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 8

Выполнил:
Журбина Марина Андреевна
Группа К3139

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Оглавление

Задачи по варианту 8

Задание № 1. Сортировка слиянием.

Текст задачи:

1 задача. Сортировка слиянием

1. 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 ($1 \leq n \leq 2 \cdot 10^4$) — 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.

2. Для проверки можно выбрать наихудший случай, когда сортируется массив размера 1000, $10^4, 10^5$ чисел порядка 10^9, отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний. Сравните, например, с сортировкой вставкой на этих же данных.

3. Перепишите процедуру `Merge` так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива L или R скопированы обратно в массив A, после чего в этот массив копируются элементы, оставшиеся в непустом массиве.

или перепишите процедуру `Merge` (и, соответственно, `Merge-sort`) так, чтобы в ней не использовались значения границ и середины - p, r и q.

Код программы:

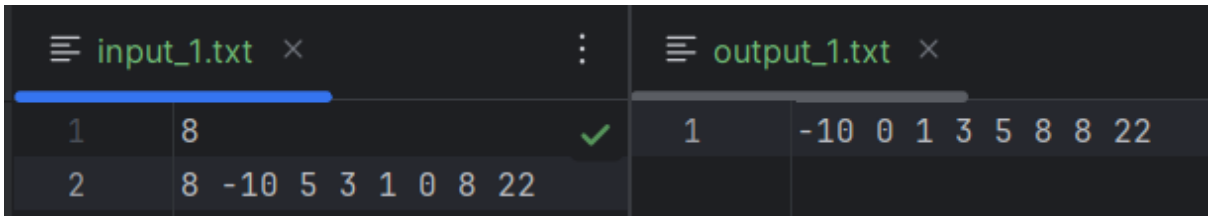
```
from time import time


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


def merge(a, left, mid, right):
    a_l = a[left:mid + 1] + [10 ** 100]
    a_r = a[mid + 1:right + 1] + [10 ** 100]

    i, j = 0, 0
    for k in range(left, right + 1):
        if a_l[i] <= a_r[j]:
            a[k] = a_l[i]
            i += 1
        else:
            a[k] = a_r[j]
            j += 1


t_start = time()
with open("input_1.txt") as f:
    n = int(f.readline())
```

```
    if not (1 <= n <= 2 * 10 ** 4):
        with open('output_1.txt', 'wt') as g:
            g.write('Введены некорректные данные')
        exit()

    a = list(map(int, f.readline().split()))
    for elem in a:
        if abs(elem) > 10 ** 9:
            with open('output_1.txt', 'wt') as g:
                g.write('Введены некорректные данные')
            exit()

    merge_sort(a, 0, len(a) - 1)
    with open('output_1.txt', 'wt') as g:
        g.write(' '.join([str(elem) for elem in a]))

t_stop = time()
if t_stop - t_start <= 2:
    print("Время выполнения:", t_stop - t_start, 'секунд')
else:
    print("Превышено время выполнения:", t_stop - t_start, 'секунд')
```

Текстовое объяснение решения:

1) Импорт модуля time для отслеживания времени работы программы.
2) Сохранение времени начала выполнения программы.
3) Считывание входных данных из файла input_1.txt.
4) Проверка корректности входных данных.
5) Вызов функции merge_sort, которой на вход подается сортируемый массив и левая и правая граница массива.
6) В функции merge_sort массив рекурсивно делится на два массива, пока массивы не будут состоять только из одного элемента, вызывается функция merge.
7) Функция merge соединяет два отсортированных подмассива в один сортированный массив.
8) Отсортированный массив записывается в файл.
9) Выводится время работы алгоритма.

Примеры работы кода:



	Время выполнения сортировки слиянием	Время выполнения сортировки вставками

Отсортированный массив максимальной длинны	Время выполнения: 0.059027671813964844 секунд	Время выполнения: 0.017602499981876463 секунд
Случайная генерация массива максимальной длинны	Время выполнения: 0.060024261474609375 секунд	Время выполнения: 16.862463299999945 секунд
Полностью не отсортированный массив максимальной длинны	Время выполнения: 0.07001781463623047 секунд	Время выполнения: 32.691591100010555 секунд

Вывод по задаче:
В решении задачи использованы переменные, операторы сравнения и if-else конструкции, цикл for, работа с файлами. Код считывает массив из файла, выполняет сортировку слиянием и сохраняет отсортированный массив в файл. Данный алгоритм сортировки выполняется существенно быстрее сортировки вставками.

Задание №3. Число инверсий.

Текст задачи:

3 задача. Число инверсий

Инверсией в последовательности чисел A называется такая ситуация, когда $i < j$, а $A_i > A_j$. 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 ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
- **Формат выходного файла (output.txt).** В выходной файл надо вывести число инверсий в массиве.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

Код:

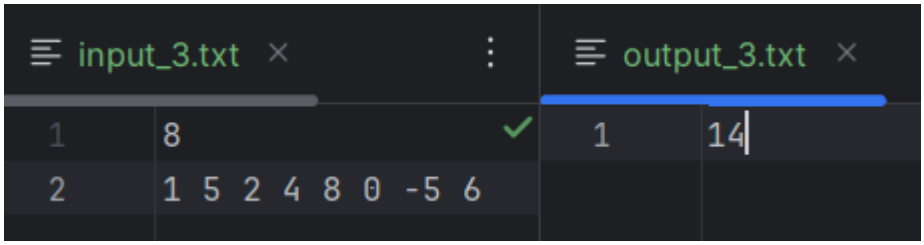
```
from time import time

cnt = 0


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


def merge(a, left, mid, right):
    global cnt
    a_l = a[left:mid + 1] + [10 ** 10]
    a_r = a[mid + 1:right + 1] + [10 ** 10]

    i, j = 0, 0
    for k in range(left, right + 1):
        if a_l[i] <= a_r[j]:
            a[k] = a_l[i]
            i += 1
        else:
            a[k] = a_r[j]
            j += 1
            cnt += len(a_l) - i - 1


t_start = time()
with open("input_3.txt") as f:
```

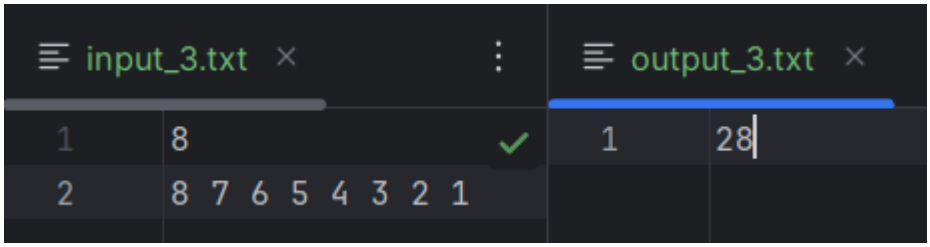
```
    n = int(f.readline())
    if not (1 <= n <= 10 ** 5):
        with open('output_3.txt', 'wt') as g:
            g.write('Введены некорректные данные')
        exit()

    a = list(map(int, f.readline().split()))
    for elem in a:
        if abs(elem) > 10 ** 9:
            with open('output_3.txt', 'wt') as g:
                g.write('Введены некорректные данные')
            exit()

    merge_sort(a, 0, len(a) - 1)
    with open('output_3.txt', 'wt') as g:
        g.write(str(cnt))

t_stop = time()
if t_stop - t_start <= 2:
    print("Время выполнения:", t_stop - t_start, 'секунд')
else:
    print("Превышено время выполнения:", t_stop - t_start, 'секунд')
```

Текстовое объяснение решения:

1) Импорт модуля time для отслеживания времени работы программы.
2) Сохранение времени начала выполнения программы.
3) Считывание входных данных из файла input_1.txt.
4) Проверка корректности входных данных.
5) Вызов функции merge_sort, которой на вход подается сортируемый массив и левая и правая граница массива.
6) В функции merge_sort массив рекурсивно делится на два массива, пока массивы не будут состоять только из одного элемента, вызывается функция merge.
7) Функция merge соединяет два отсортированных подмассива в один сортированный массив и подсчитывает количество инверсий в неотсортированном массиве из двух подмассивов.
8) Отсортированный массив записывается в файл.
9) Выводится время работы алгоритма.

Примеры работы кода:





Отсортированный массив максимальной длинны	Время выполнения: 0. 2980680465698242 секунд
Случайная генерация массива максимальной длинны	Время выполнения: 0. 3010875606536865 секунд
Полностью не отсортированный массив максимальной длинны	Время выполнения: 0. 3112924098968506 секунд
Пример из задачи	Время выполнения: 0. 0010001659393310547 секунд

Вывод по задаче:
В решении задачи использованы переменные, операторы сравнения и if-else конструкции, цикл for, работа с файлами. Код считывает массив из файла, выполняет сортировку слиянием, параллельно подсчитывая количество инверсий количество инверсий в файл.

Задание №4. Бинарный поиск.

Текст задачи:

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 ($1 \leq n \leq 10^5$) — число элементов в массиве, и последовательность $a_0 < a_1 < ... < a_{n-1}$ из n **различных** положительных целых чисел в порядке возрастания, $1 \leq a_i \leq 10^9$ для всех $0 \leq i < n$. Следующая строка содержит число k, $1 \leq k \leq 10^5$ и k положительных целых чисел $b_0, ...b_{k-1}$, $1 \leq b_j \leq 10^9$ для всех $0 \leq j < k$.
- **Формат выходного файла (output.txt).** Для всех i от 0 до $k-1$ вывести индекс $0 \leq j \leq n-1$, такой что $a_i = b_j$ или -1, если такого числа в массиве нет.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
5 1 5 8 12 13 5 8 1 23 1 11	2 0 -1 0 -1

В этом примере есть возрастающая последовательность из $a_0 = 1, a_1 = 5, a_2 = 8, a_3 = 12$ и $a_4 = 13$ длиной в $n = 5$ и пять чисел для поиска: 8 1 23 1 11. Видно, что $a_2 = 8$ и $a_0 = 1$, но чисел 23 и 11 нет в последовательности a, поэтому они имеют индекс -1. В итоге ответ: 2 0 -1 0 -1.

Код:

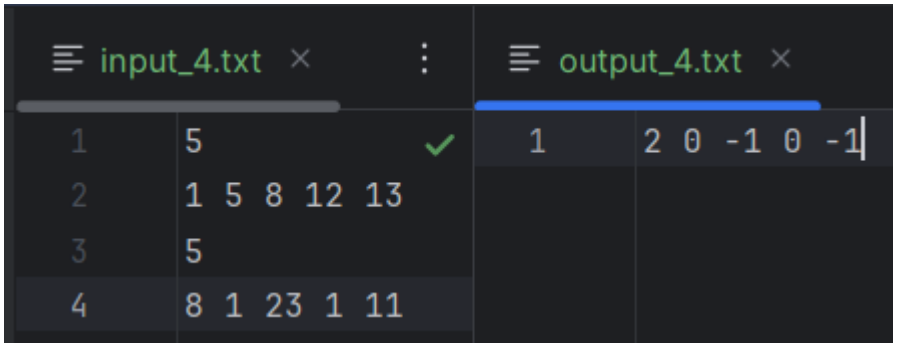
```
from time import time


def binary_search(a, value):
    left = 0
    right = len(a) - 1
    while left <= right:
        mid = (left + right) // 2
        if value == a[mid]:
            return mid

        if value > a[mid]:
            left = mid + 1
        else:
            right = mid - 1
    return -1


t_start = time()
with open("input_4.txt") as f:
    n = int(f.readline())
    if not (1 <= n <= 10 ** 5):
        with open('output_4.txt', 'wt') as g:
            g.write('Введены некорректные данные')
        exit()

    a = list(map(int, f.readline().split()))
    for elem in a:
```

```
        if not (1 <= elem <= 10 ** 9):
            with open('output_4.txt', 'wt') as g:
                g.write('Введены некорректные данные')
            exit()

    k = int(f.readline())
    if not (1 <= k <= 10 ** 5):
        with open('output_4.txt', 'wt') as g:
            g.write('Введены некорректные данные')
        exit()

    b = list(map(int, f.readline().split()))
    for elem in b:
        if not (1 <= elem <= 10 ** 9):
            with open('output_4.txt', 'wt') as g:
                g.write('Введены некорректные данные')
            exit()

    res = []
    for elem in b:
        res.append(str(binary_search(a, elem)))
    with open('output_4.txt', 'wt') as g:
        g.write(' '.join([str(elem) for elem in res]))

t_stop = time()
if t_stop - t_start <= 2:
    print("Время выполнения:", t_stop - t_start, 'секунд')
else:
    print("Превышено время выполнения:", t_stop - t_start, 'секунд')
```

Текстовое объяснение решения:

1) Импорт модуля time для отслеживания времени работы программы.
2) Сохранение времени начала выполнения программы.
3) Считывание входных данных из файла input_1.txt.
4) Проверка корректности входных данных.
5) Циклом for проходим по всем элементам списка b, для каждого вызывая функцию binary_search, записывая ее результат в конечный массив ответов.
6) Функция binary_search ищет среди отсортированного массива заданный элемент, если не находит его возвращает -1, если находит возвращает индекс этого элемента в отсортированном списке.
7) Массив ответов записывается в файл.
8) Выводится время работы алгоритма.

Пример работы кода:



Нижняя граница диапазона значений входных данных из текста задачи	Время выполнения: 0. 0009987354278564453 секунд
Пример из задачи	Время выполнения: 0. 002009153366088867 секунд
Верхняя граница диапазона значений входных данных из текста задачи	Время выполнения: 0.3376758098602295 секунд

Вывод по задаче:

В решении задачи использованы переменные, операторы сравнения и if-else конструкции, цикл for, работа с файлами. Код считывает отсортированный массив из файла и массив элементов, которые мы должны найти в отсортированном массиве и сохраняет массив индексов найденных элементов в выходной файл.

Дополнительные задачи

Задание №2. Сортировка слиянием +.

Текст задачи:

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 ($1 \le n \le 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
- **Формат выходного файла (output.txt).** Выходной файл состоит из нескольких строк.
  - 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: I_f, I_l, V_f, V_l, где I_f — индекс начала области слияния, I_l — индекс конца области слияния, V_f — значение первого элемента области слияния, V_l — значение последнего элемента области слияния.
  - Все индексы начинаются с единицы (то есть, $1 \le I_f \le I_l \le n$). **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.

Код:

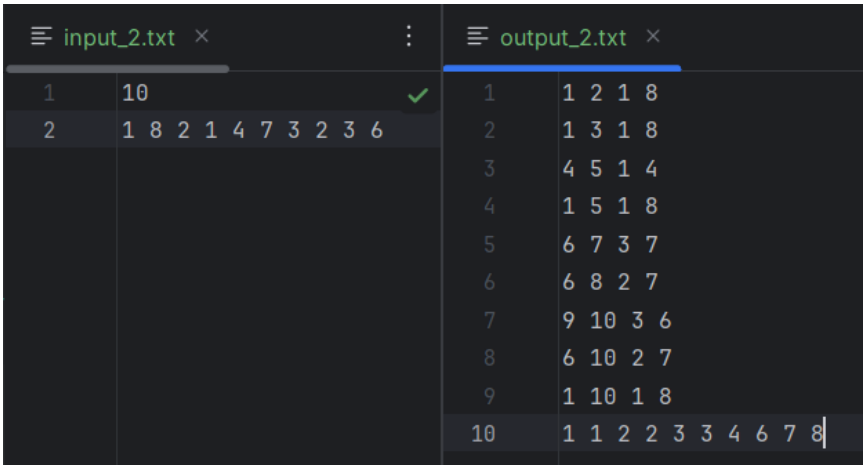
```
from time import time


def merge_sort(a, left, right, output_file):
    if left < right:
        mid = (left + right) // 2
        merge_sort(a, left, mid, output_file)
        merge_sort(a, mid + 1, right, output_file)
        merge(a, left, mid, right, output_file)


def merge(a, left, mid, right, output_file):
    a_l = a[left:mid + 1] + [10 ** 100]
    a_r = a[mid + 1:right + 1] + [10 ** 100]

    i, j = 0, 0
    for k in range(left, right + 1):
        if a_l[i] <= a_r[j]:
            a[k] = a_l[i]
            i += 1
        else:
            a[k] = a_r[j]
            j += 1

    output_file.write(f'{left + 1} {right + 1} {a[left]} {a[right]}\n')
```

```
t_start = time()
with open("input_2.txt") as f:
    n = int(f.readline())
    if not (1 <= n <= 10 ** 5):
        with open('output_2.txt', 'wt') as g:
            g.write('Введены некорректные данные')
        exit()

    a = list(map(int, f.readline().split()))
    for elem in a:
        if abs(elem) > 10 ** 9:
            with open('output_2.txt', 'wt') as g:
                g.write('Введены некорректные данные')
            exit()

    g =  open('output_2.txt', 'wt')
    g.write("")
    merge_sort(a, 0, len(a) - 1, g)
    g.write(' '.join([str(elem) for elem in a]))
    g.close()

t_stop = time()
if t_stop - t_start <= 2:
    print("Время выполнения:", t_stop - t_start, 'секунд')
else:
    print("Превышено время выполнения:", t_stop - t_start, 'секунд')
```

Текстовое объяснение решения:

1) Импорт модуля time для отслеживания времени работы программы.
2) Сохранение времени начала выполнения программы.
3) Считывание входных данных из файла input_1.txt.
4) Проверка корректности входных данных.
5) Вызов функции merge_sort, которой на вход подается сортируемый массив и левая и правая граница массива.
6) В функции merge_sort массив рекурсивно делится на два массива, пока массивы не будут состоять только из одного элемента, вызывается функция merge.
7) Функция merge соединяет два отсортированных подмассива в один сортированный массив и в файл вывода записывается левая и правая граница отсортированного массива и значение этих границ.
8) Отсортированный массив записывается в файл.
9) Выводится время работы алгоритма.

Пример работы кода:



Нижняя граница диапазона значений входных данных из текста задачи	Время выполнения: 0. 0009996891021728516 секунд
Пример из задачи	Время выполнения: 0.0010004043579101562 секунд
Верхняя граница диапазона значений входных данных из текста задачи	Время выполнения: 0. 48410654067993164 секунд

Вывод по задаче:
В решении задачи использованы переменные, операторы сравнения и if-else конструкции, цикл for, работа с файлами. Код считывает массив из файла, выполняет сортировку слиянием и сохраняет отсортированный массив в файл, а также сохраняет каждый вызов функции merge, где записывается левая и правая граница отсортированного подмассива и значения этих границ.

Задание №5. Представитель большинства.

Текст задачи:

5 задача. Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов $a_1, a_2, ...a_n$, и нужно проверить, содержит ли она элемент, который появляется больше, чем $n/2$ раз. Наивный метод это сделать:

```
Majority(A):
for i from 1 to n:
        current_element = a[i]
        count = 0
        for j from 1 to n:
                if a[j] = current_element:
                        count = count+1
        if count > n/2:
                return a[i]
return "нет элемента большинства"
```

Очевидно, время выполнения этого алгоритма квадратично. Ваша цель - использовать метод "Разделяй и властвуй"для разработки алгоритма проверки, содержится ли во входной последовательности элемент, который встречается больше половины раз, за время $O(n \log n)$.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \le n \le 10^5$) — число элементов в массиве. Во второй строке находятся n положительных целых чисел, по модулю не превосходящих 10^9, $0 \le a_i \le 10^9$.
- **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.

Код:

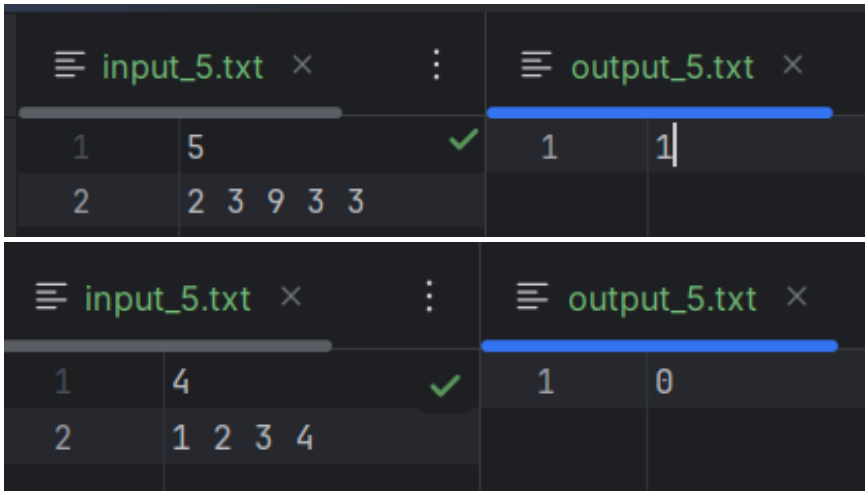
```python
from time import time


def popular_elem(a, start, end):
    if start == end:
        return a[end]
    mid = (start + end) // 2
    left = popular_elem(a, start, mid)
    right = popular_elem(a, mid + 1, end)

    if left == right:
        return left
    left_count = sum(1 for i in range(start, end + 1) if a[i] == left)
    right_count = sum(1 for i in range(start, end + 1) if a[i] == right)

    return left if left_count > right_count else right


t_start = time()
with open("input_5.txt") as f:
    n = int(f.readline())
    if not (1 <= n <= 10 ** 5):
        with open('output 5.txt', 'wt') as g:
            g.write('Введены некорректные данные')
        exit()

    a = list(map(int, f.readline().split()))
    for elem in a:
        if abs(elem) > 10 ** 9:
            with open('output_5.txt', 'wt') as g:
                g.write('Введены некорректные данные')
            exit()

    answ = popular_elem(a, 0, n - 1)
```

```
    with open('output_5.txt', 'wt') as g:
        if a.count(answ) > n / 2:
            g.write('1')
        else:
            g.write('0')

t_stop = time()
if t_stop - t_start <= 2:
    print("Время выполнения:", t_stop - t_start, 'секунд')
else:
    print("Превышено время выполнения:", t_stop - t_start, 'секунд')
```

Текстовое объяснение решения:

1) Импорт модуля time для отслеживания времени работы программы.
2) Сохранение времени начала выполнения программы.
3) Считывание входных данных из файла input_1.txt.
4) Проверка корректности входных данных.
5) Вызов функции popular_elem, которая принимает на вход массив, начальный и конечный индекс и рекурсивно находит самый часто встречающийся элемент в массиве.
6) Проверка, что полученный элемент встречается в массиве больше чем n / 2 раз.
7) Ответ записывается в файл.
8) Выводится время работы алгоритма.

Пример работы кода:



	Время выполнения
Нижняя граница диапазона значений входных данных из текста задачи	Время работы: 0.002001523971557617 секунд

Пример из задачи	Время работы: 0.0009987354278564453 секунд
Верхняя граница диапазона значений входных данных из текста задачи	Время работы: 0.2918057441711426 секунд

Вывод по задаче:
В решении задачи использованы переменные, операторы сравнения и if-else конструкции, цикл for, работа с файлами. Код считывает массив из файла, находит в нем самый популярный элемент, проверяет, что этот элемент встречается больше половины раз и записывает ответ в файл.

Вывод:
В данной лабораторной работе я практиковалась в работе с файлами, с логическими конструкциями, с модулем time, для оценки скорости работы алгоритмов, отработала подход «Разделяй и властвуй», работу со строками и списками, бинарный поиск. Научилась писать различные сортировки массивов.