Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Навчально-науковий інститут комп’ютерних систем

Кафедра інформаційних систем

Лабораторна робота № 2

з дисципліни: «Логарифмічні 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 злиттям та швидкого сортування, які мають логарифмічну обчислювальну складність Аналізується доцільність використання цих алгоритмів, їх переваги та недоліки.

Завдання:

1. Реалізуйте ітеративний та рекурсивний алгоритм сортування

злиттям (псевдокод наданий в Таблиці 1) мовою Python, додайте підрахунок

базових операцій алгоритмів.

2. Наведіть приклад використання Python-коду для масиву за

варіантом завдання, передбачте можливість трасування.

3. Візуалізуйте результати трасування (Таблиця 2) у вигляді схем

виконання ітеративного та рекурсивного алгоритмів (наприклад з

використанням Tree Diagram).

4. Реалізуйте ітеративний та рекурсивний алгоритм сортування

злиттям (псевдокод наданий в Таблиці 1) мовою Python, додайте підрахунок

базових операцій алгоритмів.

5. Наведіть приклад використання Python-коду для масиву за

варіантом завдання, передбачте можливість трасування.

6. Візуалізуйте результати трасування (Таблиця 2) у вигляді схем

виконання ітеративного та рекурсивного алгоритмів (наприклад з

використанням Tree Diagram).

**Результати виконання завдання:**

Лістинг 1.1 — Python-код ітеративної реалізації алгоритму сортування злиттям

|  |
| --- |
| def merge\_sort\_iterative(arr):  n = len(arr)  comparisons = 0  assignments = 0  size = 1    while size < n:  left = 0  while left < n:  mid = min(left + size, n)  right = min(left + 2 \* size, n)  comp, assign = merge(arr, left, mid, right)  comparisons += comp  assignments += assign  left += 2 \* size  size \*= 2  return arr, comparisons, assignments  def merge(arr, left, mid, right):  comparisons = 0  assignments = 0  left\_part = arr[left:mid]  right\_part = arr[mid:right]  assignments += (mid - left) + (right - mid)    i = j = 0  k = left  assignments += 3    while i < len(left\_part) and j < len(right\_part):  comparisons += 1  if left\_part[i] <= right\_part[j]:  arr[k] = left\_part[i]  assignments += 1  i += 1  else:  arr[k] = right\_part[j]  assignments += 1  j += 1  k += 1  assignments += 1    while i < len(left\_part):  arr[k] = left\_part[i]  assignments += 1  i += 1  k += 1  assignments += 1    while j < len(right\_part):  arr[k] = right\_part[j]  assignments += 1  j += 1  k += 1  assignments += 1  return comparisons, assignments  # Використання  A = [7, 89, 4, 68, 70, 49, 10, 62, 51]  sorted\_A, comps, assigns = merge\_sort\_iterative(A.copy())  print("Відсортований масив (Iterative Merge):", sorted\_A)  print(f"Порівнянь: {comps}")  print(f"Присвоєнь: {assigns}") |

Результат виконання:

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

Порівнянь: 16

Присвоєнь: 156

Лістинг 1.2 — Python-код рекурсивної реалізації алгоритму сортування злиттям

|  |
| --- |
| def merge\_sort\_recursive(arr):  comparisons = 0  assignments = 0  recursive\_calls = 1    if len(arr) <= 1:  return arr, comparisons, assignments, recursive\_calls    mid = len(arr) // 2  assignments += 1    left\_arr, comp\_left, assign\_left, calls\_left = merge\_sort\_recursive(arr[:mid])  right\_arr, comp\_right, assign\_right, calls\_right = merge\_sort\_recursive(arr[mid:])    comparisons += comp\_left + comp\_right  assignments += assign\_left + assign\_right  recursive\_calls += calls\_left + calls\_right    merged\_arr, comp\_merge, assign\_merge = merge\_recursive(left\_arr, right\_arr)  comparisons += comp\_merge  assignments += assign\_merge    return merged\_arr, comparisons, assignments, recursive\_calls  def merge\_recursive(left, right):  merged = []  comparisons = 0  assignments = 0  i = j = 0    while i < len(left) and j < len(right):  comparisons += 1  if left[i] <= right[j]:  merged.append(left[i])  assignments += 1  i += 1  else:  merged.append(right[j])  assignments += 1  j += 1    while i < len(left):  merged.append(left[i])  assignments += 1  i += 1    while j < len(right):  merged.append(right[j])  assignments += 1  j += 1    return merged, comparisons, assignments  # Використання  A = [7, 89, 4, 68, 70, 49, 10, 62, 51]  sorted\_A, comps, assigns, calls = merge\_sort\_recursive(A.copy())  print("Відсортований масив (Recursive Merge):", sorted\_A)  print(f"Порівнянь: {comps}")  print(f"Присвоєнь: {assigns}")  print(f"Рекурсивних викликів: {calls}") |

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

Порівнянь: 16

Присвоєнь: 72

Рекурсивних викликів: 17

Таблиця 1.1 — Трасування ітеративної версії сортування злиттям

|  |
| --- |
| Початковий масив: [7, 89, 4, 68, 70, 49, 10, 62, 51]  Розмір = 1:  Об'єднуємо [7] та [89] → [7, 89]  Об'єднуємо [4] та [68] → [4, 68]  Об'єднуємо [70] та [49] → [49, 70]  Об'єднуємо [10] та [62] → [10, 62]  Залишається [51]  Масив: [7, 89, 4, 68, 49, 70, 10, 62, 51]  Розмір = 2:  Об'єднуємо [7, 89] та [4, 68] → [4, 7, 68, 89]  Об'єднуємо [49, 70] та [10, 62] → [10, 49, 62, 70]  Залишається [51]  Масив: [4, 7, 68, 89, 10, 49, 62, 70, 51]  Розмір = 4:  Об'єднуємо [4, 7, 68, 89] та [10, 49, 62, 70] → [4, 7, 10, 49, 62, 68, 70, 89]  Залишається [51]  Масив: [4, 7, 10, 49, 62, 68, 70, 89, 51]  Розмір = 8:  Об'єднуємо [4, 7, 10, 49, 62, 68, 70, 89] та [51] → [4, 7, 10, 49, 51, 62, 68, 70, 89]  Масив: [4, 7, 10, 49, 51, 62, 68, 70, 89]  Фінальний відсортований масив: [4, 7, 10, 49, 51, 62, 68, 70, 89] |

Таблиця 1.2 — Трасування рекурсивної версії сортування злиттям

|  |
| --- |
| Початковий масив: [7, 89, 4, 68, 70, 49, 10, 62, 51]  Рівень 1: [7, 89, 4, 68, 70, 49, 10, 62, 51]  Ліва половина: [7, 89, 4, 68]  Права половина: [70, 49, 10, 62, 51]  Рівень 2 (ліво): [7, 89, 4, 68]  Ліва: [7, 89]  Права: [4, 68]  Рівень 3 (ліво-ліво): [7, 89]  Ліва: [7]  Права: [89]  Злиття: [7, 89]  Рівень 3 (ліво-право): [4, 68]  Ліва: [4]  Права: [68]  Злиття: [4, 68]  Рівень 2 (ліво) - злиття: [7, 89] + [4, 68] → [4, 7, 68, 89]  Рівень 2 (право): [70, 49, 10, 62, 51]  Ліва: [70, 49]  Права: [10, 62, 51]  Рівень 3 (право-ліво): [70, 49]  Ліва: [70]  Права: [49]  Злиття: [49, 70]  Рівень 3 (право-право): [10, 62, 51]  Ліва: [10]  Права: [62, 51]  Рівень 4 (право-право-право): [62, 51]  Ліва: [62]  Права: [51]  Злиття: [51, 62]  Рівень 3 (право-право) - злиття: [10] + [51, 62] → [10, 51, 62]  Рівень 2 (право) - злиття: [49, 70] + [10, 51, 62] → [10, 49, 51, 62, 70]  Рівень 1 - злиття: [4, 7, 68, 89] + [10, 49, 51, 62, 70] → [4, 7, 10, 49, 51, 62, 68, 70, 89]  Фінальний відсортований масив: [4, 7, 10, 49, 51, 62, 68, 70, 89] |

Лістинг 2.1 — Python-код рекурсивної реалізації алгоритму швидкого сортування

|  |
| --- |
| def quicksort\_hoare(arr, low=0, high=None):  comparisons = 0  assignments = 0  recursive\_calls = 1    if high is None:  high = len(arr) - 1    if low < high:  pivot\_index, comp\_part, assign\_part = partition\_hoare(arr, low, high)  comparisons += comp\_part  assignments += assign\_part    comp\_left, assign\_left, calls\_left = quicksort\_hoare(arr, low, pivot\_index)  comp\_right, assign\_right, calls\_right = quicksort\_hoare(arr, pivot\_index + 1, high)    comparisons += comp\_left + comp\_right  assignments += assign\_left + assign\_right  recursive\_calls += calls\_left + calls\_right    return comparisons, assignments, recursive\_calls  def partition\_hoare(arr, low, high):  comparisons = 0  assignments = 0    pivot = arr[low]  assignments += 1  i = low - 1  j = high + 1  assignments += 2    while True:  i += 1  assignments += 1  while arr[i] < pivot:  comparisons += 1  i += 1  assignments += 1  comparisons += 1    j -= 1  assignments += 1  while arr[j] > pivot:  comparisons += 1  j -= 1  assignments += 1  comparisons += 1    comparisons += 1  if i >= j:  return j, comparisons, assignments    arr[i], arr[j] = arr[j], arr[i]  assignments += 3  # Використання  A = [7, 89, 4, 68, 70, 49, 10, 62, 51]  comps, assigns, calls = quicksort\_hoare(A.copy())  print("Відсортований масив (Quick Sort):", A)  print(f"Порівнянь: {comps}")  print(f"Присвоєнь: {assigns}")  print(f"Рекурсивних викликів: {calls}") |

Результати виконання:

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

Порівнянь: 52

Присвоєнь: 76

Рекурсивних викликів: 17

Таблиця 2.1 — Трасування швидкого сортування за схемою Хоара

|  |
| --- |
| Початковий масив: [7, 89, 4, 68, 70, 49, 10, 62, 51]  Виклик 1: [7, 89, 4, 68, 70, 49, 10, 62, 51], low=0, high=8  Опорний: 7  Після розділення: [4, 7, 89, 68, 70, 49, 10, 62, 51], pivot\_index=1  Виклик 2 (лівий): [4, 7], low=0, high=1  Опорний: 4  Після розділення: [4, 7], pivot\_index=0  Виклик 3 (правий): [89, 68, 70, 49, 10, 62, 51], low=2, high=8  Опорний: 89  Після розділення: [51, 68, 70, 49, 10, 62, 89], pivot\_index=7  Виклик 4 (лівий): [51, 68, 70, 49, 10, 62], low=2, high=7  Опорний: 51  Після розділення: [10, 49, 51, 70, 68, 62], pivot\_index=4  Виклик 5 (лівий): [10, 49], low=2, high=4  Опорний: 10  Після розділення: [10, 49], pivot\_index=2  Виклик 6 (правий): [70, 68, 62], low=5, high=7  Опорний: 70  Після розділення: [62, 68, 70], pivot\_index=6  Виклик 7 (лівий): [62, 68], low=5, high=6  Опорний: 62  Після розділення: [62, 68], pivot\_index=5  Фінальний відсортований масив: [4, 7, 10, 49, 51, 62, 68, 70, 89] |

Таблиця 3.1 — Порівняння обчислювальної складності алгоритмів сортування для послідовності з 9 елементів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Теоретична складність | Порівняння | Присвоєння | Рекурсивні виклики |
| Merge Sort (Ітеративний) | O(n log n) | 16 | 156 | 0 |
| Merge Sort (Рекурсивний) | O(n log n) | 16 | 72 | 17 |
| Quick Sort (Хоара) | O(n log n) – Середній  O(n²) -найгірший | 52 | 76 | 17 |

**Висновки:** Сортування злиттям демонструє стабільну продуктивність O(n log n) незалежно від вхідних даних. Ітеративна версія виконує більше операцій присвоювання через необхідність створення тимчасових масивів, але уникнення рекурсії робить її більш ефективною з точки зору використання пам'яті.

Рекурсивне сортування злиттям виконує менше операцій присвоювання, але створює значне навантаження на стек через рекурсивні виклики (17 викликів для масиву з 9 елементів).

Швидке сортування за схемою Хоара в даному випадку показало більшу кількість порівнянь, але меншу кількість присвоювань порівняно з ітеративним сортуванням злиттям. Алгоритм ефективно використовує пам'ять, оскільки сортує "на місці".

Вибір алгоритму залежить від конкретних потреб.

**Посилання на GitHub: https://github.com/gutigomd/Algoritms/tree/main/lr2**