МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

О.І. Марченко, О.О. Марченко

СТРУКТУРИ ДАНИХ ТА АЛГОРИТМИ. КУРСОВА РОБОТА

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ СОРТУВАННЯ НА БАГАТОВИМІРНИХ МАСИВАХ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник
для здобувачів ступеня бакалавра за спеціальністю
F7 «Комп'ютерна інженерія»

Київ
КПІ ім. Ігоря Сікорського
2026

УДК 004.422; 004.424
Рецензент: *Заболотня Т.М., канд. техн. наук, доцент*
Відповідальний редактор: *Тарасенко-Клятченко Оксана Володимирівна,* канд. техн. наук, доцент, доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 4 від 05.02.2026 р.) за поданням Вченої ради Факультету програмних систем та прикладної математики (протокол № 8 від 26.01.2026 р.)
Електронне мережне навчальне видання

Марченко Олександр Іванович, канд. техн. наук, доцент
Марченко Олексій Олександрович, асистент

СТРУКТУРИ ДАНИХ ТА АЛГОРИТМИ. КУРСОВА РОБОТА
ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ СОРТУВАННЯ НА БАГАТОВИМІРНИХ МАСИВАХ

Структури даних та алгоритми. Курсова робота: Дослідження ефективності алгоритмів сортування на багатовимірних масивах: [Електронний ресурс] : навч. посіб. для студ. спеціальності F7 «Комп'ютерна інженерія», 3-тє видання, виправлене та доповнене / О. І. Марченко, О. О. Марченко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,61 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2026. – 122 с.

Навчальний посібник розроблено для ознайомлення студентів із завданнями курсової роботи по кредитному модулю «Структури даних та алгоритми: складні структури. Курсова робота». Навчальний посібник містить інформацію щодо виконання курсової роботи, яка включає постановку завдання та вимоги до проєкту програми, що повинні розробити студенти, вказівки до вимірювання часу роботи алгоритмів, вказівки до оформлення звіту та тестування розроблених алгоритмів і відповідного проєкту програми, варіанти індивідуальних завдань. Навчальний посібник призначений для студентів очної форми навчання за спеціальністю F7 «Комп'ютерна інженерія» факультету програмних систем та прикладної математики КПІ ім. Ігоря Сікорського.

ЗМІСТ

ВСТУП

Курсова робота з дисципліни «Структури даних та алгоритми» носить дослідницький характер і виконується на тему «Дослідження ефективності алгоритмів сортування на багатовимірних масивах».

Кредитний модуль «Структури даних та алгоритми: складні структури. Курсова робота» призначений для фундаментального засвоєння теоретичних знань і практичних навичок, отриманих під час вивчення дисципліни «Структури даних та алгоритми». Курсова робота дозволяє сформувати у студентів компетенції, необхідні для розв'язання задач професійної діяльності, пов'язаної із застосуванням на практиці знань основних структур даних та алгоритмів сортування для виконання оцінювання алгоритмів та порівняльного аналізу алгоритмів. Метою кредитного модуля є формування у студентів здатностей:

− аналізувати зміст поставлених задач;

− обирати найбільш придатні для розв'язку задач структури даних;

− розроблювати чіткі і ефективні структуровані алгоритми розв'язку поставлених задач;

− виконувати аналіз ефективності алгоритмів.

Кредитний модуль «Структури даних та алгоритми: складні структури. Курсова робота» базується на матеріалі кредитного модуля «Структури даних та алгоритми», який вивчався в осінньому семестрі. Цей кредитний модуль забезпечує вивчення дисциплін

«Паралельне програмування», «Об'єктно-орієнтоване програмування», «Системне програмування» навчального плану освітнього ступеня «Бакалавр» за спеціальністю F7 «Комп'ютерна інженерія».

Кредитний модуль «Структури даних та алгоритми: складні структури. Курсова робота» складається із виконання студентами комплексної курсової роботи з метою закріплення на практиці теоретичних положень навчальної дисципліни і набуття студентами умінь і досвіду оперувати складними алгоритмами та структурами даних, які необхідні для створення великих програмних систем. Курсова робота охоплює всі теми дисципліни «Структури даних та алгоритми» і включає навчальні аспекти як розробки складних програмних систем, так і написання документації для них.

Курсова робота складається з практичної частини, яка полягає у реалізації заданих алгоритмів для розв'язку задач на багатовимірних масивах з використанням принципів структурного та модульного програмування, а також з теоретичної дослідницької частини, яка полягає у вимірюванні часу роботи алгоритмів як для звичайних випадків одновимірного масиву, так і різних випадків поведінки алгоритмів на багатовимірних масивах, а також у дослідженні причин неоднакової поведінки цих алгоритмів на одновимірних і багатовимірних масивах.

Курсова робота дозволяє отримати досвід реалізації і дослідження складних алгоритмів на складних структурах даних, а також практичного засвоєння принципів структурного і модульного

програмування і оформлення документації на створений програмний продукт.

Інформація до виконання курсової роботи включають:

- технічне завдання на курсову роботу;
- постановку завдання та вимоги до виконання програми, що повинні розробити студенти;
- інструкції до вимірювання часу роботи алгоритмів;
- вказівки до оформлення звіту та тестування розробленого алгоритму і відповідної йому програми;
- варіанти індивідуальних завдань;
- додатки.

ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ

I. Описати теоретичні положення, від яких відштовхується дослідження, тобто принцип та схему роботи кожного із досліджуваних алгоритмів сортування для одновимірного масиву, навести загальновідомі властивості цих алгоритмів та оцінки кількості операцій порівняння та присвоєння для них.

II. Скласти алгоритми рішення задачі сортування в багатовимірному масиві заданими за варіантом методами та написати на мові програмування за цими алгоритмами програму, яка відповідає вимогам розділу «Вимоги до програми курсової роботи».

III. Виконати налагодження та тестування коректності роботи написаної програми.

IV. Провести практичні дослідження швидкодії складених алгоритмів, тобто виміри часу роботи цих алгоритмів для різних випадків та геометричних розмірів багатовимірних масивів.

V. За результатами досліджень скласти порівняльні таблиці за різними ознаками.

- Одна таблиця результатів (вимірів часу сортування впорядкованого, випадкового і обернено-впорядкованого масиву) для масиву з заданими геометричними розмірами повинна бути такою:

Таблиця № для масиву A[P,M,N], де P= ; M= ; N= ;

	Відсортований	Випадковий	Обернено відсортований
Назва алгоритму 1			
Назва алгоритму 2			
Назва алгоритму 3			

УВАГА! Колонки таблиць виміру часу в програмі та у звіті курсової роботи місцями НЕ переставляти! За невиконання цієї вимоги нараховується 3 (три) штрафних бали.

- Для варіантів курсової роботи, де крім алгоритмів порівнюються також способи обходу, в назвах рядків таблиць потрібно вказати як назви алгоритмів, так і номери способів обходу.
- Для виконання ґрунтовного аналізу алгоритмів потрібно зробити виміри часу та побудувати таблиці для декількох масивів з різними геометричними розмірами.
- Зробити виміри часу для стандартного випадку одновимірного масиву, довжина якого вибирається такою, щоб можна було виконати коректний порівняльний аналіз з рішенням цієї ж задачі для багатовимірного масиву.
- Кількість необхідних таблиць для масивів з різними геометричними розмірами залежить від задачі конкретного варіанту курсової роботи і вибираються так, щоб виконати всебічний та ґрунтовний порівняльний аналіз заданих алгоритмів.
- Випадки дослідження та конкретні розміри масивів дослідження для кожної із задач наведені у розділі «Випадки дослідження».

VI. Для наочності подання інформації за отриманими результатами рекомендується також будувати стовпчикові діаграми та графіки.

VII. Виконати порівняльний аналіз поведінки заданих алгоритмів за отриманими результатами (вимірами часу):

- для одновимірного масиву відносно загальновідомої теорії;
- для багатовимірних масивів відносно результатів для одновимірного масиву;
- для заданих алгоритмів на багатовимірних масивах між собою;
- дослідити вплив різних геометричних розмірів багатовимірних масивів на поведінку алгоритмів та їх взаємовідношення між собою;
- **для всіх вищезазначених пунктів порівняльного аналізу пояснити, ЧОМУ алгоритми в розглянутих ситуаціях поводять себе саме так, а не інакше**.

УВАГА! Саме якість виконання порівняльного аналізу дозволяє отримати оцінки від B (85 балів) до A (100 балів). Якщо в звіті в якості порівняльного аналізу наведений просто словесний опис фактів, що й так видно з таблиць, діаграм та графіків, то оцінка курсової роботи буде не вище, ніж C (75 балів).

VIII. Зробити висновки за зробленим порівняльним аналізом.

IX. Програму курсової роботи під час її захисту ОБОВ'ЯЗКОВО мати при собі на електронному носії інформації.

СТРУКТУРА ЗВІТУ КУРСОВОЇ РОБОТИ

1. Перший титульний аркуш (додаток 2).
2. Другий титульний аркуш (додаток 3).
3. Технічне завдання (ТЗ) на курсову роботу. Загальна частина та структура індивідуальної частини ТЗ наведені у додатку 4.
4. Опис теоретичних положень.
5. Схема імпорту/експорту модулів програми курсової роботи.
6. Структурна схема взаємовикликів процедур та функцій програми курсової роботи.
7. Опис призначення всіх функцій і процедур та їх параметрів.
8. Текст програми з коментарями.
9. Тести, що демонструють коректну роботу кожного із заданих алгоритмів для кожного випадку початкової впорядкованості масиву.
10. Результати (таблиці виміру часу та побудовані за даними таблицями стовпчикові діаграми та графіки).
11. Порівняльний аналіз алгоритмів **(це ключовий пункт для отримання високих оцінок з діапазону 85 – 100 балів!!!)**
12. Висновки по отриманих результатах.
13. Список літератури.
14. Сторінки звіту пронумерувати починаючи з другої.

ЗАГАЛЬНІ ВИМОГИ ДО ОФОРМЛЕННЯ ТА НАДСИЛАННЯ РЕЗУЛЬТАТІВ КУРСОВОЇ РОБОТИ

1. Звіт курсової роботи виконується у текстовому редакторі MS Word (файл формату **.docx**) або Libre Office Writer (файл формату **.odt**).

2. Файл звіту потрібно записати у папку з файлами проєкту програми курсової роботи. Після цього, всю цю папку, в якій знаходяться як файл звіту, так файли з кодом мовою програмування та файли вхідних даних (якщо є), потрібно зархівувати у архів (.zip, .rar, .7z), ідентифікатор якого записується ТІЛЬКИ ЛАТИНСЬКИМИ буквами і має такий вид: **KV-XX_Prizvyshche**.

УВАГА! Оскільки більшість поштових клієнтів забороняють пересилання бінарних файлів з точки зору безпеки, то перед архівуванням папки з файлами звіту та проєкту програми потрібно видалити з цієї папки всі бінарні файли (з розширеннями .exe та .obj), які були згенеровані під час компіляції проєкту.

3. Згідно з вказівками викладача, наданими на початку семестру, сформований архів результатів повинен бути або надісланий на електронну пошту викладача, або завантажений у навчальну систему, яка використовується в конкретному навчальному році.

4. Якщо архів надсилається електронною поштою, то **архів повинен бути надісланий обов'язково тільки БЕЗПОСЕРЕДНЬО як додаток до листа**, а НЕ якимось непрямим способом, наприклад, через посилання на google-диск тощо, який буде доступний

викладачу тільки обмежений проміжок часу і на якому студент може замінити свій архів також у будь-який час.

5. У випадку очного чи змішаного режиму навчання в конкретному навчальному році, крім надсилання електронної версії, звіт курсової роботи також повинен бути надрукований. Листки надрукованого звіту повинні бути скріплені довільним способом так, щоб звіт можна було читати як книжку і він не розпадався.

ВИМОГИ ДО ПРОЄКТУ ПРОГРАМИ КУРСОВОЇ РОБОТИ

1. Всі алгоритми, що задані в технічному завданні за варіантом, повинні бути реалізовані в рамках ОДНОГО програмного проєкту.
2. Програмний проєкт курсової роботи повинен мати модульну структуру. Після декомпозиції задачі курсової роботи на підзадачі, для реалізації підзадач верхнього рівня повинні бути використані програмні модулі. Для мов C/C++ модулі реалізуються у вигляді пари файлів з однаковими іменами, але різними розширеннями .h та .c, наприклад, “name.h” та “name.c”. При реалізації модулів повинен бути строго дотриманий принцип «приховування інформації» так, як це було викладено на лекціях.
3. Реалізація вимірів часу роботи алгоритмів сортування у програмному проєкті курсової роботи повинна строго задовольняти усім вимогам, зазначеним у розділі «Інструкції з виміру часу роботи алгоритмів».
4. Бінарний код програмного проєкту, який буде запускатись з метою виміру часу сортування заданими алгоритмами, повинен бути скомпільований без використання жодних режимів оптимізації вихідного коду.

   Як вимкнути всі режими оптимізації коду в компіляторах GUI CodeBlocks та Visual Studio, описано у розділі «Вимикання режимів оптимізації компіляторів».

5. В програмному коді повинні бути написані коментарі до основних її структурних компонентів:
   - до всіх оголошених структур даних;
   - до всіх модулів;
   - до всіх процедур та функцій;
   - до основних смислових фрагментів алгоритмів.
6. В програмному проєкті курсової роботи повинен бути реалізований текстовий діалоговий інтерфейс, який має декілька вкладених меню, що надають можливість обрати варіанти режимів тестування або виміру часу сортування заданими алгоритмами.
7. Режим тестування повинен забезпечити обрання варіантів тестування коду реалізації кожного з алгоритмів, тобто демонстрація коректності сортування кожним з алгоритмів для кожного з випадків початкової відсортованості масиву.
8. Режим виміру часу сортування заданими алгоритмами повинен забезпечити обрання варіантів виміру часу сортування кожним з алгоритмів для кожного з випадків початкової відсортованості масиву.

   Крім того, режим виміру часу сортування заданими алгоритмами повинен забезпечити також виконання виміру часу одразу усіма алгоритмами для усіх випадків початкової відсортованості масиву (пакетний варіант для вектора і пакетний варіант для тривимірного масиву) з побудовою результуючої таблиці, яка отримується для масиву з конкретними геометричними розмірами.

Ця таблиця повинна мати структуру згідно зі зразком, що наведений вище у технічному завданні.

9. Загальна структура вкладених меню повинна надавати наступні можливості вибору варіантів тестування та виміру часу сортування заданими алгоритмами:

Головне меню

Оберіть режим роботи:

1) Тестування алгоритмів сортування
2) Вимір часу сортування заданими алгоритмами

Меню «Тестування алгоритмів сортування»

Оберіть варіант тестування алгоритму сортування:

1) Сортування вектора алгоритмом №?? «Назва алгоритму 1»
2) Сортування вектора алгоритмом №?? «Назва алгоритму 2»
3) Сортування вектора алгоритмом №?? «Назва алгоритму 3»
4) Сортування тривимірного масиву алгоритмом №?? «Назва алгоритму 1»
5) Сортування тривимірного масиву алгоритмом №?? «Назва алгоритму 2»
6) Сортування тривимірного масиву алгоритмом №?? «Назва алгоритму 3»

Після обрання будь-якого з варіантів тестування повинно з’являтись меню з вибором:

Оберіть випадок початкової відсортованості масива:

1) Елементи масиву впорядковані строго за збільшенням
2) Масив містить випадкові значення елементів
3) Елементи масиву впорядковані строго за зменшенням

Меню

«Вимір часу сортування заданими алгоритмами»

Оберіть варіант виміру часу:

1) Сортування вектора алгоритмом №?? «Назва алгоритму 1»
2) Сортування вектора алгоритмом №?? «Назва алгоритму 2»
3) Сортування вектора алгоритмом №?? «Назва алгоритму 3»
4) Сортування тривимірного масиву алгоритмом №?? «Назва алгоритму 1»
5) Сортування тривимірного масиву алгоритмом №?? «Назва алгоритму 2»
6) Сортування тривимірного масиву алгоритмом №?? «Назва алгоритму 3»
7) Пакетний варіант для вектора (отримання одразу всієї таблиці результатів виміру часу сортування усіма алгоритмами)
8) Пакетний варіант для тривимірного масиву (отримання одразу всієї таблиці результатів виміру часу сортування усіма алгоритмами)

Після обрання будь-якого з перших шести варіантів виміру часу сортування (тобто, крім пакетних) повинно з’являтись меню з вибором:

Оберіть випадок початкової відсортованості масива:

1) Елементи масиву впорядковані строго за збільшенням
2) Масив містить випадкові значення елементів
3) Елементи масиву впорядковані строго за зменшенням

Після обрання одного з двох пакетних (сьомого або восьмого) варіантів виміру часу сортування таке меню не з’являється, а відразу починається виконання всіх вимірів часу для всіх алгоритмів та випадків відсортованості з побудовою таблиці такого виду, як показано в технічному завданні курсової роботи.

ВИМОГИ ДО ПОРІВНЯЛЬНОГО АНАЛІЗУ КУРСОВОЇ РОБОТИ

Порівняльний аналіз дослідження курсової роботи, який є її основною частиною для отримання високих оцінок, повинен містити наступні види порівняння алгоритмів:

1. Порівняння отриманих самостійно результатів швидкодії заданих алгоритмів для одновимірного масиву (вектора) порівняно до теоретичних та експериментальних результатів, що були отримані Ніклаусом Віртом і які надавались на лекції (див. підручник [1], с. 234-236, відеолекція https://youtu.be/1Bp2hJxLj5E).
2. Порівняння отриманих самостійно результатів швидкодії кожного із заданих алгоритмів для тривимірного масиву порівняно до отриманих самостійно результатів швидкодії цих же алгоритмів для одновимірного масиву (вектора).
3. Порівняння швидкодії трьох заданих алгоритмів для тривимірного масиву між собою на основі отриманих самостійно результатів швидкодії кожного із цих алгоритмів.

В рамках другого та третього видів порівняння заданих трьох алгоритмів (Алгоритм 1, Алгоритм 2, Алгоритм 3) для заданих трьох випадків початкової відсортованості масиву (прямо відсортований, випадковий, обернено відсортований) порівняльний аналіз повинен бути виконаний за наступними напрямами:

1. Порівняльний аналіз часу сортування трьох заданих алгоритмів між собою окремо для кожного із трьох випадків відсортованості для тривимірного масиву порівняно до цих же алгоритмів та випадків відсортованості для одновимірного масиву.

   Наприклад, чи змінилась пропорція часу сортування Алгоритму1 до часу сортування Алгоритму2 для випадку випадкового масиву при сортуванні тривимірного масиву до такої ж пропорції при сортуванні одновимірного масиву? І **чому** так відбулось?

   І так проаналізувати для кожної пари алгоритмів для кожного окремого випадку відсортованості масиву.
2. Порівняльний аналіз часу сортування трьох заданих випадків відсортованості масиву між собою окремо для кожного із трьох алгоритмів для тривимірного масиву порівняно до цих же алгоритмів та випадків відсортованості для одновимірного масиву.

   Наприклад, чи змінилась пропорція часу сортування прямо відсортованого масиву до часу сортування випадкового масиву для Алгоритму1 при сортуванні тривимірного масиву до такої ж пропорції при сортуванні одновимірного масиву? І **чому** так відбулось?

   І так проаналізувати для кожної пари випадків відсортованості масиву для кожного окремого алгоритму.

ГРАФІК ВИКОНАННЯ КУРСОВОЇ РОБОТИ

Курсова робота виконується студентами самостійно протягом семестру згідно з графіком. Графік виконання курсової роботи наведений у таблиці 1.

Таблиця 1. Графік виконання курсової роботи

Тиждень семестру	Назва етапу роботи
1-3	Отримання теми та завдання
4	Розробка структурних схем модулів та взаємовикликів функцій програми
4	Розробка та налагодження інтерфейсів модулів та функцій програми
5-6	Розробка, тестування та налагодження програми курсової роботи в цілому
7	Виконання вимірів швидкодії алгоритмів та систематизація отриманих результатів у таблицях
8-9	Виконання порівняльного аналізу алгоритмів. Оформлення звіту курсової роботи
10	Подання курсової роботи на перевірку з максимальною оцінкою «відмінно»
11	Подання курсової роботи на перевірку з максимально можливою оцінкою «дуже добре», незалежно від набраних балів за звіт
12	Подання курсової роботи на перевірку з максимально можливою оцінкою «добре», незалежно від набраних балів за звіт

13	Подання курсової роботи на перевірку з максимально можливою оцінкою «задовільно», незалежно від набраних балів за звіт
14	Тиждень захистів курсової роботи. Подання курсової роботи на перевірку на цьому тижні оцінюється з максимально можливою оцінкою «достатньо», незалежно від набраних балів за звіт

Завдання на курсову роботу видається на перших лекціях другого семестру навчання. Точні дати надсилання курсових робіт на кожну з максимальних оцінок, визначаються кожен рік, в залежності від календаря на поточний навчальний рік, і оголошуються викладачем на початку семестру. Захисти курсових робіт відбуваються, як правило, на передостанньому тижні семестру.

ОЦІНЮВАННЯ КУРСОВОЇ РОБОТИ

Семестровий рейтинг з кредитного модуля «Структури даних та алгоритми: складні структури. Курсова робота» визначається згідно з таблицями 2 та 3 і не може перевищувати 100 балів.

Якщо студент з урахуванням заохочувальних балів отримує більше 100 балів, йому виставляється 100 балів.

Перелік контрольних заходів, що наведений у таблиці 2, та суттєво некоректних ситуацій, допущених студентом при розробці проєкту програми та оформлення звіту курсової роботи, що наведений в таблиці 3, а також бали їх оцінювання, є орієнтовними, оскільки вимоги до розробки рейтингових систем оцінювання з дисциплін можуть змінюватися від одного навчального року до іншого.

Точний перелік пунктів таблиць 2 та 3 та відповідних їм балів буде визначатися актуальною РСО (рейтинговою системою оцінювання) курсової роботи, затвердженою для конкретного навчального року.

Остаточна оцінка за курсову роботу складається з двох оцінок: оцінки за виконання розробки програмного проєкту та оформлення звіту і оцінки за захист курсової роботи.

Важливо! *Для отримання повної кількості балів за вказаними у таблиці 2 категоріями оцінювання звіту, повинні бути дотримані* ***повна відповідність наданої роботи варіанту****, виданому студенту,* ***повністю коректна реалізація всіх алгоритмів, як для тривимірного масиву, так для вектора, а також повністю всі вимоги оформлення звіту згідно з технічним завданням варіанту****.*

Порушення цих вимог виконання курсової роботи вважається суттєво некоректними ситуаціями, при виникненні яких певна кількість із зазначених в таблиці 2 балів не нараховується, в залежності від того, які категорії оцінювання і в якому ступені не були виконані. При грубому невиконанні певних категорій оцінювання звіту, від яких залежить коректність всієї роботи, робота може бути незарахована взагалі і відправлена на переопрацювання. Перелік найбільш характерних основних (але не всіх можливих) суттєво некоректних ситуацій та їх оцінювання зазначений у таблиці 3.

Таблиця 2. Оцінювання звіту курсової роботи

№	Контрольний захід	Бали
	ПЕРЕВІРКА ЗВІТУ	
1	Загальний вигляд та правильність структури звіту.	**1**
2	Правильність оформлення титульних листів.	**1**
3	Наявність переліку використаної літератури.	**1**
4	Наявність нумерації сторінок звіту.	**1**
5	Правильність оформлення технічного завдання.	**1**
6	Наявність та якість викладення теоретичних положень.	**2**
7	Наявність та якість схеми взаємозв'язків модулів програми курсової роботи.	**2**
8	Наявність та якість схеми взаємовикликів процедур і функцій програми курсової роботи.	**2**

9	Якість опису призначення процедур і функцій програми курсової роботи.	**1**
10	Якість форматування тексту програми згідно із стандартами форматування програм, написаних мовою С.	**2**
11	Якість коментарів у програмі.	**1**
12	Наявність та логічність використання модулів у програмах.	**2**
13	Всі алгоритми реалізовані однаково ефективно.	**3**
14	Коректність реалізації виміру швидкодії алгоритмів (див. розділ «Інструкції з виміру часу роботи алгоритмів»).	**5**
15	Наявність достатньої кількості тестів та їх коректність.	**2**
16	Наявність достатньої кількості таблиць результатів виміру часу, що покривають всі випадки дослідження згідно з технічним завданням та відповідність структури таблиць до структури, заданої в технічному завданні.	**5**
17	Наявність вимірів швидкодії алгоритмів для вектора (вектор – це одновимірний масив, а НЕ тривимірний масив з одним перерізом і одним рядком!).	**2**
18	Правильність вибору розмірів структур даних (масивів) для отримання результатів, які дозволяють коректне порівняння результатів.	**5**
19	Наявність порівняльного аналізу заданих алгоритмів сортування для багатовимірних масивів.	**2**
20	Наявність порівняльного аналізу заданих алгоритмів сортування для вектора.	**2**

21	Наявність порівняльного аналізу між поведінкою алгоритмів на векторі і на багатовимірному масиві.	**2**
22	Повнота, коректність та конкретність порівняльного аналізу результатів у висновках.	**15**
	РАЗОМ ЗА РОЗРОБКУ ПРОЄКТУ ТА ЗВІТ	**60**
	ЗА ЗАХИСТ КУРСОВОЇ РОБОТИ	**40**
	РАЗОМ ЗА ВСЮ КУРСОВУ РОБОТУ	**100**

Таблиця 3. Суттєво некоректні ситуації звіту і програми курсової роботи та їх оцінювання

№	Суттєво некоректна ситуація	Оцінювання
1	Один з трьох реалізованих алгоритмів сортування тривимірного масиву не відповідає заданому варіанту. **Якщо невідповідно до заданого варіанту реалізовано більше одного алгоритму, робота не зараховується взагалі.**	Оскільки порушені п.п. 13, 15-16, 19, 21-22 таблиці 2, то сумарно не нараховується 1/3 загальної суми цих пунктів, тобто **10 балів**.
2	Відсутність у програмі або звіті реалізації одного із заданих алгоритмів сортування. **Якщо є відсутність реалізації більше одного алгоритму, робота не зараховується взагалі.**	Оскільки порушені п.п. 13-22 таблиці 2, то сумарно не нараховується 1/3 загальної суми цих пунктів, тобто **14 балів**
3	Некоректність реалізації одного або двох алгоритмів сортування. **Якщо є некоректність реалізації**	Оскільки частково порушені п.п. 13, 19-22

	більше двох алгоритмів, робота не зараховується взагалі.	таблиці 2, то не нараховується **5 балів за кожен такий алгоритм**
4	Відсутність у програмі та/або звіті реалізації одного або більше алгоритмів сортування вектора. (Вектор – це одновимірний масив, а НЕ тривимірний масив з одним перерізом і одним рядком!)	Оскільки частково порушені п.п. 13, 15-17, 19-22 таблиці 2, то не нараховується **5 балів за кожен такий алгоритм**
5	Відсутність у звіті повного тексту програми курсової роботи.	Оскільки порушені п.п. 10-14 таблиці 2, то сумарно не нараховується **10 балів**
6	Старт і стоп таймера виконані у неправильних місцях програми (див. розділ «Інструкції з виміру часу роботи алгоритмів») або серед операторів, час роботи яких вимірюється, є непотрібні оператори, які суттєво уповільнюють роботу алгоритму.	Оскільки частково порушені п.п. 13-14, 22 таблиці 2, то не нараховується **2 бали за кожен такий алгоритм**
7	Виміри часу з тактів переведені у секунди (час поділений на CLOCKS_PER_SEC) та/або округлені з отриманням великої похибки вимірів, якщо це призвело до втрати адекватності отриманих вимірів.	Оскільки порушені п.п. 19-22 таблиці 2 (наявність та повнота порівняльного аналізу), то може бути не нараховано **до 20 балів**, в залежності від ступеня некоректності отриманих вимірів.

8	Колонки та рядки в таблицях виміру часу розташовані інакше, ніж у заданому в технічному завданні зразку.	Оскільки частково порушені п.п. 16, 18, то не нараховується **1 бал за кожну таку таблицю**

Заохочувальні бали можуть бути нараховані за дострокове здавання курсової роботи не менше ніж на тиждень до встановленої дати здавання на максимальну оцінку (5 балів); за глибину та особливу докладність порівняльного аналізу (від 1-го до 5-ти балів); за виконання курсової роботи за індивідуальним завданням підвищеної складності (10 балів).

УВАГА! Заохочувальні бали не нараховуються, якщо виявлена несамостійність виконання курсової роботи. Крім того, заохочувальні бали нараховуються тільки якщо наданий звіт відповідає повністю всім вимогам до звіту, зазначених у навчальному посібнику [1].

У випадку виявлення несамостійності виконання курсової роботи бали за захист курсової роботи студенту не нараховуються, а також оцінка за звіт знижується в залежності від рівня несамостійності або відразу виставляється оцінка «незадовільно».

Студент допускається до захисту курсової роботи, якщо:

1) його рейтинг за звіт з урахуванням заохочувальних балів складає не менше, ніж 60% максимальної оцінки за звіт, тобто 60 % від 60 балів, що дорівнює 36 балів;

2) якщо студент продемонстрував хоча б мінімальну здатність самостійно виконувати програмістські роботи протягом семестру і до тижня захистів курсових робіт набрав за захисти лабораторних робіт та виконання модульної контрольної роботи не менше 40 балів; інакше курсова робота студента вважається виконаною несамостійно.

Відповідність рейтингових балів оцінкам за університетською шкалою наведена в таблиці 4.

Таблиця 4. Відповідність між рейтингом та оцінкою за курсову роботу.

Рейтинг	**Оцінка ECTS**	**Традиційна оцінка**
95≤RD≤100	A – відмінно	Відмінно
85≤RD≤94	B – дуже добре	Добре
75≤RD≤84	C – добре	
65≤RD≤74	D – задовільно	Задовільно
60≤RD≤64	E – достатньо	
40≤RD≤59	FX – незадовільно	Незадовільно
0≤RD≤39	F – незадовільно (потрібна додаткова робота)	

ІНСТРУКЦІЇ
З ВИМІРУ ЧАСУ РОБОТИ АЛГОРИТМІВ

УВАГА!

ВАЖЛИВІ ВИМОГИ ДО ВИМІРУ ЧАСУ:

1. Старт і стоп таймера повинні бути виконані прямо в тексті функції сортування безпосередньо перед першим оператором алгоритму сортування (старт) і безпосередньо після останнього оператора алгоритму сортування (стоп), як показано нижче у прикладі функції ExchangeSort. Виключенням є тільки рекурсивний алгоритм методу Швидкого сортування (Хоара), для якого старт таймера повинен бути безпосередньо перед першим рекурсивним викликом функції сортування, а стоп таймера – безпосередньо після нього.

2. Отриманий час роботи алгоритмів переводити у секунди (тобто ділити на CLOCKS_PER_SEC) не потрібно.

ЩЕ БІЛЬША УВАГА!

Якщо в курсовій роботі ці дві важливі вимоги до виміру часу роботи алгоритмів будуть проігноровані і, незважаючи на цю вказівку, старт і стоп таймера буде виконано не так, як вказано в цих вимогах та прикладі та/або отриманий час сортування переведений в секунди (ділення на CLOCKS_PER_SEC), то це вважається суттєво некоректними ситуаціями в реалізації вимірів часу, в результаті чого не нараховується суттєва кількість балів згідно з таблицею 3.

Врахування похибки вимірювання

Оскільки операційна система Windows є багатозадачною і у ній, окрім прикладної програми, одночасно працює багато системних процесів, то отриманий час має похибку.

Для врахування цієї похибки рекомендується використовувати методику усереднення значень вимірів, програма реалізації якої наведена нижче. Сама методика викладена у коментарях цієї програми.

“Common_Vector.h”

```
#ifndef __CommonVector_H__
#define __CommonVector_H__

#define VectorLength 10000

int Vector[VectorLength];

#endif
```

“Measurement.h”

```
#ifndef __Measurement_H__
#define __Measurement_H__

#include <time.h>

// Загальна кількість вимірів часу роботи алгоритму
#define measurements_number 28

// Кількість відкинутих початкових вимірів
#define rejected_number 2
```

```
// Кількість відкинутих вимірів з мінімальними значеннями.
// Вона ж дорівнює кількості відкинутих вимірів
// з максимальними значеннями.
#define min_max_number 3

// Масив значень часу роботи алгоритму
extern clock_t  Res[measurements_number];

// Функція обробки і усереднення значень вимірів
// часу роботи алгоритму.
// Повертає усереднене значення часу роботи алгоритму.
float MeasurementProcessing();

#endif
```

“Measurement.c”

```
#include "Measurement.h"
#include <stdio.h>

// Масив значень часу роботи алгоритму
clock_t  Res[measurements_number];

float MeasurementProcessing()
{
   long int Sum;
   float AverageValue;
//////////////////////////////////////////////////////////////
// Завдяки цим операторам виводу ви можете зрозуміти навіщо
// потрібне таке обчислення результатів виміру
   printf("Initial Measurement Results:\n");
   for (int i = 0; i< measurements_number; i++)
      printf("   %ld",Res[i]);
   printf("\n\n");
// У курсовій роботі ці оператори виводу потрібно видалити
//////////////////////////////////////////////////////////////
//------------------------------------------------------------
```

/* Методика усереднення результатів виміру:
Крок 1. Відкидається **rejected_number** перших вимірів з індексами від 0 до **rejected_number-1**.
Крок 2. Серед інших елементів знаходимо **min_max_number** вимірів з мінімальними значеннями та **min_max_number** вимірів з максимальними значеннями, і також відкидаємо їх.
Крок 3. Знаходимо середнє значення вимірів, що залишилися.
*/
//---

/* **Крок 1.** Для виконання Кроку 1 просто починаємо алгоритм з індексу **rejected_number**.
*/
/* **Крок 2.**
Для знаходження **min_max_number** мінімальних і максимальних значень вимірів виконуємо **min_max_number** ітерацій головного циклу алгоритму Шейкерного сортування в діапазоні індексів від **rejected_number** до **measurements_number-1**.
В результаті, **min_max_number** мінімальних значень вимірів будуть знаходитись на позиціях в діапазоні індексів від **rejected_number** до **rejected_number+min_max_number-1**, а **min_max_number** максимальних значень вимірів – на позиціях в діапазоні індексів від **measurements_number-min_max_number** до **measurements_number-1**.
*/

```
   clock_t  buf;
     int L = rejected_number, R = measurements_number - 1;
     int k = rejected_number;
     for (int j=0; j < min_max_number; j++) {
          for (int i = L; i < R; i++)    {
               if (Res[i] > Res[i + 1]) {
                    buf = Res[i];
                    Res[i] = Res[i + 1];
                    Res[i + 1] = buf;
                    k = i;
               }
          }
          R = k;
```

```
            for (int i = R - 1; i >= L; i--) {
                if (Res[i] > Res[i + 1]) {
                    buf = Res[i];
                    Res[i] = Res[i + 1];
                    Res[i + 1] = buf;
                    k = i;
                }
            }
            L = k + 1;
        }
////////////////////////////////////////////////////////////
// Завдяки цим операторам виводу ви можете зрозуміти навіщо
// потрібне таке обчислення результатів виміру
    printf("Measurement Results after sorting:\n");
    for (int i = 0; i< measurements_number; i++)
        printf("   %ld",Res[i]);
    printf("\n\n");
// У курсовій роботі ці оператори виводу потрібно видалити
////////////////////////////////////////////////////////////
```

// Крок 3.

/* Знаходимо середнє значення вимірів після відкидання **rejected_number** перших вимірів та **min_max_number** вимірів з мінімальними значеннями і **min_max_number** вимірів з максимальними значеннями, тобто середнє значення вимірів в діапазоні індексів від **rejected_number + min_max_number** до **measurements_number - min_max_number - 1**.

*/

```
    Sum=0;
    for (int i = rejected_number + min_max_number;
          i < measurements_number - min_max_number; i++)
        Sum = Sum + Res[i];
```

/* Кількість вимірів, що залишилась для обчислення середнього значення, дорівнює

measurements_number – 2 * min_max_number - rejected_number

*/

```
    AverageValue = (float)Sum / (float)(measurements_number -
2*min_max_number - rejected_number);
```

```
//////////////////////////////////////////////////////////////
// Завдяки цим операторам виводу ви можете зрозуміти навіщо
// потрібне таке обчислення результатів виміру
    printf("Initial rejected values: ");
    for (int i = 0; i < rejected_number; i++)
        printf("%ld   ", Res[i]);
    printf("\n");

    printf("Rejected minimal values: ");
    for (int i = rejected_number; i < rejected_number + min_max_number;
          i++)
        printf("%ld   ", Res[i]);
    printf("\n");

    printf("Rejected maximal values: ");
    for (int i = measurements_number - min_max_number;
          i < measurements_number; i++)
        printf("%ld   ", Res[i]);
    printf("\n\n\n");

    printf("Sum of the rest values, Sum = %ld;\n", Sum);
    printf("AverageValue = %11.2f\n\n\n", AverageValue);
// У курсовій роботі ці оператори виводу потрібно видалити
//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// А цей пустий і нібито не потрібний оператор друку
// НЕ ВИДАЛЯТИ В ЖОДНОМУ РАЗІ.
// Навіщо він потрібний буду розповідати на лекції
//    printf("");
// В останніх версіях gcc помилку, яка виникала без цього оператора,
// схоже вже виправили
//////////////////////////////////////////////////////////////

    return AverageValue;

}
```

“main.c”

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "CommonVector.h"
#include "Measurement.h"

void FillVector (int *A, int N)    {
//    for(int i=0; i<N; i++) A[i] = i;

    for(int i=0; i<N; i++) A[i] = rand();

//    for(int i=0; i<N; i++) A[i] = N-i;
//    for(int i=0; i<N; i++) printf(" %d",A[i]);
}

clock_t ExchangeSort(int *A, int N)
{
    int R, i, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(R = N-1; R > 0; R--) {
        for( i = 0 ; i < R; i++)
            if (A[i] > A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
            }
    }

    time_stop = clock();
    return time_stop - time_start;
}
```

```
void ExchangeSortMeasurement()
{
   for (int i = 0; i < measurements_number; i++) {
      FillVector(Vector, VectorLength);
      Res[i] = ExchangeSort(Vector, VectorLength);
   }
}

void OutTable(float ordered, float random, float backordered)
{
// Усереднений результат кожного з трьох вимірів буде виведено
// на екран у потрібній позиції
   printf("\t\t Ordered \t Random \t BackOrdered \n");

// В даному прикладі в усіх трьох колонках таблиці вимірів
// друкується одне й те ж саме значення
   printf("Exchange\t %7.2f \t %7.2f \t %7.2f \n", ordered, random,
backordered);
   printf("\n\n");
}

int main()
{
   float SortingTime;

   srand(time(NULL));
   ExchangeSortMeasurement();
   SortingTime = MeasurementProcessing();
// В даному прикладі в усіх трьох колонках таблиці вимірів
// друкується одне й те ж саме значення
   OutTable(SortingTime, SortingTime, SortingTime);

   return 0;
}
```

Виділення пам’яті для тривимірного масиву

Для того, щоб виміри були найбільш коректними, розміри тривимірного масиву треба брати максимально можливими. Для цього, тривимірний масив повинен бути оголошений як динамічний масив, а також пам’ять для нього повинна бути виділена перед початком роботи і звільнена в кінці роботи програми наступним чином:

```
#include <stdio.h>
#include <stdlib.h>

#define P 3      // Кількість перерізів
#define M 10    // Кількість рядків
#define N 20    // Кількість стовпчиків

int main()
{
// Оголошення динамічного тривимірного масиву
// та виділення для нього пам'яті
   int ***Arr3D;
   Arr3D = (int***) malloc(P*sizeof(int**));
   for (int k=0; k<P; k++) {
      Arr3D[k] = (int**) malloc(M*sizeof(int*));
      for (int i=0; i<M; i++)
         Arr3D[k][i] = (int*) malloc(N*sizeof(int));
   }
```

```
// Далі до масиву можна звертатись звичайним способом за
// допомогою трьох індексів, наприклад A[k][i][j].
// Наприклад, заповнення кожного перерізу масиву для Задачі 1,
// елементами, що відсортовані наскрізно по стовпчиках строго за
// збільшенням, при такому оголошенні масиву виглядає наступним
// чином:
            for (int k=0; k<P; k++) {
               int number=1;
               for (int j=0; j<N; j++)
                  for (int i=0; i<M; i++)
                     Arr3D[k][i][j] = number++;
            }
// Після того, як робота з масивом закінчена і він став не потрібним,
// необхідно звільнити його пам'ять.
// Це робиться так
   for (int k=0; k<P; k++) {
      for (int i=0; i<M; i++)
         free(Arr3D[k][i]);
      free(Arr3D[k]);
   }
   free(Arr3D);

// Пам'ять звільнено, можна закінчувати програму
   return 0;
}
```

При такому оголошенні, виділенні та звільненні пам’яті динамічного тривимірного масиву, звертання до його елементів нічим не відрізняється від звертання до елементів звичайного масиву, тобто звертання виконується як A[k][i][j].

Вимикання режимів оптимізації компіляторів

При будь-якому увімкненому режимі оптимізації коду у компіляторі, він може згенерувати досить змінений «перекручений» вихідний код, який хоча й виконує потрібну задачу сортування, але який фактично виконує це сортування вже за сильно зміненим алгоритмом, а не тим, який заданий у варіанті і код якого записаний мовою С в початковій програмі. Це може призводити до ситуацій, коли теоретично повільніший алгоритм, ніж інший, стане швидшим, ніж він, або навпаки.

В результаті, отримані виміри часу сортування покажуть спотворені взаємозалежності між заданими за варіантом алгоритмами, які пояснити з теоретичної точки зору неможливо.

Тому, для отримання адекватних результатів виміру часу сортування алгоритмів, **ВСІ режими оптимізації компілятора, яким компілюється проєкт курсової роботи, повинні бути вимкнені при компіляції проєкту перед виконанням вимірів часу сортування заданими алгоритмами**.

Розглянемо, як вимкнути всі режими оптимізації в компіляторах GUI CodeBlocks та MS Visual Studio.

Вимикання режимів оптимізації компілятора GUI CodeBlocks

У GUI CodeBlocks режими оптимізації, які встановлені за замовченням, можуть відрізнятися в залежності від версії GUI та того, з якого сайту його взяли.

Для увімкнення/вимкнення режимів оптимізації коду компілятора в GUI CodeBlocks потрібно відкрити меню Settings і виконати команду Compiler… , як показано на рис. 1. В результаті відкривається вікно Compiler settings, яке показане на рис. 2.

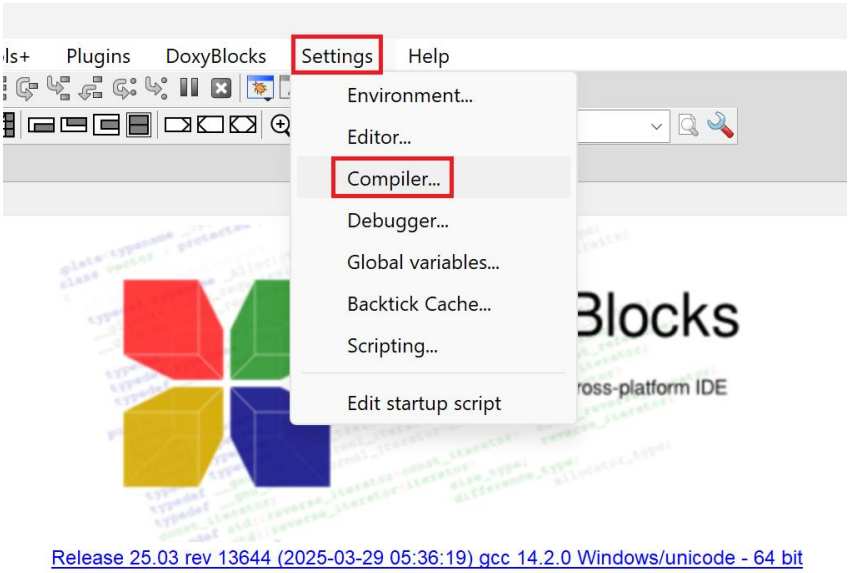

Рис. 1. Виконання команди Settings/ Compiler…

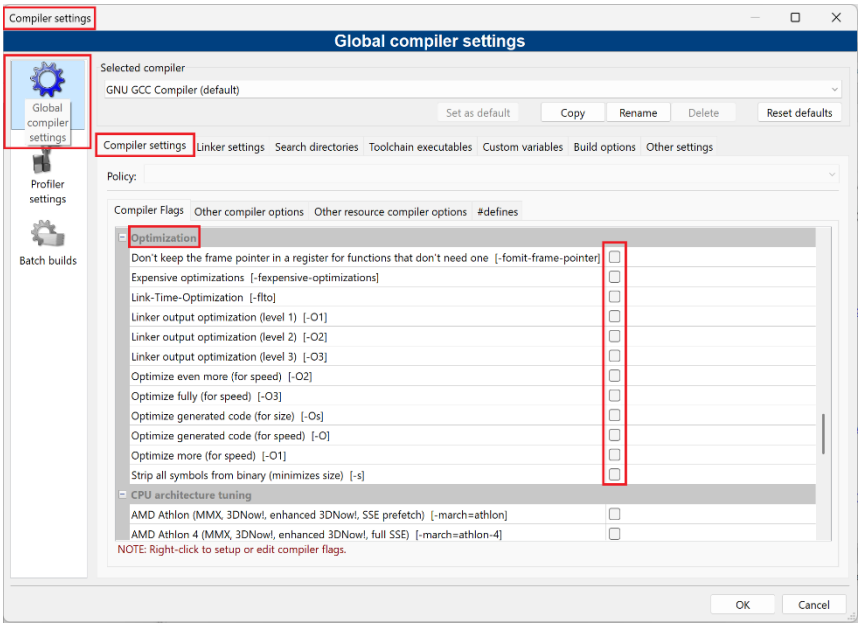

Рис. 2. Вікно Compiler settings

У вікні Compiler settings, що відкрилося, потрібно обрати вкладку Global compiler settings у лівому верхньому куті цього вікна (ця вкладка, як правило, встановлюється за замовченням), потім у цій вкладці обрати вкладку Compiler settings другого рівня, а потім вкладку опцій компілятора Compiler Flags.

Після цього у вікні опцій компілятора прокрутити перелік опцій до групи опцій оптимізації Optimization і зняти «пташки» з квадратиків абсолютно усіх опцій оптимізації, щоб вони були пустими, як це показано на рис. 2.

Вимикання режимів оптимізації компілятора GUI MS Visual Studio

У MS Visual Studio є два режими компіляції, Debug (режим компіляції для налагодження) і Release (режим компіляції остаточної версії програми/проєкту). Після стандартної інсталяції MS Visual Studio встановлюється режим Debug. Опції компілятора (зокрема опції оптимізації) для цих режимів після інсталяції є різними. Ці опції можна змінювати і встановлювати як окремо для кожного з режимів Debug і Release, так і одночасно встановлювати однакові опції для обох режимів.

Для встановлення (увімкнення/вимкнення) опцій компілятора для оптимізації коду одночасно для обох режимів Debug і Release треба виконати наступну послідовність дій в GUI MS Visual Studio:

1. При відкритому проєкту курсової роботи відкрити меню Project і обрати команду Properties цього проєкту (рис. 3).

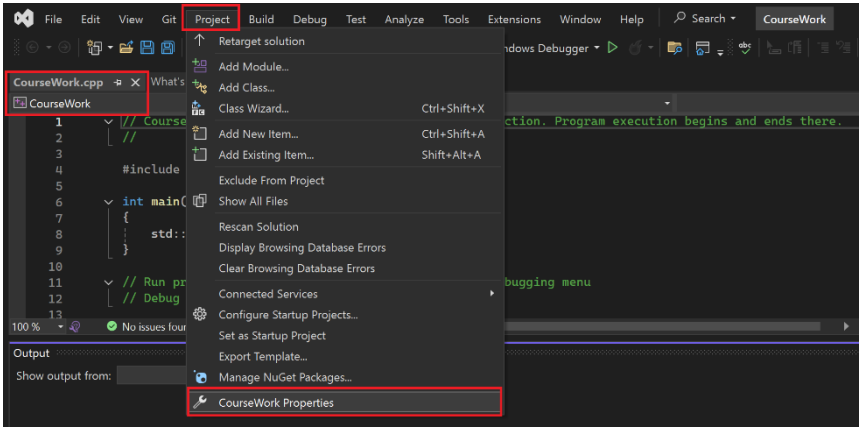

Рис. 3. Обрання команди Property поточного проєкту

2. В результаті відкривається вікно Property Pages поточного проєкту, яке показане на рис. 4. В полі Configuration видно, що стандартним активним режимом компіляції є режим Debug. Для того, щоб встановити інші значення опцій компілятора для обох режимів компіляції (Debug і Release) одночасно, потрібно в полі Configuration обрати значення All Configurations, як показано на рис. 4.

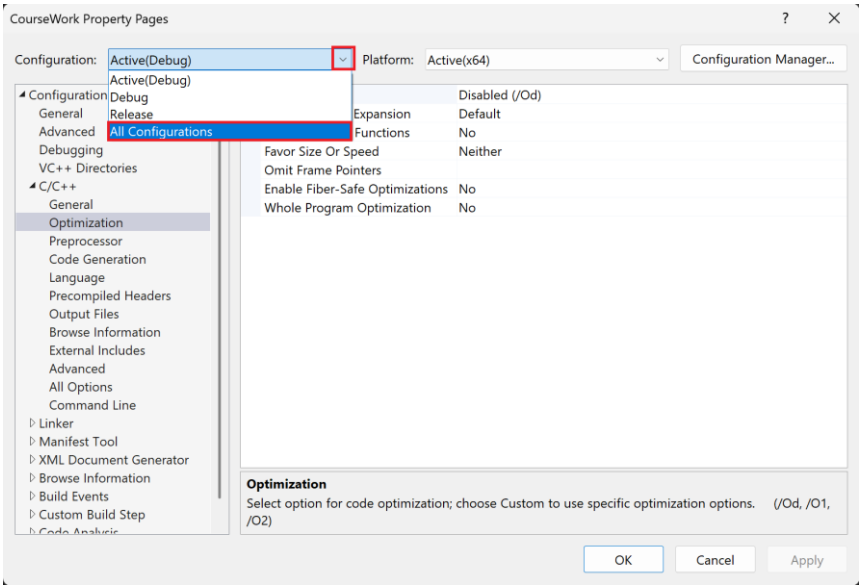

Рис. 4. Обрання значення All Configurations у вікні Property Pages поточного проєкту

3. Після цього на лівій панелі Configuration Properties потрібно спочатку обрати групу опцій C/C++, а потім в цій групі обрати підгрупу опцій оптимізації Optimization, як показано на рис. 5. Як видно з цього рисунку, в стандартних

налаштуваннях оптимізації не всі опції оптимізації є вимкненими.

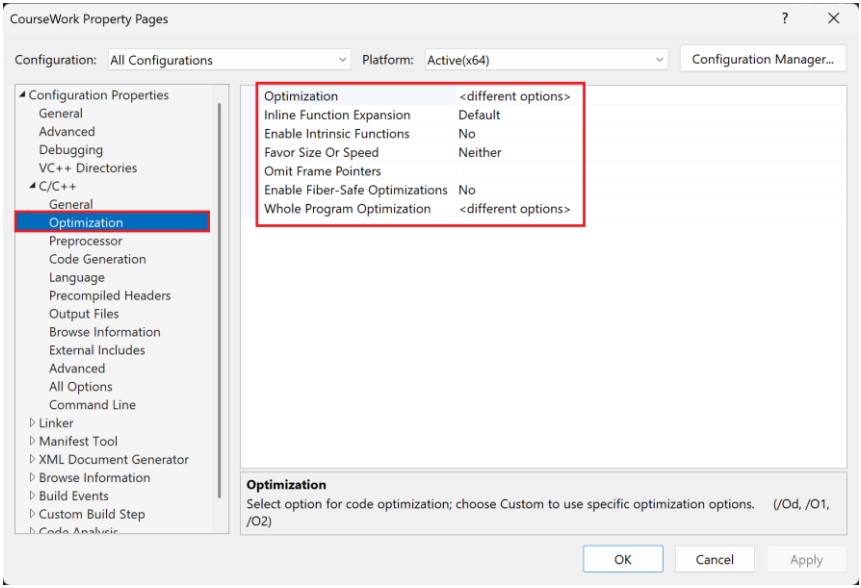

Рис. 5. Вікно групи опцій оптимізації Optimization

4. Для зміни значення певної опції оптимізації, наприклад, опції загальної оптимізації Optimization, як показано на рис. 6, потрібно клікнути спочатку на назві цієї опції, а потім на «пташці» з правої стороні відповідного рядка. В результаті відкриється меню можливих значень цієї опції, серед яких можна обрати потрібне значення.
5. Значення, які потрібно обрати для вимкнення усіх опцій оптимізації при компіляції поточного проєкту, показано на рис. 7.

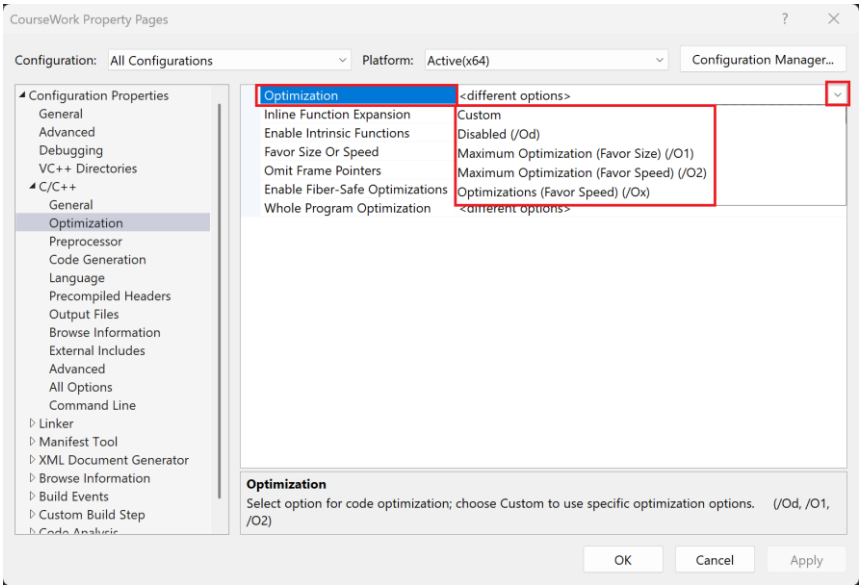

Рис. 6. Зміна значення певної опції оптимізації

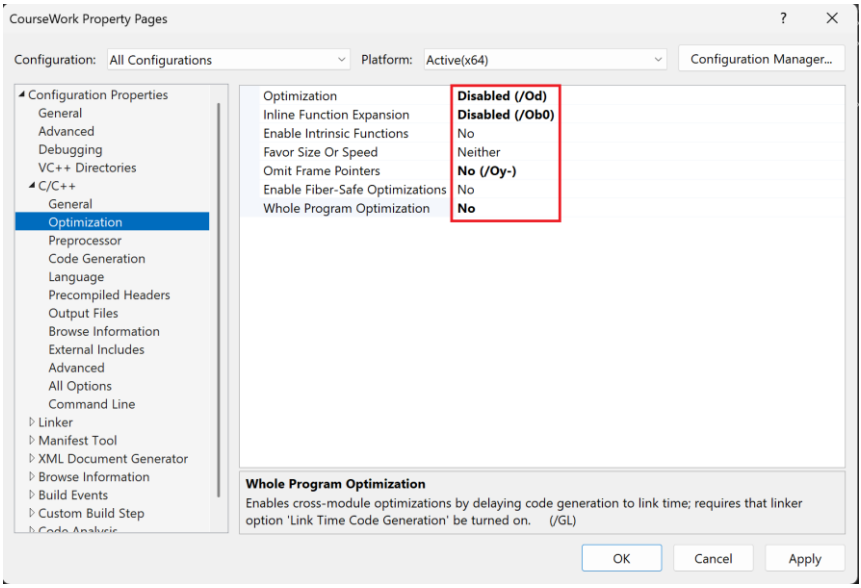

Рис. 7. Значення для вимкнення усіх опцій оптимізації

ЗАДАЧІ

Задача 1. Відсортувати окремо кожен переріз тривимірного масиву Arr3D [P][M][N] наскрізно по стовпчиках за незменшенням.

Задача 2. Відсортувати окремо кожен переріз тривимірного масиву Arr3D [P][M][N] таким чином: переставити стовпчики перерізу за незменшенням значень елементів його рядка з індексом 0 (нуль).

Задача 3. Відсортувати окремо кожен переріз тривимірного масиву Arr3D [P][M][N] таким чином: переставити стовпчики перерізу за незменшенням сум їх елементів.

Задача 4. Відсортувати тривимірний масив Arr3D [P][M][N] таким чином: переставити перерізи масиву за незменшенням значень вектора елементів з нульовими індексами кожного перерізу A [*][0][0].

Задача 5. Відсортувати тривимірний масив Arr3D [P][M][N] таким чином: переставити перерізи масиву за незменшенням сум їх елементів.

Зауваження:

1) Першим елементом тривимірного масиву є елемент з індексами Arr3D[0][0][0], останнім елементом – елемент з індексами Arr3D [P-1][M-1][N-1]. Зміна третього індексу відбувається найшвидше за інші, а першого індексу – найповільніше, ніж інших індексів.
2) Перерізи тривимірного масиву Arr3D [P][M][N] треба брати тільки по першому виміру (індексу). Тобто у масиві Arr3D [P][M][N] треба брати P перерізів.

АЛГОРИТМИ СОРТУВАННЯ

Порядкові номери алгоритмів цього переліку використовуються в таблиці 5 варіантів курсової роботи для визначення конкретних алгоритмів, що задані у певному варіанті для реалізації у курсовій роботі.

1. Алгоритм №1 сортування прямим методом вставки (Додаток 1, рис. Д1-1).

В цьому алгоритмі пошук місця вставки виконується методом лінійного пошуку, починаючи від початку масиву (зліва).

2. Алгоритм №2 сортування прямим методом вставки (Додаток 1, рис. Д1-2).

В цьому алгоритмі пошук місця вставки виконується методом лінійного пошуку, починаючи від елемента, що вставляється (справа без бар'єра).

3. Алгоритм №3 сортування прямим методом вставки (Додаток 1, рис. Д1-3).

В цьому алгоритмі пошук місця вставки виконується методом лінійного пошуку з використанням бар'єра, починаючи від елемента, що вставляється (справа з бар'єром).

4. Алгоритм №4 сортування прямим методом вставки (Додаток 1, рис. Д1-4).

В цьому алгоритмі пошук місця вставки виконується методом з двійковим пошуком місця вставки.

5. Алгоритм №1 сортування прямим методом вибору (Додаток 1, рис. Д1-5).

В цьому алгоритмі під час пошуку запам'ятовується як поточний мінімальний елемент масиву, так і його індекс.

6. Алгоритм №2 сортування прямим методом вибору (Додаток 1, рис. Д1-6).

В цьому алгоритмі під час пошуку запам'ятовується тільки індекс поточного мінімального елемента масиву.

7. Алгоритм №3 сортування прямим методом вибору (Додаток 1, рис. Д1-7).

В цьому алгоритмі пошук мінімального елемента виконується так само, як у Алгоритмі №1 методу вибору, але перестановка мінімального елемента з першим елементом поточної невідсортованої частини виконується тільки за умовою необхідності, тобто якщо позиції цих двох елементів не збігаються.

8. Алгоритм №4 сортування прямим методом вибору (Додаток 1, рис. Д1-8).

В цьому алгоритмі пошук мінімального елемента виконується так само, як у Алгоритмі №2 методу вибору, але перестановка мінімального елемента з першим елементом поточної невідсортованої частини виконується тільки за умовою необхідності, тобто якщо позиції цих двох елементів не збігаються.

9. Алгоритм №5 сортування прямим методом вибору (Додаток 1, рис. Д1-9).

Цей алгоритм працює за принципом Алгоритму №1 методу вибору але з одночасним пошуком як мінімального, так і максимального елементів невідсортованої частини масиву і перестановкою мінімального елемента з першим елементом невідсортованої частини масиву, а максимального елемента – з останнім елементом невідсортованої частини масиву.

10. Алгоритм №6 сортування прямим методом вибору (Додаток 1, рис. Д1-10).

Цей алгоритм працює за принципом Алгоритму №2 методу вибору але з одночасним пошуком як мінімального, так і максимального елементів невідсортованої частини масиву і перестановкою мінімального елемента з першим елементом невідсортованої частини масиву, а максимального елемента – з останнім елементом невідсортованої частини масиву.

11. Алгоритм №7 сортування прямим методом вибору (Додаток 1, рис. Д1-11).

Цей алгоритм працює за принципом Алгоритму №3 методу вибору але з одночасним пошуком як мінімального, так і максимального елементів невідсортованої частини масиву і перестановкою мінімального елемента з першим елементом невідсортованої частини масиву, а максимального елемента – з останнім елементом невідсортованої частини масиву.

12. Алгоритм №8 сортування прямим методом вибору (Додаток 1, рис. Д1-12).

Цей алгоритм працює за принципом Алгоритму №4 методу вибору але з одночасним пошуком як мінімального, так і максимального елементів невідсортованої частини масиву і перестановкою мінімального елемента з першим елементом невідсортованої частини масиву, а максимального елемента – з останнім елементом невідсортованої частини масиву.

13. Алгоритм №1 сортування прямим методом обміну (Додаток 1, рис. Д1-13).

Цей алгоритм є найпростішим алгоритмом сортування прямим методом обміну без додаткових модифікацій.

14. Алгоритм №2 сортування прямим методом обміну (Додаток 1, рис. Д1-14).

У цьому алгоритмі використовується прапорець для фіксації факту відсутності перестановок, що означає закінчення сортування.

15. Алгоритм №3 сортування прямим методом обміну (Додаток 1, рис. Д1-15).

У цьому алгоритмі використовується запам’ятовування місця останньої перестановки елементів для переставляння границі відсортованої частини масиву на більшу кількість елементів і зменшення кількості проходів.

16. Алгоритм №4 сортування прямим методом обміну (Додаток 1, рис. Д1-16).

Цей алгоритм має власну назву «Шейкерне сортування» і використовує чергування напряму проходів по масиву зліва направо та справа наліво.

17. Гібридний алгоритм сортування «вставка – обмін» (Додаток 1, рис. Д1-17).

В основі цього алгоритму лежить алгоритм №2 сортування прямим методом вставки (додаток 1, рис. 2), але в якому для звільнення місця вставки замість зсуву елементів виконується обмін місцями елементів, що порівнюються.

18. Гібридний алгоритм сортування «вибір№1 – обмін» (Додаток 1, рис. Д1-18).

В основі цього алгоритму лежить алгоритм №1 сортування прямим методом вибору (додаток 1, рис. 5), але в якому під час пошуку мінімального елемента у невідсортованій частині масиву замість запам'ятовування поточного мінімального елемента та його індексу кожен раз відразу виконується обмін місцями поточного мінімального елемента і першого елемента невідсортованої частини масиву.

19. Гібридний алгоритм сортування «вибір№2 – обмін» (Додаток 1, рис. Д1-19).

В основі цього алгоритму лежить алгоритм №2 сортування прямим методом вибору (додаток 1, рис. 6), але в якому під час пошуку мінімального елемента у невідсортованій частині масиву замість запам'ятовування індексу поточного мінімального елемента кожен раз відразу виконується обмін місцями поточного мінімального елемента і першого елемента невідсортованої частини масиву.

20. Гібридний алгоритм сортування «вибір№5 – обмін» (Додаток 1, рис. Д1-20).

В основі цього алгоритму лежить алгоритм №5 сортування прямим методом вибору (додаток 1, рис. 9), але в якому під час пошуку мінімального та максимального елементів у невідсортованій частині масиву, замість запам'ятовування значень поточних мінімального та максимального елементів та їх індексів, вони кожен раз відразу обмінюються місцями з першим та останнім елементами невідсортованої частини масиву відповідно.

21. Гібридний алгоритм сортування «вибір№6 – обмін» (Додаток 1, рис. Д1-21).

В основі цього алгоритму лежить алгоритм №6 сортування прямим методом вибору (додаток 1, рис. 10), але в якому під час пошуку мінімального та максимального елементів у невідсортованій частині масиву, замість запам'ятовування поточних значень їх індексів, поточні мінімальний та максимальний елементи кожен раз відразу обмінюються місцями з першим та останнім елементами невідсортованої частини масиву відповідно.

22. Алгоритм №1 сортування методом Шелла (класичний варіант) (Додаток 1, рис. Д1-22).

В основі цього алгоритму лежить алгоритм №2 сортування прямим методом вставки (додаток 1, рис. 2), який виконується у декілька етапів. Кількість етапів сортування та відстані між елементами, що сортуються на кожному етапі, взяти в залежності від довжини масиву ключів сортування.

23. Алгоритм №2 сортування методом Шелла (варіант на основі гібридного алгоритму «вставка-обмін») (Додаток 1, рис. Д1-23).

В основі цього алгоритму лежить гібридний алгоритм сортування «вставка – обмін» (додаток 1, рис. 17), який виконується у декілька етапів. Кількість етапів сортування та відстані між елементами, що сортуються на кожному етапі, взяти в залежності від довжини масиву ключів сортування.

24. Алгоритм сортування методом «швидкого сортування» (сортуванням Хоара) (Додаток 1, рис. Д1-24).

В завданні використовується класичний варіант швидкого сортування, який реалізований на основі рекурсивної функції QuickSort.

СПОСОБИ ВИКОНАННЯ ПРОЦЕСУ СОРТУВАННЯ МАСИВУ

1. Переписати елементи заданого двовимірного масиву у додатковий одновимірний масив. Виконати сортування. Повернути результат у початковий масив.
2. Не використовуючи додаткового масиву, виконати сортування перетворюючи один індекс ***k*** елементів "уявного" вектора у відповідні індекси ***i, j*** елементів конкретного перерізу заданого тривимірного масиву за допомогою формул:

   i = k % M;

   j = k / M;

   де ***M*** – кількість рядків (довжина стовпчика) переріза.
3. Виконати сортування, здійснюючи обхід безпосередньо по елементах заданого двовимірного масиву, не використовуючи додаткових масивів і перетворень індексів, тобто пристосувавши до наскрізного сортування двовимірного масиву сам принцип сортування заданим методом та алгоритмом.
4. Використовуючи елементи рядка з індексом 0 (нуль) кожного перерізу масиву як ключі сортування, переставляти відповідні стовпчики кожен раз, коли треба переставляти ключі. *При перестановці стовпчиків потрібно саме копіювати їх елементи, а не копіювати вказівники на них, використовуючи операції з вказівниками мови C/C++.*

5. В якості першого етапу сортування сформувати додатковий вектор Sum, довжина якого дорівнює кількості стовпчиків і значеннями якого є суми елементів відповідних стовпчиків. Використовуючи елементи вектора Sum як ключі сортування, переставляти відповідні стовпчики кожен раз, коли треба переставляти ключі. *При перестановці стовпчиків потрібно саме копіювати їх елементи, а не копіювати вказівники на них, використовуючи операції з вказівниками мови C/C++.* Час формування додаткового вектора Sum входить у загальний час сортування.
6. Використовуючи значення вектора елементів з нульовими індексами кожного перерізу Arr3D[*][0[0] як ключі сортування, переставляти відповідні перерізи кожен раз, коли треба переставляти ключі. *При перестановці перерізів потрібно саме копіювати їх елементи, а не копіювати вказівники на них, використовуючи операції з вказівниками мови C/C++.*
7. В якості першого етапу сортування сформувати додатковий вектор Sum, довжина якого дорівнює кількості перерізів і значеннями якого є суми елементів відповідних перерізів. Використовуючи елементи вектора Sum як ключі сортування, переставляти відповідні перерізи кожен раз, коли треба переставляти ключі. *При перестановці перерізів потрібно саме копіювати їх елементи, а не копіювати вказівники на них, використовуючи операції з вказівниками мови C/C++.* Час формування додаткового вектора Sum входить у загальний час сортування.

ВИПАДКИ ВІДСОРТОВАНОСТІ МАСИВУ

Для усіх варіантів курсової роботи (усіх задач та усіх алгоритмів) потрібно виконати виміри часу сортування для трьох випадків відсортованості ключів сортування тривимірного масиву:

Випадок 1. Початковий масив містить елементи, при яких ключі сортування масиву мають значення, що є прямо відсортованими згідно з умовою задачі (тобто **строго за збільшенням**).

Випадок 2. Початковий масив містить елементи, при яких ключі сортування масиву мають значення, що є випадковими числами (як правило, достатньо випадкових чисел **з діапазону від 0 до M*N**).

Випадок 3. Початковий масив містить елементи, при яких ключі сортування масиву мають значення, що є обернено відсортованими згідно з умовою задачі (тобто **строго за зменшенням**).

ЗАПОВНЕННЯ МАСИВУ ЗГІДНО З ВИПАДКАМИ ВІДСОРТОВАНОСТІ ДЛЯ КОЖНОЇ ІЗ ЗАДАЧ

На етапі виконання виміру часу сортування масиву різними заданими алгоритмами, коли потрібно брати масиви дуже великих розмірів, рекомендується заповнювати масив згідно з вищезазначеними трьома випадками відсортованості масиву за допомогою лічильника, який послідовно зростає (Випадок 1), послідовно зменшується (Випадок 3), або за допомогою функції взяття випадкових чисел із заданого діапазону (Випадок 2).

Розглянемо способи заповнення масиву, які відповідають заданим випадкам відсортованості для кожної із п’яти задач курсової роботи.

Заповнення масиву для Задачі 1

Оскільки в Задачі 1 кожен переріз сортується окремо, то достатньо витримати умови випадків відсортованості окремо для кожного перерізу. Тим більше, що такий підхід дозволяє оголошувати тривимірні масиви більшого розміру, не порушуючи цих умов випадків відсортованості.

Для **Випадку 1** заповнення кожного перерізу масиву наскрізно по стовпчиках прямо відсортованими елементами, тобто строго за збільшенням, можна використати наступний спосіб:

```
for (int k=0; k<P; k++) {
    int number=1;
    for (int j=0; j<N; j++)
        for (int i=0; i<M; i++)
            Arr3D[k][i][j] = number++;
}
```

Для **Випадку 2** заповнення кожного перерізу масиву випадковими елементами з діапазону від 0 до M*N, можна використати наступний спосіб:

```
for (int k=0; k<P; k++)
    for (int j=0; j<N; j++)
        for (int i=0; i<M; i++)
            Arr3D[k][i][j] = rand() % (M*N);
```

Для **Випадку 3** заповнення кожного перерізу масиву наскрізно по стовпчиках обернено відсортованими елементами, тобто строго за зменшенням, можна використати наступний спосіб:

```
for (int k=0; k<P; k++) {
    int number = M*N;
    for (int j=0; j<N; j++)
        for (int i=0; i<M; i++)
            Arr3D[k][i][j] = number--;
}
```

Заповнення масиву для Задач 2 та 3

Оскільки в Задачі 2 ключами сортування є нульові рядки кожного перерізу, то саме ці рядки обов'язково повинні відповідати випадкам відсортованості. А значення інших елементів варто обирати так, щоб було легко відслідкувати коректність процесу сортування. А для Задачі 3 важливо, щоб для Випадків 1 та 3 була початкова відсортованість стовпчиків за сумами їх елементів.

Для **Випадку 1** для обох Задач 2 та 3 заповнення масиву згідно з умовою прямої відсортованості ключів сортування, тобто строго за збільшенням, можна використати спосіб, при якому всі елементи певного стовпчика заповнюється значеннями його номера j:

```
for (int k=0; k<P; k++) {
   for (int j=0; j<N; j++)
      for (int i=0; i<M; i++)
         Arr3D[k][i][j] = j;
}
```

Для **Випадку 2** для обох Задач 2 та 3 заповнення масиву згідно з умовою випадкової відсортованості ключів сортування, можна використати спосіб, при якому кожен переріз масиву заповнюється випадковими елементами з діапазону від 0 до M*N:

```
for (int k=0; k<P; k++)
      for (int j=0; j<N; j++)
         for (int i=0; i<M; i++)
            Arr3D[k][i][j] = rand() % (M*N);
```

Для **Випадку 3** для обох Задач 2 та 3 заповнення масиву згідно з умовою оберненої відсортованості ключів сортування, тобто строго за зменшенням, можна використати спосіб, при якому всі елементи певного стовпчика заповнюється значеннями, які є симетричними до його номера j відносно кількості стовпчиків N:

```
for (int k=0; k<P; k++) {
   for (int j=0; j<N; j++)
      for (int i=0; i<M; i++)
         Arr3D[k][i][j] = N-1-j;
}
```

Заповнення масиву для Задач 4 та 5

Оскільки в Задачі 4 ключами сортування є нульові елементи кожного перерізу Arr3D[*][0[0], то саме цей вектор обов'язково повинен відповідати випадкам відсортованості. А значення інших елементів варто обирати так, щоб було легко відслідкувати коректність процесу сортування. А для Задачі 5 важливо, щоб для Випадків 1 та 3 була початкова відсортованість перерізів за сумами їх елементів.

Для **Випадку 1** для обох Задач 4 та 5 заповнення масиву згідно з умовою прямої відсортованості ключів сортування (вектора Arr3D[*][0[0] або сум елементів перерізів), тобто строго за збільшенням, можна використати спосіб, при якому всі елементи певного перерізу заповнюється значеннями його номера k:

```
for (int k=0; k<P; k++) {
   for (int j=0; j<N; j++)
      for (int i=0; i<M; i++)
         Arr3D[k][i][j] = k;
}
```

Для **Випадку 2** для обох Задач 4 та 5 заповнення масиву згідно з умовою випадкової відсортованості ключів сортування (вектора Arr3D[*][0[0] або сум елементів перерізів), можна використати спосіб, при якому кожен переріз масиву заповнюється випадковими елементами з діапазону від 0 до M*N:

```
for (int k=0; k<P; k++)
      for (int j=0; j<N; j++)
         for (int i=0; i<M; i++)
            Arr3D[k][i][j] = rand() % (M*N);
```

Для **Випадку 3** для обох Задач 4 та 5 заповнення масиву згідно з умовою оберненої відсортованості ключів сортування (вектора Arr3D[*][0[0] або сум елементів перерізів), тобто строго за зменшенням, можна використати спосіб, при якому всі елементи певного перерізу заповнюється значеннями, які є симетричними до його номера k відносно кількості перерізів P:

```
for (int k=0; k<P; k++) {
   for (int j=0; j<N; j++)
      for (int i=0; i<M; i++)
         Arr3D[k][i][j] = P-1-k;
}
```

ВИПАДКИ ДОСЛІДЖЕННЯ

Як зазначалося в розділі «Виділення пам’яті для тривимірного масиву», для того, щоб виміри були найбільш коректними, розміри тривимірного масиву, як правило, треба брати максимально можливими. Але з іншої сторони, на деяких сучасних комп’ютерах, обсяг пам’яті яких досягає вже сотні гігабайтів, час отримання однієї порівняльної таблиці алгоритмів сортування масиву з максимально можливим розміром (згідно з методикою, коли виміри повторюються до тридцяти разів на один випадок) вже може досягати декількох годин. Причому цей час сильно залежить від характеристик комп’ютера/ноутбука, який є у конкретного студента.

Тому, перед початком виконання всіх вимірів часу сортування алгоритмів для всі випадків дослідження своєї задачі, кожному студенту рекомендується спочатку виконати невелику кількість експериментальних вимірів для кожного алгоритму для того, щоб з’ясувати розмір тривимірного масиву, який є доцільним для виконання вимірів саме на його конкретному комп’ютері/ноутбуку. Під доцільністю в даному випадку розуміються дві ситуації:

1) для масиву взятого розміру час сортування всіх алгоритмів для всіх випадків відсортованості не дорівнює значенням, близьким до нуля;
2) час отримання однієї порівняльної таблиці сортування (такого виду, як зазначено в технічному завданні) для трьох алгоритмів і трьох випадків початкової відсортованості не перевищує однієї години.

Якщо на комп'ютері студента не вистачає обсягу оперативної пам'яті для оголошення зазначених нижче розмірів масивів для певних випадків дослідження або час отримання однієї порівняльної таблиці алгоритмів сортування перевищує годину, то дозволяється оголосити масиви з меншими (але максимально можливими для цього комп'ютера) розмірами, **але обов'язково зберігаючи пропорції між кількістю перерізів, рядків та стовпчиків і загальну логіку описаних нижче випадків та варіантів дослідження**.

Послідовність дій експериментального підбору розмірів тривимірного масиву для виконання дослідження з вимірів часу сортування на конкретному комп'ютері/ноутбуку:

1. Серед всіх рекомендованих розмірів масиву для випадків дослідження своєї задачі взяти масив **мінімального** розміру і зробити виміри, отримавши порівняльну таблиці алгоритмів всіх сортування. Якщо найменші з отриманих значень є не меншими 100-200 тактів, то такий розмір підходить в якості мінімального для отримання всіх наступних таблиць результатів вимірів часу. Якщо є три або більше значень менших 100 тактів, то перевірити наступний за мінімальністю розмір із рекомендованих.
2. Серед всіх рекомендованих розмірів масиву для випадків дослідження своєї задачі взяти масив **максимального** розміру і зробити виміри, отримавши порівняльну таблиці алгоритмів всіх сортування. Якщо час отримання такої

таблиці не перевищує однієї години, то такий розмір підходить в якості максимального для отримання всіх наступних таблиць результатів вимірів часу. Якщо перевищує, то перевірити попередній за максимальністю розмір із рекомендованих.

Зауваження: загалом кожному студенту потрібно бути готовим то того, що на отримання результатів всіх вимірів часу сортування усього дослідження може знадобитись 1-2 дні і враховувати цей факт при плануванні своєї роботи над курсовою роботою.

З врахуванням вищеописаного, розглянемо доцільні випадки дослідження, які потрібно опрацювати в курсовій роботі. Для виконання вимірів часу сортування алгоритмів і подальшого порівняльного аналізу рекомендується взяти перелічені нижче варіанти розмірів тривимірного масиву в якості випадків дослідження.

Випадки дослідження для Задачі 1

Випадок дослідження I. Залежність часу роботи алгоритмів від форми перерізу масиву.

Рекомендовані розміри масиву для досліджень:

P = const = 3, M = var, N = var, M*N = const = 100000000.

1) M = 10; N = 10000000;

2) M = 100; N = 1000000;

3) M = 1000; N = 100000;

4) M = 10000; N = 10000;

5) M = 100000; N = 1000;

6) M = 1000000; N = 100;

7) M = 10000000; N = 10;

Вектор довжиною NV = M*N = 100000000.

Для порівняння час сортування вектора довжиною NV = M*N помножити на P.

Випадок дослідження II. Залежність часу роботи алгоритмів від кількості перерізів масиву

Рекомендовані розміри масиву для досліджень:

P = var, M = const = 2000, N = const = 2000 (кількість елементів у перерізі M*N = 4000000)

1) P = 2;

2) P = 4;

3) P = 8;

4) P = 16;

5) P = 32;

6) P = 64;

7) P = 128;

Час сортування вектора довжиною NV = 4000000 (дорівнює кількості елементів у перерізі) помножити на P. Вимір і порівняння для вектора достатньо зробити тільки для найбільшого P.

Випадок дослідження III. Залежність часу роботи алгоритмів від розміру перерізів масиву

Порівняння з вектором для цього випадку не обов'язкове.

Для бажаючих порівняти з вектором: тривимірний масив кожного розміру P*M*N (де M=N) потрібно окремо порівнювати з вектором відповідної довжини M*N, і час сортування такого вектора помножити на кількість перерізів P.

Рекомендовані розміри масиву для досліджень:

P = const = 3

1) M = N = 4 (M*N = 16)

2) M = N = 8 (M*N = 64)

3) M = N = 16 (M*N = 256)

4) M = N = 32 (M*N = 1024)

5) M = N = 64 (M*N = 4096)

6) M = N = 128 (M*N = 16384)

7) M = N = 256 (M*N = 65536)

8) M = N = 512 (M*N = 262144)

9) M = N = 1024 (M*N = 1048576)

10) M = N = 2048 (M*N = 4194304)

11) M = N = 4096 (M*N = 16777216)

12) M = N = 8192 (M*N = 67108864)

Випадки дослідження для Задач 2 та 3

Випадок дослідження I. Залежність часу роботи алгоритмів від довжини стовпчиків масиву

Рекомендовані розміри масиву для досліджень:

Кількість ключів у перерізі (N) і загальна кількість ключів (N*P) є константами.

P = const = 3, N = const = 20000

1) M = 1;

2) M = 4;

3) M = 16;

4) M = 64;

5) M = 256;

6) M = 1024;

7) M = 4096;

8) M = 8192;

Вектор довжиною кількості ключів у перерізі N = 20000.

Для порівняння час сортування такого вектора помножити на P.

Випадок дослідження II. Залежність часу роботи алгоритмів від форми перерізів масиву

Порівняння з вектором для цього випадку не потрібне.

Рекомендовані розміри масиву для досліджень:
Кількість елементів у кожному перерізі (M*N) є константою.
P = const = 3, M = var, N = var, M*N = const = 100000000.
1) M = 10; N = 10000000;
1) M = 100; N = 1000000;
2) M = 1000; N = 100000;
3) M = 10000; N = 10000;
4) M = 100000; N = 1000;
5) M = 1000000; N = 100;
6) M = 10000000; N = 10;

Випадок дослідження III. Залежність часу роботи алгоритмів від кількості ключів у кожному перерізі масиву при однаковій загальній кількості ключів у всьому масиві

Порівняння з вектором для цього випадку не потрібне.

Рекомендовані розміри масиву для досліджень:
Кількість ключів у перерізі N = var
Загальна кількість ключів у матриці P*N = const = 1000000

Довжина стовпчика M = const = 1000

P = var, N = var, M = const = 1000, P*N = const = 1000000

1) P = 10; N = 100000;

2) P = 100; N = 10000;

3) P = 1000; N = 1000;

4) P = 10000; N = 100;

5) P = 100000; N = 10;

Випадки дослідження для Задач 4 та 5

Випадок дослідження I. Залежність часу роботи алгоритмів від розміру перерізів масиву

Рекомендовані розміри масиву для досліджень:

Кількість ключів (перерізів) P = const = 2000

Форма перерізу – однакова (квадрат)

M = var, N = var, M = N

1) M = N = 8 (M*N = 64)

2) M = N = 16 (M*N = 256)

3) M = N = 32 (M*N = 1024)

4) M = N = 64 (M*N = 4096)

5) M = N = 128 (M*N = 16384)

6) M = N = 256 (M*N = 65536)

7) M = N = 512 (M*N = 262144)

8) M = N = 1024 (M*N = 1048576)

Вектор довжиною P = 2000

Випадок дослідження ІІ. Залежність часу роботи алгоритмів від форми перерізів масиву

Рекомендовані розміри масиву для досліджень:

Кількість ключів (перерізів) P = const = 2000

M = var, N = var, M*N = const

Загальна кількість елементів P*M*N = const

1) M = 4; N = 65536;

2) M = 32; N = 8192;

3) M = 256; N = 1024;

4) M = 1024; N = 256;

5) M = 8192; N = 32;

6) M = 65536; N = 4;

Вектор довжиною P = 2000

ВАРІАНТИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

Варіанти індивідуальних завдань наведені у таблиці 5.

Початковий номер варіантів для крупи з шифром **КВ-XY** вираховується за формулою:

(25*X + 32*(Y – 1) + 1) mod 160

Таблиця 5. Варіанти курсової роботи

Варіант	Задача	Номери алгоритмів зазначені згідно з їх порядковими номерами у розділі «Алгоритми сортування» (в дужках – окремий спосіб обходу)	Способи виконання сортування
1	4	17, 12, 22	6
2	3	5, 10, 24	5
3	5	4, 10, 23	7
4	1	10, 23, 24	2
5	2	19, 12, 24	4
6	4	15, 22, 23	6
7	1	8, 13, 8	3
8	5	2, 12, 22	7
9	1	2, 13, 12	3
10	2	21, 6, 22	4
11	3	7, 23, 24	5
12	1	5, 13, 20	3
13	4	9, 8, 22	6
14	2	2, 8, 24	4
15	1	8, 14, 9	3

16	5	21, 6, 23	7
17	1	1, 14, 10	3
18	3	16, 6, 24	5
19	1	5, 14, 12	3
20	4	2, 6, 24	6
21	2	9, 8, 12	4
22	1	7, 14, 21	3
23	5	19, 8, 22	7
24	1	21, 23, 24	2
25	3	20, 8, 12	5
26	4	19, 10, 22	6
27	1	2, 15, 10	3
28	2	15, 10, 24	4
29	1	21	1 – 3
30	5	17, 6, 24	7
31	3	3, 12, 24	5
32	1	7, 15, 20	3
33	2	21, 12, 23	4
34	1	2, 17, 9	3
35	4	15, 12, 24	6
36	3	7, 10, 23	5
37	1	6 (3), 21 (3), 24 (2)	в дужках
38	5	14, 8, 23	7
39	1	7, 17, 12	3
40	2	2, 10, 23	4
41	4	11, 10, 24	6

42	1	1, 17, 21	3
43	5	21, 8, 12	7
44	3	14, 12, 23	5
45	1	22, 23, 24	2
46	4	9, 12, 23	6
47	1	2, 13, 10	3
48	5	19, 10, 24	7
49	1	6, 13, 12	3
50	4	4, 23, 24	6
51	2	4, 12, 22	4
52	1	7, 13, 20	3
53	3	16, 10, 22	5
54	1	2, 14, 9	3
55	4	2, 8, 23	6
56	5	17, 12, 23	7
57	1	5, 14, 10	3
58	3	18, 22, 24	5
59	2	9, 6, 23	4
60	1	7, 14, 12	3
61	5	15, 6, 24	7
62	1	1, 14, 21	3
63	4	21, 6, 22	6
64	1	9, 22, 24	2
65	3	20, 6, 22	5
66	2	11, 8, 22	4
67	1	6, 15, 10	3

68	2	15, 22, 24	4
69	1	11	1 – 3
70	5	11, 8, 23	7
71	3	1, 8, 24	5
72	1	1, 15, 20	3
73	4	19, 10, 23	6
74	1	6, 17, 9	3
75	1	1 (3), 11 (3), 24 (2)	в дужках
76	2	17, 10, 22	4
77	5	5, 22, 24	7
78	3	3, 12, 22	5
79	1	1, 17, 12	3
80	1	5, 17, 21	3
81	4	16, 6, 23	6
82	3	9, 8, 22	5
83	5	3, 8, 24	7
84	1	12	1 – 3
85	2	18, 8, 23	4
86	4	14, 10, 24	6
87	1	7, 13, 9	3
88	5	1, 23, 24	7
89	1	1, 13, 11	3
90	2	20, 23, 24	4
91	3	11, 12, 23	5
92	1	2 (3), 12 (3), 22 (2)	в дужках
93	4	5, 12, 23	6

94	2	1, 10, 23	4
95	1	6, 13, 21	3
96	5	20, 10, 24	7
97	1	20	1 – 3
98	3	17, 10, 22	5
99	1	2, 13, 11	3
100	4	1, 22, 24	6
101	2	5, 12, 22	4
102	1	6, 14, 20	3
103	5	18, 12, 23	7
104	1	5 (3), 20 (3), 23 (2)	в дужках
105	3	21, 8, 23	5
106	4	18, 10, 23	6
107	1	1, 15, 9	3
108	2	14, 6, 23	4
109	1	5, 15, 11	3
110	5	16, 10, 22	7
111	3	4, 6, 22	5
112	1	6, 15, 12	3
113	2	18, 10, 22	4
114	1	10, 15, 21	3
115	4	14, 6, 22	6
116	3	11, 8, 24	5
117	1	5, 17, 10	3
118	5	15, 12, 24	7
119	1	6, 17, 11	3

120	2	1, 6, 22	4
121	4	7, 8, 22	6
122	1	8, 17, 20	3
123	5	20, 6, 23	7
124	3	15, 6, 24	5
125	1	9	1 – 3
126	4	5, 6, 24	6
127	1	1, 13, 9	3
128	5	18, 8, 22	7
129	1	5, 13, 11	3
130	4	3, 10, 22	6
131	2	3, 8, 24	4
132	1	14 (3), 9 (3), 22 (2)	в дужках
133	3	17, 8, 23	5
134	1	8, 13, 21	3
135	4	1, 12, 24	6
136	5	16, 22, 23	7
137	1	10	1 – 3
138	3	19, 12, 24	5
139	2	5, 22, 23	4
140	1	6, 14, 11	3
141	5	14, 10, 22	7
142	1	8, 14, 20	3
143	4	20, 8, 12	6
144	1	15 (3), 10 (3), 23 (2)	в дужках
145	3	21, 10, 23	5

146	2	7, 10, 24	4
147	1	5, 15, 9	3
148	2	14, 12, 23	4
149	1	7, 15, 11	3
150	5	7, 12, 24	7
151	3	2, 22, 23	5
152	1	8, 15, 12	3
153	4	18, 8, 24	6
154	1	2, 15, 21	3
155	3	4, 6, 23	5
156	5	9, 6, 22	7
157	1	7, 17, 10	3
158	2	16, 6, 24	4
159	1	8, 17, 11	3
160	1	2, 17, 20	3

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Базова література

1. Марченко О. І. Структури даних та алгоритми: підручник. У 2-х ч. Ч. 1. [Електронний ресурс] / О. І. Марченко, О. О. Марченко; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 9,88 Мбайт). – Київ : Просвіта, 2024. – 268 с. ISBN 978-617-7010-34-9 (Online). – Режим доступу до ресурсу: https://ela.kpi.ua/handle/123456789/66355

2. Креневич А. П., Алгоритми і структури даних. Підручник. – К.: ВПЦ "Київський Університет", 2021. – 200 с.

3. Мелешко Є. В., Якименко М. С., Поліщук Л. І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В. Ф., 2019. – 156 с.

4. Коротєєва Т. О. Алгоритми та структури даних : навч. посібник/ Т. О. Коротєєва. – Львів : Видавництво Львівської політехніки, 2014. – 280 с.

5. Алгоритми і структури даних: практикум: навч. посіб. / Н. К. Стратієнко, М. Д. Годлевський, І. О. Бородіна. – Харків: НТУ «ХПІ», 2017. – 224 с.

6. Програмування. Завдання для виконання лабораторних робіт. Для студентів спеціальності "Комп'ютерна інженерія". Укл.: О. І. Марченко, О. В. Тарасенко-Клятченко, Я. М. Клятченко. – Київ.: НТУУ "КПІ ім. Ігоря Сікорського", 2016. – 58 с.

7. Програмування мовою С: інструкції до виконання лабораторних робіт з дисципліни «Програмування-2. Програмування мовою С» [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 Комп'ютерна інженерія / КПІ ім. Ігоря Сікорського; В. О. Романкевич, О. В. Тарасенко-Клятченко, Я. М. Клятченко, – Електронні текстові дані (1 файл: 2,8 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2021. – 142 с.

8. Програмування мовою С. Задачі до практичних занять з кредитного модуля «Програмування-1. Основи програмування» [Електронний ресурс] : навч. посіб.для студ. спеціальності «123 Комп'ютерна інженерія» / О. В. Тарасенко-Клятченко, Я. М. Клятченко, О. С.Михайлюк ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,2 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 43 с.

Додаткова література

1. N. Wirth. Algorithms and Data Structures, Prentice Hall, 1985.

2. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Introduction to algorithms. .—3rd ed. The MIT Press Cambridge, Massachusetts London, England, 2009. – 1292 p.

3. Robert Sedgewick. Algorithms in C++, Parts 1-4: Fundamentals, Data Structure, Sorting, Searching, Third Edition, Addison-Wesley Professional, 1998. – 740 p.

4. Donald E. Knuth. The art of programming. Volume 1. Fundamental Algorithms. Addison-Wesley Publishing Company, 1968.

5. Donald E. Knuth. The art of programming. Volume 2. Seminumerical Algorithms. Addison-Wesley Publishing Company, 1969.

6. Donald E. Knuth. The art of programming. Volume 3. Sorting and Search Algorithms. Addison-Wesley Publishing Company, 1970.

7. Martin J., McClure C. Structured techniques : The basis for CASE., 1988.

8. A.V. Aho, J.E. Hopkroft, J.D. Ullman. Data Structures and Algorithms. Addison-Wesley Publishing Company, 1995.

Інформаційні ресурси

1. Марченко О. І. Youtube канал «Олександр Іванович Марченко». Режим доступу : https://www.youtube.com/@Oleksandr_Marchenko

2. Марченко О. І. Структури даних та алгоритми . Курсова робота. Пояснення завдання. Частина 1. Режим доступу: https://youtu.be/DFrcUTMBko4

3. Марченко О. І. Структури даних та алгоритми . Курсова робота. Пояснення завдання. Частина 2. Режим доступу: https://youtu.be/FPOLlG6A2x4

4. Марченко О. І. Структури даних та алгоритми . Курсова робота. Пояснення завдання. Частина 3. Режим доступу: https://youtu.be/-Tq2IA8tNsY

5. Електронний кампус НТУУ «КПІ». Матеріали з дисципліни «Структури даних та алгоритми 1. Основи алгоритмізації». – Режим доступу : http://login.kpi.ua.

6. Структури даних та алгоритми - 1. Основи алгоритмізації: Інструкції та завдання до виконання лабораторних робіт з

дисципліни «Структури даних та алгоритми» : [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 – «Комп'ютерна інженерія», 2-ге видання, виправлене та доповнене / О. І. Марченко, О. О. Марченко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 0,8 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2023. – 74 с. – Режим доступу до ресурсу:
https://ela.kpi.ua/handle/123456789/54661

7. Структури даних та алгоритми - 2. Складні структури: Інструкції та завдання до виконання лабораторних робіт з дисципліни «Структури даних та алгоритми» : [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 – «Комп'ютерна інженерія», 2-ге видання, виправлене та доповнене / О. І. Марченко, О. О. Марченко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,07 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2023. – 125 с. – Режим доступу до ресурсу: https://ela.kpi.ua/handle/123456789/54662.

ДОДАТОК 1.
АЛГОРИТМИ ДЛЯ ВИКОНАННЯ КУРСОВОЇ РОБОТИ

Для всіх нижченаведених алгоритмів, крім двох, оголошення одновимірного масиву (вектора) чисел та виклик функцій даних алгоритмів виконується так, як в прикладі виміру часу в розділі «Інструкції з виміру часу роботи алгоритмів».

Виключеннями є «Алгоритм сортування №3 методу прямої вставки (з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», з бар'єром)» та «Алгоритм методу «швидкого сортування» (сортування Хоара)».

У алгоритмі прямої вставки №3 використовується «бар'єр», для чого масив оголошується розміром на один елемент більше і сортування виконується в діапазоні індексів від 1 до N. Оголошення вектора для цього алгоритму наведено на рис. 3 зверху від функції алгоритму.

Другим виключенням є «Алгоритм методу «швидкого сортування» (сортування Хоара)», який є рекурсивним алгоритмом. У зв'язку з цим, оскільки функція `QuickSort` є рекурсивною, то виклики функції `clock()` для фіксації часу початку і закінчення сортування, на відміну від інших алгоритмів, як показано на рис.24, винесені з власне функції сортування `QuickSort` у функцію `QuickSortMeasurement`, яка виконує формування масиву і виклик рекурсивної функції `QuickSort`.

УВАГА! Винесення викликів функції `clock()` з власне сортуючої функції у функцію, з якої вона викликається, є коректним тільки для «Алгоритму методу «швидкого сортування» (сортування Хоара)», який є рекурсивним алгоритмом. Для всіх інших 23-ох алгоритмів виклики функції `clock()` повинні бути виконані строго як показано в прикладі виміру часу в розділі «Інструкції з виміру часу роботи алгоритмів» (див. «ВАЖЛИВІ ВИМОГИ ДО ВИМІРУ ЧАСУ» цього розділу).

```
clock_t Insert1(int *A, int N)
{
    int Elem, j;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=1; i<N; i++){
        Elem=A[i];
        j=0;
        while (Elem>A[j]) j=j+1;
        for (int k=i-1; k>=j; k--)
            A[k+1]=A[k];
        A[j]=Elem;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-1. Алгоритм сортування №1 методу прямої **вставки** (з лінійним пошуком місця вставки від початку послідовності, що сортується , або «зліва»)

```
clock_t Insert2(int *A, int N)
{
    int Elem, j;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=1; i<N; i++){
        Elem=A[i];
        j=i;
        while (j>0 && Elem<A[j-1]) {
            A[j]=A[j-1];
            j=j-1;
        }
        A[j]=Elem;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-2. Алгоритм сортування №2 методу прямої **вставки** (з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», без бар'єра).

```
#define VectorLength 10

int Vector[VectorLength+1];

--------------------------------------

clock_t Insert3(int *A, int N)
{
    int j;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=2; i<N+1; i++){
        A[0]=A[i];
        j=i;
        while (A[0]<A[j-1]) {
            A[j]=A[j-1];
            j=j-1;
        }
        A[j]=A[0];
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-3. Алгоритм сортування №3 методу прямої **вставки** (з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», з бар'єром).

```
clock_t Insert4(int *A, int N)
{
    int Elem, L, R, j;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=1; i<=N-1; i++){
        Elem=A[i];
        L=0;  R=i;
        while (L<R){
            j=(L+R)/2;
            if (A[j]<=Elem){
                L=j+1;
            } else R=j;
        }
        for (int k=i-1; k>=R; k--) {
            A[k+1]=A[k];
        }
        A[R]=Elem;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-4. Алгоритм сортування №4 методу прямої **вставки** (з двійковим пошуком місця вставки).

```
clock_t Select1(int *A, int N)
{
    int Min, imin;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        Min=A[s]; imin=s;
        for(int i=s+1; i<N; i++)
         if (A[i]<Min){
                Min=A[i];
                imin=i;
            }
        A[imin]=A[s];
        A[s]=Min;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-5. Алгоритм сортування №1 методу прямого **вибору**.

```
clock_t Select2(int *A, int N)
{
    int imin, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        imin=s;
        for(int i=s+1; i<N; i++)
         if (A[i]<A[imin]) imin=i;
        tmp=A[imin];
        A[imin]=A[s];
      A[s]=tmp;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-6. Алгоритм сортування №2 методу прямого **вибору**.

```
clock_t Select3(int *A, int N)
{
    int Min, imin;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        Min=A[s]; imin=s;
        for(int i=s+1; i<N; i++)
         if (A[i]<Min){
                Min=A[i];
                imin=i;
            }
        if (imin!=s) {
            A[imin]=A[s];
            A[s]=Min;
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-7. Алгоритм сортування №3 методу прямого **вибору**.

```
clock_t Select4(int *A, int N)
{
    int imin, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        imin=s;
        for(int i=s+1; i<N; i++)
         if (A[i]<A[imin]) imin=i;
        if (imin!=s) {
            tmp=A[imin];
            A[imin]=A[s];
            A[s]=tmp;
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-8. Алгоритм сортування №4 методу прямого **вибору**.

```
clock_t Select5(int *A, int N)
{
    int Min, Max;
    int L, R, imin, imax;

    clock_t time_start, time_stop;
    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        Min=A[L]; imin=L;
        Max=A[L]; imax=L;
        for(int i=L+1; i<R+1; i++){
            if (A[i] < Min){
                Min=A[i];
                imin=i;
            }
            else
                if (A[i] > Max){
                    Max=A[i];
                    imax=i;
                }
        }
        A[imin]=A[L];
        A[L]=Min;
        if (imax==L) A[imin]=A[R];
        else         A[imax]=A[R];
        A[R]=Max;

        L=L+1; R=R-1;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-9. Алгоритм сортування №5 методу прямого **вибору**.

```
clock_t Select6(int *A, int N)
{
    int L, R, imin, imax, tmp;

    clock_t time_start, time_stop;
    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        imin=L; imax=L;

        for(int i=L+1; i<R+1; i++)
            if (A[i]<A[imin]) imin=i;
            else
                if (A[i]>A[imax]) imax=i;

        tmp=A[imin];
        A[imin]=A[L];
        A[L]=tmp;
        if (imax==L) {
            tmp=A[imin];
            A[imin]=A[R];
            A[R]=tmp;
        }
        else {
            tmp=A[imax];
            A[imax]=A[R];
            A[R]=tmp;
        }
        L=L+1; R=R-1;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-10. Алгоритм сортування №6 методу прямого **вибору**.

```
clock_t Select7(int *A, int N)
{
    int Min, Max;
    int L, R, imin, imax;

    clock_t time_start, time_stop;
    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        Min=A[L]; imin=L;
        Max=A[L]; imax=L;
        for(int i=L+1; i<R+1; i++){
            if (A[i] < Min){
                Min=A[i];
                imin=i;
            }
            else
                if (A[i] > Max){
                    Max=A[i];
                    imax=i;
                }
        }
        if (imin!=L) {
            A[imin]=A[L];
            A[L]=Min;
        }
        if (imax!=R){
            if (imax==L) A[imin]=A[R];
            else         A[imax]=A[R];
            A[R]=Max;
        }
        L=L+1; R=R-1;
    }
    time_stop = clock();
    return time_stop - time_start;
}
```

Рис. Д1-11. Алгоритм сортування №7 методу прямого **вибору**.

```
clock_t Select8(int *A, int N)
{
    int L, R, imin, imax, tmp;

    clock_t time_start, time_stop;
    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        imin=L; imax=L;

        for(int i=L+1; i<R+1; i++)
            if (A[i]<A[imin]) imin=i;
            else
                if (A[i]>A[imax]) imax=i;

        if (imin!=L) {
            tmp=A[imin];
            A[imin]=A[L];
            A[L]=tmp;
        }
        if (imax!=R)
            if (imax==L) {
                tmp=A[imin];
                A[imin]=A[R];
                A[R]=tmp;
            }
            else {
                tmp=A[imax];
                A[imax]=A[R];
                A[R]=tmp;
            }
        L=L+1; R=R-1;
    }
    time_stop = clock();
    return time_stop - time_start;
}
```

Рис. Д1-12. Алгоритм сортування №8 методу прямого **вибору**.

```
clock_t Exchange1(int *A, int N)
{
    int tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int R=N-1; R>0; R--){
        for(int i=0; i<R; i++)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
            }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-13. Алгоритм сортування №1 методу прямого **обміну** (без модифікацій).

```
clock_t Exchange2(int *A, int N)
{
    int R, flag, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    R=N-1; flag=1;
    while(flag == 1){
        flag=0;
        for(int i=0; i<R; i++)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
                flag=1;
            }
        R--;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-14. Алгоритм сортування №2 методу прямого **обміну** (з використанням прапорця).

```
clock_t Exchange3(int *A, int N)
{
    int R, k, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    R=N-1;
    while(R>0){
        k=0;
        for(int i=0; i<R; i++)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
                k=i;
            }
        R=k;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-15. Алгоритм сортування №3 методу прямого **обміну** (із запам'ятовуванням місця останньої перестановки).

```
clock_t Exchange4(int *A, int N)
{
    int L, R, k, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    L=0; R=N-1; k=0;
    while(L<R){
        for(int i=L; i<R; i++)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
                k=i;
            }
        R=k;
        for(int i=R-1; i>=L; i--)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
                k=i;
            }
        L=k+1;
     }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-16. Алгоритм сортування №4 методу прямого **обміну** (Шейкерне сортування).

```
clock_t InsertExchange(int *A, int N)
{
    int j, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=1; i<N; i++){
        j=i;
        while (j>0 && A[j]<A[j-1]) {
            tmp=A[j];
            A[j]=A[j-1];
            A[j-1]=tmp;
            j=j-1;
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-17. Гібридний алгоритм "вставка – обмін".

```
clock_t Select1Exchange(int *A, int N)
{
    int Min;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        Min=A[s];
        for(int i=s+1; i<N; i++)
            if (A[i]<Min){
                Min=A[i];
                A[i]=A[s];
                A[s]=Min;
            }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-18. Гібридний алгоритм "вибір№1 – обмін".

```
clock_t Select2Exchange(int *A, int N)
{
    int tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        for(int i=s+1; i<N; i++)
            if (A[i]<A[s]){
                tmp=A[i];
                A[i]=A[s];
                A[s]=tmp;
            }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-19. Гібридний алгоритм "вибір№2 – обмін".

```
clock_t Select5Exchange(int *A, int N)
{
    int Min, Max, tmp;
    int L, R;

    clock_t time_start, time_stop;

    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        if (A[L] > A[R]) {
            tmp=A[L];
            A[L]=A[R];
            A[R]=tmp;
        }
        Min=A[L]; Max=A[R];
        for(int i=L+1; i<R+1; i++){
            if (A[i] < Min){
                Min=A[i];
                A[i]=A[L];
                A[L]=Min;
            }
            else
                if (A[i] > Max){
                    Max=A[i];
                    A[i]=A[R];
                    A[R]=Max;
                }
        }
        L=L+1; R=R-1;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-20. Гібридний алгоритм "вибір№5 – обмін".

```
clock_t Select6Exchange(int *A, int N)
{
    int Min, Max;
    int L, R;

    clock_t time_start, time_stop;

    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        for(int i=L; i<R+1; i++){
            if (A[i] < A[L]){
                Min=A[i];
                A[i]=A[L];
                A[L]=Min;
            }
            else
                if (A[i] > A[R]){
                    Max=A[i];
                    A[i]=A[R];
                    A[R]=Max;
                }
        }
        L=L+1; R=R-1;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-21. Гібридний алгоритм "вибір№6 – обмін".

```
clock_t Shell_1(int *A, int N)
{
    int Elem, t, j, k;
    clock_t time_start, time_stop;

    time_start = clock();

    if (N<4) t=1;
    else t=(int)log2f((float)N)-1;

    Stages[t-1]=1;
    for (int i=t-2; i>=0; i--)
        Stages[i]=2*Stages[i+1]+1;

    for (int p=0; p<t; p++){
        k=Stages[p];
        for (int i=k; i<N; i++){
            Elem=A[i];
            j=i;
            while (j>=k && Elem<A[j-k]) {
                A[j]=A[j-k];
                j=j-k;
            }
            A[j]=Elem;
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-22. Алгоритм №1 методу сортування Шелла.

```
clock_t Shell_2(int *A, int N)
{
    int tmp, t, j, k;
    clock_t time_start, time_stop;

    time_start = clock();

    if (N<4) t=1;
    else t=(int)log2f((float)N)-1;

    Stages[t-1]=1;
    for (int i=t-2; i>=0; i--)
      Stages[i]=2*Stages[i+1]+1;

    for (int p=0; p<t; p++){
        k=Stages[p];
        for (int i=k; i<N; i++){
            j=i;
            while (j>=k && A[j]<A[j-k]) {
                tmp=A[j];
                A[j]=A[j-k];
                A[j-k]=tmp;
                j=j-k;
            }
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. Д1-23. Алгоритм №2 методу сортування Шелла.

```
void QuickSort(int L, int R)
{
    int B, tmp, i, j;

    B=Vector[(L+R)/2];
    i=L; j=R;
    while (i<=j) {
        while (Vector[i] < B) i=i+1;
        while (Vector[j] > B) j=j-1;
        if (i<=j) {
            tmp=Vector[i];
            Vector[i]=Vector[j];
            Vector[j]=tmp;
            i=i+1;
            j=j-1;
        }
    }
    if (L<j) QuickSort(L,j);
    if (i<R) QuickSort(i,R);
}

void QuickSortMeasurement()
{
    clock_t time_start, time_stop;

    for (int i=0; i < measurements_number; i++)
    {
        FillVector(Vector, VectorLength);
        time_start = clock();
        QuickSort(0, VectorLength-1);
        time_stop = clock();
        Res[i] = time_stop - time_start;
    }
}
```

Рис. Д1-24. Алгоритм методу «швидкого сортування» (сортування Хоара).

ДОДАТОК 2. ПЕРШИЙ ТИТУЛЬНИЙ ЛИСТ КУРСОВОЇ РОБОТИ.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРОГРАМНИХ СИСТЕМ
ТА ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування
і спеціалізованих комп'ютерних систем

КУРСОВА РОБОТА

з дисципліни "Структури даних та алгоритми"

Виконав: **Прізвище Ініціали**

Група: КВ-**??**

Номер студентського квитка: КВ-**????**

Допущений до захисту

_____

2 семестр 20__/20__ навч.року

ДОДАТОК 3. ДРУГИЙ ТИТУЛЬНИЙ ЛИСТ КУРСОВОЇ РОБОТИ.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРОГРАМНИХ СИСТЕМ
ТА ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування
і спеціалізованих комп'ютерних систем

Узгоджено	ЗАХИЩЕНА "__"_____20__р.
Керівник роботи	з оцінкою_____
_____/Марченко О.І./	_____ /Марченко О.І./

Дослідження ефективності методів сортування
(вказати назви конкретних методів сортування за варіантом)
на багатовимірних масивах

Виконавець роботи:_____
(підпис)
Прізвище Ім'я По батькові
_____20__ р.

ДОДАТОК 4. СТРУКТУРА ТЕХНІЧНОГО ЗАВДАННЯ КУРСОВОЇ РОБОТИ.

ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ

I. Описати теоретичні положення, від яких відштовхується дослідження, тобто принцип та схему роботи кожного із досліджуваних алгоритмів сортування для одновимірного масиву, навести загальновідомі властивості цих алгоритмів та оцінки кількості операцій порівняння та присвоєння для них.

II. Скласти алгоритми рішення задачі сортування в багатовимірному масиві заданими за варіантом методами та написати на мові програмування за цими алгоритмами програму, яка відповідає вимогам розділу «Вимоги до програми курсової роботи».

III. Виконати налагодження та тестування коректності роботи написаної програми.

IV. Провести практичні дослідження швидкодії складених алгоритмів, тобто виміри часу роботи цих алгоритмів для різних випадків та геометричних розмірів багатовимірних масивів.

V. За результатами досліджень скласти порівняльні таблиці за різними ознаками.

- Одна таблиця результатів (вимірів часу сортування впорядкованого, випадкового і обернено-впорядкованого масиву) для масиву з заданими геометричними розмірами повинна бути такою:

Таблиця № для масиву A[P,M,N], де P= ; M= ; N= ;

	Відсортований	Випадковий	Обернено відсортований
Назва алгоритму 1			
Назва алгоритму 2			
Назва алгоритму 3			

УВАГА! Колонки таблиць виміру часу в програмі та у звіті курсової роботи місцями НЕ переставляти! За невиконання цієї вимоги нараховується 3 (три) штрафних бали.

- Для варіантів курсової роботи, де крім алгоритмів порівнюються також способи обходу, в назвах рядків таблиць потрібно вказати як назви алгоритмів, так і номери способів обходу.
- Для виконання ґрунтовного аналізу алгоритмів потрібно зробити виміри часу та побудувати таблиці для декількох масивів з різними геометричними розмірами.
- Зробити виміри часу для стандартного випадку одновимірного масиву, довжина якого вибирається такою, щоб можна було виконати коректний порівняльний аналіз з рішенням цієї ж задачі для багатовимірного масиву.
- Кількість необхідних таблиць для масивів з різними геометричними розмірами залежить від задачі конкретного варіанту курсової роботи і вибираються так, щоб виконати всебічний та ґрунтовний порівняльний аналіз заданих алгоритмів.

- Рекомендації випадків дослідження з різними геометричними розмірами масивів наведені у розділі «Випадки дослідження».

VI. Для наочності подання інформації за отриманими результатами рекомендується також будувати стовпчикові діаграми та графіки.

VII. Виконати порівняльний аналіз поведінки заданих алгоритмів за отриманими результатами (вимірами часу):

- для одновимірного масиву відносно загальновідомої теорії;
- для багатовимірних масивів відносно результатів для одновимірного масиву;
- для заданих алгоритмів на багатовимірних масивах між собою;
- дослідити вплив різних геометричних розмірів багатовимірних масивів на поведінку алгоритмів та їх взаємовідношення між собою;
- **для всіх вищезазначених пунктів порівняльного аналізу пояснити, ЧОМУ алгоритми в розглянутих ситуаціях поводять себе саме так, а не інакше**.

VIII. Зробити висновки за зробленим порівняльним аналізом.

IX. Програму курсової роботи під час її захисту ОБОВ'ЯЗКОВО мати при собі на електронному носії інформації.

Варіант № ___

Задача

Умова задачі за варіантом.

Досліджувані методи та алгоритми

Перелік методів та алгоритмів за варіантом.

Способи обходу

Перелік способів обходу за варіантом.

Випадки дослідження

Перелік випадків дослідження, які потрібно розглянути для заданої за варіантом задачі згідно розділу «Випадки дослідження».

ДОДАТОК 5. ФОРМАТУВАННЯ КОДУ.

Загальні вимоги до форматування коду

1. Не рекомендується писати рядки коду довші, ніж 80 символів.

2. Для відступів використовуйте завжди пробіли і не використовуйте символи табуляції.

В різних текстових редакторах символу табуляції може відповідати різна кількість позицій. Тому при зміні редактора вид (форматування) вашого коду може суттєво змінитися. Особливо негарною ця зміна буде у випадку, якщо в коді для відступів використовувалася суміш пробілів та символів табуляції.

3. В рамках коду однієї програми використовуйте завжди однакову кількість пробілів для відступів при написанні вкладених операторів.

Компанії, що розроблюють програмне забезпечення, як правило, встановлюють певні правила кодування та форматування коду, які є обов'язковими для дотримання всіма програмістами компанії.

Два найбільш використаних варіанти кількості пробілів для відступів – це 4 пробіли та 2 пробіли.

4. Є два загальноприйнятих варіанти запису фігурних дужок для об'єднання блоків операторів:

1) і відкриваюча, і закриваюча дужки ставляться під першим символом відповідного оператора

```
while (умова)
{
    Оператори  // відступ 4 пробіли
}
```

2) відкриваюча дужка ставиться в кінці рядочка оператора, а закриваюча дужка під першим символом цього оператора

```
while (умова) {
    Оператори  // відступ 4 пробіли
}
```

Форматування функцій

1. Якщо параметрів у функції небагато і заголовок функції не дуже довгий, то він записується в одну лінію

```
res_type func(type1 par1, type2 par2)
{
    Оператори  // відступ 4 пробіли
}
```

2. Якщо функція має багато параметрів і імена функції та параметрів довгі, то параметри записуються в стовпчик

```
result_type very_long_func_name(type1 parameter1,
                                type2 parameter2)
                                type3 parameter3)
{
  Оператори  // відступ 4 пробіли
}
```

Форматування складних булевих виразів

1. Запис довгих булевих виразів розбивається на декілька рядків, щоб логіка виразу читалася легше і була більш зрозумілою. Перенос виразу на інший рядок робиться, як правило, на бінарних булевих операціях && та !!.

```
if (operand1 > operand2 &&
    operand3 == operand4 &&
    operand5)
{
  ...
}
```

2. Також гарною ідеєю є використання додаткових дужок для підвищення читабельності складних виразів, особливо, якщо в них є операції різного пріоритету.

```
if ((operand1 > operand2) && (operand3 == operand4)!!
    (operand5 <= operand6) && (operand7 != operand8))
{
  ...
}
```

Форматування умовних операторів

1. Якщо вкладені оператори ***if*** **є винятково тільки в гілках *else***, то можна використовувати так-звану форму ***else if***, коли гілки пишуться без відступів.

```
if (condition1)
{
    Оператори умови 1  // відступ 4 пробіли
}
else if (condition2)
{
    Оператори умови 2  // відступ 4 пробіли
}
else if (condition3)
{
    Оператори умови 3  // відступ 4 пробіли
}
else {
    Оператори else  // відступ 4 пробіли
}
```

2. Якщо ж вкладені оператори ***if*** **<u>є в обох гілках зовнішнього оператора</u>** ***if***, то всі оператори пишуться за загальною вимогою з відступом у 4 чи 2 пробіли.

```
if (умова1)
{
    if (умова1_1)           // відступ 4 пробіли
    {
        Оператори умови 1_1 // відступ 8 пробілів
    }
    else
    {
        Оператори else умови 1_1// відступ 8 пробілів
    }
}
else
{
    if (умова2_1)           // відступ 4 пробіли
    {
        Оператори умови 2_1 // відступ 8 пробілів
    }
    else
    {
        Оператори else умови 2_1// відступ 8 пробілів
    }
}
```

3. Фігурні дужки рекомендується ставити завжди, навіть, якщо у відповідній гілці буде тільки один простий оператор.

```
if (умова)
{
    Тільки один простий оператор // відступ 4 пробіли
}
else
{
    Тільки один простий оператор // відступ 4 пробіли
}
```

Це дозволяє уникнути помилок запису, якщо пізніше знадобиться в певній гілці замість одного оператора записати декілька.

Виключенням є тільки дуже прості випадки умовних операторів, гілки яких навряд чи будуть розширюватися в майбутньому, наприклад

```
if (умова) a = 0;
else       a = N;
```

Форматування операторів вибору (перемикання)

```
switch (var) {
  case 0:  // відступ 2 пробіли для case
  {
    Оператори // відступ 4 пробіли для операторів
    break;
  }
  case 1:
  {
    Оператори // відступ 4 пробіли для операторів
    break;
  }
  default:
  {
    Оператори // відступ 4 пробіли для операторів
  }
```

```
}
```

Форматування вкладених циклів

Загальні правила форматування вкладених циклів є такими самими, як і вкладених умов. Ті ж правила застосовуються й до комбінацій циклів та умов.

Приклад 1.

```
while (умова1)
{
    for (int i=0; i<n; ++i) // відступ 4 пробіли
    {
        Оператори for       // відступ 8 пробілів
    }

    do
    {
        Оператори do while  // відступ 8 пробілів
    } while (умова2);
}
```

Приклад 2.

```
while (умова1)
{
    for (int i=0; i<n; ++i)    // відступ 4 пробіли
    {
        Оператори for          // відступ 8 пробілів
    }

    if (умова2)                // відступ 4 пробіли
    {
        Оператори умови2       // відступ 8 пробілів
    }
    else
    {
        Оператори else умови2 // відступ 8 пробілів
    }
}
```

Приклад 3. Виключеннями, коли фігурні дужки можуть не ставитися, є цикли з дуже простим тілом циклу, яке навряд чи ускладниться в майбутньому (наприклад, цикл лінійного пошуку).

```
i = 0;
while (i<n && A[i]!=X) i++;
```