САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2

по курсу «Алгоритмы и структуры данных»

Тема: Сортировка слиянием. Метод декомпозиции
Вариант 1

Выполнил:
Бен Шамех Абделазиз
Группа: K3239

Проверила:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание

1 Задача 1. Сортировка слиянием

1.1 Условие

Реализовать алгоритм сортировки слиянием (*Merge Sort*) без использования сигнальных значений. Проверить корректность на различных наборах данных.

1.2 Листинг кода

```python
def merge_sort(arr, p, r):
    if p < r:
        q = (p+r)//2
        merge_sort(arr,p,q)
        merge_sort(arr,q+1,r)
        merge(arr, p, q, r)

def merge(arr, p, q, r):
    arr1 = arr[p:q+1]
    arr2 = arr[q+1:r+1]
    i = j = 0
    while i < len(arr1) and j < len(arr2):
        if arr1[i] < arr2[j]:
            arr[p+i+j] = arr1[i]; i += 1
        else:
            arr[p+i+j] = arr2[j]; j += 1
    while i < len(arr1):
        arr[p+i+j] = arr1[i]; i += 1
    while j < len(arr2):
        arr[p+i+j] = arr2[j]; j += 1
```

1.3 Текстовое объяснение

Используется метод «Разделяй и властвуй». Массив рекурсивно делится пополам до единичных элементов. При возврате из рекурсии выполняется слияние двух отсортированных половин в одну общую последовательность.

1.4 Результаты выполнения

Ввод: `6 \n 31 41 59 26 41 58`. **Вывод:** `26 31 41 41 58 59`.

Метрики эффективности:

Сценарий	Время (сек)	Память (Mb)
Нижняя граница (n=1)	0.00022 sec	15.54 Mb
Пример из задачи	0.00016 sec	15.54 Mb
Верхняя граница	0.02904 sec	16.51 Mb

2 Задача 2. Сортировка слиянием+

2.1 Условие

Модифицировать алгоритм так, чтобы после каждого слияния выводились индексы начала и конца области, а также значения первого и последнего элементов этой области.

2.2 Листинг кода

```
def merge_with_logs(arr, left, mid, right, logs):
    L = arr[left:mid+1]
    R = arr[mid+1:right+1]
    i = j = 0; k = left
    while i < len(L) and j < len(R):
        if L[i] <= R[j]:
            arr[k] = L[i]; i += 1
        else:
            arr[k] = R[j]; j += 1
        k += 1
    while i < len(L):
        arr[k] = L[i]; i += 1; k += 1
    while j < len(R):
        arr[k] = R[j]; j += 1; k += 1
    logs.append(f"{left+1} {right+1} {arr[left]} {arr[right]}")
```

2.3 Текстовое объяснение

Для выполнения условий задачи в процедуру слияния добавлен логгер. После того как участок массива $[left, right]$ отсортирован, информация о границах (в 1-индексации) и значениях сохраняется в список `logs` для итогового вывода.

2.4 Результаты выполнения

Ввод: `10 \n 1 8 2 1 4 7 3 2 3 6`
Вывод:

```
1 2 1 8
4 5 1 4
...
1 10 1 8
1 1 2 2 3 3 4 6 7 8
```

3 Задача 3. Число инверсий

3.1 Условие

Подсчитать количество инверсий в массиве ($i < j$ и $A[i] > A[j]$) за время $O(n \log n)$.

3.2 Листинг кода

```
def merge_count(arr, p, q, r):
    global k
    L = arr[p:q+1]; R = arr[q+1:r+1]
    i = j = 0
    while i < len(L) and j < len(R):
        if L[i] <= R[j]:
            arr[p+i+j] = L[i]; i += 1
        else:
            arr[p+i+j] = R[j]; j += 1
            k += len(L) - i #
                      L[i]
```

3.3 Текстовое объяснение

Задача решается модификацией *Merge Sort*. Когда элемент из правого подмассива меньше элемента из левого, это означает, что он образует инверсию со всеми оставшимися элементами в левом подмассиве, так как они уже отсортированы и заведомо больше него.

3.4 Результаты выполнения

Ввод: `10 \n 1 8 2 1 4 7 3 2 3 6`. **Вывод:** `17`.

4 Задача 4. Бинарный поиск

4.1 Листинг кода

```python
def binary_search(arr, key):
    lt, rt = -1, len(arr)
    while rt - lt > 1:
        m = (rt + lt) // 2
        if key > arr[m]: lt = m
        else: rt = m
    return rt if rt < len(arr) and arr[rt] == key else -1
```

4.2 Текстовое объяснение

Для отсортированного массива используется итеративный подход. Мы делим область поиска пополам на каждом шаге. Если искомое значение больше центрального, смещаем левую границу, иначе — правую.

4.3 Результаты выполнения

Ввод: `5 1 5 8 12 13 \n 5 8 1 23 1 11`. **Вывод:** `2 0 -1 0 -1`.

5 Задача 5. Представитель большинства

5.1 Текстовое объяснение

Рекурсивно разбиваем массив до базовых случаев (один элемент). На этапе объединения проверяем "победителей"левой и правой половин. Если победители разные, подсчитываем их количество во всем текущем подмассиве и возвращаем того, кто встречается чаще.

5.2 Результаты выполнения

Ввод: `5 \n 2 3 9 2 2`. **Вывод:** `1`.

6 Задача 6. Поиск максимальной прибыли

6.1 Текстовое объяснение

Применяется рекурсивный алгоритм поиска максимального подмассива. Мы ищем максимальную сумму в левой половине, правой половине или на пересечении середины (*cross-subarray*). Это позволяет найти оптимальные дни покупки и продажи акций.

6.2 Результаты выполнения

Пример Газпром:
Вывод:

```
Фирма: ПАО Газпром
Дата покупки: 08.01.2024
Дата продажи: 09.01.2024
Прибыль: 38
```

7 Задача 7. Поиск максимального подмассива за линейное время

7.1 Листинг кода

```python
def max_subarray_linear(arr):
    min_pref = min(arr[0], 0)
    max_sum = arr[0]
    cur_sum = arr[0]
    for i in range(1, len(arr)):
        cur_sum += arr[i]
        max_sum = max(max_sum, cur_sum - min_pref)
        min_pref = min(cur_sum, min_pref)
    return max_sum
```

7.2 Текстовое объяснение

Используется алгоритм Кадана. Мы проходим по массиву один раз, поддерживая текущую префиксную сумму и минимальный префикс, встреченный ранее. Максимальный подмассив, заканчивающийся в текущем индексе, — это разность текущей суммы и минимального префикса.

7.3 Результаты выполнения

Ввод: `6 \n -2 1 -3 4 -1 2`. **Вывод:** `5`.

8 Вывод

1. В ходе лабораторной работы был изучен метод «Разделяй и властвуй», позволивший эффективно реализовать сортировку слиянием со сложностью $O(n \log n)$.

2. Модификации алгоритма слияния позволили решить задачу подсчета инверсий и поиска представителя большинства в рамках того же временного лимита.

3. Было проведено сравнение логарифмического и линейного подходов для поиска максимального подмассива, подтвердившее преимущество алгоритма Кадана на больших объемах данных.