Лабораторна робота №1

Квадратичні алгоритми сортування

У роботі розглядаються алгоритми сортування вибором та вставкою. Наводиться доцільність використання цих алгоритмів, їх переваги та недоліки.

Питання:

1. Загальні відомості про завдання сортування поняття їх часової складності.
2. Сортування вибором.
3. Сортування вставками.
4. Аналіз обчислювальної складності алгоритмів сортування вибором та вставками
5. Завдання.
6. Вимоги до представлення звіту.

1. Загальні відомості про завдання сортування, поняття їх часової.

Нехай є послідовність a_0, a_1... a_n та функція порівняння, яка на будь-яких двох елементах послідовності набуває одного з трьох значень: менше, більше або дорівнює. Задача сортування полягає у перестановці членів послідовності у такий спосіб, щоб виконувалася умова: $a_i <= a_{i+1}$, для всіх i від 0 до n.

Можлива ситуація, коли елементи послідовності складаються з кількох полів, наприклад *x* та *y* :

```
struct element {
  field x;
  field y;
}
```

Якщо значення функції порівняння залежить тільки від поля *x*, то *x* називають ключем, по якому проводиться сортування. На практиці, у якості *x* часто виступає число, а поле *y* зберігає будь-які дані, що ніяк не впливають на роботу алгоритму.

Розглянемо параметри, за якими оцінюються алгоритми сортування.

1. *Час* сортування – основний параметр, що характеризує швидкодію алгоритму.
2. *Пам'ять* – ряд алгоритмів вимагає виділення додаткової пам'яті під тимчасове зберігання даних.
3. *Стійкість* – стійке сортування не змінює взаємного розташування рівних елементів. Така властивість може бути дуже корисною, якщо вони складаються з декількох полів, а сортування відбувається по одному з них, наприклад, по *x*. (рис. 1)
4. *Природність поведінки* – ефективність методу при обробці вже відсортованих, або частково відсортованих даних. Алгоритм веде себе

природно, якщо враховує цю характеристику вхідної послідовності та працює краще.

Рисунок 1 – Приклад даних стійкого та нестійкого сортування

Як бачимо, при стійкому сортування взаємне розташування рівних елементів з ключем «1» і додатковими полями «a», «b», «c» залишилося таким же як до сортування самим: елемент з полем «a», потім – з «b», потім – з «c». При приведенні сортування за нестійким алгоритмом взаємне розташування рівних елементів із ключем «1» та додатковими полями «a», «b», «c» змінилося.

Часова складність алгоритму визначає час роботи, який використовує алгоритм, як функцію від довжини послідовності, що представляє вхідні дані. Часова складність алгоритму звичайно виражається із використанням нотації «*O*» *велике*, яка виключає коефіцієнти та члени меншого порядку. Якщо складність виражена у такий спосіб, говорять про *асимптотичний* опис часової складності, тобто у разі наближення розміру вхідної послідовності до нескінченності. Наприклад, якщо час, який потрібний алгоритму для виконання роботи, для всіх вхідних послідовностей довжини n не перевищує $5n^3 + 3n$ для деякого n (більшого деякого n_0), асимптотична часова складність такого алгоритму дорівнює O (n^3).

Часова складність найчастіше оцінюється шляхом підрахунку числа *елементарних операцій*, здійснюваних алгоритмом, де елементарна операція займає для виконання фіксований час. Тоді повний час виконання та кількість елементарних операцій, виконаних алгоритмом, відрізняються максимум на постійний множник.

Алгоритми сортування *бульбашкою, вставками та вибором* відносяться до сортувань із квадратичним часом складності $O(n^2)$

2. Сортування вибором

Дії алгоритму *сортування вибором* починаються з пошуку (вибору) найменшого елемента у списку та обміну його з першим елементом (отже найменший елемент поміщається в остаточну позицію у відсортованому списку). Потім ми скануємо список, починаючи з другого елемента, у пошуках найменшого серед n-1 елементів, що залишилися, і обмінюємо знайдений найменший елемент з другим, тобто поміщаємо другий найменший елемент у остаточну позицію у відсортованому списку. У загальному випадку, при i-ому проході по списку ($0 \le i \le n$-2) алгоритм

знаходить найменший елемент серед останніх *n-i* елементів і обмінює його з A_i. (рис. 2). Після виконання *n*-1 проходів список виявляється відсортованим.

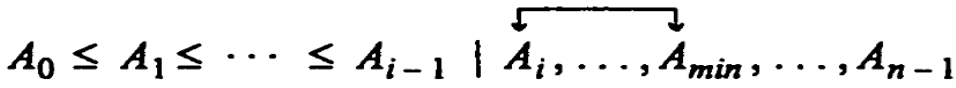


Рисунок 2 – Алгоритм сортування вибором

Алгоритм сортування вибором представимо за допомогою наступного *псевдокоду*

Лістинг 1 – Псевдокод алгоритму сортування вибором.

```
Алгоритм Selection_sort (A [0..n - 1])
// Сортування масиву методом вибору
// Вхідні дані: Масив A [0.. n - 1] елементів, що впорядковуються
// Вихідні дані: Масив A [0.. n - 1], відсортований у неспадному
порядку
for i← 0 to n-2 do // зовнішній цикл
{
     min ← i
          for j ← i+1 to n -1 do // внутрішній цикл
               if A[j] < A[min]
                    min← j
     B ← A[i] // Обмін A[i] та A [min]
     A[i]← A[min]
     A[min]← B
}
```

Як приклад на рисунку 3 наведено сортування вибором наступної послідовності: **A=[89_0, 45_1, 68_2, 90_3, 29_4, 34_5, 17_6] n=7**

№ ітерації/ позиції	0	1	2	3	4	5	6
i = 0	89 |	45	68	90	29	34	**17**
1	17 |	45	68	90	**29**	34	89
2	17	29 |	68	90	45	**34**	89
3	17	29	34 |	90	**45**	68	89
4	17	29	34	45 |	90	**68**	89
5	17	29	34	45	68 |	90	**89**
6	17	29	34	45	68	89 |	90

Рисунок 3 – Приклад сортування вибором

На рисунку наведено наступні позначення. Кожен рядок відповідає одній ітерації (*i*) зовнішнього циклу агоритму *Selection_sort*. Спершу робиться

пропозиція, що мінімальний елемент послідовності знаходиться в позиції *i* Далі во внутрішньому циклу агоритму *Selection_sort* (ітерації *j*) відбувається сканування послідовності кожен раз праворуч від вертикальної межі (*j*←*i*+1). Напівжирним шрифтом виділено найменші елементи, які обираються під час сканування. При цьому на кожній ітерації по *j* фіксується положення меншого елементу. Елементи ліворуч від вертикальної межі знаходяться в остаточних позиціях та не скануються. Обмін значеннями елементів в позиціях *i* та *min* виконується з використанням додаткової змінної B

Ускладнимо завдання необхідністю порахувати кількості порівнянь comparisons та присвоювань assignments, які виконує алгоритм сортування. Бо саме ці операції відносять до елементарних за якими визначається складність алгоритму.

Алгоритм сортування вибором (Selection_sort) на Python виглядатиме наступним чином.

Лістинг 2 – Python-код алгоритму сортування вибором.

```
def selection_sort(arr):
    n = len(arr)
    comparisons = 0
    assignments = 0

    # Зовнішній цикл ітерується по всьому списку
    # від 0 до n-2 (n-1 проходів)
        for i in range(n - 1):
        # Припускаємо, що поточний елемент є мінімальним
        min_index = i
        assignments += 1  # Присвоєння змінній min_index

        # Внутрішній цикл шукає найменший елемент в решті списку
        for j in range(i + 1, n):
            comparisons += 1  # Операція порівняння
            if arr[j] < arr[min_index]:
                min_index = j
                assignments += 1  # Присвоєння змінній min_index

        # Обмін елементів, якщо знайдено новий мінімальний
          comparisons += 1  # Операція порівняння
          if min_index != i:
            # Обмін елементів
            arr[i], arr[min_index] = arr[min_index], arr[i]
            assignments += 3  # Три присвоєння при обміні
    return arr, comparisons, assignments
# Приклад використання
my_list = [89, 45, 68, 90, 29, 34, 17]
sorted_list, comps, assigs = selection_sort(my_list.copy())
  # Використання .copy(), щоб не змінювати оригінал
print("Оригінальний список:", my_list)
print("Відсортований список:", sorted_list)
print(f"Кількість порівнянь: {comps}")
print(f"Кількість присвоєнь: {assigs}")
```

Результати виконання Python-коду показано далі:

– Оригінальний список: [89, 45, 68, 90, 29, 34, 17]
– Відсортований список: [17, 29, 34, 45, 68, 89, 90]
– Кількість порівнянь: 27
– Кількість присвоєнь: 33

Як бачимо сумарна кількість базових операцій дорівнює 60 (27+33), що відповідає припущенню про квадратичну складність алгоритму сортування вибором для $n=7$ $(n^2=49)$

Для кращого розуміння дій алгоритму проведемо так зване «трасування» та чисельне моделювання наведених псевдокоду та Python-коду (Лістінгі 1 та 2)

Трасування (англ. tracing) – процес моніторингу та запису послідовності виконання програми або алгоритму. Його метою є детально відстежити, як змінюються дані та як алгоритм проходить через свої кроки, щоб допомогти розробникам зрозуміти логіку, виявити помилки або оцінити ефективність.

До основних елементів трасування відносяться відстеження:

– стану змінних або візуалізація значень змінних на кожному етапі виконання.
– ходу виконання: візуалізація рядки коду, які виконуються, які цикли та умовні оператори спрацьовують.
– візуалізація проміжних результатів або станів.

Чисельне моделювання – використання математичної моделі для імітації поведінки реальної системи, часто з використанням комп'ютера. У контексті алгоритмів, це може означати ручне або автоматизоване проходження алгоритму з конкретними вхідними даними, щоб побачити, які проміжні результати виходять, і перевірити, чи відповідає це очікуванням. У лабораторній роботі чисельне моделювання — це ручне виконання кроків алгоритму сортування вибором (Selection Sort) для конкретного масиву чисел. Для виконання трасування Python-коду (Лістінг 2) необхідно самостійно додати команди print у відповідних місцях коду.

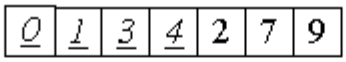
Таблиця 1 Результати чисельного моделювання та трасування алгоритму сортування вибором.

Чисельне моделювання псевдокоду Лістінг 1	Трасування Python-коду Лістінг 2
Вихідні дані A=[89_0, 45_1, 68_2, 90_3, 29_4, 34_5, 17_6] n=7 **(for *i*)** 1. i=0; *min*=0; *j*=0+1=1; A[1]<A[0] (45<89) min=1 1. j=2 A[2]<A[1] (68<45) min=1 2. j=3 A[3]<A[1] (90<45) min=1 3. j=4 A[4]<A[1] (29<45) min=4 4. j=5 A[5]<A[4] (34<29) min=4 5. j=6 A[6]<A[4] (17<29) min=6 (end for *j*) b=A[0] A[0]=A[6] A[6]=b A[0]=17 A[6]=89 A=[17_0, 45_1, 68_2, 90_3, 29_4, 34_5, 89 $_6$]	Початковий масив: [89, 45, 68, 90, 29, 34, 17] ---------- Ітерація 0: Поточний елемент (для обміну): arr[0] = 89 Шукаємо мінімальний елемент у частині: [89, 45, 68, 90, 29, 34, 17] Знайдено мінімальний елемент: arr[6] = 17 Обмін arr[0] (89) і arr[6] (17) Масив після ітерації 0: [17, 45, 68, 90, 29, 34, 89] ----------

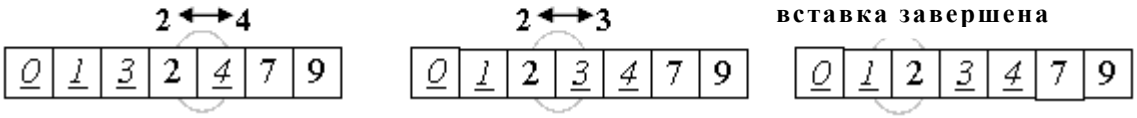
Чисельне моделювання псевдокоду Лістінг 1	Трасування Python-коду Лістінг 2
2. i=1; *min*=1; *j*=1+1=2; A[2]<A[1] (68<45) min=1 1. j=3 A[3]<A[1] (90<45) min=1 2. j=4 A[4]<A[1] (29<45) min=4 3. j=5 A[5]<A[4] (34<29) min=4 4. j=6 A[6]<A[4] (89<29) min=4 (end for *j*) b=A[1] A[1]=A[4] A[4]=b A[1]=29 A[4]=45 A=[17_0, 29_1, 68_2, 90_3, 45_4, 34_5, 89_6] 3. i=2; *min=2; j=2*+1=3; A[3]<A[2] (68<90) min=2 1. j=4 A[4]<A[1] (45<68) min=4 2. j=5 A[5]<A[4] (34<45) min=5 3. j=6 A[6]<A[4] (89<34) min=5 (end for *j*) b=A[2] A[2]=A[5] A[5]=b A[2]=34 A[5]=68 A=[17_0, 29_1, 34_2, 90_3, 45_4, 68_5, 89_6] 4. i=3; *min=3; j=3*+1=4; A[4]<A[3] (45<90) min=4 1. j=5 A[5]<A[4] (68<45) min=4 2. j=6 A[6]<A[4] (89<45) min=4 (end for *j*) b=A[3] A[3]=A[4] A[3]=b A[3]=45 A[4]=90 A=[17_0, 29_1, 34_2, 45_3, 90_4, 68_5, 89_6] 5. i=4; *min=4; j=4*+1=5; A[5]<A[4] (68<90) min=5 1. j=6 A[6]<A[5] (89<68) min=5 (end for *j*) b=A[4] A[4]=A[5] A[4]=b A[4]=68 A[5]=90 A=[17_0, 29_1, 34_2, 45_3, 68_4, 90_5, 89_6] 6. **i=5**; *min=5;* ***j=5*+1=6***; A[6]<A[5] (89<90) min=6 (end for *j*) b=A[5] A[5]=A[6] A[5]=b A[5]=89 A[6]=90 A=[17_0, 29_1, 34_2, 45_3, 68_4, 89_5, 90_6] **(end for *i*)**	Ітерація 1: Поточний елемент (для обміну): arr[1] = 45 Шукаємо мінімальний елемент у частині: [45, 68, 90, 29, 34, 89] Знайдено мінімальний елемент: arr[4] = 29 Обмін arr[1] (45) і arr[4] (29) Масив після ітерації 1: [17, 29, 68, 90, 45, 34, 89] ---------- Ітерація 2: Поточний елемент (для обміну): arr[2] = 68 Шукаємо мінімальний елемент у частині: [68, 90, 45, 34, 89] Знайдено мінімальний елемент: arr[5] = 34 Обмін arr[2] (68) і arr[5] (34) Масив після ітерації 2: [17, 29, 34, 90, 45, 68, 89] ---------- Ітерація 3: Поточний елемент (для обміну): arr[3] = 90 Шукаємо мінімальний елемент у частині: [90, 45, 68, 89] Знайдено мінімальний елемент: arr[4] = 45 Обмін arr[3] (90) і arr[4] (45) Масив після ітерації 3: [17, 29, 34, 45, 90, 68, 89] ---------- Ітерація 4: Поточний елемент (для обміну): arr[4] = 90 Шукаємо мінімальний елемент у частині: [90, 68, 89] Знайдено мінімальний елемент: arr[5] = 68 Обмін arr[4] (90) і arr[5] (68) Масив після ітерації 4: [17, 29, 34, 45, 68, 90, 89] ---------- Ітерація 5: Поточний елемент (для обміну): arr[5] = 90 Шукаємо мінімальний елемент у частині: [90, 89] Знайдено мінімальний елемент: arr[6] = 89 Обмін arr[5] (90) і arr[6] (89) Масив після ітерації 5: [17, 29, 34, 45, 68, 89, 90] ---------- Сортування завершено. Фінальний відсортований масив: [17, 29, 34, 45, 68, 89, 90] Загальна кількість порівнянь: 27 Загальна кількість присвоєнь: 33

3. Сортування вставками

Розглянемо алгоритм вставками на прикладі, починаючі з дій на і-му кроці. Послідовність А на цей момент розділена на дві частини: відсортовану A[0]...A[i] та невідсортовану A[i+1]...A[n-1]. Далі на (i+1)-му кроці алгоритму беремо a[i+1] і вставляємо на потрібне місце у частину масиву A[0]...A[i].
Пошук відповідного місця для чергового елемента вхідної послідовності здійснюється шляхом послідовних порівнянь з елементом, що стоїть перед ним. Залежно від результату порівняння елемент або залишається на поточному місці (вставка завершена), або вони змінюються місцями і процес повторюється (Рис. 4).



Послідовність на поточний момент. Частина a[0]..a[3] вже впорядкована.



Вставка числа 2 у відсортовану послідовність. Пари, що порівнюються виділені.

Рисунок 4 – Приклад сортування вставками

Таким чином, у процесі вставки ми "просіваємо" елемент *A* до початку масиву, зупиняючись у разі, коли знайдений елемент, менший за *A* або досягнуто початку послідовності.
Алгоритм сортування вставками складається з n-1 проходів (n – розмірність масиву). Кожен із проходів включає чотири дії:

1. взяття чергового *i*-го невідсортованого елемента та збереження його у додатковій змінній;
2. пошук позиції *j* у відсортованій частині масиву, у якій присутність взятого елемента не порушить упорядкованості елементів;
3. зсув елементів масиву від *i*-1-го до *j*-1-го вправо, щоб звільнити знайдену позицію вставки;
4. вставка взятого елементу в знайдену *j*-ю позицію.

Псевдокод алгоритму сортування вставками представлений нижче
Лістинг 3 – Псевдокод алгоритму сортування вставками.

```
Insertion_Sort
for j = 2 to A.length do
    key = A[j]
    i = j-1
    while (int i > 0 and A[i] > key) do
        A[i + 1] = A[i]
        i = i - 1
    end while
    A[i+1] = key
end
```

На вхід подається масив A [1..n], що містить послідовність з n чисел, що сортуються (кількість елементів масиву A позначено в цьому коді як A.length). Вхідні числа сортуються без використання додаткової пам'яті. Їхня

перестановка проводиться в межах масиву, і обсяг використовуваної при цьому додаткової пам'яті не перевищує постійну величину, яка потрібна для зберігання комірки `key`. Зверність увагу на наявність двох вкладених циклів. Перший зовнішній виконує ітерації за елементами невідсортованої частини масиву, а другий внутрішній за перестановками у відсортованій частині масиву. По закінченні роботи алгоритму Insertion_Sort вхідний масив містить відсортовану послідовність.

Ускладнимо завдання необхідністю порахувати кількості порівнянь `comparisons` та присвоювань `assignments`, які виконує алгоритм сортування. Алгоритм сортування вибором insertion_sort на Python виглядатиме наступним чином.

Лістинг 4 – Python-код алгоритму сортування вставками

```
def insertion_sort(arr):
    n = len(arr)
    comparisons = 0
    assignments = 0

    # Цикл ітерується від другого елемента до кінця
    # i - це індекс елемента, який потрібно вставити
    for i in range(1, n):
        # Зберігаємо поточний елемент для вставки
        key = arr[i]
        assignments += 1

        # j - індекс попереднього елемента
        j = i - 1
        assignments += 1

        # Пересуваємо елементи, що більші за key,
        # вправо, щоб звільнити місце для вставки
        while j >= 0 and arr[j] > key:
            comparisons += 1  # Порівняння в умові while
            arr[j + 1] = arr[j]
            assignments += 1
            j -= 1
            assignments += 1
        # Додаткове порівняння, коли умова while стає false
        # (якщо j не стало менше 0)
        if j >= 0:
            comparisons += 1

        # Вставляємо key на його правильне місце
        arr[j + 1] = key
        assignments += 1

    return arr, comparisons, assignments
```

Результати виконання Python-коду показано далі:

- Оригінальний список: [89, 45, 68, 90, 29, 34, 17]
- Відсортований список: [17, 29, 34, 45, 68, 89, 90]
- Кількість порівнянь: 19
- Кількість присвоєнь: 50

Для кращого розуміння дій алгоритму проведемо так зване «трасування» та чисельне моделювання наведених псевдокоду та Python-коду (Лістінгі 3 та 4)

Таблиця 2 Результати чисельного моделювання та трасування алгоритму сортування вставками.

Чисельне моделювання псевдокоду Лістінг 3	Трасування Python-коду Лістінг 4
Вихідні дані A=[89_1, 45_2, 68_3, 90_4, 29_5, 34_6, 17_7] n=7 A.length=7 **(for j)** 1. j = 2; key = A[2] = 45; i = 2 – 1 = 1; i>0 (True) A[1] > key (89 > 45 – True) → A[2] = A[1] = 89; A=[89_1, 89_2, 68_3, 90_4, 29_5, 34_6, 17_7] i = 1 – 1 = 0; i>0 (False) A[1] = key = 45 A=[45_1, 89_2, 68_3, 90_4, 29_5, 34_6, 17_7] j = 3; key = A[3] = 68; i = 3 – 1 =2; i>0 (True) A[2] > key (89 > 68 – True) → A[3] = A[2] = 89; A=[45_1, 89_2, 89_3, 90_4, 29_5, 34_6, 17_7] i = 2 – 1 = 1; i>0 (True) A[1] > key (45 > 68 – False); A[2] = key = 68 A=[45_1, 68_2, 89_3, 90_4, 29_5, 34_6, 17_7] j = 4; key = A[4] = 90; i = 4 – 1 =3; i>0 (True) A[3] > key (89 > 90 – False) A[4] = key = 90 A=[45_1, 68_2, 89_3, 90_4, 29_5, 34_6, 17_7] j = 5; key = A[5] = 29; i = 5 – 1 =4; i>0 (True) A[4] > key (90 > 29 –True) → A[5] = A[4] = 90; A=[45_1, 68_2, 89_3, 90_4, 90_5, 34_6, 17_7] i = 4 – 1 = 3; i>0 (True) A[3] > key (89 > 29 –True) → A[4] = A[3] = 89; A=[45_1, 68_2, 89_3, 89_4, 90_5, 34_6, 17_7] i = 3 – 1 = 2; i>0 (True) A[2] > key (68 > 29 –True) → A[3] = A[2] = 68; A=[45_1, 68_2, 68_3, 89_4, 90_5, 34_6, 17_7] i = 2 – 1 = 1; i>0 (True) A[1] > key (45 > 29 –True) → A[2] = A[1] = 45; A=[45_1, 45_2, 68_3, 89_4, 90_5, 34_6, 17_7] i = 1 – 1 = 0; i>0 (False) A[1] = key = 29 A=[29_1, 45_2, 68_3, 89_4, 90_5, 34_6, 17_7]	Початковий масив: [89, 45, 68, 90, 29, 34, 17] ---------- Ітерація 1: Елемент для вставки (key): 45 Відсортована частина: [89] Порівняння: 89 > 45. True. Зсуваємо 89 вправо. Досягнуто початку масиву. Цикл завершено. Вставка 45 на позицію 0. Масив після ітерації 1: [45, 89, 68, 90, 29, 34, 17] ---------- Ітерація 2: Елемент для вставки (key): 68 Відсортована частина: [45, 89] Порівняння: 89 > 68. True. Зсуваємо 89 вправо. Порівняння: 45 > 68. False. Цикл завершено. Вставка 68 на позицію 1. Масив після ітерації 2: [45, 68, 89, 90, 29, 34, 17] ---------- Ітерація 3: Елемент для вставки (key): 90 Відсортована частина: [45, 68, 89] Порівняння: 89 > 90. False. Цикл завершено. Вставка 90 на позицію 3. Масив після ітерації 3: [45, 68, 89, 90, 29, 34, 17] ---------- Ітерація 4: Елемент для вставки (key): 29 Відсортована частина: [45, 68, 89, 90] Порівняння: 90 > 29. True. Зсуваємо 90 вправо. Порівняння: 89 > 29. True. Зсуваємо 89 вправо. Порівняння: 68 > 29. True. Зсуваємо 68 вправо. Порівняння: 45 > 29. True. Зсуваємо 45 вправо. Досягнуто початку масиву. Цикл завершено. Вставка 29 на позицію 0. Масив після ітерації 4: [29, 45, 68, 89, 90, 34, 17] ----------

j = 6; key = A[6] = 34; i = 6 – 1 =5; i>0 (True) A[5] > key (90 > 34 –True) → A[6] = A[5] = 90; A=[29_1, 45_2, 68_3, 89_4, 90_5, 90_6, 17_7] i = 5 – 1 = 4; i>0 (True) A[4] > key (89 > 34 –True) → A[5] = A[4] = 89; A=[29_1, 45_2, 68_3, 89_4, 89_5, 90_6, 17_7] i = 4 – 1 = 3; i>0 (True) A[3] > key (68 > 34 –True) → A[4] = A[3] = 68; A=[29_1, 45_2, 68_3, 68_4, 89_5, 90_6, 17_7] i = 3 – 1 = 2; i>0 (True) A[2] > key (45 > 34 –True) → A[3] = A[2] = 45; A=[29_1, 45_2, 45_3, 68_4, 89_5,90_6, 17_7] i = 2 – 1 = 1; i>0 (True) A[1] > key (29 > 34 –False) → A[3] = A[2] = 45; A[2] = key = 34 A=[29_1, 34_2, 45_3, 68_4, 89_5,90_6, 17_7] j = 7; key = A[7] = 17; i = 7 – 1 =6; i>0 (True) A[6] > key (90 > 17 –True) → A[7] = A[6] = 90; A=[29_1, 34_2, 45_3, 68_4, 89_5,90_6, 90_7] i = 6 – 1 = 5; i>0 (True) A[5] > key (89 > 17 –True) → A[6] = A[5] = 89; A=[29_1, 34_2, 45_3, 68_4, 89_5,89_6, 90_7] i = 5 – 1 = 4; i>0 (True) A[4] > key (68 > 17 –True) → A[5] = A[4] = 68; A=[29_1, 34_2, 45_3, 68_4,68_5,89_6, 90_7] i = 4 – 1 = 3; i>0 (True) A[3] > key (45 > 17 –True) → A[4] = A[3] = 45 A=[29_1, 34_2, 45_3, 45_4,68_5,89_6, 90_7] i = 3 – 1 = 2; i>0 (True) A[2] > key (34 > 17 –True) → A[3] = A[2] = 34 A=[29_1, 34_2, 34_3, 45_4,68_5,89_6, 90_7] i = 2 – 1 = 1; i>0 (True) A[1] > key (29 > 17 –True) → A[2] = A[1] = 29 A=[29_1, 29_2, 34_3, 45_4,68_5,89_6, 90_7] i = 1 – 1 = 0; i>0 (False) A[1] = key = 17 A=[17_1, 29_2, 34_3, 45_4,68_5,89_6, 90_7] **(End for j)**	Ітерація 5: Елемент для вставки (key): 34 Відсортована частина: [29, 45, 68, 89, 90] Порівняння: 90 > 34. True. Зсуваємо 90 вп раво. Порівняння: 89 > 34. True. Зсуваємо 89 вп раво. Порівняння: 68 > 34. True. Зсуваємо 68 вп раво. Порівняння: 45 > 34. True. Зсуваємо 45 вп раво. Порівняння: 29 > 34. False. Цикл завершен о. Вставка 34 на позицію 1. Масив після ітерації 5: [29, 34, 45, 68, 89, 90, 17] ---------- Ітерація 6: Елемент для вставки (key): 17 Відсортована частина: [29, 34, 45, 68, 89 , 90] Порівняння: 90 > 17. True. Зсуваємо 90 вп раво. Порівняння: 89 > 17. True. Зсуваємо 89 вп раво. Порівняння: 68 > 17. True. Зсуваємо 68 вп раво. Порівняння: 45 > 17. True. Зсуваємо 45 вп раво. Порівняння: 34 > 17. True. Зсуваємо 34 вп раво. Порівняння: 29 > 17. True. Зсуваємо 29 вп раво. Досягнуто початку масиву. Цикл завершено. Вставка 17 на позицію 0. Масив після ітерації 6: [17, 29, 34, 45, 68, 89, 90] ---------- Сортування завершено. Фінальний відсортований масив: [17, 29, 34, 45, 68, 89, 90] Загальна кількість порівнянь: 19 Загальна кількість присвоєнь: 50

Як бачимо сумарна кількість базових операцій дорівнює 69 (19+50), що відповідає припущенню про квадратичну складність алгоритму сортування вставками для $n=7$ $(n^2=49)$

4. Аналіз обчислювальної складності алгоритмів сортування вибором та вставками

Оскільки результати лабораторної роботи 1 можуть суттєво відрізнятися залежно від вхідних даних, то зробити загальні висновки відносно обчислювальної складності алгоритмів сортування вибором та вставками не можливо. Тому порівняємо отримані характеристики обох алгоритмів маючи на увазі різну машинну складність операцій присвоювання та порівняння.

Відомо, що загалом, операція присвоювання (запис даних у пам'ять) є значно дорожчою, ніж операція порівняння тому, що запис у пам'ять вимагає більшої кількості машинних циклів, ніж зчитування. Операція порівняння вимагає зчитування двох значень з пам'яті та їх порівняння, що виконується дуже швидко процесором. Операція присвоювання вимагає запису значення в певну комірку пам'яті, що в свою чергу вимагає звернення до шини пам'яті та запису даних, що є повільнішою операцією. Таким чином, алгоритм, який виконує меншу кількість операцій присвоювання, може бути ефективнішим, навіть якщо він робить більше порівнянь.

З теоретичної точки зору алгоритм **Selection Sort** мінімізує кількість операцій присвоювання. На кожному кроці він робить лише один обмін. При цьому кількість операцій порівнянь завжди дорівнює $O(n^2)$, тобто згідно алгоритму завжди перевіряється кожен елемент, щоб знайти мінімальний. Кі лькість операцій присвоювання завжди дорівнює $O(n)$, тобто кількість обмінів є фіксованою, оскільки на кожному проході відбувається лише один обмін.
З теоретичної точки зору алгоритм **Insertion Sort** мінімізує кількість порівнянь, якщо масив вже частково або повністю відсортований. Однак, він може виконувати багато операцій присвоювання (зсувів), щоб звільнити місце для елемента. При цьому кількість операцій порівнянь у найкращому випадку (масив відсортований) дорівнює $O(n)$. У найгіршому та середньому випадку це буде $O(n^2)$. Кількість операцій присвоювання у найкращому випадку дорівнює $O(n)$. У найгіршому та середньому випадку це буде $O(n^2)$.
Сформуємо за результатами лабораторної роботи Таблицю 3

Таблиця 3. Порівняння обчислювальної складності алгоритмів сортування вибором та вставками

Назва алгоритму	Теоретичні відомості		Результати експерименту n=7 A=[89_0, 45_1, 68_2, 90_3, 29_4, 34_5, 17_6]	
	порівняння	присвоювання	порівняння	присвоювання
Selection Sort	$O(n^2)$,	$O(n)$	27	33
Insertion Sort	$O(n)$ (Best) $O(n^2)$ (Worst, Average)	$O(n)$ (Best) $O(n^2)$ (Worst, Average)	19	50

5.Завдання

1. Відсортувати за допомогою сортування вибором
2. Відсортувати за допомогою сортування вставками
3. Порівняти між собою алгоритми сортування за кількістю операцій порівняння та присвоювання, проаналізувати яким чином виконується сортування для відсортованих послідовностей в прямому та зворотному порядку

Варіанти вхідних послідовностей для індивідуальних завдань співпадають із порядковим номером студента в групі.

Варіант	Послідовність
1	38, 15, 50, 99, 41, 52, 47, 65, 95
2	79, 97, 82, 18, 20, 2, 88, 61, 17
3	68, 97, 12, 15, 31, 40, 22, 50, 53
4	35, 95, 16, 33, 28, 76, 27, 10, 5
5	46, 11, 49, 78, 77, 4, 62, 8, 69
6	58, 5, 50, 99, 61, 32, 27, 45, 75
7	41, 68, 67, 10, 7, 69, 95, 43, 98
8	69, 52, 97, 27, 10, 88,29, 1, 24
9	12, 23, 67, 65, 50, 70, 80, 61, 92
10	87, 79, 97, 82,98, 40, 42, 88, 61
11	47, 50, 61, 41,53, 12, 68, 63, 3
12	53, 100, 44, 74, 53, 38, 82, 65, 28
13	90, 10, 15, 80, 100, 6, 57, 5, 29
14	10, 90, 95, 30, 45, 60, 57, 28, 5
15	50, 80, 19, 86, 35, 7, 60, 48, 51
16	54, 65, 7, 33, 86, 29, 11, 91, 12
17	7, 89, 4, 68, 70, 49, 10, 62, 51
18	21, 44, 22, 50, 63, 68, 97, 12, 15
19	80, 27, 37, 36, 91, 53, 86, 66, 98
20	86, 36, 14, 50, 64, 21, 2, 83, 82
21	50, 57, 78, 34, 41, 68, 47, 61, 38
22	19, 75, 43, 31, 5, 66, 62, 34, 76
23	77, 89, 74, 68, 70, 49, 5, 62, 51
24	53, 5, 44, 47, 35, 83, 82, 85, 28
25	11, 42, 67, 55, 65, 78, 25, 50, 69
26	41, 52, 47, 65, 95, 38, 15, 50, 99
27	18, 20, 2, 88, 61, 17, 79, 97, 82
28	31, 40, 22, 50, 53, 68, 97, 12, 15
29	28, 76, 27, 10, 5, 35, 95, 16, 33
30	78, 77, 4, 62, 8, 69, 46, 11, 49

31	61, 32, 27, 45, 75, 58, 5, 50, 99
32	95, 43, 98, 41, 68, 67, 10, 7, 69
33	29, 1, 24, 69, 52, 97, 27, 10, 88
34	70, 80, 61, 92, 12, 23, 67, 65, 50
35	98, 40, 42, 88, 61, 87, 79, 97, 82
36	53, 12, 68, 63, 3, 47, 50, 61, 41

6.Вимоги до представлення звіту

Звіт повинен містити:

1. Титульну частину (лист) з вказівкою на назву лабораторної роботи, ПІБ студента, групу, номер варіанту та послідовність, яку необхідно отсортувати.
2. Псевдокоди сортування вибором та вставками, результат їх чисельного моделювання та трасування за варіантами завдань
3. Результати порівнянь алгоритмів сортування вибором та вставками між собою за кількістю операцій порівняння та присвоювання.
4. *Яким чином зміняться наведені в п. 3 результати для відсортованих послідовностей в прямому та зворотному порядку*
5. Висновки