Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Навчально-науковий інститут комп’ютерних систем

Кафедра інформаційних систем

Лабораторна робота № 1

з дисципліни: «Квадратичні алгоритми сортування»

Варіант № 17

Виконав:

Студент групи АІ-243

Кучерявенко О.В.

Перевірили:

Смик С. Ю.

Арсірій О.О.

Одеса 2025

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

Завдання:

Відсортувати за допомогою сортування вибором послідовність: [7, 89, 4, 68, 70, 49, 10, 62, 51].

Відсортувати за допомогою сортування вставками послідовність: [7, 89, 4, 68, 70, 49, 10, 62, 51].

Виконати моделювання, трасування та підрахувати кількість операцій порівняння та присвоювання для обох алгоритмів.

Порівняти між собою алгоритми сортування за отриманими результатами.

Проаналізувати, яким чином зміняться результати для відсортованих послідовностей в прямому та зворотному порядку.

**Результати виконання завдання:**

Лістинг 1.1 – Псевдокод алгоритму сортування вибором

|  |
| --- |
| Алгоритм Selection\_Sort(A[0..n-1])  // 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  min ← i  for j ← i+1 to n-1 do  if A[j] < A[min]  min ← j  B ← A[i]  A[i] ← A[min]  A[min] ← B |

Лістинг 1.2 — Python-код алгоритму сортування вибором з підрахунком операцій

|  |
| --- |
| def selection\_sort(arr):  n = len(arr)  comparisons = 0  assignments = 0  for i in range(n - 1):  min\_index = i  assignments += 1  for j in range(i + 1, n):  comparisons += 1  if arr[j] < arr[min\_index]:  min\_index = j  assignments += 1  comparisons += 1  if min\_index != i:  arr[i], arr[min\_index] = arr[min\_index], arr[i]  assignments += 3  return arr, comparisons, assignments  # Вхідні дані  A = [7, 89, 4, 68, 70, 49, 10, 62, 51]  sorted\_A, comps, assigns = selection\_sort(A.copy())  print("Відсортований масив (Selection Sort):", sorted\_A)  print(f"Кількість порівнянь: {comps}")  print(f"Кількість присвоєнь: {assigns}")  print(f"Загальна кількість операцій: {comps + assigns}") |

Відсортований масив (Selection Sort): [4, 7, 10, 49, 51, 62, 68, 70, 89]

Кількість порівнянь: 36

Кількість присвоєнь: 42

Загальна кількість операцій: 78

Таблиця 1.1 — Чисельне моделювання роботи алгоритму сортування вибором для масиву A = [7, 89, 4, 68, 70, 49, 10, 62, 51]

|  |
| --- |
| Початковий масив: [7, 89, 4, 68, 70, 49, 10, 62, 51]  Ітерація 0 (i=0):  Пошук мінімуму в [7, 89, 4, 68, 70, 49, 10, 62, 51]:  min=0 (7)  j=1: 89 < 7? False  j=2: 4 < 7? True -> min=2  j=3: 68 < 4? False  j=4: 70 < 4? False  j=5: 49 < 4? False  j=6: 10 < 4? False  j=7: 62 < 4? False  j=8: 51 < 4? False  Мінімум знайдено на позиції 2 (4)  Обмін A[0] (7) та A[2] (4)  Масив після ітерації: [4, 89, 7, 68, 70, 49, 10, 62, 51]  Ітерація 1 (i=1):  Пошук мінімуму в [89, 7, 68, 70, 49, 10, 62, 51]:  min=1 (89)  j=2: 7 < 89? True -> min=2  j=3: 68 < 7? False  j=4: 70 < 7? False  j=5: 49 < 7? False  j=6: 10 < 7? False  j=7: 62 < 7? False  j=8: 51 < 7? False  Мінімум знайдено на позиції 2 (7)  Обмін A[1] (89) та A[2] (7)  Масив після ітерації: [4, 7, 89, 68, 70, 49, 10, 62, 51]  Ітерація 2 (i=2):  Пошук мінімуму в [89, 68, 70, 49, 10, 62, 51]:  min=2 (89)  j=3: 68 < 89? True -> min=3  j=4: 70 < 68? False  j=5: 49 < 68? True -> min=5  j=6: 10 < 49? True -> min=6  j=7: 62 < 10? False  j=8: 51 < 10? False  Мінімум знайдено на позиції 6 (10)  Обмін A[2] (89) та A[6] (10)  Масив після ітерації: [4, 7, 10, 68, 70, 49, 89, 62, 51]  Ітерація 3 (i=3):  Пошук мінімуму в [68, 70, 49, 89, 62, 51]:  min=3 (68)  j=4: 70 < 68? False  j=5: 49 < 68? True -> min=5  j=6: 89 < 49? False  j=7: 62 < 49? False  j=8: 51 < 49? False  Мінімум знайдено на позиції 5 (49)  Обмін A[3] (68) та A[5] (49)  Масив після ітерації: [4, 7, 10, 49, 70, 68, 89, 62, 51]  Ітерація 4 (i=4):  Пошук мінімуму в [70, 68, 89, 62, 51]:  min=4 (70)  j=5: 68 < 70? True -> min=5  j=6: 89 < 68? False  j=7: 62 < 68? True -> min=7  j=8: 51 < 62? True -> min=8  Мінімум знайдено на позиції 8 (51)  Обмін A[4] (70) та A[8] (51)  Масив після ітерації: [4, 7, 10, 49, 51, 68, 89, 62, 70]  Ітерація 5 (i=5):  Пошук мінімуму в [68, 89, 62, 70]:  min=5 (68)  j=6: 89 < 68? False  j=7: 62 < 68? True -> min=7  j=8: 70 < 62? False  Мінімум знайдено на позиції 7 (62)  Обмін A[5] (68) та A[7] (62)  Масив після ітерації: [4, 7, 10, 49, 51, 62, 89, 68, 70]  Ітерація 6 (i=6):  Пошук мінімуму в [89, 68, 70]:  min=6 (89)  j=7: 68 < 89? True -> min=7  j=8: 70 < 68? False  Мінімум знайдено на позиції 7 (68)  Обмін A[6] (89) та A[7] (68)  Масив після ітерації: [4, 7, 10, 49, 51, 62, 68, 89, 70]  Ітерація 7 (i=7):  Пошук мінімуму в [89, 70]:  min=7 (89)  j=8: 70 < 89? True -> min=8  Мінімум знайдено на позиції 8 (70)  Обмін A[7] (89) та A[8] (70)  Масив після ітерації: [4, 7, 10, 49, 51, 62, 68, 70, 89]  Фінальний відсортований масив: [4, 7, 10, 49, 51, 62, 68, 70, 89] |

Лістинг 2.2 — Python-код алгоритму сортування вставками з підрахунком операцій

|  |
| --- |
| def insertion\_sort(arr):  n = len(arr)  comparisons = 0  assignments = 0  for i in range(1, n):  key = arr[i]  assignments += 1  j = i - 1  assignments += 1  while j >= 0:  comparisons += 1  if arr[j] > key:  arr[j + 1] = arr[j]  assignments += 1  j -= 1  assignments += 1  else:  break  if j >= 0:  comparisons += 1  arr[j + 1] = key  assignments += 1  return arr, comparisons, assignments  # Вхідні дані  A = [7, 89, 4, 68, 70, 49, 10, 62, 51]  sorted\_A, comps, assigns = insertion\_sort(A.copy())  print("Відсортований масив (Insertion Sort):", sorted\_A)  print(f"Кількість порівнянь: {comps}")  print(f"Кількість присвоєнь: {assigns}")  print(f"Загальна кількість операцій: {comps + assigns}") |

**Результат виконання коду:**

Відсортований масив (Insertion Sort): [4, 7, 10, 49, 51, 62, 68, 70, 89]

Кількість порівнянь: 24

Кількість присвоєнь: 59

Загальна кількість операцій: 83

Таблиця 2.1 — Трасування роботи алгоритму сортування вставками для масиву A = [7, 89, 4, 68, 70, 49, 10, 62, 51]

|  |
| --- |
| Початковий масив: [7, 89, 4, 68, 70, 49, 10, 62, 51]  Ітерація 1 (j=1, key=89):  Відсортована частина: [7]  Порівняння: 7 > 89? False  Вставка 89 на позицію 1  Масив після ітерації: [7, 89, 4, 68, 70, 49, 10, 62, 51]  Ітерація 2 (j=2, key=4):  Відсортована частина: [7, 89]  Порівняння: 89 > 4? True -> зсув 89  Порівняння: 7 > 4? True -> зсув 7  Досягнуто початок масиву  Вставка 4 на позицію 0  Масив після ітерації: [4, 7, 89, 68, 70, 49, 10, 62, 51]  Ітерація 3 (j=3, key=68):  Відсортована частина: [4, 7, 89]  Порівняння: 89 > 68? True -> зсув 89  Порівняння: 7 > 68? False  Вставка 68 на позицію 2  Масив після ітерації: [4, 7, 68, 89, 70, 49, 10, 62, 51]  Ітерація 4 (j=4, key=70):  Відсортована частина: [4, 7, 68, 89]  Порівняння: 89 > 70? True -> зсув 89  Порівняння: 68 > 70? Falsе  Вставка 70 на позицію 3  Масив після ітерації: [4, 7, 68, 70, 89, 49, 10, 62, 51]  Ітерація 5 (j=5, key=49):  Відсортована частина: [4, 7, 68, 70, 89]  Порівняння: 89 > 49? True -> зсув 89  Порівняння: 70 > 49? True -> зсув 70  Порівняння: 68 > 49? True -> зсув 68  Порівняння: 7 > 49? False  Вставка 49 на позицію 2  Масив після ітерації: [4, 7, 49, 68, 70, 89, 10, 62, 51]  Ітерація 6 (j=6, key=10):  Відсортована частина: [4, 7, 49, 68, 70, 89]  Порівняння: 89 > 10? True -> зсув 89  Порівняння: 70 > 10? True -> зсув 70  Порівняння: 68 > 10? True -> зсув 68  Порівняння: 49 > 10? True -> зсув 49  Порівняння: 7 > 10? False  Вставка 10 на позицію 2  Масив після ітерації: [4, 7, 10, 49, 68, 70, 89, 62, 51]  Ітерація 7 (j=7, key=62):  Відсортована частина: [4, 7, 10, 49, 68, 70, 89]  Порівняння: 89 > 62? True -> зсув 89  Порівняння: 70 > 62? True -> зсув 70  Порівняння: 68 > 62? True -> зсув 68  Порівняння: 49 > 62? False  Вставка 62 на позицію 4  Масив після ітерації: [4, 7, 10, 49, 62, 68, 70, 89, 51]  Ітерація 8 (j=8, key=51):  Відсортована частина: [4, 7, 10, 49, 62, 68, 70, 89]  Порівняння: 89 > 51? True -> зсув 89  Порівняння: 70 > 51? True -> зсув 70  Порівняння: 68 > 51? True -> зсув 68  Порівняння: 62 > 51? True -> зсув 62  Порівняння: 49 > 51? False  Вставка 51 на позицію 5  Масив після ітерації: [4, 7, 10, 49, 51, 62, 68, 70, 89]  Фінальний відсортований масив: [4, 7, 10, 49, 51, 62, 68, 70, 89] |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Назва Алгоритму | Теоретична складність (порівняння) | Теоретична складність (присвоювання) | Результати есперименту (n=9) Порівняння | Результати есперименту (n=9) Присвоювання |
| Selection Sort | O(n2) | O(n) | 36 | 42 |
| Insertion Sort | O(n) (Best)  O(n2) (Worst, avg.) | O(n2) (Best)  O(n2) (Worst, avg.) | 24 | 59 |

**Для послідовності, відсортованої у прямому порядку [4, 7, 10, 49, 51, 62, 68, 70, 89]:**

Selection Sort:

Порівняння: Залишиться O(n²), оскільки внутрішній цикл завжди повністю сканує невідсортовану частину, щоб знайти мінімум (навіть якщо він вже на своєму місці). ~36 порівнянь.

Присвоювання: Залишиться O(n), оскільки буде виконано n-1 обмін, але лише якщо min\_index != i. Для відсортованого масиву умова min\_index != i буде хибною, тому обмінів не буде. Присвоєнь буде значно менше, близько 18-20.

Висновок: Алгоритм не є природним, його ефективність не покращується на вже відсортованих даних.

Insertion Sort:

Порівняння: O(n). На кожному кроці key порівнюється лише з одним елементом (попереднім), і умова A[i] > key одразу хибна. ~8 порівнянь (n-1).

Присвоювання: O(n). Виконується лише зчитування key та вставка його на ту саму позицію (жодних зсувів). ~18 присвоєнь (2\*(n-1)).

Висновок: Алгоритм є природним і на вже відсортованих даних працює дуже ефективно, майже за лінійний час.

Для послідовності, відсортованої у зворотному порядку [89, 70, 68, 62, 51, 49, 10, 7, 4]:

Selection Sort:

Порівняння: Як і завжди, O(n²). ~36 порівнянь.

Присвоювання: O(n). Кількість обмінів залишиться невеликою, але умова min\_index != i завжди істинна, тому відбудеться n-1 обмін. ~39 присвоєнь.

Висновок: Ефективність залишається стабільною та незалежною від вхідних даних.

Insertion Sort:

Порівняння: O(n²). На кожному кроці елемент key доведеться порівнювати та зсувати через усю відсортовану частину до самого початку. ~36 порівнянь.

Присвоювання: O(n²). Кожен крок вимагатиме максимальної кількості зсувів. ~90 присвоєнь.

Висновок: Це найгірший випадок для Insertion Sort, де його ефективність деградує до квадратичної.

**Висновки:** Обидва алгоритми, Selection Sort та Insertion Sort, у загальному випадку мають квадратичну складність O(n²), що підтверджується результатами експерименту для послідовності з 9 елементів.

Selection Sort виконує більше операцій порівняння, але має фіксовану та малу кількість операцій обміну (присвоєнь) – O(n). Це робить його вигідним, коли операція обміну є дуже «дорогою».

Insertion Sort виконує менше порівнянь у середньому, але значно більше присвоєнь через необхідність зсуву елементів. Він є природнім алгоритмом.

На практиці вибір алгоритму залежить від характеру даних.

**Посилання на GitHub: https://github.com/gutigomd/Algoritms/tree/main/lr1**