

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ**

**РОБОЧА ПРОГРАМА,  
методичні вказівки та індивідуальні завдання  
до вивчення дисципліни  
«Економічна кібернетика (поглиблений курс)»  
для студентів спеціальності 051 – економіка**

**Дніпро НМетАУ 2019**

УДК 330.46(07)

Робоча програма, методичні вказівки та індивідуальні завдання до вивчення дисципліни «Економічна кібернетика (поглиблений курс)» для студентів спеціальності 051 – економіка / Укл.: Л.І. Лозовська. – Дніпро: НМетАУ, 2019. – 20 с.

Викладені робоча програма, методичні вказівки до виконання контрольної та курсової роботи з дисципліни «Економічна кібернетика (поглиблений курс)», література, що рекомендується для вивчення дисципліни.

Призначена для студентів напрямку 051 – економіка заочної форми навчання.

Укладач: Л.І. Лозовська, канд. ф.-м. наук, доц.

Відповідальна за випуск Л.М. Бандоріна, канд. екон. наук, доц.

Рецензент К.Ф. Ковальчук, д-р екон. наук, проф. (НМетАУ)

## ВСТУП

Проблема складності є головною проблемою, яку доводиться вирішувати при створенні великих і складних систем будь-якої природи, в тому числі програмних систем. Жоден розробник не в змозі вийти за межі людських можливостей і зрозуміти всю систему в цілому. Єдиний ефективний підхід до вирішення цієї проблеми полягає в побудові складної системи з невеликого числа великих частин, кожна з яких, в свою чергу, будується з частин меншого розміру, і т.д., до тих пір, поки самі невеликі частини можна буде будувати з наявного матеріалу. Цей підхід відомий під різними назвами, серед них такі як «розділяй і володарюй», ієрархічна декомпозиція. По відношенню до проектування складної програмної системи це означає, що її необхідно розділяти (декомпозиція) на невеликі підсистеми, кожен з яких можна розробляти незалежно від інших.

Значна частина теоретичного матеріалу присвячена класичним аспектам аналізу часової і просторової ефективності алгоритмів. Тут викладені основні етапи асимптотичного аналізу обчислювальної складності алгоритмів: від підрахунку кількості операцій до застосування асимптотичних позначень. Розглянуті методи оцінки ефективності рекурсивних алгоритмів: аналіз дерева рекурсивних викликів і застосування основної теореми.

Також розглядаються задачі сортування і пошуку. Наведені тривіальні алгоритми сортування з квадратичною складністю і основні асимптотично оптимальні алгоритми: сортування злиттям і пірамідальне сортування. Проаналізовані лінійний і бінарний алгоритми розв'язання задачі пошуку.

В останньому модулі дисципліни розглянуті питання реалізації найважливіших структур даних. Викладення матеріалу починається зі введення поняття абстрактних типів даних та їх класифікації. На прикладі реалізації лінійних типів даних розглянуті основні операції над зв'язними списками, статичними масивами і кільцевими буферами. Далі наведено опис бінарних дерев пошуку і хеш-таблиць, як основних засобів реалізації абстрактних типів даних множина і асоціативний масив (словник).

Головна мета дисципліни полягає в формуванні у студентів фундаментальних знань класичних алгоритмів, які використовуються при програмуванні, навчити, як техніці програмування, так і мистецтву програмування з використанням даних різних типів, різних структур даних.

**1.1 Мета вивчення дисципліни**

Навчальна дисципліна «Економічна кібернетика (поглиблений курс)» входить до циклу дисциплін фахової підготовки.

**Мета вивчення дисципліни** – сформувати у студентів фундаментальні знання класичних алгоритмів, які використовуються при програмуванні, навчити, як техніці програмування, так і мистецтву програмування з використанням даних різних типів, різних структур даних. Вивчити основні поняття, які визначають суть алгоритмізації процесів обробки інформації, засоби представлення алгоритмів. В дисципліні також розглядається поняття ефективності алгоритмів та методологія оцінки часу їх роботи.

В результаті вивчення дисципліни студент повинен:

**знати:**

- сучасний інструментарій ПК;
- типові алгоритмічні конструкції, способи розробки алгоритмів;
- етапи обробки програм на комп'ютері;
- скалярні та структурні типи даних;

**вміти:**

- розв'язувати спектр проблем при рішенні практичних задач від проблеми формалізації задачі до проблем, які постають у час виконання закінченої програми;
- застосовувати вибраний або розроблений алгоритм до конкретних вихідних даних задачі, яка вирішується.

**Зв'язок з іншими дисциплінами** – дисципліна оснований на вивченні дисциплін «Вища та прикладна математика», «Економічна кібернетика», «Основи алгоритмізації та програмування», «Технологія проектування програмних систем», «Об'єктно-орієнтоване програмування».

**Критерії успішності** – отримання позитивних оцінок при виконанні контрольних робіт, при захисті лабораторних робіт, виконанні та захисті курсової роботи.

Набуті знання і вміння використовуються при вивченні комплексу фахових дисциплін, підготовки випускної роботи.

## 1.2 Розподіл годин за навчальним планом

Дисципліна вивчається у 9 семестрі. Відповідно до навчального плану на вивчення дисципліни «Економічна кібернетика (поглиблений курс)» для всіх форм навчання заплановано 120 годин, які для студентів заочної форми навчання розподілені по видах занять у такий спосіб:

аудиторні заняття – 20 годин;

з них:

лекції – 8 годин;

лабораторні заняття – 12 годин;

самостійна робота – 100 годин.

## 1.3 Зміст дисципліни

*ТЕМА 1.* Алгоритми і їх ефективність.

Задача, алгоритм, програма. Показники ефективності алгоритму. Підрахунок кількості операцій алгоритму. Швидкість росту функцій.

*ТЕМА 2.* Аналіз рекурсивних алгоритмів

Рекурсивні алгоритми. Сортування злиттям. Розв'язання рекурентних рівнянь.

*ТЕМА 3.* Сортування. Задача сортування

Властивості і види алгоритмів сортування. Сортування вставкою. Сортування вибором. Нижня границя часу сортування порівнянням. Швидке сортування. Сортування злиттям. Пірамідальне сортування. Сортування підрахунком. Вибір алгоритму сортування.

*ТЕМА 4.* Пошук. Задача пошуку

Лінійний пошук. Бінарний пошук.

*ТЕМА 5.* Абстрактні типи даних

Базові типи даних. Структури даних. Абстрактні типи даних. Вибір абстрактного типу даних.

*ТЕМА 6.* Списки. АТД список

Реалізація списків на базі масивів. Зв'язні списки. Однозв'язні списки. Двозв'язні списки. Реалізація АТД список.

*ТЕМА 7.* Стеки. АТД стек

Реалізація стеку на базі масиву. Реалізація стеку на базі зв'язного списку. Порівняння реалізацій.

*ТЕМА 8.* Черга. АТД черга

Реалізація черги на базі зв'язного списку. Реалізація черги на базі кільцевого буфера. Порівняння реалізацій.

#### *ТЕМА 9. Бінарні дерева*

Кореневі дерева. Представлення дерева в пам'яті. Обходи бінарних дерев.

Бінарні дерева пошуку. Структура бінарного дерева пошуку. Створення вузла. Додавання вузла. Пошук вузла по ключу. Пошук мінімального і максимального вузлів. Пошук наступного і попереднього вузлів. Обхід дерева в упорядкованій послідовності. Видалення вузла. Видалення дерева. Висота бінарного дерева пошуку.

### **1.4 Рекомендована література**

1. [CLRS] Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ. – 3-е изд. – М.: Вильямс, 2013. – 1328 с.
2. [Levitin] Левитин А.В. Алгоритмы: введение в разработку и анализ. – М.: Вильямс, 2006. – 576 с.
3. [Aho] Ахо А.В., Хопкрофт Д., Ульман Д.Д. Структуры данных и алгоритмы. – М.: Вильямс, 2001. – 384 с.
4. Кормен Т.Х. Алгоритмы: Вводный курс. - М.: Вильямс, 2014. - 208 с.
5. Дасгупта С., Пападимитриу Х., Вазирани У. Алгоритмы. - М.: МЦНМО, 2014. - 320с.
6. Кнут Д. Искусство программирования. Том {1, 3}, 3-е изд. - М.: Вильямс, 2010.
7. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: ДияСофт, 2001. – 688 с.
8. Скиена С.С. Алгоритмы. Руководство по разработке. – 2-е изд. – СПб: БХВ, 2011 – 720 с.
9. Макконнелл Дж. Основы современных алгоритмов. – 2е изд. – М.: Техносфера, 2004. – 368 с.
10. Миллер Р. Последовательные и параллельные алгоритмы: общий подход. – М.: БИНОМ, 2006. – 406 с.
11. Керниган Б., Пайк Р. Практика программирования. - М.: Издательский дом “Вильямс”, 2004. - 288 с.

- 12.Бентли Дж. Жемчужины программирования. - СПб.: Питер, 2002. - 272 с.
- 13.Уоррен Г.С. Алгоритмические трюки для программистов. - М.: Вильямс, 2014. - 512 с.
- 14.Миллер Р. Последовательные и параллельные алгоритмы: общий подход. – М.: БИНОМ, 2006. – 406 с.
- 15.Бакнелл Д.М. Фундаментальные алгоритмы и структуры данных в Delphi. – СПб.: ДияСофт, 2003. – 560 с.

## **2 МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ КОНТРОЛЬНОЇ ТА КУРСОВОЇ РОБОТИ**

### **2.1 Загальні положення**

Протягом семестру після установочних занять студент виконує контрольну роботу. Контрольна робота, що виконана в повному обсязі, здається в деканат заочного факультету для реєстрації. Після реєстрації в деканаті контрольна робота передається на кафедру економічної інформатики для перевірки. Контрольна робота, що виконана з помилками й відхиленнями від вимог методичних вказівок, повертається студентові для доробки. Захист контрольної роботи здійснюється під час екзаменаційної сесії. Контрольна та курсова робота виконуються на аркушах формату А4 з використанням комп'ютера. До контрольної та курсової роботи додається CD-диск з текстами програм.

### **2.2 Завдання і методичні вказівки до виконання контрольної роботи**

Контрольна робота складається з трьох тематичних частин, кожна з яких містить кілька завдань. Перші дві частини мають практичну направленість, третя містить теоретичні питання, на які студент має дати відповіді.

**Частина 1.** Дослідження часової складності алгоритмів.

#### **Завдання**

1. За допомогою пакету Microsoft Excel побудувати графіки залежності функцій  $n$ ,  $n^2$ ,  $\sqrt{n}$ ,  $\log_2 n$ ,  $\log_{10} n$ ,  $\sqrt[3]{n}$ ,  $\log_2(\log_2 n)$ ,  $(\log_2 n)^3$ ,  $\frac{3}{2^n}$ ,  $2^n$ .

Проаналізувати швидкість зростання цих функцій. Упорядкуйте функції за неспаданням швидкості їх зростання

2. Є обчислювач (процесор), який виконує  $5 \cdot 10^{12}$  операцій в секунду, і три алгоритми з кількістю операцій

$$T_1(n) = 500n \cdot \log_2 n + 2048, T_2(n) = 4n^2 + 10n + 48, T_3(n) = n! \log_2 n + 2,$$

Визначте скільки часу (днів, годин, хвилин, секунд) буде виконуватися кожен алгоритм при  $n = 10$ ,  $n = 100$ ,  $n = 500$ .

3. Використовуючи визначення асимптотичних позначень  $O$  і  $\Theta$ , доведіть справедливість наступних відношень:

$$n^4 + 3n^3 - 10n^2 - 50n + 3 = O(n^4);$$

$$n^2 = O(2^n);$$

$$2^{n-1} = \Theta(2^n).$$

4. Порівняйте порядки зростання функцій через межу

$$f(n) = 2^{3n}, g(n) = 2^n;$$

$$f(n) = \ln(n), g(n) = 3n;$$

$$f(n) = 2^{5n}, g(n) = n^{20} \text{ (використайте правило Лопітала 20 разів);}$$

$$f(n) = \log_a(n), g(n) = \sqrt[n]{n}, a > 1.$$

**Частина 2.** Розв'язання рекурентних рівнянь.

### Теоретичні відомості.

#### Порівняння порядку зростання функцій через межу.

Нехай, як і раніше, ми маємо дві функції  $f(n)$  і  $g(n)$ . Для відповіді на питання, яка з функцій має вищий порядок зростання, досить обчислити межу їх відношення при  $n \rightarrow \infty$ . [4, 13]

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0, & \text{якщо } f(n) = o(g(n)), \\ c, & \text{якщо } f(n) = \Theta(g(n)), \\ \infty, & \text{якщо } f(n) = \omega(g(n)). \end{cases} \quad (1)$$



Якщо межа дорівнює 0, то  $f(n) = o(g(n))$  і  $f(n) = O(g(n))$ . Другий випадок:  $f(n) = \Theta(g(n))$ , отже,  $f(n) = O(g(n))$  і  $f(n) = \Omega(g(n))$ . З третього випадку слідує  $f(n) = \omega(g(n))$  і  $f(n) = \Omega(g(n))$ .

**Приклад.** Порівняємо порядки (швидкості) зростання функцій  $f(n) = n^3(n-2)/3$  і  $g(n) = n^4$ .

$$\lim_{n \rightarrow \infty} \frac{n^3(n-2)/3}{n^4} = \frac{1}{3} \lim_{n \rightarrow \infty} \left(1 - \frac{2}{n}\right) = \frac{1}{3}.$$

Межа дорівнює позитивній константі, тоді за (1.19) обидві функції мають однаковий порядок зростання  $n^3(n-2)/3 = \Theta(n^4)$ .

**Приклад.** Порівняємо порядки зростання функції  $f(n) = n-2$  і функції  $g(n) = \log_a n$  при  $a > 1$ .

$$\lim_{n \rightarrow \infty} \frac{n-2}{\log_a n} = \lim_{n \rightarrow \infty} \left( \frac{1}{\frac{1}{n \ln_n a}} \right) = \lim_{n \rightarrow \infty} (n \ln_n a) = \infty.$$

Для обчислення останньої межі використано правило Лопітала – перейшли до розгляду межі відношення похідних функцій  $f(n)$  і  $g(n)$ . Межа дорівнює нескінченності. Це означає, що функція  $f(n)$  має більший порядок зростання, ніж  $g(n)$ . Іншими словами,  $n-2 = \omega(\log n)$ . Зверніть увагу, що тут навмисно опущено основу логарифма, так як при використанні спільно з асимптотичними позначеннями вона не має сенсу. За властивостями логарифмів зміна основи приводить до множення логарифма на константу, а за властивостями асимптотичних позначень константи можуть бути відкинуті:

$$\log_a n = \frac{1}{\log_c a} \log_c n. \quad (2)$$

### Властивості асимптотичних відношень.

Факт належності функції  $f(n)$  деякій множині ми записували через знак рівності, наприклад,  $f(n) = O(g(n))$ . Це позначення є зручним на практиці, але потрібно пам'ятати, що сенс його полягає в тому, що функція, що стоїть зліва від знаку рівності, належить множині, що вказана справа:

$$f(n) \in O(g(n)) \quad (3)$$

Тобто це не є рівність в звичайному розумінні, а несиметричне бінарне відношення між функціями  $f(n)$  і  $g(n)$ . Наприклад, запис  $f(n) = O(g(n))$  є коректним, в той же час вираз  $O(g(n)) = f(n)$  позбавлений сенсу.

Якщо асимптотичне позначення  $O$ ,  $\Omega$  або  $\Theta$  присутнє у формулі, то замість нього можна подумки підставити будь-яку функцію з цієї множини. Наприклад, у виразі  $5n^3 + 3n^2 + 55n + 5 = 5n^3 + \Theta(n^2)$ , замість  $\Theta(n^2)$  можна підставити будь-яку функцію з квадратичним порядком зростання. Асимптотичні бінарні відношення  $O$ ,  $\Omega$  та  $\Theta$  мають багато властивостей бінарних відношень між дійсними числами (для визначеності позначимо числа через  $f$  і  $g$ ):

$$f(n) = O(g(n)) \text{ відповідає } f \leq g ; \quad (4)$$

$$f(n) = \Omega(g(n)) \text{ відповідає } f \geq g ; \quad (5)$$

$$f(n) = \Theta(g(n)) \text{ відповідає } f = g ; \quad (6)$$

$$f(n) = o(g(n)) \text{ відповідає } f < g ; \quad (7)$$

$$f(n) = \omega(g(n)) \text{ відповідає } f > g . \quad (8)$$

### **Транзитивність**

$$f(n) = O(g(n)) \text{ і } g(n) = O(h(n)), \text{ то } f(n) = O(h(n)); \quad (9)$$

$$f(n) = \Omega(g(n)) \text{ і } g(n) = \Omega(h(n)), \text{ то } f(n) = \Omega(h(n)); \quad (10)$$

$$f(n) = \Theta(g(n)) \text{ і } g(n) = \Theta(h(n)), \text{ то } f(n) = \Theta(h(n)); \quad (11)$$

$$f(n) = o(g(n)) \text{ і } g(n) = o(h(n)), \text{ то } f(n) = o(h(n)); \quad (12)$$

$$f(n) = \omega(g(n)) \text{ і } g(n) = \omega(h(n)), \text{ то } f(n) = \omega(h(n)). \quad (13)$$

### **Рефлексивність**

$$f(n) = O(f(n)); \quad (14)$$

$$f(n) = \Omega(f(n)); \quad (15)$$

$$f(n) = \Theta(f(n)). \quad (16)$$

### **Симетричність**

$$f(n) = \Theta(g(n)) \text{ тоді і лише тоді, коли } g(n) = \Theta(f(n)). \quad (17)$$

### **Перестановочна симетрія**

$$f(n) = O(g(n)) \text{ тоді і лише тоді, коли } g(n) = \Omega(f(n)); \quad (18)$$

$$f(n) = o(g(n)) \text{ тоді і лише тоді, коли } g(n) = \omega(f(n)). \quad (19)$$

Розглянемо властивості  $O$ -позначення, які зручно використовувати на практиці:

– добуток – якщо  $f_1(n) = O(g_1(n))$  і  $f_2(n) = O(g_2(n))$ , то

$$f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n)); \quad (20)$$

– сума – якщо  $f_1(n) = O(g_1(n))$  і  $f_2(n) = O(g_2(n))$ , то

$$\begin{aligned} f_1(n) + f_2(n) &= O(|g_1(n)| + |g_2(n)|) \text{ і} \\ f_1(n) + f_2(n) &= O(\max\{g_1(n); g_2(n)\}); \end{aligned} \quad (21)$$

– множення на константу – якщо  $f(n) = O(g(n))$  і  $k > 0$ , то

$$k \cdot f(n) = O(g(n)). \quad (22)$$

При використанні асимптотичних позначень константи в функції  $T(n)$  ігноруються.

З властивостей слідує: для будь-якого полінома  $p(n)$  ступеня  $k \in \{0, 1, 2, \dots\}$  при  $a_k > 0$  справедливо

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n^1 + a_0 = \Theta(n^k). \quad (23)$$

## Етапи асимптотичного аналізу

Асимптотичний аналіз складності алгоритму полягає в побудові *асимптотичних оцінок його обчислювальної складності* (computational complexity) або *обсягу необхідної йому пам'яті* (space complexity) в гіршому, середньому або кращому випадку. Розглянемо основні кроки асимптотичного аналізу.

1. Визначаються параметри, від яких залежить час виконання алгоритму або обсяг необхідної йому пам'яті: встановлюється, скільки аргументів буде у функції  $T(n)$ . Наприклад,  $T(n)$ ,  $T(n, m)$ ,  $T(n, m, k)$ .

2. Для гіршого, середнього або кращого випадку обчислюється кількість операцій алгоритму або обсяг необхідної йому пам'яті, що записується як функція від параметрів, встановлених на попередньому кроці. Наприклад,  $T(n) = 4n^5 + 3n^2 + 10n + 5$ . У першу чергу обчислюють кількість операцій алгоритму для гіршого випадку. Якщо на практиці виникнення цього випадку

малоймовірно, то підраховується кількість операцій алгоритму для середнього випадку. На даному етапі можна обмежитися підрахунком кількості тільки **базових операцій** (*basic operations*) – найбільш важливих операцій, від яких залежить час виконання алгоритму. Наприклад, можна не враховувати константну кількість операцій читання значень з пам'яті і операцій привласнення.

3. До побудованої функції  $T(n)$  застосовуються асимптотичні позначення для класифікації порядку її зростання. За допомогою  $\Theta$ -позначення будується асимптотично точна оцінка для функції  $T(n)$ . У випадку виникнення труднощів побудови такої оцінки, обмежуються визначенням хоча б  $O$ -позначення – асимптотичної верхньої межі функції  $T(n)$ .

На практиці в процесі асимптотичного аналізу алгоритмів зустрічається невеликий набір класів складності, які наведені в табл. 1.3 (в порядку зростання швидкості росту).

**Приклад. Факторіал числа.** Проведемо асимптотичний аналіз обчислювальної складності алгоритму обчислення факторіала цілого числа. За визначенням факторіал числа є

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n. \quad (24)$$

Алгоритм 1 – Обчислення факторіала кількості

```
1.int factorial(int n)
2.{
3.    int fact = 1;
4.    for (int i = 1; i <= n; i++)
5.    {
6.        fact *= i;
7.    }
8.    return fact;
9.}
```

Крок 1. Час виконання алгоритму `factorial` залежить тільки від значення  $n$ , отже, функція  $T(n)$  буде мати один аргумент.

Крок 2. Обчислимо кількість  $T(n)$  операцій алгоритму. Найгірший, середній і кращий випадки для цього алгоритму збігаються. Одна операція йде

на ініціалізацію змінної *fakt*, далі на кожній з  $n - 1$  ітерації циклу виконується дві операції (множення і привласнення), після чого відбувається повернення значення з функції – ще одна операція. отже,

$$T(n) = 1 + 2(n - 1) + 1 = 2n. \quad (25)$$

Крок 3. Отримаємо асимптотично точну оцінку обчислювальної складності алгоритму. Доведемо, що  $2n = \Theta(n)$ . Візьмемо  $c_1 = 1$ ,  $c_2 = 2$  і  $n_0 = 1$ , ці константи забезпечує виконання нерівності

$$0 \leq n \leq 2n \leq 2n, \quad \text{при } n \geq 1. \quad (26)$$

Результатом асимптотичного аналізу обчислювальної складності алгоритму `factorial` є асимптотично точна оцінка  $\Theta(n)$ , яка говорить про те, що алгоритм має лінійну обчислювальну складність. Тепер виконаємо асимптотичний аналіз складності за пам'яттю алгоритму. Обчислимо кількість  $M(n)$  осередків пам'яті **RAM**-машини, яка потрібна алгоритму. У рядку 3 ініціалізується змінна *fakt* – це вимагає одного осередку пам'яті, для змінної *i* також потрібен один осередок пам'яті. Остаточоно отримуємо

$$M(n) = 2 = \Theta(1). \quad (27)$$

Алгоритм `factorial` має константну складність за пам'яттю – він не створює в пам'яті структур даних, розмір яких залежить від  $n$ .

**Приклад. Сорткування вибором.** Проведемо асимптотичний аналіз обчислювальної складності **алгоритму сортування вибором** (*selection sort*).

Алгоритм 2 – Сорткування вибором

```

1.int selectionSort(double *mas, int n)
2.{
3.    double tmp = 0;
4.    for (int i = 0; i < n - 1; i++)
5.    {
6.        int min = i;
7.        for (int j = i + 1; j < n; j++)
8.        {
9.            if (mas[j] < mas[min])
10.            {
11.                min = j;
12.            }
13.        }
14.        if (min != i)
```

```

15.      {
16.          tmp = mas[i];
17.          mas[i] = mas[min];
18.          mas[min] = tmp;
19.      }
20.  }
21.  return 0;
22.}

```

Крок 1. Час виконання алгоритму `selectionSort` залежить від розміру масиву і значень елементів в ньому.

Крок 2. Обчислимо кількість  $T(n)$  операцій алгоритму, що виконуються ним в гіршому випадку. Тобто, у випадку коли необхідно виконати максимальну кількість операцій при виконанні на всіх ітераціях циклів умов в рядках 9 і 14. Найгірший випадок для алгоритму `selectionSort` – це масив, елементи якого впорядковані за спаданням – в порядку, зворотному напрямку сортування. В такому випадку зовнішній цикл `for` виконується  $n-1$  раз. На кожній ітерації цього циклу одна операція доводиться на запис значення в змінну (рядок 6), що разом дає  $n-1$  операцію. На рядки 14–19 доводиться чотири операції (перевірка умови і три привласнення), що дає ще  $4(n-1)$  операцій. тоді

$$T(n) = (n-1) + 4(n-1) + T_{inner\_loop}(n). \quad (28)$$

де функція  $T_{inner\_loop}(n)$  визначає кількість операцій, що припадають на внутрішній цикл, за весь час виконання алгоритму.

Обчислимо, скільки операцій припадає на внутрішній цикл в рядках 7-13. Розглянемо процес виконання зовнішнього циклу:

- при  $i=1$  внутрішній цикл виконується  $n-1$  раз (змінна  $j$  приймає значення від 2 до  $n$ );
- при  $i=2$  внутрішній цикл виконується  $n-2$  рази (змінна  $j$  приймає значення від 3 до  $n$ );
- ...
- при  $i=n-2$  внутрішній цикл виконується 2 рази (змінна  $j$  приймає значення від  $n-1$  до  $n$ );
- при  $i=n-1$  внутрішній цикл виконується 1 раз (змінна  $j$  приймає значення від  $n$  до  $n$ ).

З огляду на, що на кожній ітерації внутрішнього циклу виконується дві операції (перевірка умови і привласнення(рядки 7–9)), отримуємо

$$T_{inner\_loop}(n) = 2(n - 1 + n - 2 + \dots + 1). \quad (29)$$

$$T(n) = (n - 1) + 4(n - 1) + 2[n - 1 + n - 2 + \dots + 1]. \quad (30)$$

У квадратних дужках записана сума членів арифметичної прогресії з різницею 1. Обчислимо її

$$T(n) = (n - 1) + 4(n - 1) + 2\left[\frac{n^2 - n}{2}\right] = n^2 + 4n - 5. \quad (31)$$

Крок 3. Отримаємо асимптотично точну оцінку обчислювальної складності алгоритму `selectionSort`. Доведемо справедливність

$$T(n) = n^2 + 4n - 5 = \Theta(n^2). \quad (32)$$

Дотримуючись визначення асимптотичних позначень, візьмемо константи  $c_1 = 1$ ,  $c_2 = 2$  і  $n_0 = 2$ , вони забезпечують виконання нерівності

$$0 \leq n^2 \leq n^2 + 4n - 5 \leq 2n^2, \quad \text{при } n \geq 2. \quad (33)$$

Таким чином, ми показали, що алгоритм сортування вибором в гіршому випадку має квадратичну обчислювальну складність, і ця оцінка є асимптотично точною.

Діючи аналогічно, можна оцінити обчислювальну складність алгоритму сортування вибором для кращого випадку, коли вхідний масив вже упорядкований в напрямку сортування. Умови всередині циклів виконуватися не будуть, і кількість операцій алгоритму дорівнюватиме

$$T(n) = (n - 1) + (n - 1) + [n - 1 + n - 2 + \dots + 1] = \Theta(n). \quad (34)$$

Складність за пам'яттю алгоритму сортування вибором є константною, так як в процесі своєї роботи алгоритму потрібно лише чотири додаткові осередки під змінні  $j$ ,  $min$  і  $tmp$  (вхідний масив  $mas$  не враховується).

## Розв'язання рекурентних рівнянь

Для розв'язання рекурентних рівнянь розроблені різні методи. При аналізі обчислювальної складності сортування злиттям ми використовували метод дерева рекурсії. Тепер ми розглянемо один із загальних підходів до розв'язання рекурентних рівнянь – **основний метод** (master method).

Основний метод широко застосовується для аналізу алгоритмів, заснованих на методі декомпозиції. Розглянемо розв'язання рекурентних

рівнянь, коли вихідну задачу розміру можна розділити на  $a \geq 1$  підзадач розміру  $n/b$ . Будемо вважати, що для розв'язання задачі розміром 1 потрібен час  $O(1)$ , а для декомпозиції задачі розміру  $n$  і комбінування (злиття) розв'язків підзадач потрібно  $f(n)$  одиниць часу. Тоді час  $T(n)$  розв'язання задачі розміром  $n$  можна записати як

$$T(n) = aT(n/b) + f(n), \quad (35)$$

де  $a \geq 1$ ,  $b > 1$ .

Записане рівняння називається **узагальненим рекурентним рівнянням декомпозиції** (general divide-and-conquer recurrence). Розв'язанням цього рівняння є порядок зростання функції  $T(n)$ , який визначається з наступної теореми.

**Теорема 1.** Якщо в узагальненому рекурентному рівнянні декомпозиції  $f(n) = \Theta(n^d)$ , де  $d \geq 0$  то

$$T(n) = \begin{cases} \Theta(n^d), & \text{якщо } a < b^d, \\ \Theta(n^d \log n), & \text{якщо } a = b^d, \\ \Theta(n^{\log_b a}), & \text{якщо } a > b^d. \end{cases} \quad (36)$$

Наприклад, в рекурентному рівнянні алгоритму сортування злиттям  $a = 2$ ,  $b = 2$ ,  $f(n) = \Theta(n)$  і  $d = 1$ . Отже, маємо випадок  $a = b^d$ . Тоді, слідуючи теоремі, обчислювальна складність сортування злиттям в гіршому випадку дорівнює

$$T(n) = \Theta(n^d \log n) = \Theta(n \log n). \quad (37)$$

Розглянемо ще один приклад. Нехай для деякого алгоритму отримане рекурентне співвідношення

$$T(n) = 2T(n/2) + 1. \quad (38)$$

Необхідно знайти асимптотично точну оцінку для  $T(n)$ . Неважко помітити, що в даному випадку  $a = 2$ ,  $b = 2$ ,  $f(n) = \Theta(1)$  і  $d = 0$ . Отже, оскільки  $a > b^d$ :

$$T(n) = \Theta(n^d \log n) = \Theta(n \log n). \quad (39)$$



## Завдання.

1. Використовуючи метод дерева рішень розв'язати рекурентні рівняння.

- 1)  $T(n) = T(n - a) + T(a) + cn$
- 2)  $T(n) = T(\alpha n) + T((1 - \alpha)n) + cn$

2. Використовуючи основну теорему розв'язати рекурентні рівняння.

- 1)  $T(n) = 2T(n / 2) + n^3$
- 2)  $T(n) = T(\frac{9}{10}n) + n$
- 3)  $T(n) = 16T(n / 4) + n^2$
- 4)  $T(n) = 7T(n / 3) + n^2$
- 5)  $T(n) = 7T(n / 2) + n^2$
- 6)  $T(n) = 2T(n / 4) + n^{\frac{1}{2}}$
- 7)  $T(n) = 4T(n / 2) + n^2 \sqrt{n}$

3. Розв'язати рекурентні рівняння.

- 1)  $T(n) = T(n - 1) + n$
- 2)  $T(n) = T(\sqrt{n}) + 1$
- 3)  $T(n) = 5T(n / 5) + n / \lg n$
- 4)  $T(n) = 2T(n / 2) + n / \lg n$

Частина 3. Теоретичні питання.

Дайте відповіді на наступні запитання.

1. Що таке обчислювальна складність алгоритму?
2. Що означають записи  $f(n) = O(g(n))$ ,  $f(n) = \Theta(g(n))$ ,  $f(n) = \Omega(g(n))$ ?
3. Який алгоритм сортування називається стійким (stable)?
4. Який алгоритм сортування називається сортуванням «на місці» (in place)?
5. Яка обчислювальна складність в найгіршому випадку у алгоритмів, які Ви реалізували?
6. Який алгоритм сортування з обчислювальною складністю  $O(n \log n)$  для найгіршого випадку вам відомі?
7. Чи відомі вам алгоритми сортування, які працюють швидше ніж  $O(n \log n)$  для найгіршого випадку?
8. Що таке словник, асоціативний масив?
9. Що таке бінарне дерево пошуку (аналіз складності основних операцій)?

## 2.3 Завдання і методичні вказівки до виконання курсової роботи

### Постановка задачі.

У відповідності до свого варіанту необхідно реалізувати алгоритм та провести експериментальне дослідження його ефективності. Розподіл алгоритмів за варіантом наведено в табл. 1.

Таблиця 1. Розподіл завдань за варіантами

Варіант	Алгоритми, які не використовують операцію порівняння	Тривіальні алгоритми сортування	«Швидкі» алгоритми сортування
1	Counting Sort	Bubble Sort	MergeSort
2	Bucket Sort	Selection Sort	HeapSort
3	Counting Sort	Insertion Sort	QuickSort
4	Bucket Sort	Shell Sort	MergeSort
5	Counting Sort	Bubble Sort	HeapSort
6	Bucket Sort	Selection Sort	QuickSort
7	Counting Sort	Insertion Sort	MergeSort
8	Bucket Sort	Shell Sort	HeapSort
9	Counting Sort	Bubble Sort	QuickSort
10	Bucket Sort	Selection Sort	MergeSort
11	Counting Sort	Insertion Sort	HeapSort
12	Bucket Sort	Shell Sort	QuickSort
13	Counting Sort	Bubble Sort	MergeSort
14	Bucket Sort	Selection Sort	HeapSort
15	Counting Sort	Insertion Sort	QuickSort
16	Bucket Sort	Shell Sort	MergeSort
17	Counting Sort	Bubble Sort	HeapSort

### Експериментальні дослідження.

1. Необхідно визначити час роботи кожного алгоритму для масивів з різною кількістю елементів – заповніть таблицю 2 для кожного алгоритму.
2. За даними таблиці 2 побудуйте для кожного алгоритму графік залежності часу його виконання від кількості елементів в масиві.
3. За результатами експериментів визначте, який алгоритм працює швидше і чому?

### ***Зауваження.***

- 1) В експериментах використовуйте масиви з елементами типу `int`.
- 2) Масиви заповнюйте псевдовипадковими числами з рівномірним розподілом з інтервалу  $[0, 100000]$ .

Таблиця 2. Результати експериментів

№	Кількість елементів в масиві	Час виконання алгоритму, с
1	50 000	
2	100 000	
3	150 000	
...	...	
20	1 000 000	

В пояснювальній записці курсової роботи повинні бути наступні розділи:

1. Опис алгоритмів – загальна характеристика алгоритмів (бажано навести псевдокод), їх властивості (*in place*, *stable*), обчислювальна складність та складність по пам'яті.
2. Організація експериментів – в цьому розділі необхідно вказати конфігурацію машини, на якій ви проводили експерименти (модель процесора, об'єм пам'яті, версію операційної системи і ключі компіляції програми).
3. Результати експериментів – таблиці, графіки та висновки про ефективність алгоритмів.

## ЗМІСТ

	ВСТУП.....	3
1	РОБОЧА ПРОГРАМА ДИСЦИПЛІНИ «ЕКОНОМІЧНА КІБЕРНЕТИКА (ПОГЛИБЛЕНИЙ КУРС)».....	4
	1.1 Мета вивчення дисципліни.....	4
	1.2 Розподіл годин за навчальним планом.....	5
	1.3 Зміст дисципліни.....	5
	1.4 Рекомендована література.....	6
2	МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ КОНТРОЛЬНОЇ ТА КУРСОВОЇ РОБОТИ.....	7
	2.1 Загальні положення.....	7
	2.2 Завдання і методичні вказівки до виконання контрольної роботи	7
	2.2 Завдання і методичні вказівки до виконання курсової роботи.....	17