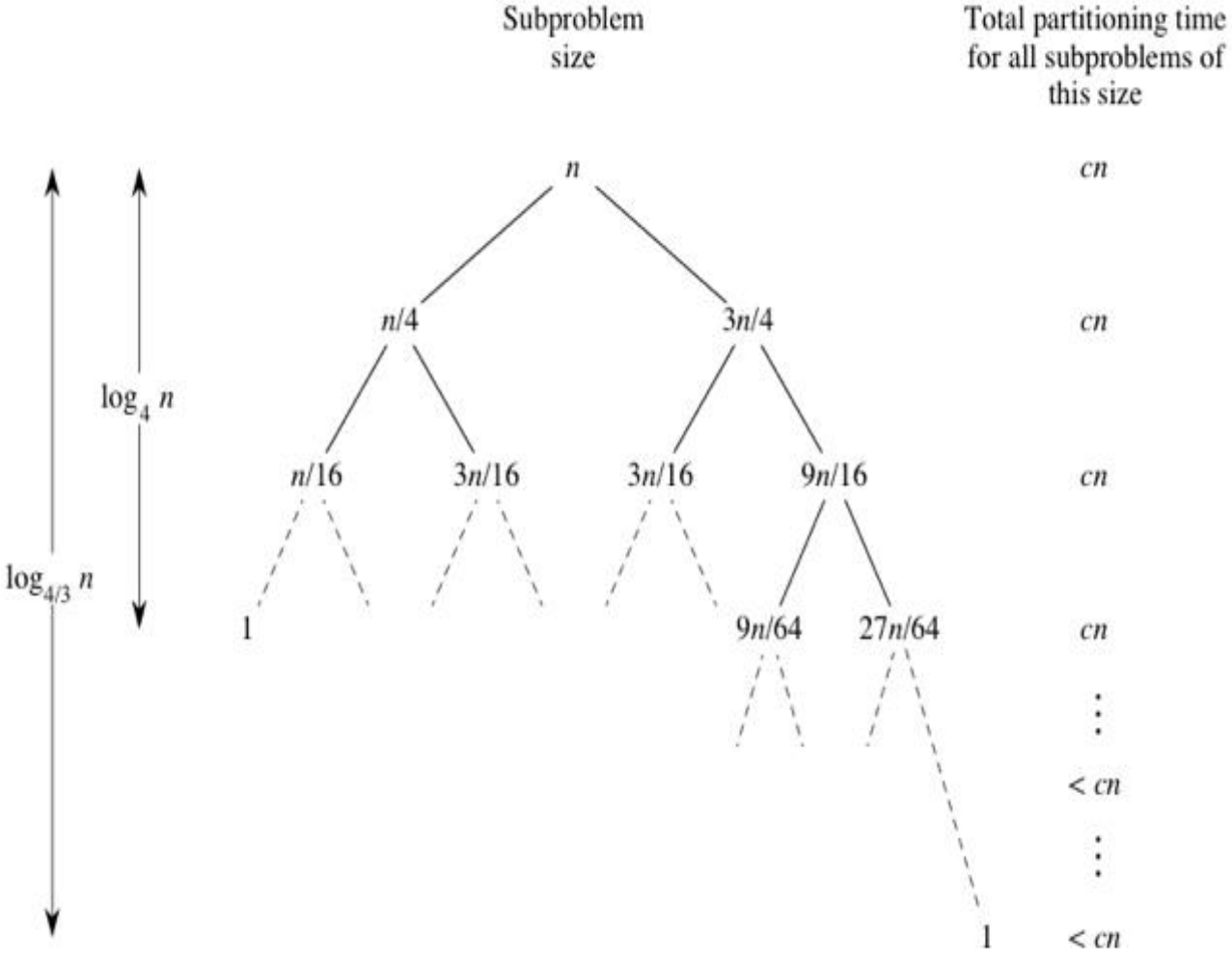

Рисунок 1.8 - Дерево розмірів підмасивів та час їх розділення в середньому

Лівий дочірній елемент кожного вузла представляє підмасив розміром *1/4*, а правий дочірній елемент представляє підмасив розміром *3/4*. Оскільки менші підмасиви знаходяться ліворуч, дотримуючись шляху лівих дітей, ми

дістаємося від кореня вниз до розміру підмасиву 1 швидше, ніж уздовж будь-якого іншого шляху.

Після $log_4 n$ рівнів, переходимо до розміру підмасиву 1 (рис.1.8). Найлегше подумати, починаючи з розміру підмасиву 1 і множивши його на 4, поки не досягнемо n. Іншими словами, запитуємо, яким значення *x* є $4^x = n$. Відповідь - $log_4 n$. Щоб піти шляхом правильних дітей на графі потрібно $log_{4/3} n$.

Оскільки кожен правий дочірній вузол становить *3/4* розміру вузла над ним (його батьківського вузла), кожен батьківський вузол у *4/3* разів більший від свого правого дочірнього вузла.

Якщо почати з підмасиву розміром *1* і помножити розмір на *4/3*, поки не досягнемо *n*. За якої величиної *x* є $(\frac{4}{3})^x = n$. Відповідь - $log_{4/3} n$.

У кожному першому $log_{4/3} n$ рівнів, є *n* елементів (включаючи опори, які насправді більше не розділяються), і тому загальний час розділення для кожного з цих рівнів *cn*.

Для решти рівнів кожен із них має менше ніж *n* вузлів, тому час розділення для кожного рівня становить максимум *cn*. Загалом, є $log_{4/3} n$ рівнів, тому загальний час розділення становить $O(nlog_{\frac{4}{3}} n)$.

Відомо, що:

$$log_a n = \frac{log_b n}{log_b a} \tag{1.2}$$

де *a, b* та *n* - додатні числа.

Підставимо *a = 4/3* та *b = 2*, отримаємо:

$$log_{4/3} n = \frac{log_2 n}{log_2 (\frac{4}{3})}, \tag{1.3}$$

де $log_{4/3}n$ та $log_2 n$ відрізняються лише на коефіцієнт $log_2(\frac{4}{3})$, який є константою.

Оскільки постійні коефіцієнти не мають значення, коли ми використовуємо нотацію O, можливо стверджувати, що якщо всі розбиття є 3 до 1, то час роботи швидкого сортування становить $O\big(n\, log_2(n)\big)$, хоча із більшим прихованим постійним фактором, ніж найкращий час роботи.

Існує завдання з'ясувати, як часто очікувати розподіл 3 до 1 або краще. Це залежить від того, як обирається опорна точка.

Припустимо, що опорна точка з рівною ймовірністю опиниться в будь-якому місці n-елементного підмасиву після розбиття (рис.1.9). Потім, щоб отримати розподіл 3 до 1 або краще, опорна точка повинна бути у середній частині (middle half).

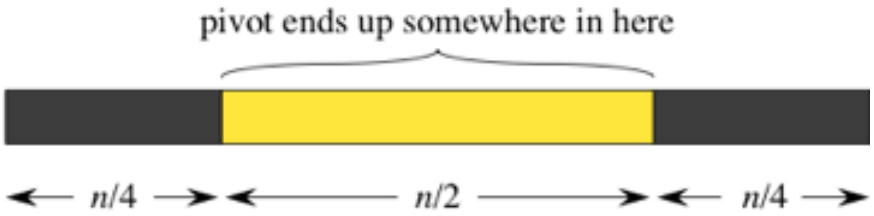

Рисунок 1.9 – Приклад розбиття вхідного масива

Отже, якщо опорна точка з однаковою ймовірністю опиниться в будь-якому місці підмасиву після розділення, тоді існує шанс отримати в гіршому випадку розподіл 3 до 1 з ймовірністю 50%. Іншими словами, приблизно в половині випадків очікується розподіл 3 до 1 або краще.

Інший випадок, для розуміння, чому середній час роботи QuickSort становить $O\big(n\, log_2(n)\big)$ — це та половина випадків, коли не отримаємо розподіл 3 до 1, а отримаємо розподіл у найгіршому випадку. Припустимо, що розбиття 3 до 1 і найгірший варіант чергуються, розглянемо вузол у

дереві з k елементів у підмасиві. Тоді замість частини дерева на рисунку 1.10 отримаємо іншу (рис.1.11).

Таким чином, навіть якби ми отримували найгірший варіант розподілу в половині випадків і розподіл у співвідношенні 3 до 1 або краще в половині випадків, час виконання буде приблизно вдвічі більший від часу отримання розподілу 3 до 1 кожного разу. Знову ж таки, це лише постійний фактор, і він поглинається в оцінці O, тому у випадку між найгіршим випадком і 3 до 1, час роботи становить $O\left(n\ log_2(n)\right)$.

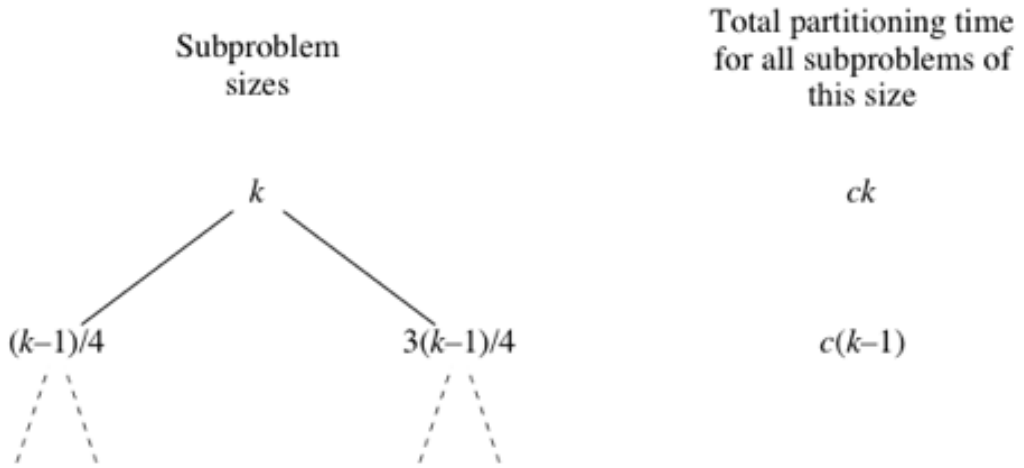

Рисунок 1.10 - Попереднє дерево з k елементів у під масиві

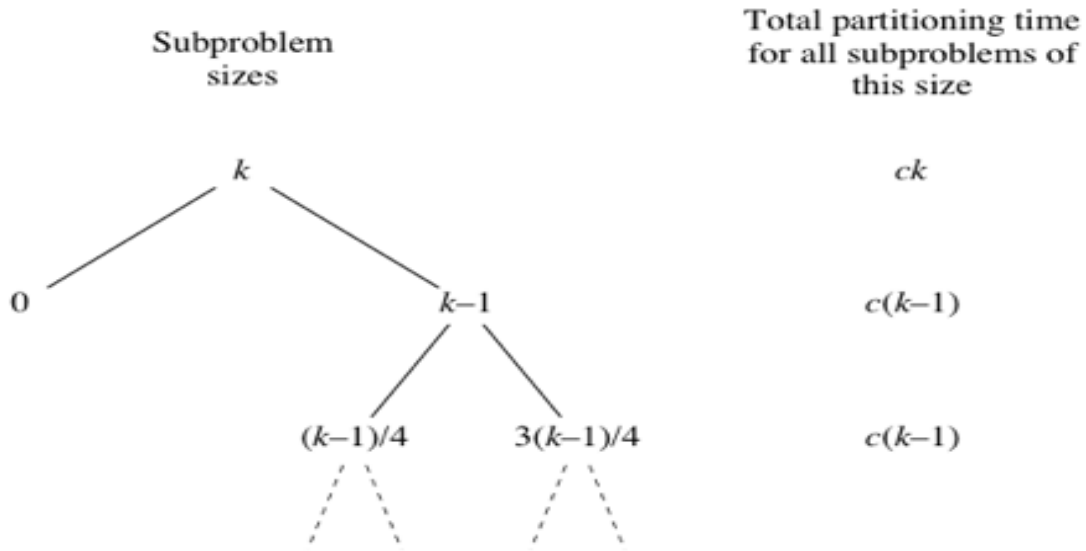

Рисунок 1.11 - Вузол у дереві з k елементів у підмасиві

Рандомізоване швидке сортування [22, 23].

Припустімо, що виникла ситуація, коли надано масив для сортування за допомогою швидкого сортування, причому, завжди обирається крайній правий елемент у кожному підмасиві як опорний, а масив впорядковано «зловмисником» так, щоб завжди отримувати розбиття в найгіршому випадку. Запобігти такій ситуації можливо таким чином.

Не обов’язково обирати крайній правий елемент у кожному підмасиві як опору. Замість цього можливо випадковим чином обирати елемент у підмасиві та використовувати цей елемент як опорний. Згадаємо, що функція розділення передбачає, що опорна точка знаходиться в крайньому правому положенні підмасиву. Для цього треба, наприклад, замінити елемент, який обрано опорним, на крайній правий елемент, а потім розділити масив, як і раніше. Отже, намірене створення такого «незручного» масиву не буде впливати на сортування, якщо стороннім («зловмисникам») не відомо, як обираються випадкові місця в підмасиві.

Також можливо збільшити шанси отримати розподіл, який у гіршому випадку буде 3 до 1. Для цього треба випадково обрати не один, а три елементи з підмасиву та узяти медіану з трьох як опору (помінявши її місцями з крайнім правим елементом).

Під медіаною ми розуміємо елемент із трійки, значення якого знаходиться посередині. Якщо обрати медіану трьох випадково вибраних елементів як опорну, буде 68,75% шансів (11/16) отримати розподіл 3 до 1 або краще.

Ще можливо обрати п’ять елементів навмання та вяти медіану як опорну, шанси на розподіл у гіршому випадку 3 до 1 покращаться приблизно до 79,3% (203/256).

Якщо взяти медіану із семи випадково обраних елементів, вона досягне приблизно 85,9% (1759/2048).

Медіана із дев'яти - близько 90,2% (59123/65536). Медіана 11 - близько 93,1% (488293/524288).

Причому, не обов’язково обирати випадковим чином велику кількість елементів і брати їх медіану, оскільки час, витрачений на це, може звести нанівець переваги отримання хороших розподілів.

Таблиця 1.1 - Збільшення шансу отримати розподіл, який у гіршому випадку буде 3 до 1

Кількість елементів для вибору медіани	Ймовірність шансу (%)	Співвідношення ймовірності шансу
3	68,750	11 / 16
5	79,297	203 / 256
7	85,889	1 759 / 2 048
9	90,215	59 123 / 65 536
11	93,135	488 293 / 524 288

1.5 Висновок до розділу 1

Достатньої актуальності набирає питання вибору ефективних методів і алгоритмів сортування масивів даних.

Використання паралельного процесу сортування є головним способом збільшення швидкості сортування масивів даних.

У якості алгоритму сортування краще за все використовувати алгоритм швидкого сортування, який має прийнятні показники: невелика обчислювальна складності алгоритму, найбільш швидкодіючий алгоритм внутрішнього сортування, простий для розуміння і реалізації, потребує небагато пам’яті, ефективний для сортування великої кількості даних.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ШВИДКОГО СОРТУВАННЯ МАСИВІВ ДАНИХ З ВИКОРИСТАННЯМ OPEN MPI

2.1 Обгрунтування вибору Open MPI для інформаційної технології швидкого сортування масивів даних

Для систем на платформах розподіленої пам’яті (на основі розподілених систем) широко використовується технологія MPI [24, 25].

Назва технології MPI означає «інтерфейс для передачі повідомлень» або просто «інтерфейс передачі повідомлень». Він є чи не найголовнішим інтерфейсом у таких системах, в яких відбувається обмін повідомленнями.

Ця технологія широко використовується в багатьох реалізаціях, а саме: WMPI, MPICH, Open MPI тощо. Опис розрахунків, наведений у MPI, дозволяє реалізувати і інші технології.

Технологія MPI дозволяє писати програми на основі двох відомих принципах розпаралелювання: MIMD та SIMD. Використання першої концепції не настільки популярно у порівнянні з другим принципом. Це пов’язано з тим, що SIMD-принцип вимагає виконувати однаковий код для усіх процесів, які працюють паралельно, а MIMD-концепція вимагає одночасну роботу різних кодів на різних процесах.

Для використання MPI-технології необхідно інсталювати та використати додаткове програмне забезпечення, яке зумовлює співпрацю паралельних процесів між собою.

В деяких випадках встановлення додаткового програмного забезпечення вимагає тільки основний комп’ютер.

Об’єднання процесів, які паралельно співпрацюють між собою, і є програмою з використанням MPI-технології.

Якщо у нашого комп’ютерного процесора декілька ядер, то такі

процеси можуть виконуватися на ньому одному.

Відправка повідомлень використовується процесами для спілкування з іншими процесами.

Комунікатор обмежує рамки співпраці процесів один з одним.

Ось чому для кожного процеса передбачено дві ознаки:

1) номер у рамках, які окреслює комунікатор,
2) безпосередньо комунікатор.

Для кожного процесу у групі передбачено свій ідентифікатор.

Повідомлення ідентифікуються такими основними ознаками:

1) назва комунікатора,
2) номер повідомлення,
3) номер процесу, який отримує повідомлення,
4) номер процесу, який відправляє повідомлення тощо.

Для обміну між процесами використовуються два основних типи повідомлень.

По-перше, треба організувати спілкування двох різних процесів. Спілкування отримало назву «точка-точка» (point-to-point). При такому типі спілкування отримувач повідомлення отримує повідомлення, а відправник відправляє повідомлення. Лише відправник з свого боку, а отримувач з свого, приймають участь у цьому обміні.

Обмін типу «точка-точка» може бути:

1) без блокування операції,
2) з блокуванням операції.

По-друге, треба організувати одночасне спілкування між множиною процесів, які підключені до комунікатора. Спілкування отримало назву «колективного обміну» (collective communication).

Колективний обмін зручно використовувати для надання однакового контенту усім процесам одночасно тощо. Цей тип обміну може бути тільки з блокуванням операцій.

Розглянемо технологію Open MP [26, 27].

Ця технологія заснована на використанні концепції багатопотоковості.

На початку створюється основний потік. Він додатково може ще створювати потоки, які виконуються паралельно. Після закінчення обрахунків ці додаткові потоки надають результати своїх обрахунків основному потоку.

Сучасна технологія Open MP дозволяє писати коди мовами Fortran, C та C++.

Також технологія Open MP має можливість та широко поєднується з різними паралельними технологіями. Наприклад, спільне використання технології Open MPI та технології OpenMP.

До складу середовища OpenMP входять такі структурні блоки. По-перше, змінні оточення. По-друге, набір функцій бібліотеки. По-третє, набір директив транслятору.

В технології OpenMP головним представленням розпаралелювання виступають диретиви транслятору.

В OpenMP використовується модель паралельного виконання «розгалуження-злиття».

Ілюстрація багатопоточності, де основний потік відгалужує кілька потоків, які виконують блоки коду паралельно надано на рисунку 2.1 [26, 27].

Існує три основних способа для контролю кількості потоків у групі.

По-перше, поєднання конструкції num_threads з директивою parallel.

По-друге, використовуючи процедуру omp_set_num_threads().

По-третє, за допомогою змінної OMP_NUM_THREADS

Існують два основних типи змінних в програмі OpenMP:

1) private-змінні або індивідуальні (локальні),
2) shared-змінні або спільні (глобальні).

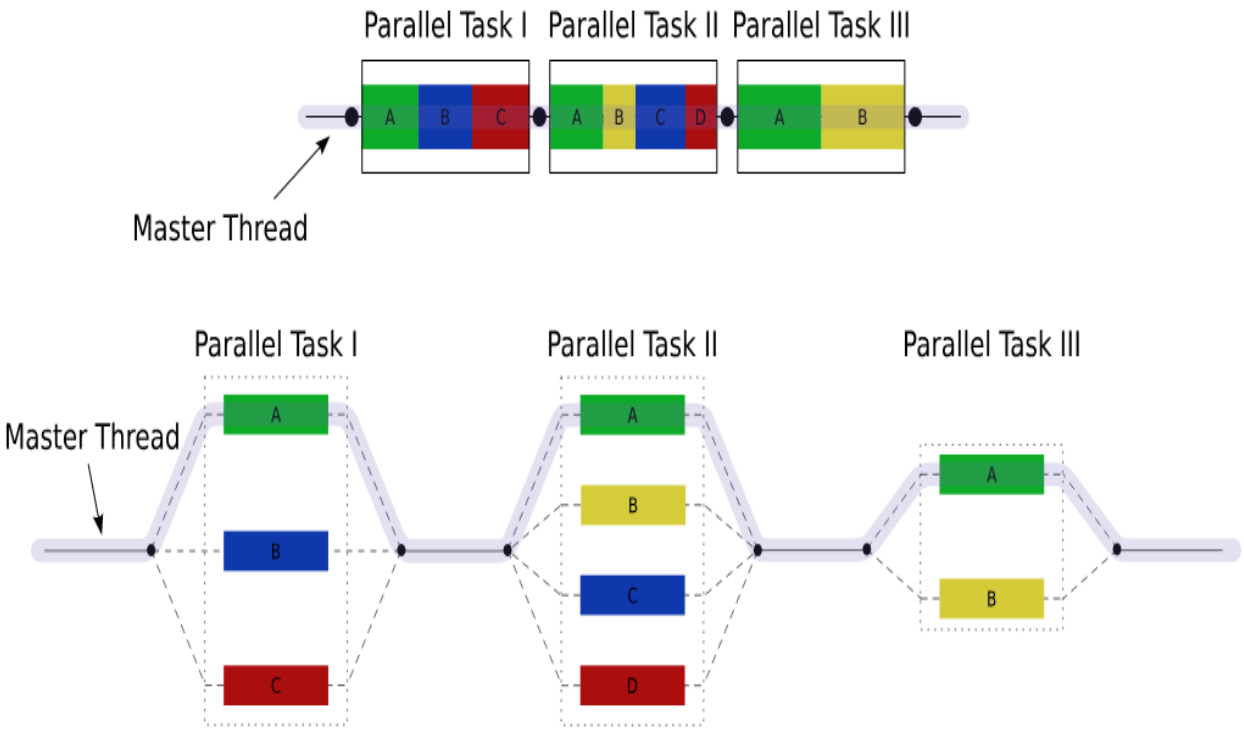

Рисунок 2.1 – Модель виконання OpenMP-програми

Змінні типу shared можна використовувати в межах однієї групи множиною потоків, як для запису, так і для зчитування.

Змінні типу private можуть використовуватися тільки одним потоком, якому вони належать.

Іноді можливе виникнення ситуації, коли зміст спільної змінної невизначено. Невизначеність значення спільної змінної виникає в ситуації її модифікації без використання синхронізації процесів запису та зчитування спільної змінної.

Усіма потоками одночасно можуть бути використані змінні типу shared. Змінними цього типу рахуються, як правило, усі змінні в OpenMP за замовчуванням.

Окрім спільних та приватних змінних існують ще змінні-лічильники, які підраховують цикли, і за допомогою циклів PARALLEL FOR та FOR

розташовуються між нитями.

На рисунку 2.2 представлено випадок належності змінних до спільної або до неспільної пам’яті.

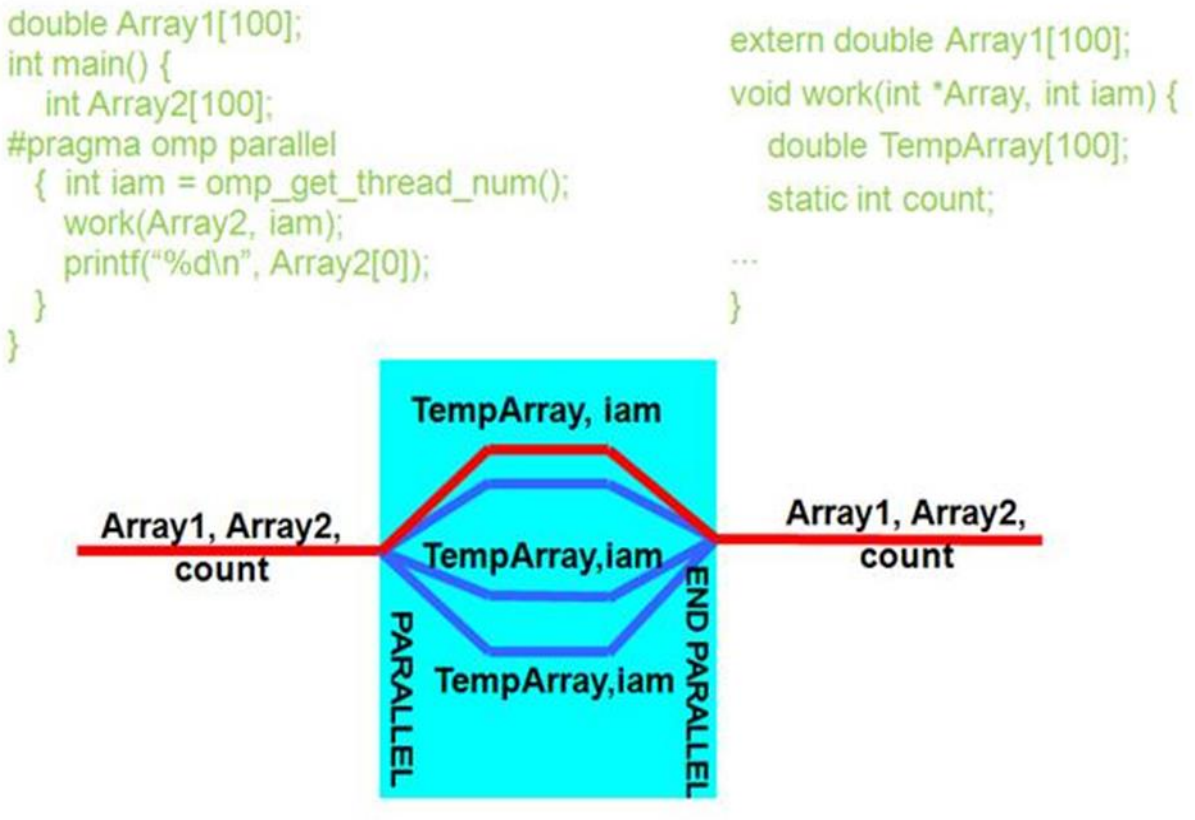

Рисунок 2.2 – Приклад розташування змінних в пам’яті

Як видно з рисунку 2.2, змінні TempArray,iam – це змінні типу private (локальні), змінні count, Array1 та Array2 – це змінні типу shared (глобальні). Причому, значення змінних TempArray,iam можуть бути різними на різних потоках.

З використанням нижченаведених конструкцій є можливість замінити клас змінних:

1) конструкцій DEFAULT (PRIVATE | SHARED | NONE);
2) конструкцій THREADPRIVATE (список змінних);
3) конструкцій LASTPRIVATE (список змінних);

4) конструкцій FIRSTPRIVATE (список змінних);
5) конструкцій PRIVATE (список змінних);
6) конструкція SHARED (список змінних).

Основними елементами Open MP є конструкції для створення потоків, розподілу робочого навантаження (розподілу роботи), керування середовищем даних, синхронізація потоків, процедури виконання на рівні користувача та змінні середовища (рис.2.3).

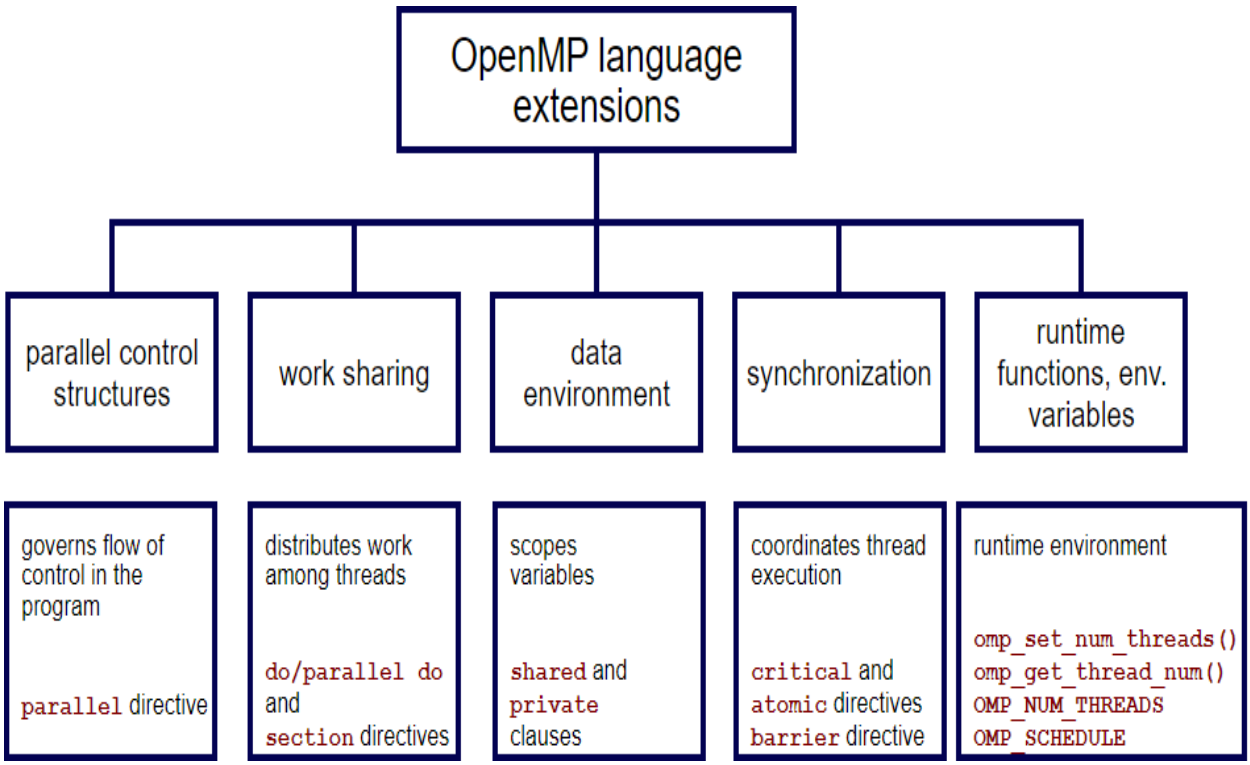

Рисунок 2.3 - Діаграма конструкцій OpenMP

Open MPI Project - це реалізація інтерфейсу передачі повідомлень з відкритим кодом , яка розроблена та підтримується консорціумом академічних, дослідницьких і промислових партнерів. Таким чином, Open MPI може поєднати досвід, технології та ресурси з усього співтовариства високопродуктивних обчислень, щоб створити найкращу доступну бібліотеку MPI. Open MPI пропонує переваги для постачальників систем і

програмного забезпечення, розробників додатків і дослідників інформатики. Функції, реалізовані або розробляються в короткостроковій перспективі для Open MPI, включають:

- повна відповідність стандартам MPI-3.1;
- безпека потоків і паралельність;
- динамічний нерест процесу;
- відмовостійкість мережі та процесів;
- підтримка неоднорідності мережі;
- єдина бібліотека підтримує всі мережі;
- інструменти виконання;
- підтримується багато планувальників завдань;
- підтримується багато ОС (32 і 64 біт);
- програмне забезпечення якості виробництва;
- висока продуктивність на всіх платформах;
- портативний і ремонтопридатний;
- налаштовується інсталяторами та кінцевими користувачами;
- компонентний дизайн, задокументовані API.

Для виконання паралельних дій було використано можливості технології MPI (Open MPI).

На рисунку 2.4 надано приклад процесу MPI COMM, що містить кілька вузлів у чотирьох кластерах, показує, як надається ранг кожному процесору [28].

Усі процесори P не мають жодної топологічної сутності один з одним і є рівними в усіх відношеннях. Вони з'єднані за допомогою єдиної мережі. Ця базова модель машини охоплює всі можливі паралельні машини. Все ще не представлено багато характеристик реальних паралельних систем, що стосуються продуктивності: кластерна структура SMP, ієрархія спільної пам’яті, спільний мережевий інтерфейс, топологія мережі.

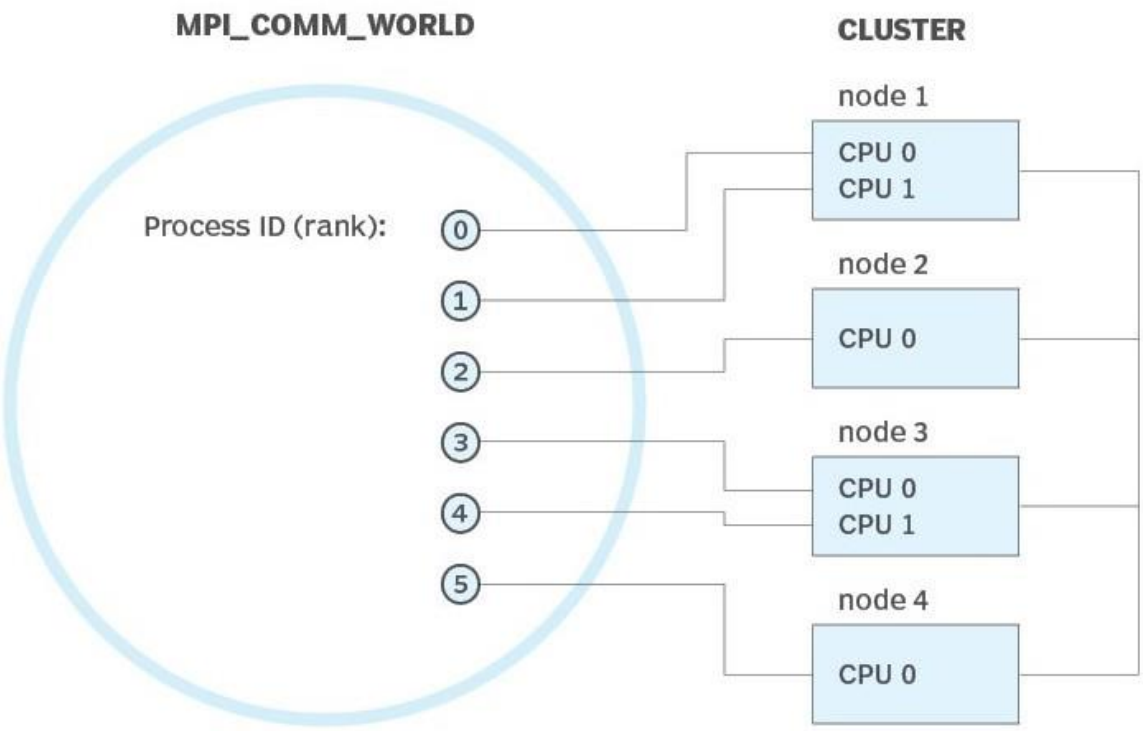

Рисунок 2.4 - Приклад процесу MPI

Модель машини MPI представлена на рисунку 2.5 [24, 29-31].

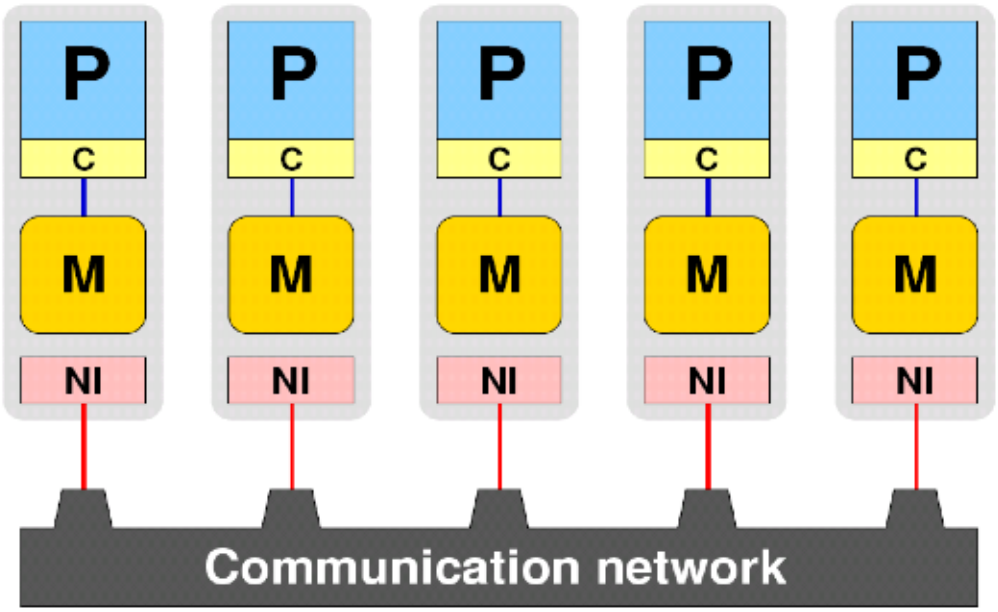

Рисунок 2.5 – Модель машини MPI

У той час як оптимізовані реалізації MPI намагаються забезпечити найкращу можливу продуктивність для кожного випадку зв'язку, той факт, що зв'язок між двома партнерами не є рівноправним через комунікатор MPI, не може бути вирішений і програміст має чітко розглянути це питання.

На рисунку 2.6 надано структуру відкритих процесів MPI та програмний стек MPI [32]:

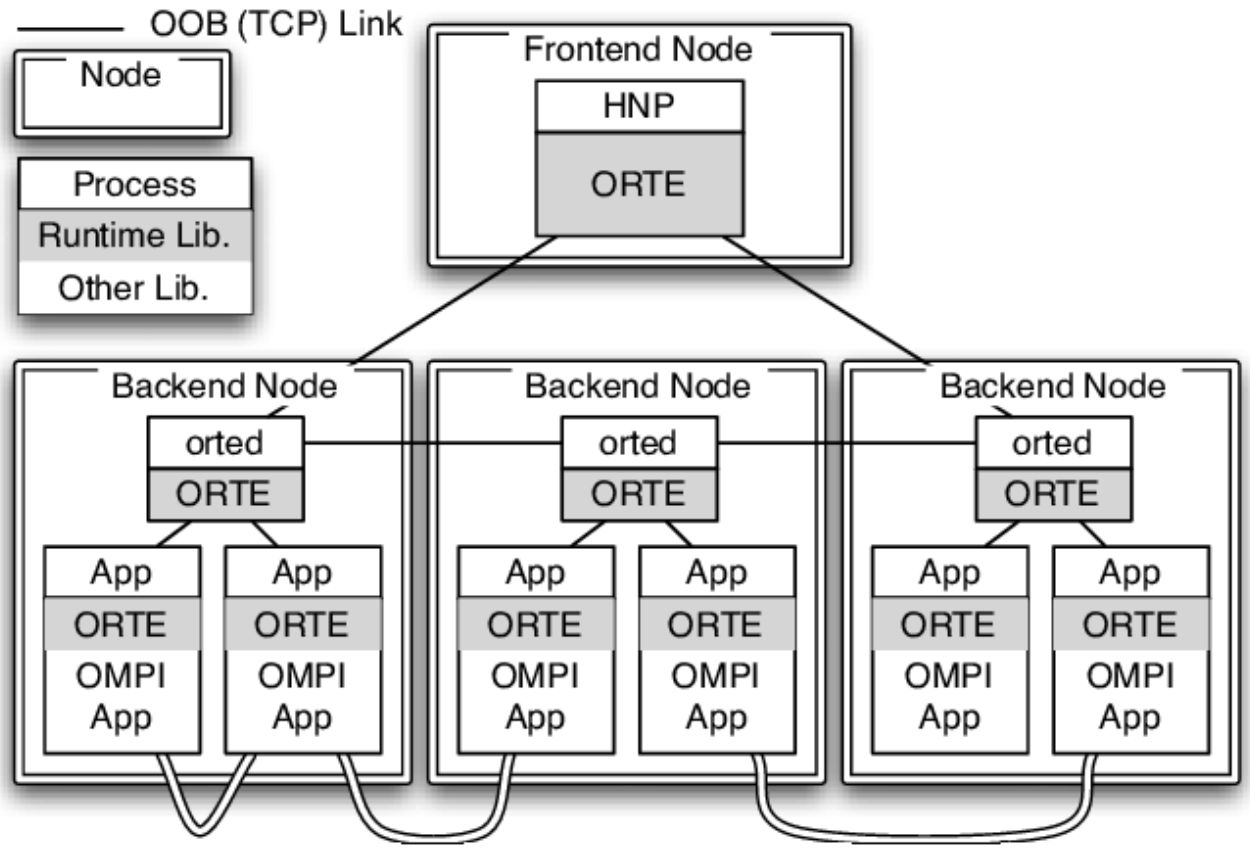

Рисунок 2.6 - Відкриті процеси MPI та програмний стек

Можна виділити три типи процесів, які взаємодіють разом для реалізації середовища виконання Open MPI. Малюнок 1 дає зображення цих процесів і їхній програмний стек.

HNP процес головного вузла складається виключно з часу виконання

середовище: зазвичай це псевдонім команди mpirun, і його основна роль полягає в розгортанні та координації демонів orte (ортований) на паралельній машині. Цей процес зазвичай запускається користувачем або пакетним планувальником на зовнішній машині зі списком обчислювальних машин, які слугуватимуть підтримці паралельної програми.

ORTE Daemons — це процеси середовища виконання: вони є ланкою між HNP і додатком процеси. Один з таких ортів запускається на обчислювальний вузол, що належить до виконання.

App (додатки) MPI – це процеси, які пов’язані з бібліотеками. Відкрита бібліотека MPI (яка реалізує стандарт MPI), та бібліотека ORTE. Для повідомлень MPI вони безпосередньо спілкуватися один з одним. Для повідомлень поза діапазоном, вони передають інформацію через рівень ORTE з локальним ортованим процесом (наприклад, пересилання стандартного вихідного повідомлення до команди mpirun або отримання порту, на якому можна встановити пряме MPI-з'єднання). Є така ж необхідна кулькість mpiapp процесів, що обробляється на кожному вузлі, щоб виконати вимоги користувачів. Усі процеси mpiapp на вузлі, який належать до однієї програми MPI, підключеної до того самого процеса.

2.2 Структура та порядок функціонування швидкого сортування масивів

Алгоритм швидкого сортування масивів працює за принципом “розділяй і володарюй”.

Швидке сортування масивів можна представити такими етапами.

Перший етап. Обираємо із масиву опорний елемент (pivot).

Другий етап. Поділяємо вхідний масив на дві частини. Причому, в одній половині повинні знаходитися елементи, які більші за значенням у порівнянні з опорним елементом, а у іншій ті елементи, які менші чи

дорівнюють за опорний елемент.

Третій етап. В кожному із отриманих підмасивів виконуємо рекурсивно другий крок у випадку, коли у підмасивах кількість елементів більше 2-х. Інакше, елементи порівнюються між собою та замінюють один-одного у разі потреби, коли елементів 2-а,

Розглянемо другий етап деталізовано.

У якості вхідного масиву візьмемо такий набір

(17, 12, 11, 18, 16, 13, 15, 14)

та виконаємо алгоритм швидкого сортування поетапно.

Перший крок (рис.2.7).

Перше, що треба зробити – обрати pivot-елемент (опорний елемент). Обираємо заключний елемент масиву – pivot = 14. Будемо використовувати лічильник по i та лічильник по j. Перше значення для j = 0, тоді значення i = -1.

Порівнюємо опорний елемент pivot=14 із j-м елементом масиву, що дорівнює (17). При ситуації, коли j-й елемент масиву більше за pivot-елемент, то збільшуємо лічильник j + 1 .

Отже, лічильник тепер j = 1, опорний елемент pivot=14 із j-м елементом масиву, що дорівнює (14). Якщо він менший за опорний, то збільшуємо лічильник i + 1.

Отримали, що i = 0, замінюємо місцем i-й та j-й елементи масиву. В прикладі замінюємо місцями нульовий і перший елементи масиву, Отже, отримали такий масив (12, 17, 11, 18, 16, 13, 15), опорний елемент pivot=14, та збільшуємо лічильник j.

Другий крок (рис.2.8).

Отримали показник j = 2, порівняємо pivot=14 та (11). Збільшуємо i, i=1, далі замінюємо місцями i-й та j-й елементи масиву (17) та (11), отримали такий масив (12, 11, 17, 18, 16, 13, 15), опорний елемент pivot=14. Інкрементуємо лічильник j.

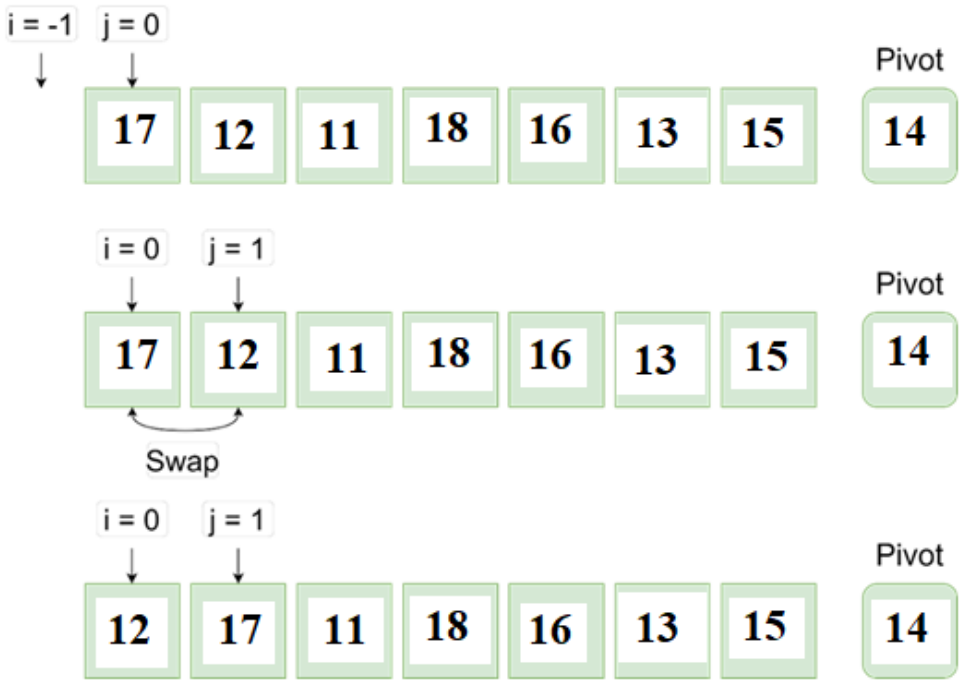

Рисунок 2.7 – Перший крок швидкого сортування

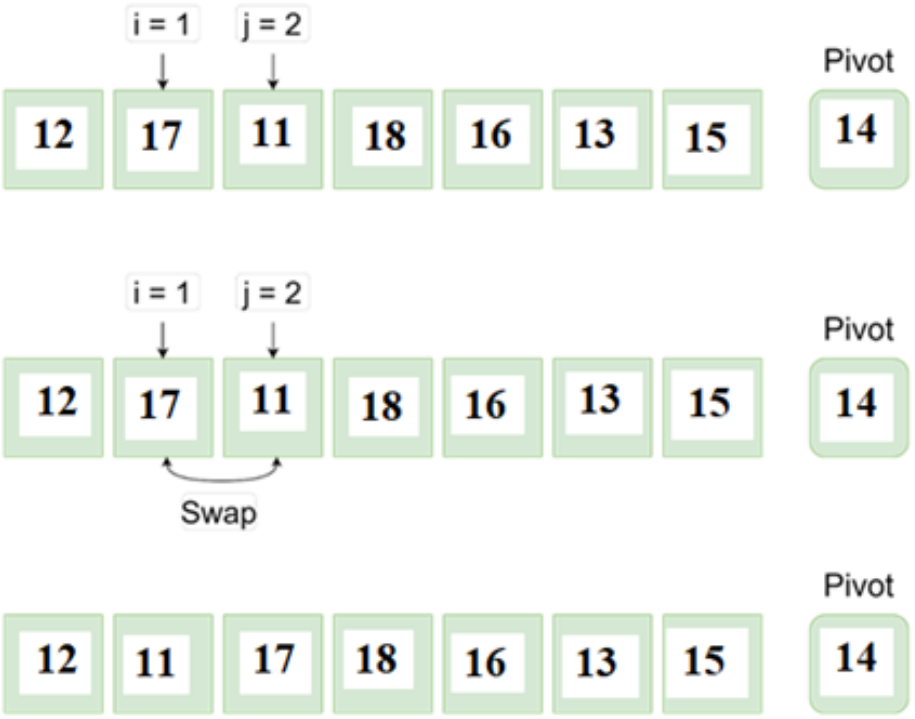

Рисунок 2.8 – Другий крок швидкого сортування

Третій крок (рис.2.9).

Значення лічильника j=3, порівнюємо pivot=14 з елементом масиву (18). Збільшуємо тільки j.

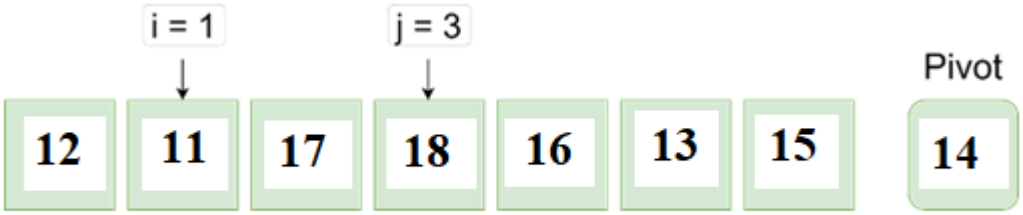

Рисунок 2.9 – Третій крок швидкого сортування

Четвертий крок (рис.2.10).

Значеннння j=4, порівнюємо pivot=14 з елементом масиву (16). Відбувається збільшення лічильника j.

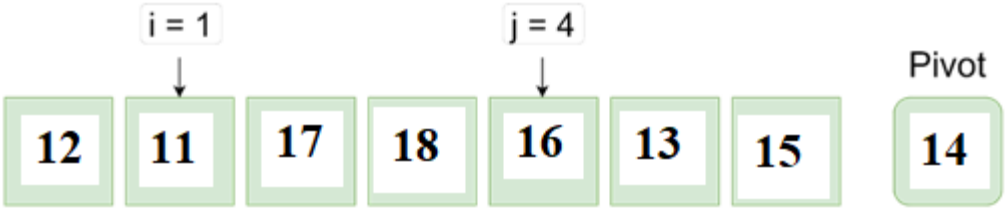

Рисунок 2.10 – Четвертий крок швидкого сортування

Значення j=5, порівнюємо pivot=14 з елементом масива (13). Зьільшуємо i, i = 2 та замінюємо місцями і-й та j-й елементи масиву (17) та (13). Отже, маємо новий масив (12, 11, 13, 18, 16, 17, 15), опорний елемент pivot=14 та інкрементуємо лічильник j.

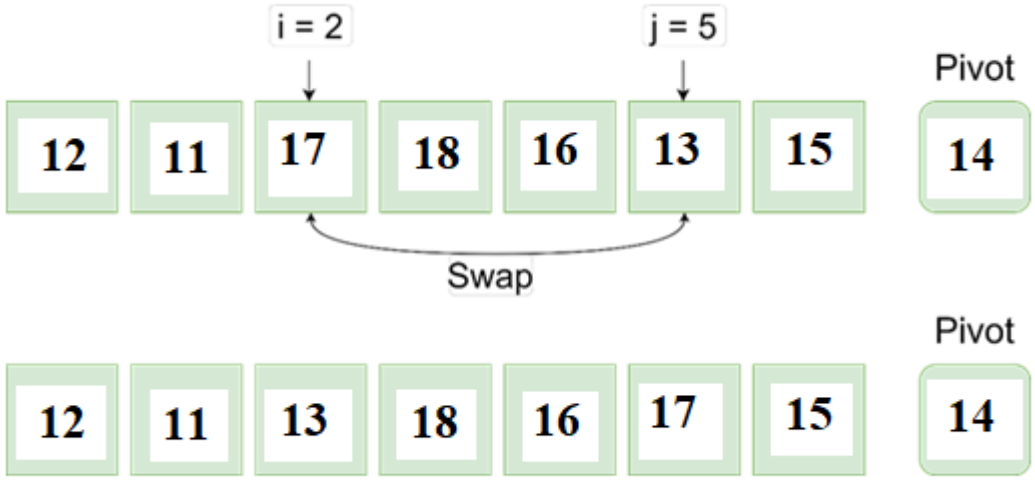

Рисунок 2.11 – П’ятий крок швидкого сортування

Шостий крок (рис.2.12).

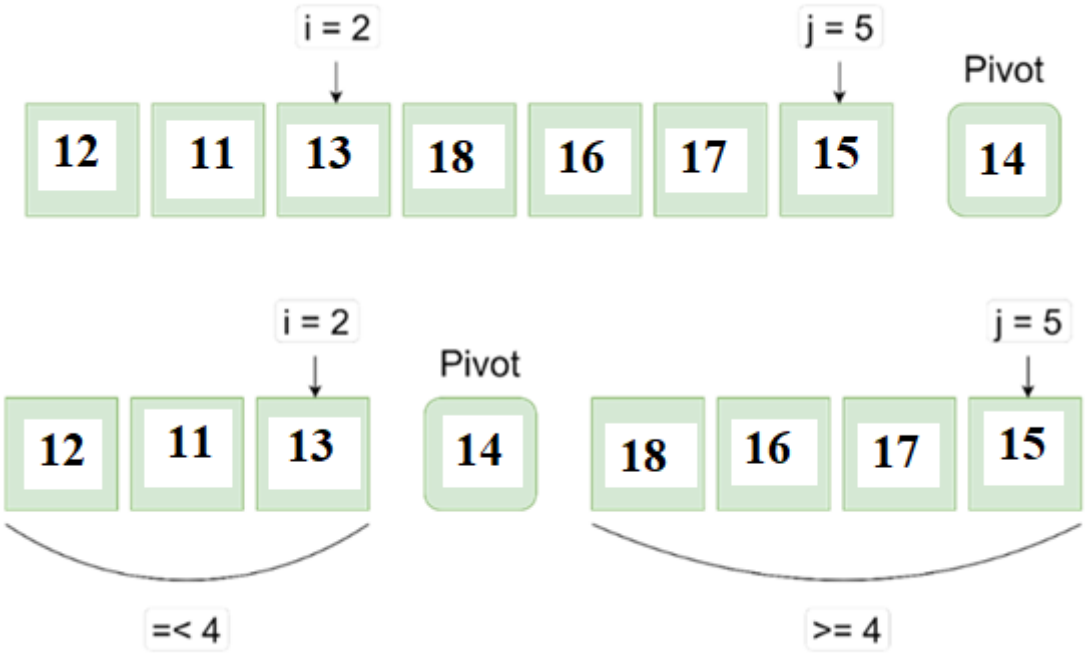

Рисунок 2.12 – Шостий крок швидкого сортування

Значення j=6, порівнюємо pivot=14 з елементом масиву (15). Інкрементацію j не робимо, томущо знаходимося в кінці масива.

Єдине, що треба зробити, це визначити позицію в масиві для нашого опорного елемента pivot=14. Індекс і збільшуємо. Маємо таку умову

$$(12, 11, 13) \leq (14) < (18, 16, 17, 15).$$

У кінцевих підмасивах три та чотири елементи, у кожному більше двох. Тому виконуємо сортування в цих двох підмасивах. Упід масиві, в якому елементи більші за опорний елемент pivot=14: (18, 16, 17, 15). Та у під масиві, у якому елементи менші чи дорівнюють опорному елементу pivot=14: (12, 11, 13). Сортуваня виконуємо до отримання результата.

Результат сортування – це масив

$$(11, 12, 13, 14, 15, 16, 17, 18).$$

Більш наочний приклад швидкого сортування надано на рисунку 2.13.

У різних реалізаціях використовуються різноманітні стратегії по обиранню pivot:

- перший елемент;
- останній елемент;
- середній елемент;
- випадковий елемент;
- медіана трьох, п'яти або більше елементів.

Якщо опорний елемент обирається одним із цих способів, збільшується ймовірність того, що підмасиви, отримані в результаті розбиття, будуть якомога однаковими, хоча найчастіше в реалізації використовується стратегії перший чи останній елемент. Також як один із варіантів покращення якості обраного опорного елементу, можна обрати у якості опорного елемента середній елемент із відсортованої трійки елементів середнього, останнього та першого елементів масиву.

Використання медіани є не практичним, оскільки щоб визначити медіану, масив спочатку потрібно відсортувати.

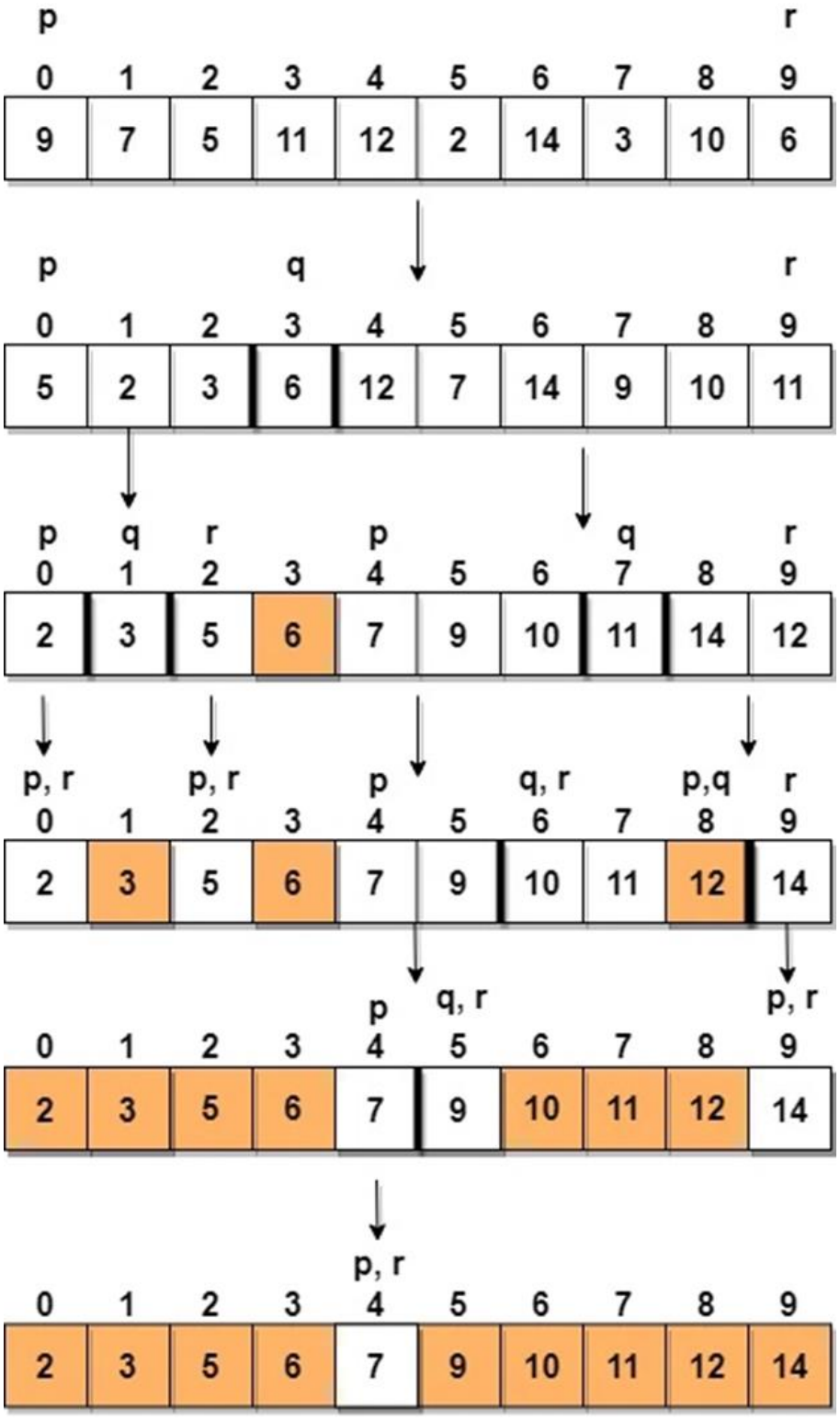

Рисунок 2.13 – Приклад швидкого сортування

2.3 Структура інформаційної технології швидкого сортування масивів даних з використанням Open MPI

Швидке сортування можна розпаралелювати різними способами.

Розглянемо пряме паралельне сортування. Масив розбивається на дві частини, і кожна частина розв'язується рекурсивно. Сортування менших масивів представляє дві повністю незалежніпідзадачі, які можна вирішувати паралельно. Таким чином, один із способів розпаралелити швидке сортування містить наступні етапи:

- виконання сортування в одному процесі;
- виконання алгоритмом рекурсивних викликів та призначення одного з підмасивів іншому процесу.

Тепер кожен із цих процесів сортує свій масив за допомогою швидкого сортування та призначає одну зі своїх підзадач іншим процесам. Алгоритм припиняє роботу, коли масиви не можуть бути розділені далі. Після завершення кожен процес зберігає елемент масиву, і відсортований порядок можна відновити шляхом обходу процесів.

Це паралельне формулювання швидкого сортування використовує n процесів для сортування n елементів. Його основним недоліком є те, що розділення масиву A[q ...r] на два менших масиви, A[q ...s] і A[s + 1 ...r], виконується одним процесом. Оскільки один процес повинен розділити вихідний масив A[1 ...n]. Ось чому з точки зору оптимальності витрат цей варіант не є оптимальним.

Основним обмеженням попереднього підходу є те, що етап розділення виконується послідовно. Виконання паралельного розділення є важливим для отримання ефективного паралельного швидкого сортування. Однак це важко для більшості моделей паралельних обчислень. Єдиними відомими алгоритмами є алгоритми для абстрактних моделей PRAM. Через накладні витрати на зв'язок, в реальних випробуваннях етап розділення займає більше

часу, ніж Q(1) на ПК зі спільним адресним простором і паралельною передачею повідомлень. Отже, розпаралелювання етапу розділення має потенціал для отримання значно швидшого паралельного алгоритму.

Результат синтезу потокового графу паралельного алгоритму надано на рисунку 2.14.

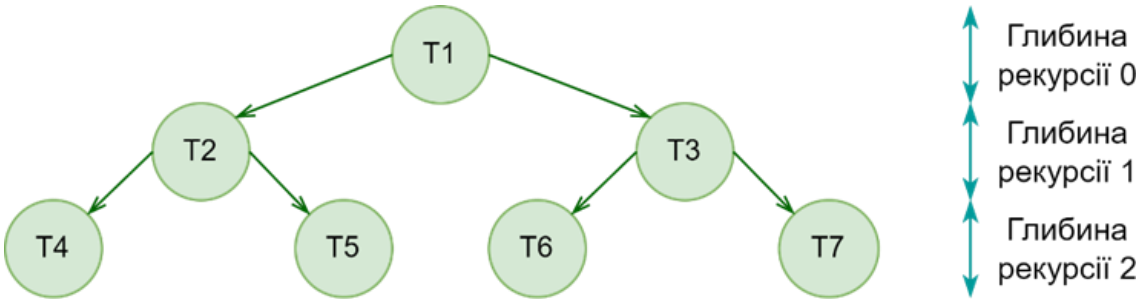

Рисунок 2.14 – Потоковий граф для двоядерного процесора

Результат синтезу потокового графу прикладу паралельного швидкого сортування надано на рисунку 2.15.

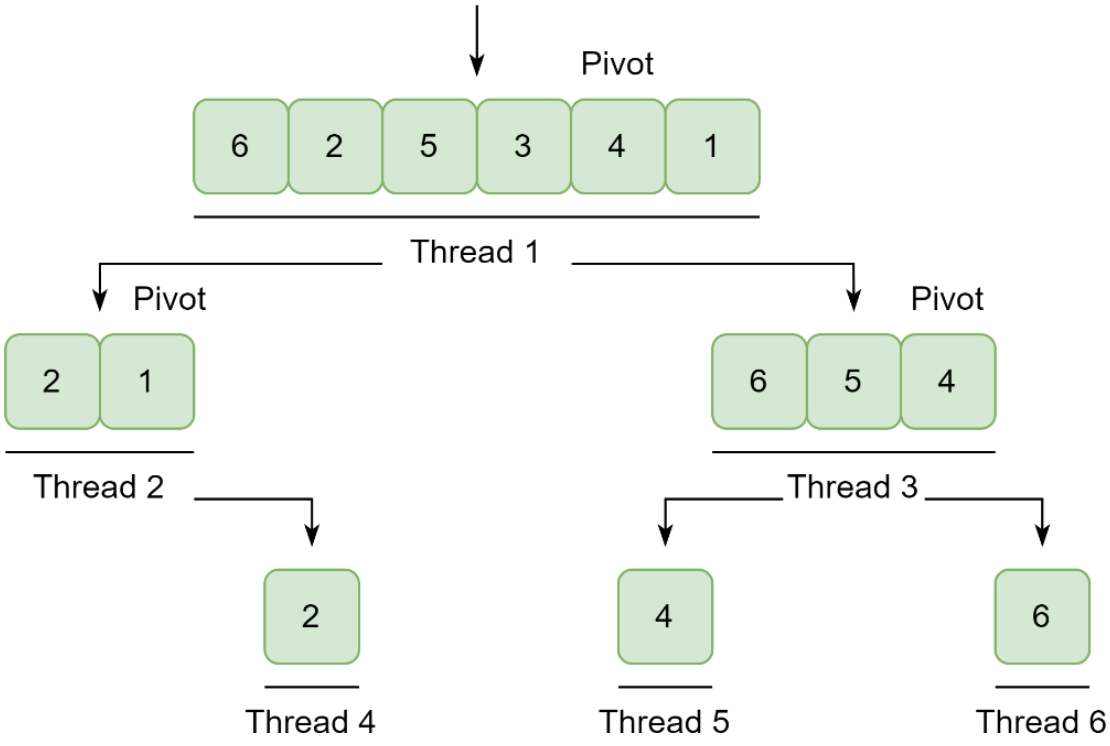

Рисунок 2.15 – Потоковий граф паралельного швидкого сортування

2.4 Розробка алгоритму роботи програмного забезпечення швидкого сортування масивів даних з використанням Open MPI

На початку роботи паралельного алгоритму відбувається побудова паралельних процесів. Кожний із процесів виконує свої обчислення над елементами свого масиву. Після закінчення обрахунків головний процес отримує від допоміжних результати та складає їх в остаточний.

На рисунку 2.15 представлена послідовність виконання алгоритму швидкого сортування [33].

Останній стан є прийнятним станом, оскільки це остання дія в бажаній послідовності виконання. Тобто, спостерігаємо послідовність виконання, де спочатку виконується метод Sorter.sort(), потім, виконується метод quickSort.Sort(), потім повертається метод Sorter.sort().

2.5 Висновок до розділу 2

У результаті проведених теоретичних та аналітичних досліджень можна зробити висновок про те, що для збільшення швидкодії сортування масивів даних доцільно обрати реалізацію алгоритма паралельного швидкого сортування з використанням технології OPEN MPI.

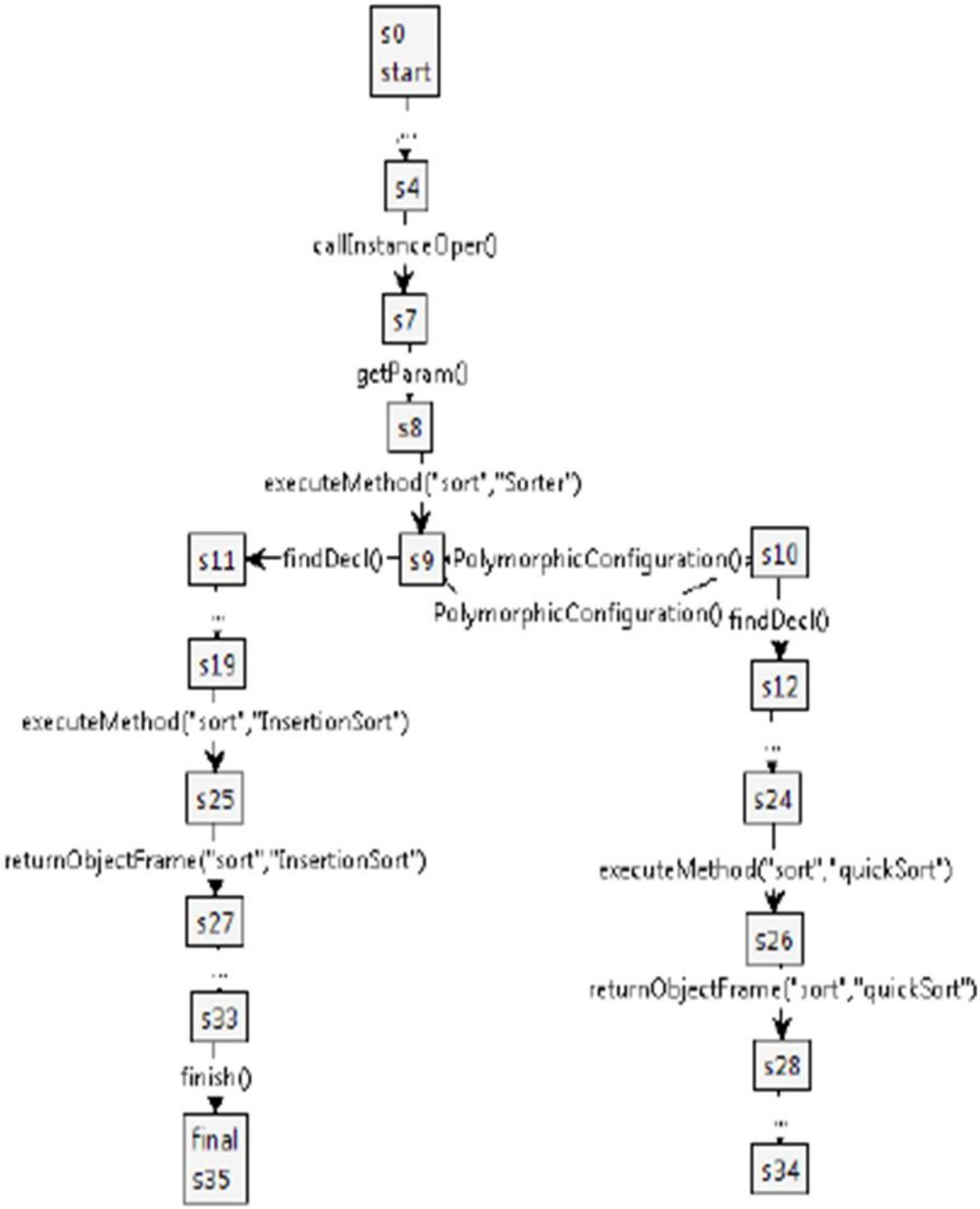

Рисунок 2.15 - Послідовність виконання алгоритму швидкого сортування

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ШВИДКОГО СОРТУВАННЯ МАСИВІВ ДАНИХ З ВИКОРИСТАННЯМ OPEN MPI

3.1 Обгрунтування вибору мови програмування та засобів розробки

Алгоритми сортування використовуються для впорядкування невпорядкованих даних будь-якого типу в певному порядку. Це дуже важливий аспект розробки програмного забезпечення, оскільки він дозволяє розробникам упорядковувати дані, порівнювати вміст і виконувати інші необхідні завдання. Популярні алгоритми сортування включають Quicksort, Insertion Sort, Heapsort і Bubble Sort, які використовуються для перевпорядкування елементів у масиві. Для їх реалізації можна використовувати різні мови програмування.

Мова програмування C++ була обрана для реалізації розробленого алгоритма. C++ подолала складність написання кодів програм складної структури [35-37]. Вона була створена Б'ярном Страуструпом в 1979 р. в компанії AT&T Bell Laboratories.

Не зважаючи на швидке зростання потужностей комп’ютерів, кількість задач, де стане в нагоді заощадження кожного такту процесора та кожного байта пам’яті стає більше. В наукових та інженерних обчислення, в задачах штучного інтелекту.

C++ є унікальною мовою.

На тепер мова програмування C++ сполучує виразність, простоту та високорівневий код із малою платою за абстракції, дозволяє самим обирати, наскільки заглиблюватися в подробиці. C++ надає відповідні інструменти: переваги ООП, функціональний стиль, узагальнене програмування, метапрограмування.

Зворотною стороною потужності паралельних систем та гнучкістю C++, є дещо велика складність в її освоєнні [35-37]..

Розбиття Гоара залишається одним із найефективніших, коли мова йде про загальне сортування [2, 3, 10, 16, 21, 38]. Приклад рекурсивної реалізації розбиття Гоара мовою C++ надано на рисунку 3.1 [39].

```
typedef std::ptrdiff_t Index;
template <class Iterator, class Comparison =
std::less<>>

void HoareQuickSort(Iterator a, Iterator finish,
Comparison compare = Comparison())

{
    Index right = (finish - a) - 1;
    Index left = 0;
    if (right <= left)
        return;
    auto pivot = a[left + (right - left) / 2];
    Index i = left;
    Index j = right;
    while (i <= j)

    {
        while (compare(a[i], pivot))
            ++i;
        while (compare(pivot, a[j]))
            --j;
        if (i <= j)
        {
            std::swap(a[i++], a[j--]);
        }
    }

    HoareQuickSort(a, a + (j + 1), compare);
    HoareQuickSort(a + i, finish, compare);
}
```

Рисунок 3.1 - Реалізація розбиття Гоара мовою C++

3.2 Програмна реалізація модуля швидкого сортування масивів даних з використанням Open MPI

Реалізовувати будемо відомий алгоритм швидкого сортування (рис.3.2) [2, 3].

На рисунку 3.3 надано приклад UML-діаграми діяльності швидкого сортування масиву даних [6, 34].

Стає цілком розумним використання технологій паралельного програмування із мовою C++ та бібліотекою MPI, що надає низькорівневий, але зручний інтерфейс програмування [39 - 41].

Було використано такі бібліотеки C++:

- #include<mpi.h> - реалізує паралельні обчислення за допомогою багатопоточності, в якому провідний потік створює набір відомих потоків і завдання розподіляється між ними;
- #include<iostream> - для організації введення-виведення для стандартної консолі в мові програмування C++;
- #include<chrono> та #include<time.h> - для використання таймера та відстеження часу виконання алгоритму;
- #include<random> - для генерації випадкових чисел.

Варіант вибору опорної точки надано на рисунку 3.4.

Функція quickSortParallel, яка визначає кількість потоків та проводить сортування представлена на рисунку 3.5, де <#pragma omp parallel sections> визначає паралельну область, що містить код, який ми будемо виконувати, використовуючи декілька потоків паралельно. Цей код буде розділено між усіма потоками.

Лістинг програми швидкого сортування масивів даних з використанням Open MPI наведено у додатку Б.

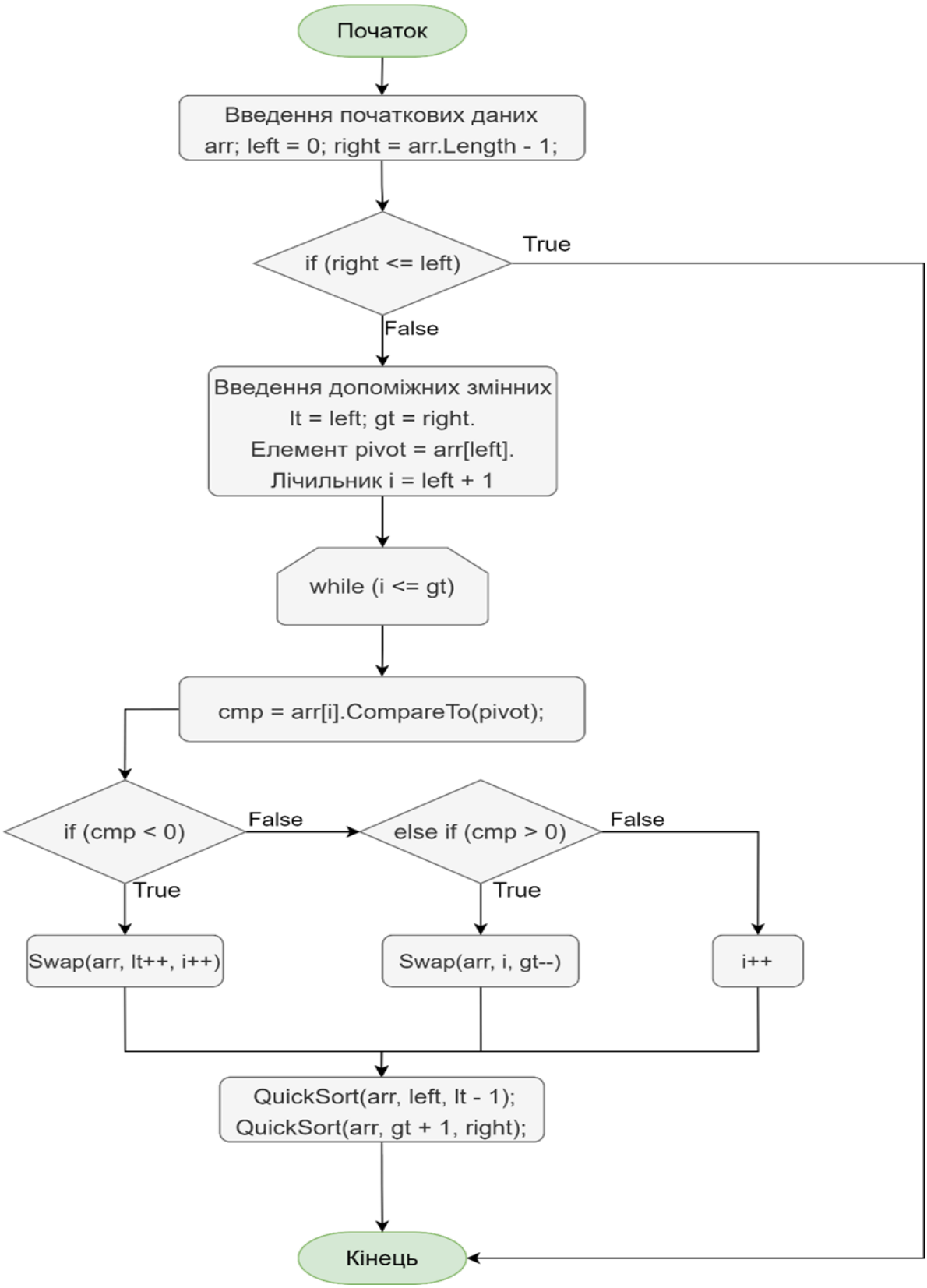

Рисунок 3.2 – Граф-схема алгоритм швидкого сортування

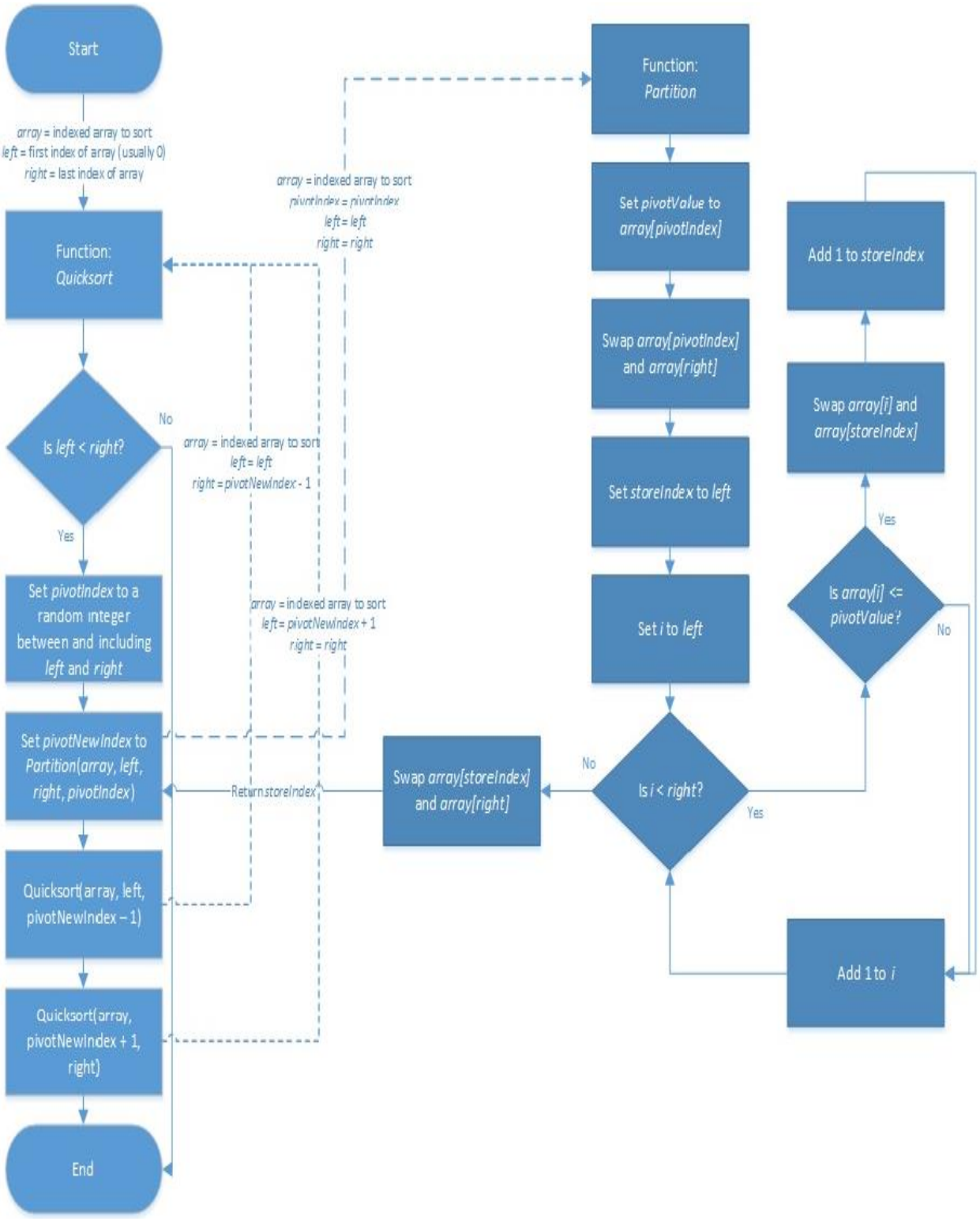

Рисунок 3.3 - Приклад UML-діаграми діяльності швидкого сортування

```
  private:
//partitioning procedure
    int partitionParallel(int arr[], int l, int r){
      int i = l + 1;
      int j = r;
      int key = arr[l];
      int temp;
      while(true){
        while(i < r && key >=
          arr[i])
        i++;
        while(key <
          arr[j]) j--;
        if(i < j){
          temp = arr[i];
          arr[i] = arr[j];
          arr[j] = temp;
        }
        else{
          temp = arr[l];
          arr[l] = arr[j];
          arr[j] = temp;
          return j;
        }
      }
    }
```

Рисунок 3.4 – Вибір опорної точки

```
public:
  void quickSortParallel(int arr[], int l,
    int r){ if(l < r){
      int p = partitionParallel(arr, l, r);
       cout <<"pivot"<< p << "found by thread no."
<<k<< endl;

      #pragma omp parallel sections
      {
        #pragma omp section
        {
          k = k + 1;
          quickSortParallel(arr, l, p-1);
        }
        #pragma omp section
        {
          k = k + 1;
          quickSortParallel(arr, p+1,
          r);
        }
      }
    }
  }
```

Рисунок 3.5 – Функція quickSortParallel, яка визначає кількість потоків та проводить сортування

3.3 Висновок до розділу 3

Для розробки алгоритму було обрано мову C++. Для виконання паралельних дій було використано технологію Open MPI. . Результатом розділу є готовий паралельний алгоритм швидкого сортування масивів даних з використанням Open MPI та реалізація його мовою C++.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТТІВ РОБОТИ ПРОГРАМИ ШВИДКОГО СОРТУВАННЯ МАСИВІВ ДАНИХ З ВИКОРИСТАННЯМ OPEN MPI

4.1 Процес тестування програми

Тестування програмного забезпечення використовує такі критерії [42].

1. Сумісність з іншим програмним забезпеченням та основними операційними системами.
2. Практичність використання в відповідній комбінації апаратно-програмної платформи.
3. Виконання необхідних функцій за прийнятний для користувача час.
4. Правильна відповідь для усього діапазону можливих вхідних даних.
5. Відповідність вимогам, проектування та розробки.

Процес використання розробленої програми містить такі етапи.

1. Зберегти лістинг коду в файл з розширенням ***.cpp.***
2. Відкрити збережений файл в зручному для користувача редакторі або онлайн компіляторі.
3. Після цього натиснути <Пуск> для запуску програми.
4. В консолі потрібно ввести розмір масиву.
5. Завантажити файл-джерело масиву даних.
6. Після цього отримуємо результат відсортованого масиву даних у файл-результат.
7. Для виходу із програми натиснути будь-яку клавішу.

На рисунку.4.1 зображено тестування на прикладі масиву із 10 елементів. Також виведено в консолі тривалість виконання алгоритмів (паралельного швидкого сортування та звичайного).

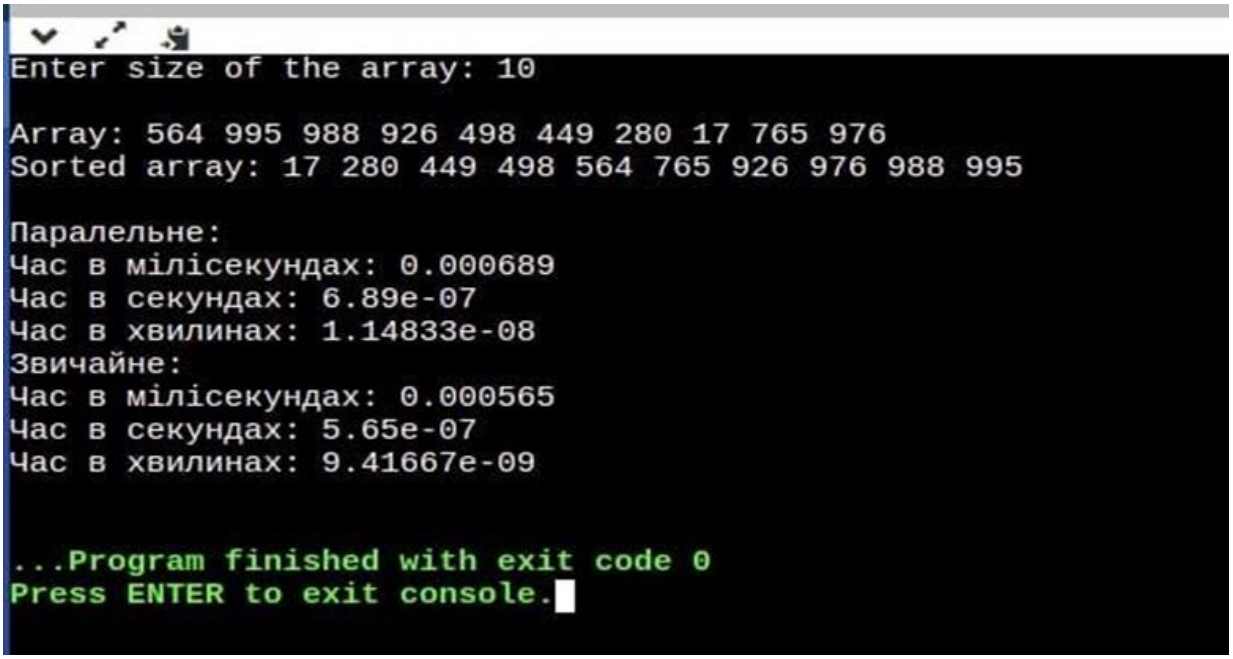

Рисунок 4.1 – Тестування програми із N=10

Тестування на прикладі із розміром масиву 1000 елементів надано на рисунку 4.2.

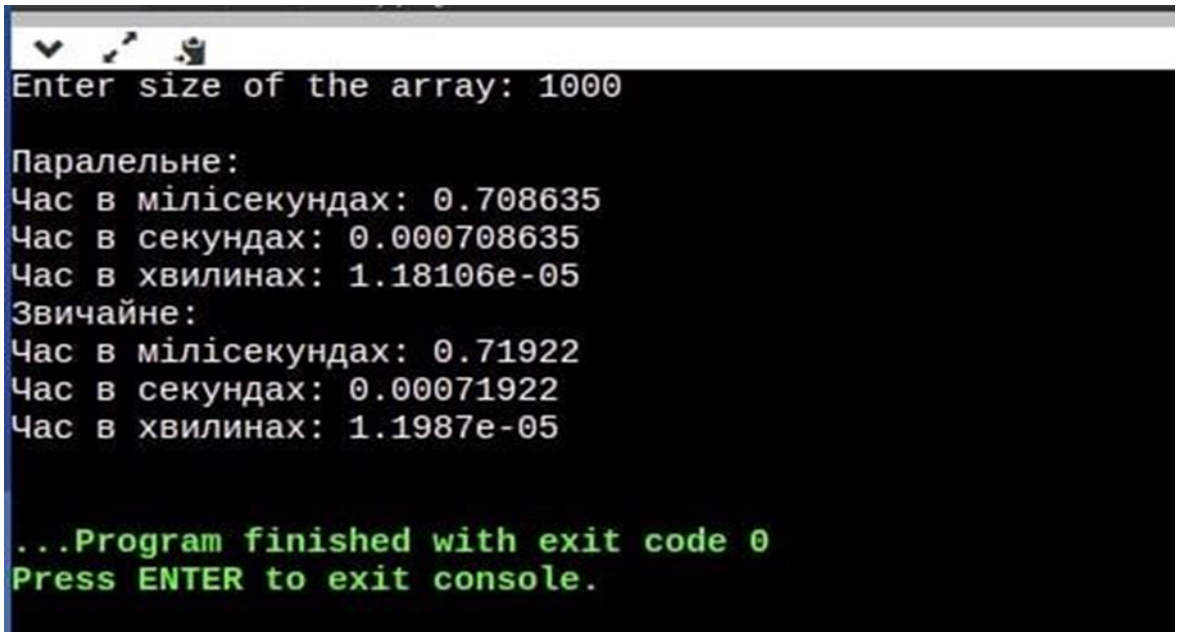

Рисунок 4.2 – Тестування програми із N=1000

Тестування на прикладі із розміром масиву 100 000 елементів надано на рисунку 4.3.

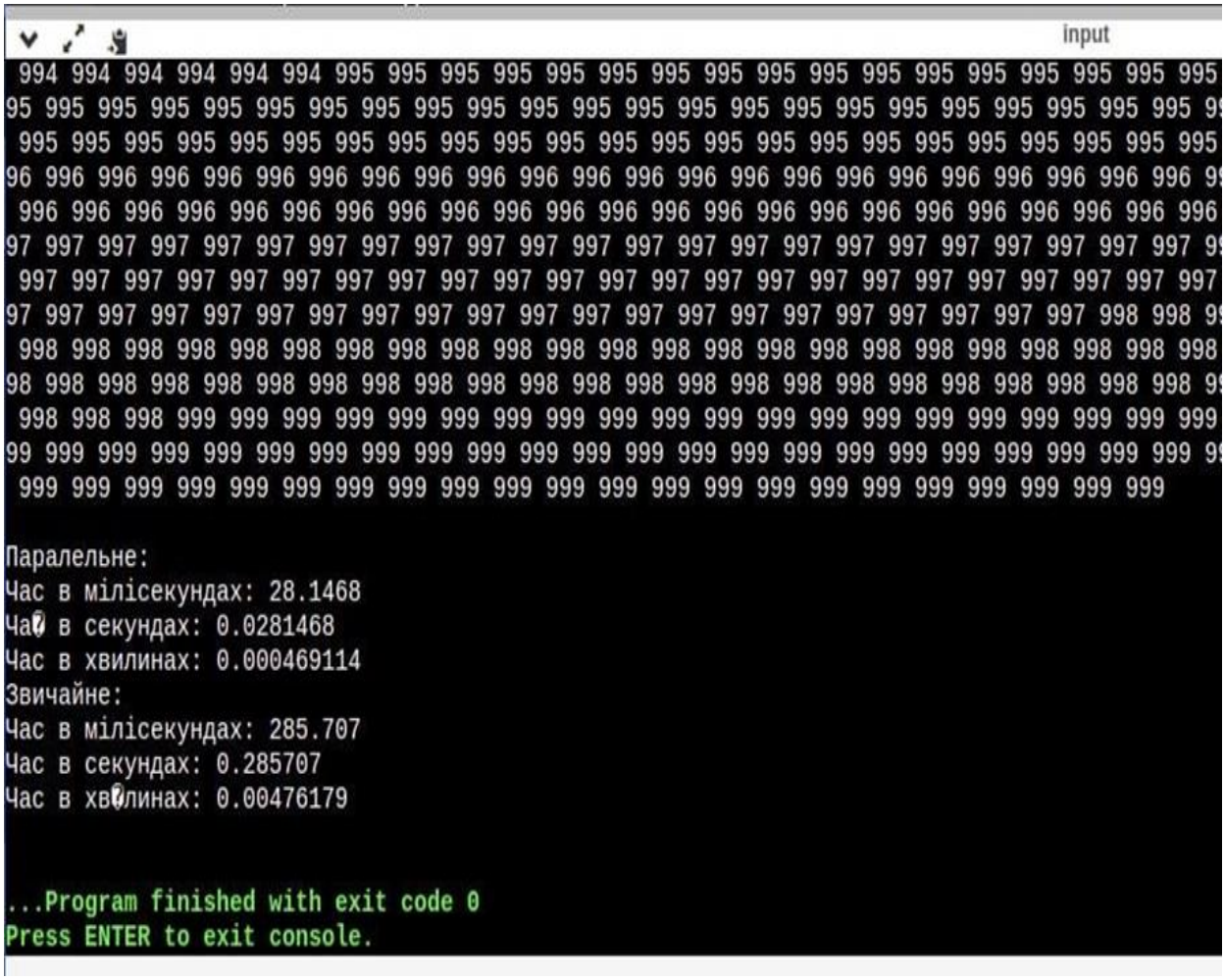

Рисунок 4.3 – Тестування програми із N=100 000

4.2 Аналіз результатів роботи програми швидкого сортування масивів даних з використанням Open MPI

Отримані результати тестування двох алгоритмів сортування масивів для значень n від 10 до 1 000 000, де кожний із елементів масивів змінюється в межах від 1 до 999 (табл.4.1).

Таблиця 4.1 – Час сортування масивів

Єлементів масива, n	Швидке сортування, мс	Паралельне швидке сортування, мс
10	0,000 656	0,000 656
100	0,068 56	0,069265
1 000	0,719 220	0,708635
10 000	81,135 8	7,2634
100 000	285,707 0	28,1468
1 000 000	3 279,630 5	153,3212

Час паралельного виконання дорівнює O(logn). Загальна часова складність становить θ(nlogn). Розроблений алгоритм може працювати на паралельних процесорах та виконується набагато швидше для великих n (при більших розмірностях масиву даних).

4.3 Висновок до розділу 4

Тестування розробленої програми довело: відповідність вимогам, якими керувалися при проектуванні та розробці; правильність результату для усіх можливих вхідних даних; виконання програмного додатку за прийнятний час; практичність використання; сумісність із програмним забезпеченням та операційними системами.

Отже, в результаті проведених тестувань можна стверджувати, що паралельні обчислення є набагато швидшими, ніж звичайне швидке сортування, а розроблена програмна реалізація паралельного алгоритму швидкого сортування працює вірно.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка може бути прийнята та успішно впроваджена, якщо вона відповідає вимогам сучасності в напрямку науково-технічного прогресу та враховує економічні аспекти. Це зумовлює необхідність оцінити економічну ефективність отриманих результатів науково-дослідної роботи.

Магістерська кваліфікаційна робота з розробки та дослідження за темою «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок визначається як пріоритетний, оскільки розроблені результати можуть бути використані різними зацікавленими сторонами, приносячи економічні вигоди. Однак для здійснення цього процесу необхідно знайти потенційного інвестора, який був би зацікавлений у втіленні цього проекту, і переконати його у виправданості вкладання інвестицій в даний проект.

Для цього визначені такі етапи виконання робіт.

1. Проведення комерційного аудиту науково-технічної розробки, що включає в себе визначення науково-технічного рівня та комерційного потенціалу.
2. Розрахунок витрат на реалізацію науково-технічної розробки.
3. Проведення розрахунку економічної ефективності науково-технічної розробки в разі її впровадження та комерціалізації потенційним інвестором, а також обґрунтування економічної доцільність комерціалізації для інвестора.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу згідно рекомендацій здійснюємо із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями (табл.5.1) [43].

Для проведення технологічного аудиту було залучено трьох незалежних експертів. Такими експертами будуть Денисюк В.О., Сілагін О.В. та Озеранський В.С.

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки надано у зведеній таблиці 5.2.

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [43].

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інформаційна технологія онтологічного моделювання бази знань з організації бібліотеки» становить 38 балів, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Отже, з отриманих даних (табл. 5.1-5.2) видно, що нова розробка має достатній рівень комерційного потенціалу.

Результатом роботи є інформаційна технологія швидкого сортування масивів даних з використанням Open MPI.

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

№ з/п	Бали (за п’ятибальною шкалою)				
	0	1	2	3	4
1	2	3	4	5	6
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогівна малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Не маєаналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижча за ціни аналогів	Ціна продукту значно нижча заціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок зпозитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конку ренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Таблиця 5.1 - Продовження

1	2	3	4	5	6
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно най мати фахівців або витрачати значні кошти тачас на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використову-ються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використову-ються у виробництві
11	Термін реалізації ідеї більшийза 10 років	Термін реалізації ідеї більшийза 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізаціїідеї від 3-х до 5- ти років. Термін окупності інвестицій більший 5-ти років	Термін реалізації ідеї меншийЗ-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї меншийЗ-х років. Термін окупності інвестицій менший 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документівна виробництвота реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво тареалізацію продукту, що потребує значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту потребує незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво тареалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво тареалізацію продукту

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Денисюк В.О., доцент кафедри КН ВНТУ	Сілагін О.В., доцент кафедри КН ВНТУ	Озеранський В.С., доцент кафедри КН ВНТУ
	Бали, виставлені експертами		
1	3	3	3
2	3	3	3
3	3	3	3
4	3	3	3
5	3	3	3
6	3	3	3
7	3	3	3
8	3	3	4
9	3	4	4
10	3	3	3
11	4	4	4
12	3	3	3
Сума балів	$СБ_1 = 37$	$СБ_2 = 38$	СБ3 = 39
Середньо-арифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 38$		

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Розробка може бути застосована для розв’язання задачі сучасних методів швидкого сортування, забезпечує подальше впровадження інформаційних технологій у різноманітних задачах.

Науково-технічний рівень розробки та її комерційний потенціал є вище середнього за рахунок:

- покращення та розширення функціональних можливостей нової науково-технічної розробки порівняно з аналогічними розробками, існуючими в цей час на ринку;
- значно вищої якості, конкурентоспроможності нової науковотехнічної розробки за рахунок збільшення швидкодії;
- суттєвого зростання продуктивності, ефективності нової науковотехнічної розробки за рахунок використання паралельних технологій.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата для розробників (дослідників) визначається за формулою (5.1):

$$З_о = \frac{M}{T_p} \cdot t, \quad (5.1)$$

де M – місячний посадовий оклад конкретного розробника;

T_p – кількість робочих днів у місяці, T_p = 22 дні;

t – число днів роботи розробника, t = 50 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.4. ($З_О$=20454,25 грн.)

Таблиця 5.4 – Розрахунки основної заробітної плати дослідників

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	10000	454,55	5	2272,75
Інженер-програміст	8000	363,63	50	18181,5
Всього:				20454,25

Основна заробітна плата робітників.

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою [43]:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6500 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду [43];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати [ММ];

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дн;

$t_{зм}$ – тривалість зміни, год.

Величина витрат на основну заробітну плату робітників наведена у таблиці 5.5 ($З_р$= 6664,80 грн.).

Додаткова заробітна плата дослідників та робітників.

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$$З_{дод} = (20454,25 + 6664,80) \cdot 11 / 100\% = 2983,10 \text{ грн.}$$

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год.	Розряд роботи	Погодинна тарифна ставка, грн.	Величина оплати на робітника, грн
1. Аналіз вимог та формулювання специфікацій	10	1	62,8	628,00
2. Швидке сортування масивів даних з використанням Open MPI	24	3	84,8	2035,20
3. Розробка програмного коду та системної архітектури	20	5	106,8	2136,00
4. Тестування та оптимізація розробленого рішення	22	3	84,8	1865,60
Всього				6664,80

Відрахування на соціальні заходи.

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$$З_Н = (\ 20454,25 + 6664,80 + 2983,10\ ) \cdot 22 / 100\% = 6622,47 \text{ грн.}$$

Сировина та матеріали.

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Інформаційна технологія онтологічного моделювання бази знань з організації бібліотеки».

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (5.6)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, (K_j = *1,1 … 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

Проведені розрахунки зведено до таблиці 5.6 *(М=519,20 грн.).*

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг/шт, грн	Норма витрат, кг/шт	Вартість витраченого матеріалу, грн
Папір для записів А4/80	100	1	200
Диск оптичний	16	2	32
Flesh-пам'ять RAM 32GB	240	1	240
Всього			472
З врахуванням коефіцієнта транспортування			519,2

Розрахунок витрат на комплектуючі.

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI» відсутні *($К_в=0$)*.

Спецустаткування для наукових (експериментальних) робіт. До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (5.7)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ – кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 10000 \cdot 1 \cdot 1,11 = 11100$ *грн.*

Отримані результати зведено до таблиці 5.7.

Таблиця 5.7 – Витрати на придбання спецустаткування

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Персональний комп’ютер	1	10000	11100
Всього			11100

Програмне забезпечення для наукових (експериментальних) робіт.

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

До балансової вартості програмного забезпечення входять витрати на його інсталяцію, тому ці витрати беруться додатково в розмірі 10…12% від вартості програмного забезпечення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10...1,12$);

k – кількість найменувань програмних засобів.

Оскільки в розробці використовувалися безкоштовні програмні засоби, тому дані витрати не враховані ($B_{прг} = 0$).

Амортизація обладнання, програмних засобів та приміщень.

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} \ , \qquad (5.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Проведені розрахунки зведено до таблиці 5.8 ($A_{обл}$ *= 1875,00 грн.*).

Таблиця 5.8 – Розрахунок амортизаційних відрахувань

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Комп'ютер	10000	3	3	833,33
Приміщення лабораторії	250000	20	1	1041,67
Всього				1875,00

Паливо та енергія для науково-виробничих цілей.

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_е = \sum_{i=1}^{n} \frac{W_{уі} \cdot t_i \cdot Ц_е \cdot К_{вні}}{\eta_i}, \qquad (5.10)$$

де $W_{уі}$ – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е$ = 7,5 грн;

$К_{вні}$ – коефіцієнт, що враховує використання потужності, $К_{вні} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i<1$.

$B_e = 0,250 \cdot 200 \cdot 7,5 \cdot 0,5 / 0,8 = 234,38$ *грн.*

Службові відрядження. До статті «Службові відрядження» дослідної роботи на тему «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \qquad (5.11)$$

де $H_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$B_{св}$ = (20454,25+ 6664,80) · 20 / 100% = 5423,81 грн.

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» відсутні.

Розрахуємо інші витрати. До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.12)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$I_в$ = (20454,25+ 6664,80) • 50 / 100% = 1355,95 грн.

Накладні (загальновиробничі) витрати. До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу тощо.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.13)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = *100%.*

$В_{нзв}$ = (20454,25+ 6664,80) • 100 / 100% = 27119,05 грн.

Витрати на проведення науково-дослідної роботи на тему «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI». розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.14)$$

$B_{заг}$ = 20454.25+6664.80+2983.10+6622.47+519.20+0+11100.00+

+0+1875.00+234.38+5423.81+0+1355.95+27119.05= 84352.01 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.15)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,5 [43].

ЗВ = 84352.01/0,5 = 168704,02 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

Позитивним результатом у ринкових умовах є отримання потенційним інвестором чистого прибутку чи його збільшення за рахунок можливого впровадження результатів науково-технічної розробки. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації розробки за темою «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI».

Результати дослідження проведені за темою розробки передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 20000,00 грн;

$\pm\Delta Ц_о$ – зміна вартості послуги від впровадження результатів, приймемо зростання на 1000,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [43]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.16)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ=40%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(1000·1+20000·200)·0,83·0,4·(1-0,18/100%)=1325941,00 грн.

Збільшення чистого прибутку 2-го року:

$$\Delta П_2 = (1000 \cdot 1 + 20000 \cdot (200+100)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1-0{,}18/100\%) = 1988745{,}80 \text{ грн.}$$

Збільшення чистого прибутку 3-го року:

$$\Delta П_3 = (1000 \cdot 1 + 20000 \cdot (200+100+50)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1-0{,}18/100\%) = 2320148{,}20 \text{ грн.}$$

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.17)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =18%;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 1325941{,}00 / (1+0{,}18)^1 + 1988745{,}80/(1+0{,}18)^2 + 2320148{,}20/(1+0{,}18)^3 =$$
$$= 3969155{,}70 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.18)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, становлять 168704,02 грн.

$$PV = k_{інв} \cdot ЗВ = 2 \cdot 168704{,}02 = 337408{,}04 \text{ грн.}$$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (5.19)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 3969155,70 грн;

PV – теперішня вартість початкових інвестицій, 337408,04 грн.

$$E_{абс} = ПП - PV = 3969155{,}70 - 337408{,}04 = 3631747{,}7 \text{ грн.}$$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.20)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+3631747,7 /337408,04)^{1/3} - 1 = 1,21$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.21)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,45.

$\tau_{мін}$ = 0,1+0,45 = 0,55 < 1,21 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Отже, інвестування в науково-дослідну роботу за темою «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI» є доцільним.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.22}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,21 = 0,83 р.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновок до розділу 5

Проведені дослідження рівня комерційного потенціалу розробки за темою «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI» становить 38 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Термін окупності становить 0,83 р., менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI».

ВИСНОВКИ

Дослідження присвячене швидкому сортуванню масивів даних з використанням Open MPI, яке спрямовано на розв’язання задач, пов'язаних з вибором ефективних методів і алгоритмів сортування масивів даних. У роботі було оцінено переваги швидкого сортуванню масивів даних з розпаралелюванням процесу сортування в порівнянні з традиційними способами сортування. Основна мета була досягнута шляхом реалізації алгоритма швидкого сортуванню масивів даних мовою С++. Для виконання паралельних дій було використано технологію Open MPI. Результатом розробки є: аналіз особливостей і параметрів, що виникають в задачах швидкого сортування, та існуючі методи вирішення задачі швидкого сортування; дослідженя математичної моделі швидкого сортування та її удосконалення; алгоритм програми і програмний продукт досліджуваної інформаційної технології швидкого сортування масиву даних; аналіз результатів, що отримані під час тестування розробленого програмного продукту.

Тестування розробленої програми довело відповідність вимогам, якими керувалися при проектуванні та розробці, правильність результату для усіх можливих вхідних даних, виконання програмного додатку за прийнятний час, практичність використання, сумісність із програмним забезпеченням та операційними системами.

Результати дослідження можуть мати практичне значення для застосування в різноманітних галузях використання інформаційних технологій та в учбовому процесі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. V.O. Denysiuk, A.A. Polishchyk. Software Implementation and Research of Quick Sorting of Data Arrays by Open MPI. Молодь в науці: дослідження, проблеми, перспективи (МН-2024). Вінницький національний технічний університет. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/19391/16149
2. D. E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching (3rd. ed.), Addison Wesley Longman Publishing Co., Inc., 1998.
3. С. В. Коляденко, В. О. Денисюк, Н. П. Юрчук. Дискретний аналіз. Частина1. Навчальний посібник. Вінниця: ВНАУ, 2019.
4. Креневич А.П. Алгоритми і структури даних. Підручник. К.: ВПЦ "Київський Університет", 2021. 200 с. URL: https://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf
5. Sorting Algorithms - Class Diagram. UML: https://creately.com/diagram/example/ilsj99zj3/sorting-algorithms-class-diagram
6. Pim van den Broek, M. Aksit. Verifying Runtime Reconfiguration Requirements on UML Models. UML: https://www.researchgate.net/publication/226948224_Verifying_Runtime_Reconfiguration_Requirements_on_UML_Models
7. Manuela Panoiu, Ionel Muscalagiu, Caius Panoiu, Maria Raich. Educational Software for Study the Performances of Some Known Parallel and Sequential Algorithms. URL: https://www.researchgate.net/publication/228821227_Educational_Software_for_Study_the_Performances_of_Some_Known_Parallel_and_Sequential_Algorithms

8. Hoare, C.A.R. (1961). Algorithm 63, Partition: Algorithm 64, Quicksort: Communications of the ACM, Vol. 4,p.321-322. URL: https://dl.acm.org/doi/pdf/10.1145/366622.366644
9. Hoare, C.A.R. (1962) Quicksort. The Computer Journal, 5, 10-15. URL: http://dx.doi.org/10.1093/comjnl/5.1.10
10. ШШвидке сортування. URL: https://uk.wikipedia.org/wiki/QuickSort
11. Alfred V. Aho, John E. Hopcroft, Jeffrey D. Data Structures and Algorithms. URL: https://doc.lagout.org/Alfred%20V.%20Aho%20-%20Data%20Structures%20and%20Algorithms.pdf
12. Теоретичні відомості про часову складність алгоритмів. URL: https://uk.wikipedia.org/wiki/Часова_складність_алгоритму
13. Sedgewick, Robert; Wayne, Kevin (2011). Algorithms (4th ed.). Addison-Wesley Professional. URL: https://bank.engzenon.com/tmp/5e7f6ee5-d4dc-4aa8-9b0a-42d3c0feb99b/6062caf3-c600-4fc2-b413-4ab8c0feb99b/Algorithms-4th-Edition.pdf
14. Семеренко, В. П. Технології паралельних обчислень: навчальний посібник / Семеренко В. П. Вінниця: ВНТУ, 2018. 104 с. URL: https://pdf.lib.vntu.edu.ua/books/IRVC/2021/Semerenko_2018_104.pdf
15. Parallel computing. URL: https://en.wikipedia.org/wiki/Parallel_computing
16. Analysis of quicksort. URL: https://www.khanacademy.org/computing/computer-science/algorithms/ quick-sort/a/analysis-of-quicksort
17. Quicksort. URL: https://algs4.cs.princeton.edu/23quicksort/
18. Parallel Quick Sort. URL: https://iq.opengenus.org/parallel-quicksort/
19. Patel, H. (2020, March 10). An Overview of QuickSort Algorithm. https://towardsdatascience.com/an-overview-of-quicksort-algorithm-b9144e314a72
20. Паралельний алгоритм швидкого сортування. URL: https://iq.opengenus.org/parallel-quicksort

21. Порівняння алгоритмів сортування. URL: https://www.dcc.fc.up.pt/~ricroc/aulas/1516/cp/apontamentos/slides_sorting.pdf
22. Shrinu Kushagra, Alejandro López-Ortiz, J. Ian Munro, Aurick Qiao. Multi-Pivot Quicksort: Theory and Experiments. URL: https://www.researchgate.net/publication/289974363_Multi-Pivot_Quicksort_Theory_and_Experiments
23. QUICKSORT DEMOS. URL: http://fpl.cs.depaul.edu/jriely/ds2/extras/demos/23DemoPartitioningDijkstra.pdf
24. Open MPI. URL: https://en.wikipedia.org/wiki/Open_MPI
25. Open MPI documentation. URL: https://www.open-mpi.org/
26. OpenMP. URL: https://en.wikipedia.org/wiki/OpenMP
27. OpenMP URL: https://www.openmp.org
28. Message passing interface (MPI). URL: https://www.techtarget.com/searchenterprisedesktop/definition/message-passing-interface-MPI
29. MPI Forum. URL: https://www.mpi-forum.org/
30. MPI: A Message-Passing Interface Standard. Version 4.1. URL: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf
31. MPI. URL: https://hpc-wiki.info/hpc/MPI#References
32. On Scalability for MPI Runtime Systems. URL: https://www.researchgate.net/publication/221201576_On_Scalability_for_MPI_Runtime_Systems
33. Selim Ciraci, Pim van den Broek, M. Aksit. University of Twente Verifying Runtime Reconfiguration Requirements on UML Models. URL: https://www.researchgate.net/publication/226948224_Verifying_Runtime_Reconfiguration_Requirements_on_UML_Models
34. Quicksort. URL: https://jeffallendevelopment.wordpress.com/flowcharts/
35. Теоретичні відомості про мову програмування C++ URL: https://uk.wikipedia.org/wiki/C%2B%2B

36. Bjarne Stroustrup. The C++ Programming Language. 4th Edition. 2013. Addison-Wesley Professional. 1376 p. URL: https://chenweixiang.github.io/docs/The_C++_Programming_Language_4th_Edition_Bjarne_Stroustrup.pdf
37. Трофименко О. Г. С++. Алгоритмізація та програмування : підручник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, О. В. Задерейко. 2-ге вид. перероб. і доповн. Одеса: Фенікс, 2019. 477 с.
38. Quicksort Algorithm. URL: https://www.programiz.com/dsa/quick-sort
39. C++ Implementations of Quicksort Methods for Sorting Arrays with Lots of Duplicates. URL: https://www.codeproject.com/Articles/1023591/Cplusplus-Implementations-of-Quicksort-Methods-for
40. Victor Eijkhout. Parallel Programming in MPI and OpenMP. 2nd edition. 2022. 673 p. URL: https://web.corral.tacc.utexas.edu/CompEdu/pdf/pcse/EijkhoutParallelProgramming.pdf
41. Implementation of Quick sort using MPI, OMP and Posix thread. URL: https://www.geeksforgeeks.org/implementation-of-quick-sort-using-mpi-omp-and-posix-thread
42. Теоретичні відомості про тестування програмного забезпечення. URL: https://internetdevels.ua/blog/software-testing-business-benefits
43. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

87

Додаток А (обов'язковий)
Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 95,24% Схожість 4,76%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Поліщук А.А.

Керівник роботи Денисюк В.О.

Опис прийнятого рішення

Магістерську кваліфікаційну роботу допущено до захисту

Особа, відповідальна за перевірку Озеранський В.С.

Додаток Б (обов'язковий)

Лістинг програми

```
#include <iostream>
#include <mpi.h>
#include <chrono>
#include <time.h>
#include <random>

using namespace std;
using  std::cout;
using  std::endl;
typedef std::chrono::high_resolution_clock Clock;

class ParallelQuickSort{
  //keep count of threads
  int k = 0;

  private:
    //partitioning procedure
    int partitionParallel(int arr[], int l, int r){
      int i = l + 1;
      int j = r;
      int key = arr[l];
      int temp; while(true){
        while(i < r && key >= arr[i])
        i++;
        while(key < arr[j])
          j--;
        if(i < j){
          temp = arr[i];
          arr[i] = arr[j];
          arr[j] = temp;
        }
        else{
          temp = arr[l];
          arr[l] = arr[j];
          arr[j] = temp;
          return j;
        }
      }
    }

    int partition(int arr[], int start, int end)
    {

      int pivot = arr[start];
```

```
    int count = 0;

    for (int i = start + 1; i <= end; i++){
         if (arr[i] <= pivot)
              count++;
    }

    // Giving pivot element its correct position
    int pivotIndex = start + count;
    swap(arr[pivotIndex], arr[start]);

    // Sorting left and right parts of the pivot element
    int i = start, j = end;

    while (i < pivotIndex && j > pivotIndex) {

         while (arr[i] <= pivot) {
              i++;
         }

         while (arr[j] > pivot) {
              j--;
         }

         if (i < pivotIndex && j > pivotIndex) {
              swap(arr[i++], arr[j--]);
         }
    }

    return pivotIndex;
  }

public:
  void quickSortParallel(int arr[], int l, int r){
    if(l < r){
      int p = partitionParallel(arr, l, r);
// cout <<"pivot"<< p << "found by thread no." <<k<< endl;

      #pragma omp parallel sections
      {
        #pragma omp section
        {
          k = k + 1;
          quickSortParallel(arr, l, p-1);
        }
        #pragma omp section
        {
          k = k + 1;
          quickSortParallel(arr, p+1, r);
        }
```

```
    }
  }
}

void quickSort(int arr[], int start, int end)
{

  // base case
  if (start >= end)
       return;

  // partitioning the array
  int p = partition(arr, start, end);

  // Sorting the left part
  quickSort(arr, start, p - 1);

  // Sorting the right part
  quickSort(arr, p + 1, end);
}

//prints array
void printArr(int arr[], int n){
   cout << endl;
   cout <<"Sorted array: ";
   for(int i = 0; i < n; i++)
      cout << arr[i] << " ";
   cout << endl;
}

//run the whole procedure
void run(){

  int length;
  cout<<"Enter size of the array: ";
  cin>>length;
  int arr[length];

  random_device rd;
  mt19937 mt(rd());
  uniform_real_distribution<double> dist(1, 1000);

  cout << "\n";
  cout <<"Array: ";
  for (int i=0; i<length; ++i)
  {
  int num = dist(mt);
  arr[i]=num;
  cout<<arr[i]<<" ";
  }
```

```
    for(int i=0;i<length;i++)
    {
    int num = rand()%1000;
    arr[i]=num;
    }

    for(int i=0;i<length;i++)
    cout<<arr[i]<<" ";
    cout<<endl;

    int n = sizeof(arr) / sizeof(arr[0]);
    auto t1 = Clock::now();
    quickSortParallel(arr, 0, n-1);
    auto t2 = Clock::now();
    auto t3 = Clock::now();
    quickSort(arr, 0, n-1);
    auto t4 = Clock::now();
    printArr(arr, n);

    cout << "\n";
    cout << "РµР°СЂР°Р»РµР»СЊРЅРµ:"
    << std::endl;
       std::chrono::duration<double,std::milli>fp_ms=+t2-t1;
       //PjC-P»C-CЃPµPєCґPSPrPë
       cout << "P§P°CЃ PI PjC-P»C-CЃPµPєCґPSPrP°C…: "
         << fp_ms.count()
         << "\n"
         << "P§P°CЃ PI CЃPµPєCґPSPrP°C…: "
         << fp_ms.count() / 1000
         << "\n"
         << "P§P°CЃ PI C…PIPëP»PëPSP°C…: "
         << fp_ms.count() / 60000
         << std::endl;
         cout << "P—PIPëC‡P°PNºPSPµ:"
         << std::endl;
       std::chrono::duration<double,std::milli>fp_ms_1 =t4-t3;
       //PjC-P»C-CЃPµPєCґPSPrPë
       cout << "P§P°CЃ PI PjC-P»C-CЃPµPєCґPSPrP°C…: "
         << fp_ms_1.count()
         << "\n"
         << "P§P°CЃ PI CЃPµPєCґPSPrP°C…: "
         << fp_ms_1.count() / 1000
         << "\n"
         << "P§P°CЃ PI C…PIPëP»PëPSP°C…: "
         << fp_ms_1.count() / 60000
         << std::endl;
   }
}
```

92

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ШВИДКОГО СОРТУВАННЯ МАСИВІВ ДАНИХ З ВИКОРИСТАННЯМ OPEN MPI

Виконав: студент 2-го курсу,
групи 2КН-22м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Поліщук А.А..
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
_____ Денисюк В.О.
(прізвище та ініціали)
«07» 12 2023 р.

Вінниця ВНТУ - 2023 рік

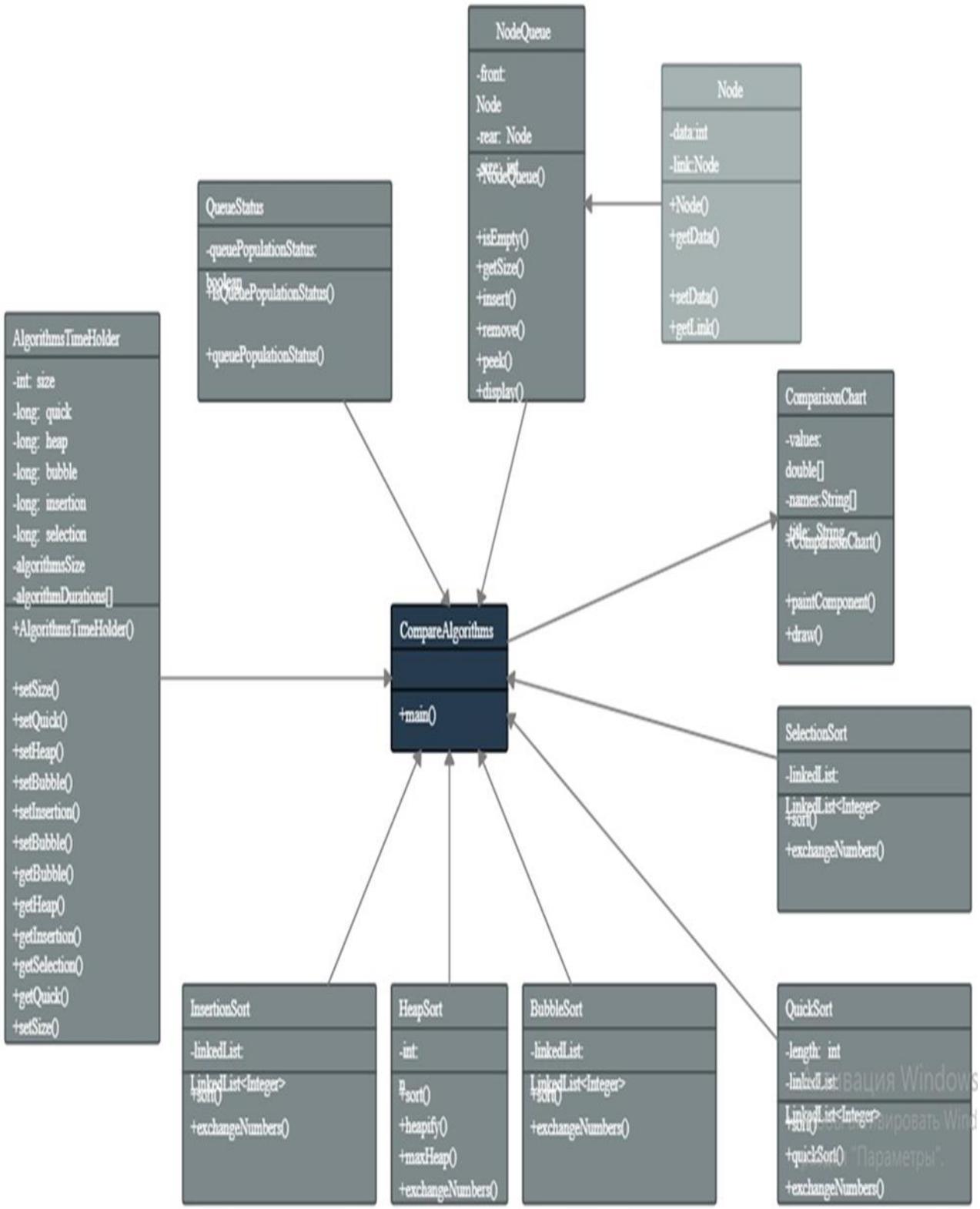

Рисунок В.1 - UML-діаграма класів алгоритмів сортування

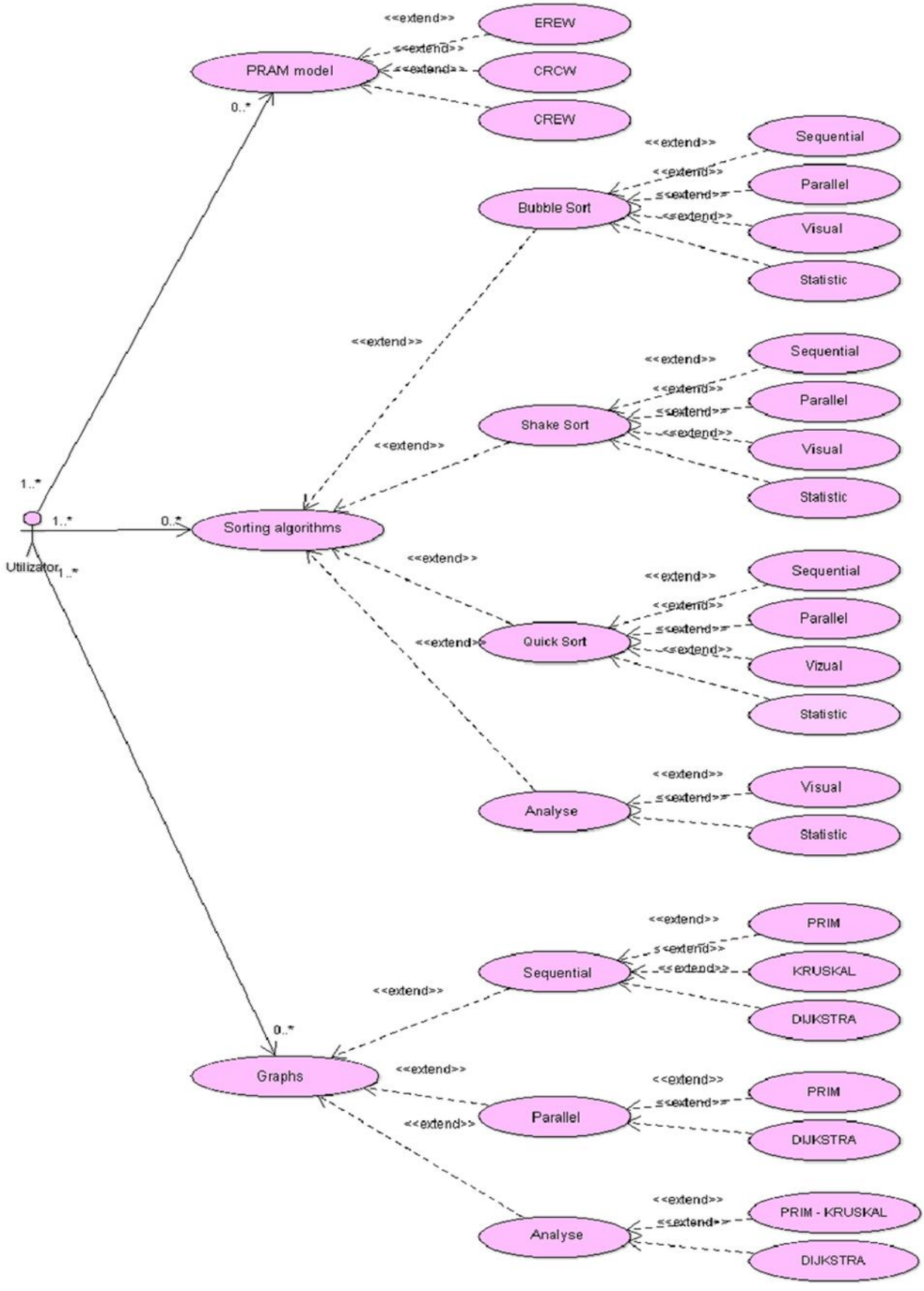

Рисунок В.2 – UML-діаграма використання різних методів сортування

ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  quicksort (A,M,N); value M,N;
             array A; integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin      integer I,J;
           if M < N then begin partition (A,M,N,I,J);
                               quicksort (A,M,J);
                               quicksort (A, I, N)
                         end
end        quicksort
```

Рисунок В.3 - Класичний алгоритм Гоара

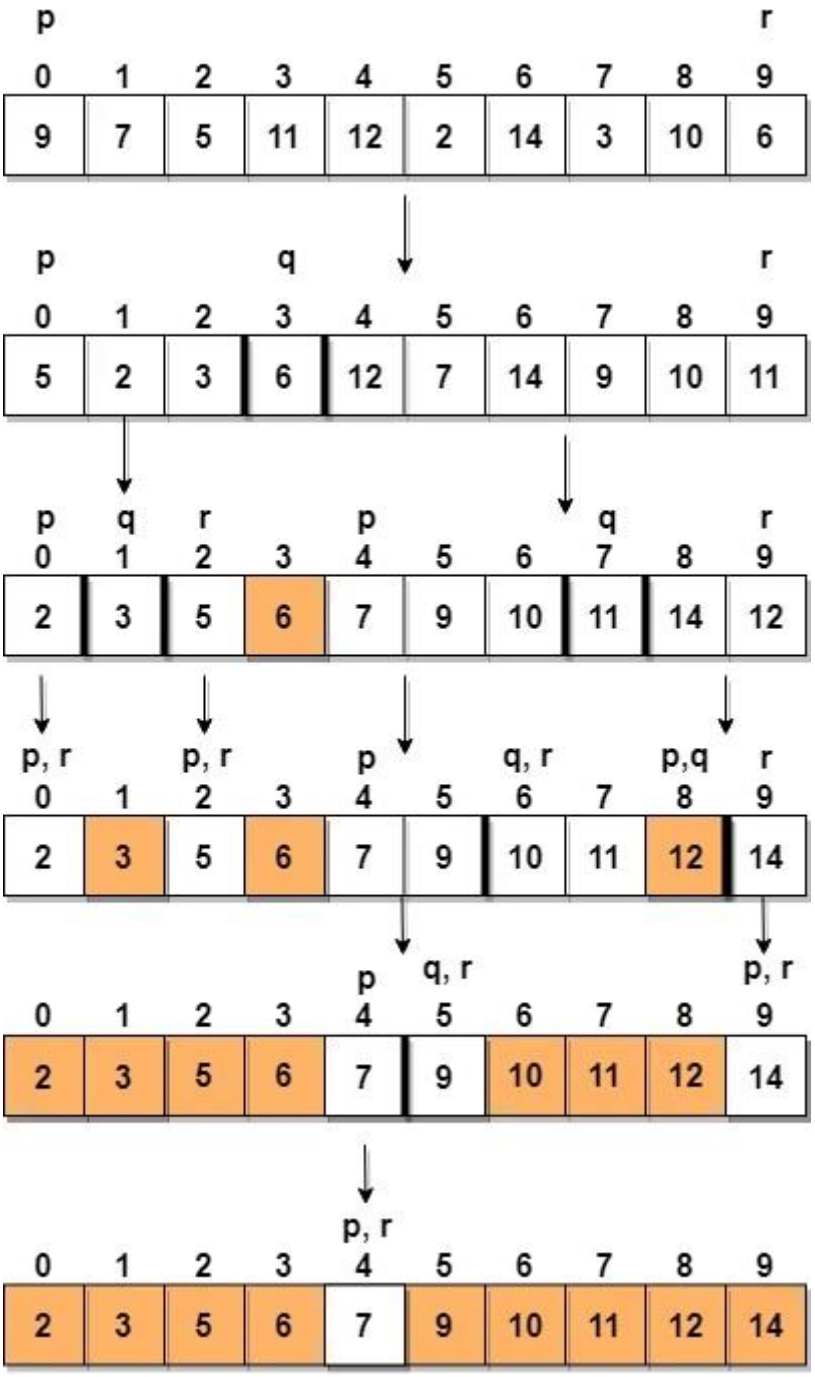

Рисунок В.4 – Приклад швидкого сортування

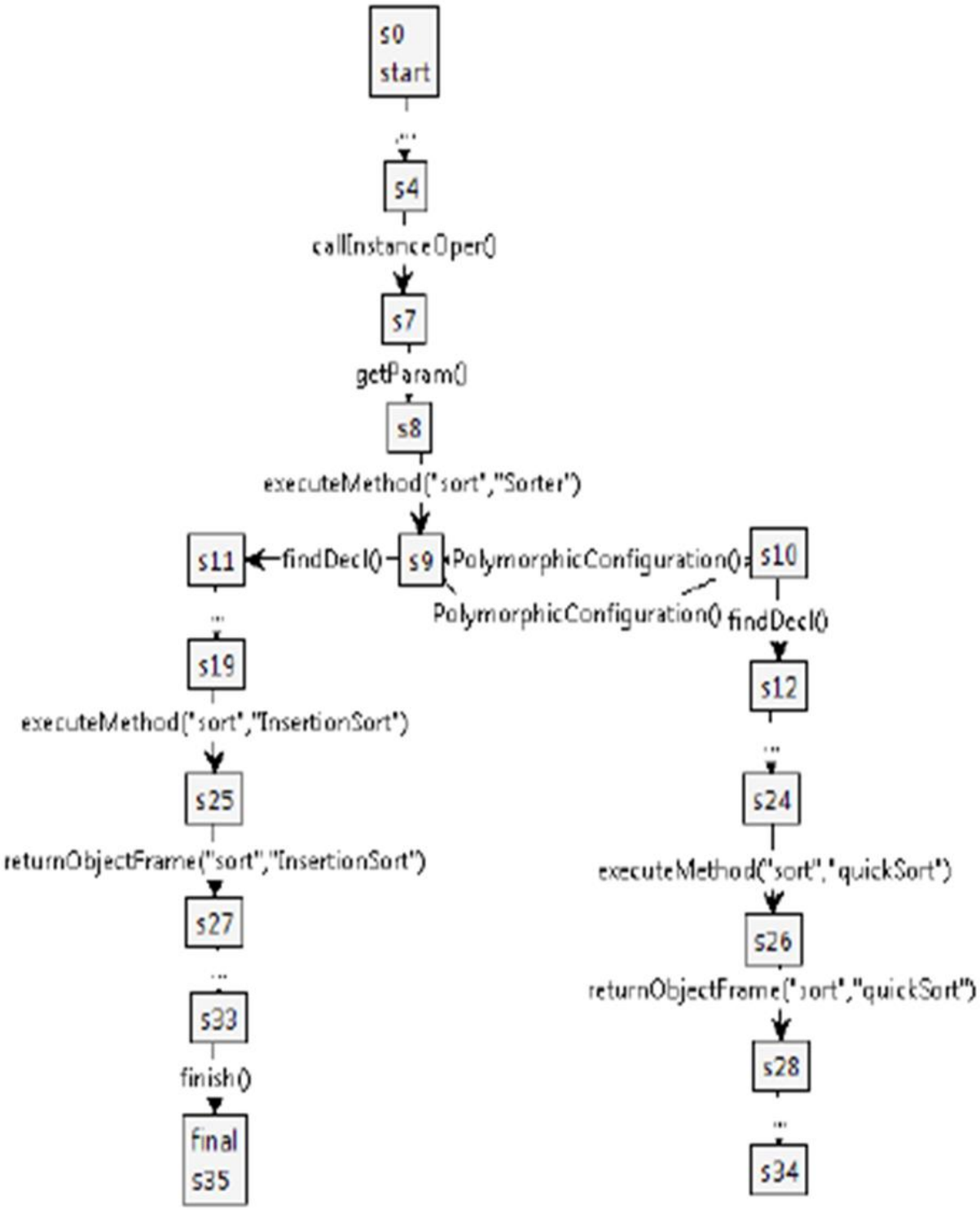

Рисунок В.5 – Послідовність виконання алгоритму швидкого сортування

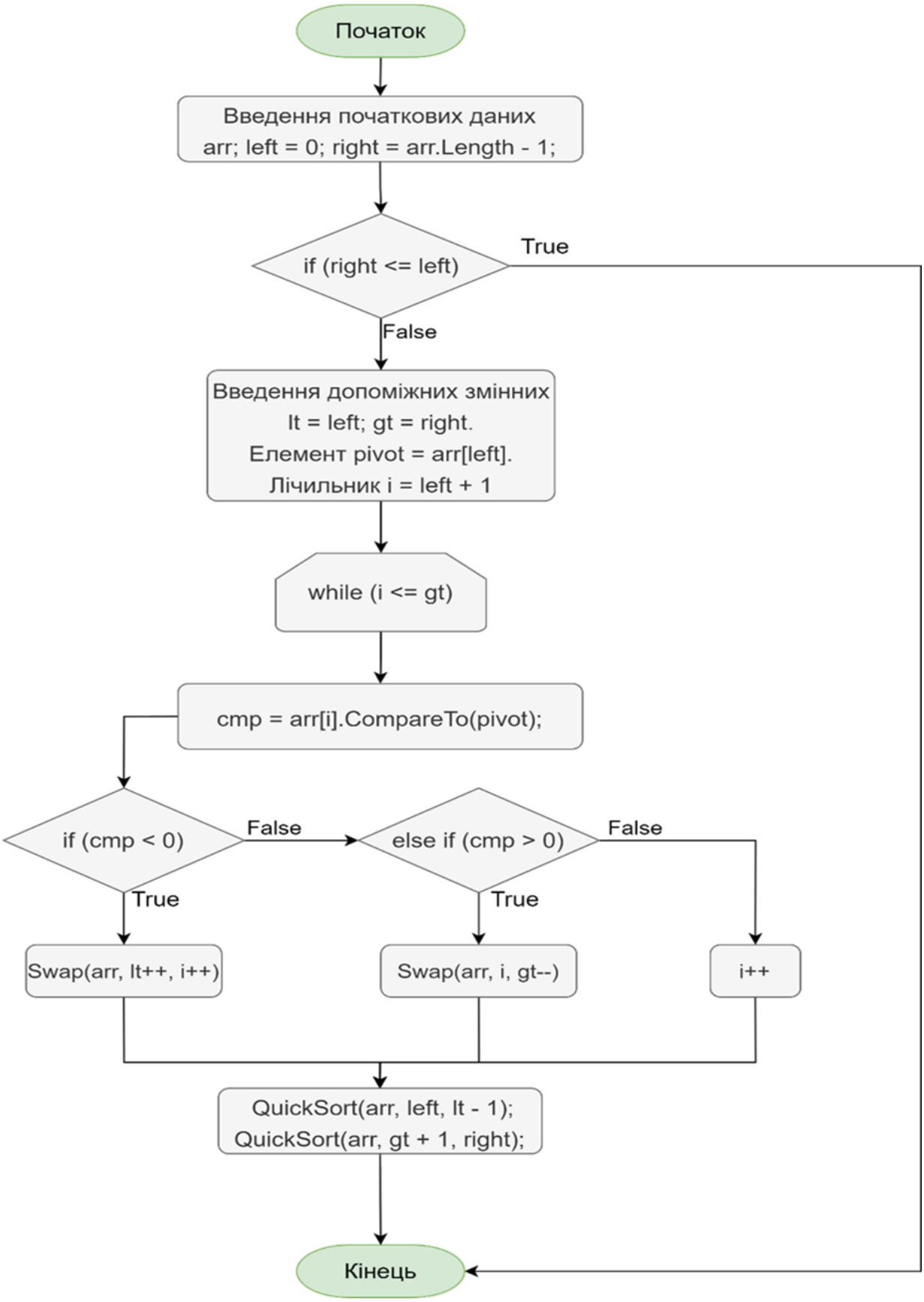

Рисунок В.6 – Граф-схема алгоритма швидкого сортування

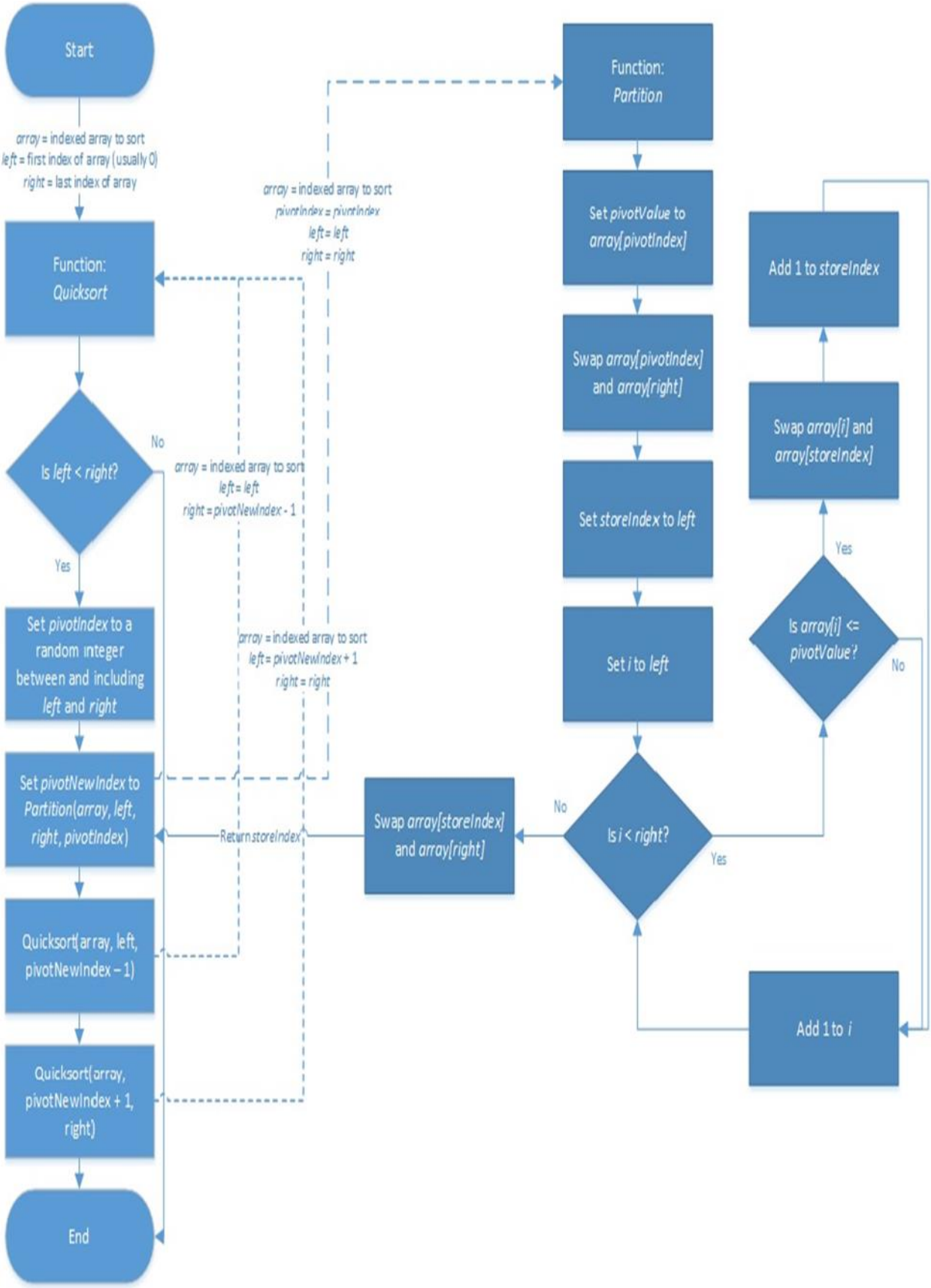

Рисунок В.7 – Приклад UML-діаграми діяльності швидкого сортування

```
private:
    //partitioning procedure
    int partition(int arr[], int l, int r){
        int i = l + 1;
        int j = r;
        int key = arr[l];
        int temp;
        while(true){
            while(i < r && key >= arr[i])
                i++;
            while(key < arr[j])
                j--;
            if(i < j){
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }else{
                temp = arr[l];
                arr[l] = arr[j];
                arr[j] = temp;
                return j;
            }
        }
    }
```

Рисунок В.8 – Скріншот функції розділення масиву

```
public:
    void quickSort(int arr[], int l, int r){
        if(l < r){
            int p = partition(arr, l, r);
            cout << "pivot " << p << " found by thread no. " << k << endl;

            #pragma omp parallel sections
            {
                #pragma omp section
                {
                    k = k + 1;
                    quickSort(arr, l, p-1);
                }
                #pragma omp section
                {
                    k = k + 1;
                    quickSort(arr, p+1, r);
                }
            }
        }
    }
```

Рисунок В.9 – Скріншот функції quickSort, яка визначає кількість потоків та проводить сортування

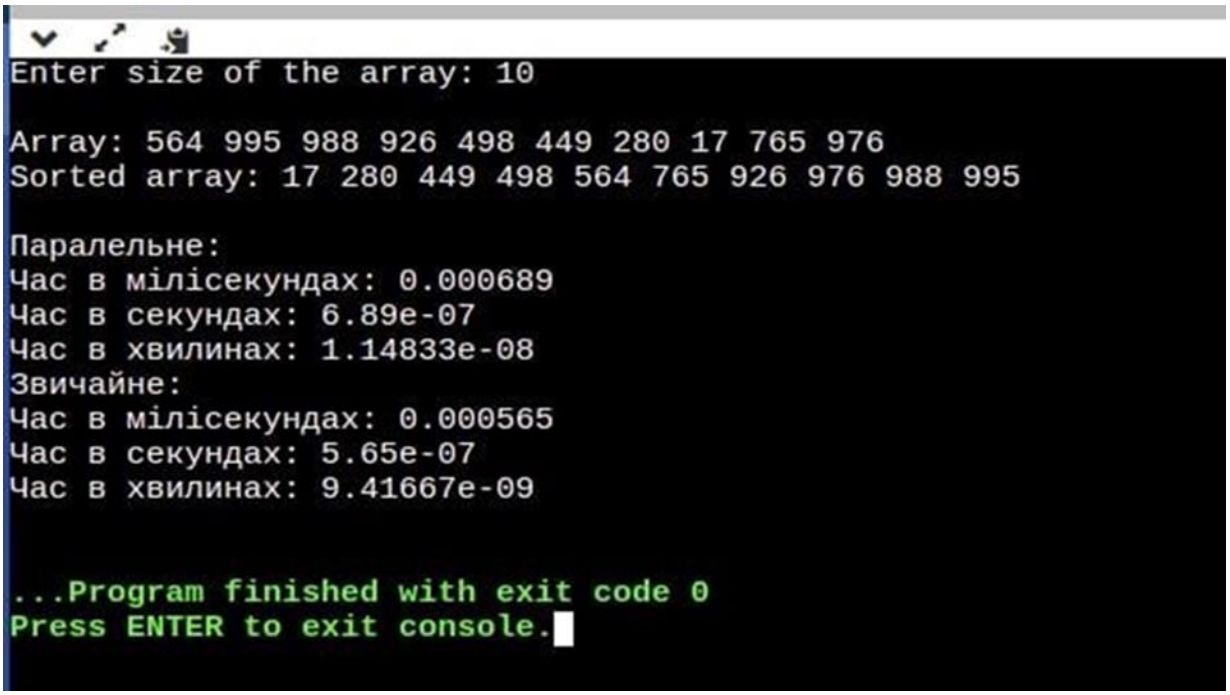

Рисунок В.10 – Тестування програми із N=10

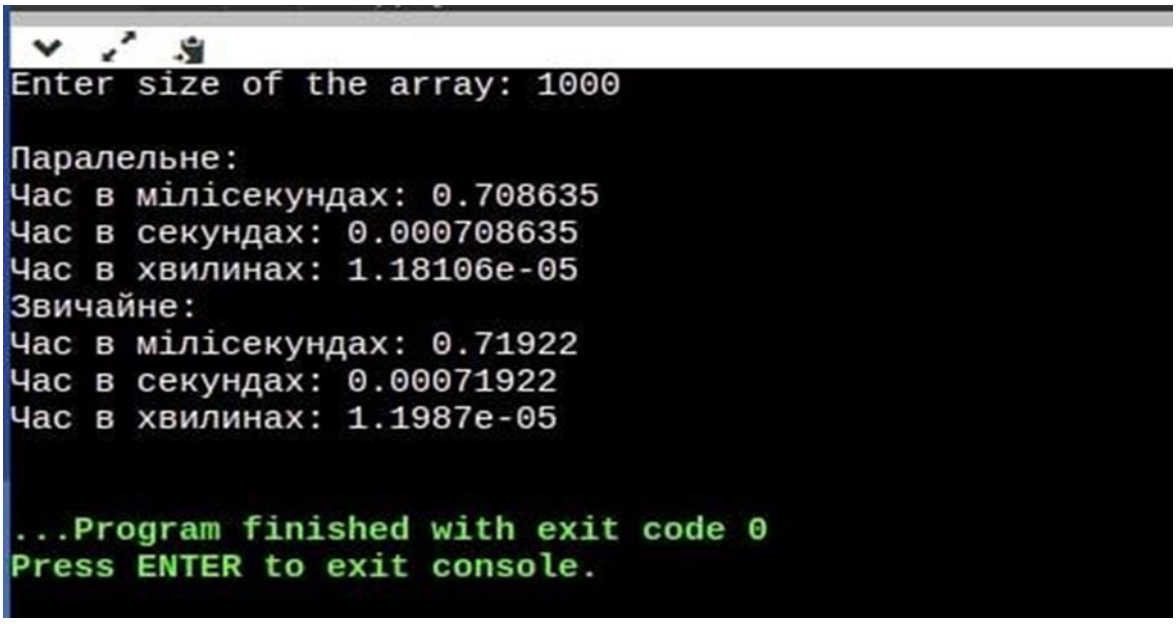

Рисунок В.11 - Тестування програми із N=1000

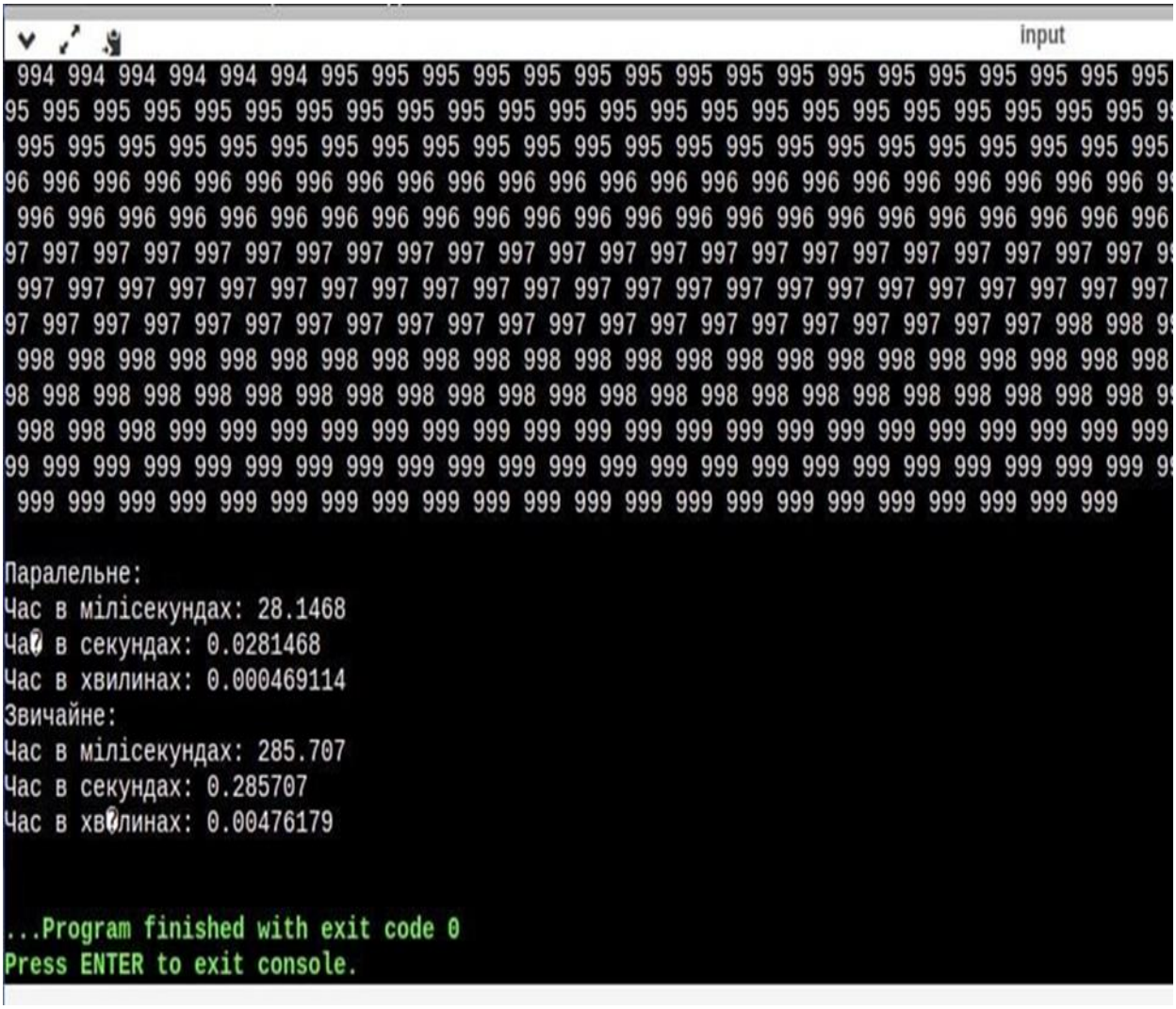

Рисунок В.12 - Тестування програми із N=100 000

Додаток Г (довідниковий)

Акт впровадження

ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ "ЮК КРИСТАЛ"
21027, Україна, Вінницька область, м. Вінниця, вул. Келецька, 51
IBAN 70 325365 000000 2600801744329, в ПАТ "КРЕДОБАНК",
Код ЄДРПОУ 35648543, Інд.под. № 356485402288

Затверджую
Директор
ТОВ «ЮК Кристал»
Довбеус Ігор Ярославович

«25» листопада 2023 р.

АКТ
впровадження результатів магістерської кваліфікаційної роботи
Поліщука Анатолія Андрійовича на тему:
«Інформаційна технологія швидкого сортування масивів даних з використанням Open MPI»

Комісія у складі директора І. Я. Довбеуса, заступника директора з питань обліку готової продукції С. С. Сидоренка та бухгалтера ТОВ «ЮК Кристал» С. Д. Лучицький вирішила: розроблене А. А. Поліщуком програмне, алгоритмічне та математичне забезпечення може бути прийнято для використання при виконанні робіт, що пов'язані з обробкою інформації для пошуку, обробки, сортуванню та обліку продукції.

Таким чином, актуальність та практична цінність роботи А. А. Поліщука не викликає сумнівів. Наукова новизна запропонованих та розроблених моделей та алгоритмів швидкого сортування масивів даних з використанням Open MPI також можна визнати достатньо високою, оскільки ні в сучасній літературі, ні в відомих математичних та програмних системах обчислень немає готових засобів для розв'язання таких задач.

Голова комісії:
директор _____ І. Я. Довбеус.

Члени комісії:

заступник директора
з питань обліку готової продукції _____ С. С. Сидоренко,

бухгалтер _____ С. Д. Лучицький