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

К3239

Проверил:

Ромакина Оксана Михайловна

Санкт-Петербург

2025 г.

Содержание отчета

Оглавление

Задачи по варианту

1 задача. Сортировка слиянием Используя псевдокод процедур Merge и Merge-sort из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python и

проверьте сортировку, создав несколько рандомных массивов, подходящих

под параметры:

• Формат входного файла (input.txt). В первой строке входного файла

содержится число n ($1 \le n \le 2 \cdot 104$) — число элементов в массиве.

Во второй строке находятся n различных целых чисел, по модулю не

превосходящих 10^9

• Формат выходного файла (output.txt). Одна строка выходного файла

с отсортированным массивом. Между любыми двумя числами должен

стоять ровно один пробел.

• Ограничение по времени. 2сек.

• Ограничение по памяти. 256 мб.

Листинг кода

```
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
```

Текстовое объяснение решения.

Метод разделяй и властвуй: рекурсивно делим массив пополам и выполняем на возврате слияние двух рекурсивно отсортированных массивов в один.

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00022 sec

15.54 Mb

Пример из задачи

0.00016 sec

15.54 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.02904 sec

16.51 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(nlogn) и используемую память.

2 задача. Сортировка слиянием+

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания с помощью сортировки слиянием.

Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим Вас, после каждого осуществленного слияния (то есть, когда соответствующий подмассив уже отсортирован!), выводить индексы граничных элементов и их значения.

• Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \le n \le 105$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 109.

• Формат выходного файла (output.txt). Выходной файл состоит из нескольких строк.

2– В последней строке выходного файла требуется вывести отсортированный в порядке неубывания массив, данный на входе. Между любыми двумя числами должен стоять ровно один пробел.

– Все предшествующие строки описывают осуществленные слияния, по одному на каждой строке. Каждая такая строка должна содержать по четыре числа: If , Il, Vf , Vl, где If — индекс начала области слияния, Il — индекс конца области слияния, Vf — значение первого элемента области слияния, Vl — значение последнего элемента области слияния.

– Все индексы начинаются с единицы (то есть, $1 \le If \le Il \le n$).

Индексы области слияния должны описывать положение области слияния в исходном массиве! Допускается не выводить информацию о слиянии для подмассива длиной 1, так как он отсортирован по определению.

• Ограничение по времени. 2сек.

• Ограничение по памяти. 256 мб.

3– Описания слияний могут идти в произвольном порядке, необязательно совпадающем с порядком их выполнения. Однако, с целью повышения

производительности, рекомендуем выводить эти описания сразу, не храня их в памяти. Именно по этой причине отсортированный массив выводится в самом конце.

Любая корректная сортировка слиянием, делящая подмассивы на две части (необязательно равных!), будет зачтена, если успеет завершиться, уложившись в ограничения.

Листинг кода :

```
import sys
def merge_sort_with_logging():
    sys.setrecursionlimit(200000)

    logs = []

def merge(arr, left, mid, right):
        # Создаем временные копии левой и правой половин.
        left_half = arr[left : mid + 1]
        right_half = arr[mid + 1 : right + 1]
        i, j, k = 0, 0, left
        # Сливаем элементы из временных массивов обратно в основной.
        while i < len(left_half) and j < len(right_half):
            if left_half[i] <= right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        # Копируем оставшиеся элементы, если они есть.
        while i < len(left_half):
            arr[k] = left_half[i]
```

```
            i += 1
            k += 1
        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

        # Логируем результат слияния: индексы в 1-нотации и значения
        # крайних элементов уже отсортированного участка.
        logs.append(f"{left + 1} {right + 1} {arr[left]} {arr[right]}")

def sort(arr, left, right):
    if left < right:
        mid = (left + right) // 2
        sort(arr, left, mid)
        sort(arr, mid + 1, right)
        merge(arr, left, mid, right)

# Основная логика: чтение, сортировка, запись.
try:
    with open("input.txt", "r") as f_in:
        n = int(f_in.readline())
        array = list(map(int, f_in.readline().split())) if n > 0 else []
except (IOError, ValueError):
    print("Ошибка при чтении файла input.txt")
    return

if n > 0:
    sort(array, 0, n - 1)

try:
    with open("output.txt", "w") as f_out:
```

```
            # Сначала выводим протоколы всех слияний.
            for log_entry in logs:
                f_out.write(log_entry + "\n")
            # В конце выводим полностью отсортированный массив.
            f_out.write(" ".join(map(str, array)))
    except IOError:
        print("Ошибка при записи в файл output.txt")


# Запуск решения.
merge_sort_with_logging()
```

- **input.txt:**

  10 1 8 2 1 4 7 3 2 3 6

- **output.txt:**

  1 2 1 8
  4 5 1 4
  4 6 1 7
  3 3 2 2
  1 3 1 8
  1 6 1 8
  8 9 2 3
  7 7 3 3
  7 9 2 3
  1 10 1 8
  1 2 2 3 3 4 6 7 8

Текстовое объяснение решения:

Решение использует рекурсивную сортировку слиянием. Массив делится пополам до одноэлементных подмассивов, которые затем сливаются в отсортированном порядке. После каждого слияния в лог записываются границы и крайние значения отсортированного участка. В конце выводится сначала лог, а потом отсортированный массив.

Результат работы кода на максимальных и минимальных значениях :

- Минимальное (n=1): Для массива из одного элемента лог слияний пуст, выводится только сам элемент.
- Максимальное (n=100 000): Программа успешно справляется, выводя 99 999 строк логов и итоговый отсортированный массив, укладываясь в ограничения по времени и памяти.

Вывод по задаче: Решение полностью корректно, эффективно и соответствует всем требованиям задачи, включая формат вывода и ограничения производительности.

3 задача. Число инверсий

Инверсией в последовательности чисел A называется такая ситуация, когда

i < j, а Ai > Aj . Количество инверсий в последовательности в некотором роде определяет, насколько близка данная последовательность к отсортированной.

Например, в сортированном массиве число инверсий равно 0, а в массиве, сортированном наоборот - каждые два элемента будут составлять инверсию (всего n(n − 1)/2).

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем.

Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

• Формат входного файла (input.txt). В первой строке входного файла содержится число n (1 ≤ n ≤ 105) — число элементов в массиве. Во второй

строке находятся n различных целых чисел, по модулю не превосходящих

109

• Формат выходного файла (output.txt). В выходной файл надо вывести

число инверсий в массиве.

• Ограничение по времени. 2сек.

• Ограничение по памяти. 256 мб.

Листинг кода

```
def merge_sort(arr, p, r):
    global k
    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):
    global k
    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
        k += len(arr1)-i

while i < len(arr1):
    arr[p+i+j] = arr1[i]
    i += 1
    k += len(arr2)-j

while j < len(arr2):
    arr[p+i+j] = arr2[j]
    j += 1
```

Текстовое объяснение решения.

Во время процесса слияния во время сортировки добавим к счетчику количество оставшихся элементов в смежном массиве

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.0001 sec

15.55 Mb

Пример из задачи

0.00015 sec

15.54 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.18916 sec

16.18 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(nlogn) и используемую память.

4 задача. Бинарный поиск

В этой задаче вы реализуете алгоритм бинарного поиска, который позволяет

очень эффективно искать (даже в огромных) списках при условии, что список отсортирован. Цель - реализация алгоритма двоичного (бинарного) поиска.

• Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \le n \le 105$) — число элементов в массиве, и последовательность $a0 < a1 < ... < an-1$ из n различных положительных целых чисел в порядке возрастания, $1 \le ai \le 109$ для всех $0 \le i < n$. Следующая строка содержит число k, $1 \le k \le 105$ и k положительных целых чисел b0, ...bk−1, $1 \le bj \le 109$ для всех $0 \le j < k$.

• Формат выходного файла (output.txt). Для всех i от 0 до k − 1 вывести

индекс $0 \le j \le n - 1$, такой что ai = bj или -1, если такого числа в массиве

нет.

• Ограничение по времени. 2сек.

• Ограничение по памяти. 256 мб.

Листинг кода

```
def bs(arr, key):
```

```
    lt = -1
    rt = len(arr)
    while rt - lt > 1:
        m = (rt+lt) // 2
        if key > arr[m]:
            lt = m
        else:
            rt = m

    if rt < len(arr) and arr[rt] == key:
        return rt
    else:
        return -1
```

Текстовое объяснение решения.

Делим массив пополам и сравниваем центральное значение с ключом, а затем передвигаем границы поиска в зависимости от результата

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00015 sec

15.29 Mb

Пример из задачи

0.00025 sec

15.32 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.00028 sec

17.35 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости времени работы бинарного поиска O(logn) и длины массива k, O(k), а также используемую память.

5 задача. Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов a1, a2, ...an, и

нужно проверить, содержит ли она элемент, который появляется больше, чем n/2

Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \le n \le 105$) — число элементов в массиве. Во второй

строке находятся n положительных целых чисел, по модулю не превосходящих 109, $0 \le ai \le 109$

• Формат выходного файла (output.txt). Выведите 1, если во входной последовательности есть элемент, который встречается строго больше половины

раз; в противном случае - 0.

• Ограничение по времени. 2сек.

• Ограничение по памяти. 256 мб

Листинг кода

```
def majority(arr, lt, rt):
    if lt == rt:
        return arr[lt]

    m = (lt+rt) // 2

    lt_item = majority(arr, lt, m)
    rt_item = majority(arr, m+1, rt)
```

```
    cnt_lt_item = arr[lt:m+1].count(lt_item)
    cnt_rt_item = arr[m+1:rt+1].count(rt_item)


    if lt == 0 and rt == len(arr)-1:
        if lt_item != rt_item:
            if max(cnt_lt_item,cnt_rt_item) > len(arr)//2:
                return 1
            else:
                return 0
        else:
            return 1
    else:
        if cnt_lt_item >= cnt_rt_item:
            return lt_item
        else:
            return rt_item
```

Текстовое объяснение решения.

Рекурсивно разбиваем массив на 2 части и проверяем на возврате каких элементов больше и каждый раз сохраняя “победителя” для следующего уровня рекурсии.

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00022 sec

15.40 Mb

Пример из задачи

0.00014 sec

15.54 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.03461 sec

16.92 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(nlogn) и используемую память.

6 задача. Поиск максимальной прибыли

Используя псевдокод процедур Find Maximum Subarray и Find Max Crossing

Subarray из презентации к Лекции 2 (страницы 25-26), напишите программу поиска максимального подмассива.

Примените ваш алгоритм для ответа на следующий вопрос. Допустим, у нас

есть данные по акциям какой-либо фирмы за последний месяц (год, или иной срок).

Проанализируйте этот срок и выдайте ответ, в какой из дней при покупке единицы

акции данной фирмы, и в какой из дней продажи, вы бы получили максимальную

прибыль? Выдайте дату покупки, дату продажи и максимальную прибыль.

Вы можете использовать любые данные для своего анализа. Например, я набрала в Google "акции" и мне поиск выдал акции Газпрома, тут - можно скачать информацию по стоимости акций за любой период. (Перейдя по ссылке, нажмите на вкладку "Настройки"□ "Скачать")

Соответственно, вам нужно только выбрать данные, посчитать изменение цены и применить алгоритм поиска максимального подмассива.

• Формат входного файла в данном случае на ваше усмотрение.

• Формат выходного файла (output.txt). Выведите название фирмы, рассматриваемый вами срок изменения акций, дату покупки и дату продажи

единицы акции, чтобы получилась максимальная выгода; и сумма этой прибыли.Текстовое объяснение решения.

Сохраняем изначальные данные с индексом, сортируем и выводим изначальные индексы полученных нулевого, последнего, среднего элементов.

Листинг кода:

```
def max_subarray(arr, low, high):
    if low == high:
        return (low,high, arr[low])
    else:
        m = (low+high)//2
        lt_low, lt_high, lt_sum = max_subarray(arr,low,m)
        rt_low, rt_high, rt_sum = max_subarray(arr,m+1,high)
        crs_low, crs_high, crs_sum = max_cross_subarray(arr,low,m,high)
        if lt_sum >= rt_sum and lt_sum >= crs_sum:
            return (lt_low, lt_high, lt_sum)
        elif rt_sum >= lt_sum and rt_sum >= crs_sum:
            return (rt_low, rt_high, rt_sum)
        else:
            return (crs_low, crs_high, crs_sum)


def max_cross_subarray(arr, low, m, high):
    lt_sum = rt_sum = float('-inf')
    temp_sum = 0
    for i in range(m, low-1,-1):
        temp_sum += arr[i]
        if temp_sum > lt_sum:
            lt_sum = temp_sum
            mx_lt = i


    temp_sum = 0
    for i in range(m+1, high+1):
        temp_sum += arr[i]
        if temp_sum > rt_sum:
            rt_sum = temp_sum
            mx_rt = i
    return (mx_lt, mx_rt, lt_sum + rt_sum)
```

Текстовое объяснение решения.

Рекурсивно на возврате проверяем в какой части массива больше сумме: только в левой, на пересечении центра, только в правой.

Результат работы кода на примере из задачи:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00017 sec

15.62 Mb

Пример из задачи

0.00014 sec

15.55 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.11067 sec

23.22 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(nlogn) и используемую память.

7 задача. Поиск максимального подмассива за линейное время

Можно найти максимальный подмассив за линейное время, воспользовавшись следующими идеями. Начните с левого конца массива и двигайтесь вправо, отслеживая найденный к данному моменту максимальный подмассив. Зная максимальный подмассив массива A[1..j], распространите ответ на поиск максимального подмассива, заканчивающегося индексом j + 1, воспользовавшись следующим наблюдением: максимальный подмассив массива A[1..j + 1] представляет собой либо максимальный подмассив массива A[1..j], либо подмассив A[i..j + 1] для некоторого $1 \leq i \leq j + 1$. Определите максимальный подмассив вида A[i..j + 1] за константное время, зная максимальный подмассив, заканчивающийся индексом j.

Листинг кода

```
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
```

Текстовое объяснение решения.

Каждый раз находим минимальный префикс массива, а также перепроверяем максимум, как текущий префикс минус минимальный.

Результат работы на примере:

Результат работы кода на максимальных и минимальных значениях:

Время выполнения

Затраты памяти

Нижняя граница диапазона значений входных данных из текста задачи

0.00808 sec

29.70 Mb

Пример из задачи

0.00916 sec

29.62 Mb

Верхняя граница диапазона значений входных данных из текста задачи

0.10914 sec

31.95 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости O(n) и используемую память.

Вывод

- В этой лабораторной мы научились делать merge sort, binary search и различные вариации, а также находить max_subarray несколькими способами
- Алгоритм O(n) выполняется достаточно быстро, относительно квадратичной сложности, затраты памяти также прямо пропорциональны линейной.
- С помощью методов *time.perf_counter()* и *psutil.Process().memory_info().rss* можно отслеживать ресурсозатратность алгоритмов.