

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 композиции
Вариант 5

Выполнил:
Артемов И. В.
К3141

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка слиянием	3
Задача №2. Сортировка слиянием+	7
Задача №7. Поиск максимального подмассива за линейное время	14
Дополнительные задачи	18
Задача №3. Число инверсий	18
Задача №4. Бинарный поиск	23
Задача №5. Представитель большинства	28
Вывод	33

Задачи по варианту

Задача №1. Сортировка слиянием

Текст задачи.

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 случайных 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 .

Листинг кода.

```
import tracemalloc
import time
from lab2.utils import open_file, write_file
t_start = time.perf_counter()
tracemalloc.start()

def merge(arr, left, mid, right):

    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

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

if __name__ == "__main__":
    n_str, m = open_file("../txtf/input.txt")
    n = int(n_str[0])
    if (1 <= n <= 2 * 10**4) and (all(abs(i) <= 10**9 for i in m)):
        merge_sort(m, 0, n - 1)
        write_file(" ".join(str(a) for a in m), "../txtf/output.txt")
    else:
        print('Введите корректные данные')

    print("Время работы: %s секунд" % (time.perf_counter() - t_start))
    print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()

```

Текстовое объяснение решения.

- 1) Считываем число элементов в массиве и сам массив из input.txt
- 2) Проверяем удовлетворяют ли полученные данные условию задачи.
Если нет, то просим пользователя ввести корректные данные
- 3) Определяем функцию merge для объединения отсортированных подмассивов:
 - Создаём два подмассива L и R, содержащих левую и правую части массива arr
 - Объединяем элементы из подмассивов обратно в arr, сортируя их по возрастанию:
 - Сравниваем текущие элементы из L и R и добавляем меньший в arr
 - После окончания сравнения добавляем оставшиеся элементы из L и R
- 4) Определяем функцию merge_sort для рекурсивной сортировки
 - Проверяем, если left < right (т.е., есть как минимум два элемента для сортировки)

- Вычисляем середину массива mid
 - Рекурсивно применяем merge_sort для левой и правой частей массива
 - Объединяем отсортированные части, вызывая merge
- 5) Отсортированный массив преобразуем в строку через пробелы и записываем результат в output.txt

Результат работы кода на максимальных и минимальных значениях:

The image shows three screenshots of a code editor with the following tabs: task1.py, input.txt, and output.txt.

Screenshot 1: The input.txt file contains two lines of input: 1 and -1000000000. The output.txt file is empty.

Screenshot 2: The input.txt file contains one line of input: -1000000000. The output.txt file is empty.

Screenshot 3: The input.txt file contains two lines of input: 20000 and a long list of 20 integers. The output.txt file contains the same two lines of input, with the second line being the sorted version of the input array.

```

1 20000
2 190404878 922225069 605487108 -100482751 -691973531 -826981630 70298
  634926947 -496627772 -376656213 -841258248 172070402 995671753 -444
  -465178792 490686580 403682065 559783798 -621336355 798801721 31153
  28701469 -70090966 -893721652 545851483 -607810053 931966771 -96362
  -570945395 264647884 -151541441 768906555 31937298 237032953 173342
  -128810251 10711019 -834068207 -661027487 -880762507 -75405129 6660
  448688895 -169159788 -101960326 -660508724 -378713896 212428797 821
  604098128 -882293593 -961223888 523249491 797569488 685403986 -7744
  906120153 804221146 -560248473 -396720420 -33581513 122680031 -5401
  -666219170 805730973 974080533 -763462127 -294851778 -23012446 -143
  350032978 -192085240 132883147 106434006 700802283 2773672 29334538
  
```

```

task1.py  input.txt  output.txt x
1  -999919772 -999858719 -999737703 -999648584 -999603152 -999575289 -9
   -998448396 -998162980 -998119422 -998114528 -998027805 -998024497 -
   -997134790 -997119039 -997111875 -996951250 -996778947 -996714288 -
   -996085349 -996034720 -995890853 -995824018 -995723081 -995457501 -
   -994583678 -994401883 -994397409 -994373870 -994339447 -994280137 -
   -992433422 -992347397 -992252639 -992111275 -991962685 -991794565 -
   -990921349 -990723600 -990676890 -990640733 -990628228 -990622518 -
   -989726179 -989711802 -989516818 -989195790 -989183357 -989164820 -
   -987990836 -987899608 -987763961 -987708292 -987577776 -987549343 -
   -986336408 -986259402 -986224487 -986114519 -985865948 -985818573 -
   -985151577 -984994617 -984944658 -984777647 -984745647 -984709950 -
   -983666385 -983582914 -983560866 -983539355 -983519271 -983496339 -

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00022191699827089906 секунд	14348 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.7021002499968745 секунд	2127886 байт

Вывод по задаче: мною был изучен алгоритм сортировки массива слиянием, а также выяснилось, что чем больше массив, тем дольше будет время выполнения и больше затраты памяти.

Задача №2. Сортировка слиянием+

Текст задачи.

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 ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .
- **Формат выходного файла (output.txt).** 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, $1 \leq I_f \leq I_l \leq n$). **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.
 - **Приведем небольшой пример:** отсортируем массив $[9, 7, 5, 8]$. Рекурсивная часть сортировки слиянием (процедура $\text{SORT}(A, L, R)$, где A — сортируемый массив, L — индекс начала области слияния, R — индекс конца области слияния) будет вызвана с $A = [9, 7, 5, 8]$, $L = 1$, $R = 4$ и выполнит следующие действия:
 - разделит область слияния $[1, 4]$ на две части, $[1, 2]$ и $[3, 4]$;
 - выполнит вызов $\text{SORT}(A, L = 1, R = 2)$:
 - * разделит область слияния $[1, 2]$ на две части, $[1, 1]$ и $[2, 2]$;
 - * получившиеся части имеют единичный размер, рекурсивные вызовы можно не делать;
 - * осуществит слияние, после чего A станет равным $[7, 9, 5, 8]$;
 - * выведет описание слияния: $I_f = L = 1$, $I_l = R = 2$, $V_f = A_L = 7$, $V_l = A_R = 9$.
 - выполнит вызов $\text{SORT}(A, L = 3, R = 4)$:
 - * разделит область слияния $[3, 4]$ на две части, $[3, 3]$ и $[4, 4]$;
 - * получившиеся части имеют единичный размер, рекурсивные вызовы можно не делать;
 - * осуществит слияние, после чего A станет равным $[7, 9, 5, 8]$;
 - * выведет описание слияния: $I_f = L = 3$, $I_l = R = 4$, $V_f = A_L = 5$, $V_l = A_R = 8$.
 - осуществит слияние, после чего A станет равным $[5, 7, 8, 9]$;
 - выведет описание слияния: $I_f = L = 1$, $I_l = R = 4$, $V_f = A_L = 5$, $V_l = A_R = 9$.

- 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.

- Пример:

input.txt	output.txt
10	1 2 1 8
1 8 2 1 4 7 3 2 3 6	3 4 1 2
	1 4 1 8
	5 6 4 7
	1 6 1 8
	7 8 2 3
	9 10 3 6
	7 10 2 6
	1 10 1 8
	1 1 2 2 3 3 4 6 7 8

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

Листинг кода.

```
import tracemalloc
import time
from lab2.utils import open_file, write_file
t_start = time.perf_counter()
tracemalloc.start()

def merge(arr, left, mid, right):

    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

    write_file(f"{left + 1} {right + 1} {arr[left]} {arr[right]}\n",
```

```

"../txtf/output.txt", mode='a')

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

if __name__ == "__main__":
    n_str, m = open_file("../txtf/input.txt")
    n = int(n_str[0])
    if (1 <= n <= 10**5) and (all(abs(i) <= 10**9 for i in m)):
        merge_sort(m, 0, n - 1)
        write_file(" ".join(str(a) for a in m), "../txtf/output.txt",
mode='a')
    else:
        print('Введите корректные данные')

    print("Время работы: %s секунд" % (time.perf_counter() - t_start))
    print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()

```

Текстовое объяснение решения.

- 1) Объявляем функцию `merge`, которая выполняет слияние двух отсортированных частей массива в один отсортированный массив
- 2) Проверяем удовлетворяют ли входные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
- 3) В файл записываются границы текущего подмассива (индексы `left + 1` и `right + 1`) и крайние значения после слияния (`arr[left]` и `arr[right]`)
- 4) Объявляем функцию `merge_sort`, которая выполняет рекурсивную сортировку слиянием
- 5) Результат сортировки записывается в файл `output.txt`

Результат работы кода на примере из текста задачи:

task2.py		input.txt	output.txt
1	10		
2	1 8 2 1 4 7 3 2 3 6		

```
task2.py  input.txt  output.txt x
1 1 2 1 8
2 1 3 1 8
3 4 5 1 4
4 1 5 1 8
5 6 7 3 7
6 6 8 2 7
7 9 10 3 6
8 6 10 2 7
9 1 10 1 8
10 1 1 2 2 3 3 4 6 7 8
```

Результат работы кода на максимальных и минимальных значениях:

```
task2.py  input.txt x  output.txt
1 1
2 -10000000000
```

```
task2.py  input.txt  output.txt x
1 -10000000000
```

```

task2.py  input.txt  output.txt
1 100000
2 -972582847 -25467689 -845621092 554839911 580009291 611895400 461865
  724424032 -601595507 200529032 -478564521 -970667503 833029696 -594
  -272126250 505776012 -879343429 -299922887 -364570243 623753210 224
    603856020 828298264 887325123 -969506156 -674320810 -371083165 -97
    802965567 -526848992 469104797 129390582 573621587 -102884163 -6509
    970630841 451602415 -160601183 -355846172 -525042730 -743230393 770
      841639250 380239786 461052968 346482827 -161913784 437710629 -8306
    832139291 -967067453 -729279412 -906207615 285731347 -20612523 -940
    166858511 552392856 -600396528 173923924 -469885690 290920070 81091
    301366264 840801092 862202773 -148257355 -318481853 -270595890 -461
    -954598025 393081698 -942005288 -791164058 757685853 770239714 -830

```

```

task2.py  input.txt  output.txt
⚠ The file size (4,3 MB) exceeds the configured limit (2,56 MB). Code insight features are not
1 1 2 -972582847 -25467689
2 3 4 -845621092 554839911
3 1 4 -972582847 554839911
4 5 6 580009291 611895400
5 5 7 461865225 611895400
6 1 7 -972582847 611895400
7 8 9 -578122897 977600731
8 8 10 -578122897 977600731
9 11 12 -91245831 572047564
10 11 13 -91245831 954705553

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00032733401167206466 секунд	13863 байт
Пример из задачи	0.000959457946009934 секунд	14400 байт
Верхняя граница диапазона значений входных данных из текста задачи	10.301613874966279 секунд	10487339 байт

Вывод по задаче: мною был изучен алгоритм сортировки массива слиянием, а также выяснилось, что чем больше массив, тем дольше будет время

выполнения и больше затраты памяти. Был также получен способ записи индексов граничных элементов и их значения.

Задача №7. Поиск максимального подмассива за линейное время

Текст задачи.

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 .

В этом случае у вас возможны 2 варианта тестирования: первый предполагает создание случайного массива чисел, аналогично задаче №1 (в этом случае формат входного и выходного файла смотрите там). Второй вариант - взять любые данные по акциям какой-либо компании, аналогично задаче №6.

Листинг кода.

```
import tracemalloc
import time
from lab2.utils import open_file, write_file
t_start = time.perf_counter()
tracemalloc.start()

def max_subarray(arr):
    max_sum = arr[0]
    current_sum = arr[0]
    start = end = 0
    temp_start = 0

    for i in range(1, len(arr)):
        if arr[i] > current_sum + arr[i]:
            current_sum = arr[i]
            temp_start = i
        else:
            current_sum += arr[i]

        if current_sum > max_sum:
            max_sum = current_sum
            start = temp_start
            end = i

    return arr[start:end + 1]

if __name__ == "__main__":
    n_list, arr = open_file("../txtf/input.txt")

    n = int(n_list[0])

    if 1 <= n <= 2 * 10**4 and (all(abs(i) <= 10**9 for i in arr)):
        write_file("", "../txtf/output.txt", mode="w")
        sub_arr = max_subarray(arr)
```

```

        write_file(sub_arr, "../txtf/output.txt")
    else:
        print('Введите корректные данные')

    print("Время работы: %s секунд" % (time.perf_counter() - t_start))
    print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()

```

Текстовое объяснение решения.

- 1) Проверяем удовлетворяют ли входные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
- 2) Объявляем функцию `max_subarray`, которая находит подмассив с наибольшей суммой в массиве `arr`. Она использует модификацию алгоритма Кадане, чтобы эффективно решить эту задачу:
 - `max_sum` – хранит наибольшую сумму подмассива, найденную на данный момент; `current_sum` – текущая сумма подмассива, которая обновляется на каждом шаге; `start` и `end` – индексы начала и конца подмассива с максимальной суммой; `temp_start` – временный индекс начала текущего подмассива, который обновляется при нахождении новой потенциально максимальной суммы
 - Далее цикл проходит по всем элементам начиная с индекса 1, так как `current_sum` и `max_sum` уже были инициализированы значением `arr[0]`
 - Если элемент `arr[i]` больше суммы `current_sum + arr[i]`, это означает, что начало нового подмассива может иметь большую сумму, чем продолжение текущего
 - В этом случае обновляем `current_sum` до `arr[i]` и устанавливаем `temp_start` на текущий индекс `i`
 - Иначе добавляем `arr[i]` к `current_sum`, продолжая текущий подмассив
 - Если `current_sum` больше, чем `max_sum`, обновляем `max_sum` и устанавливаем `start` и `end` как границы подмассива с максимальной суммой
 - Возвращаем подмассив `arr` от `start` до `end` включительно, так как это подмассив с наибольшей суммой
- 3) Вызываем `max_subarray` для поиска подмассива с наибольшей суммой, сохраняем его в `sub_arr` и записываем в `output.txt`

Результат работы кода на максимальных и минимальных значениях:

```
task7.py  input.txt  output.txt
1 1
2 -10000000000
```

```
task7.py  input.txt  output.txt
1 -10000000000
2
```

```
task7.py  input.txt  output.txt
1 20000
2 -605493159 -246210590 490789079 -474247206 119243682 -425619254 -173
-191993966 -539544017 823254592 -311541584 3099570 -200828839 -7440
-360652765 1104180 -374159303 991488586 431666210 -755598349 520167
961157750 589003997 837767409 -782257483 960789800 -770207432 -156
-954480614 813562590 -866171795 12604000 -526337322 491660673 15477
-695920595 -773189495 158552901 -517207713 -782612994 89730406 -237
233829513 -819344859 55224481 612902176 666248824 -83310047 8164027
516713816 -984418984 773161175 474089795 476108767 -877976848 -3983
```

```
task7.py  input.txt  output.txt
1 353274827 596664392 898455540 -63578098 587564271 -769218359 -409983
-455009617 692884086 417274282 -512348954 211409799 -337361770 -932
-330928027 890591480 369041328 143987364 994617730 -424365259 -509
-788027708 916236154 496274429 -389593005 826952756 -441315335 -794
168973865 878586906 488945761 -678225669 941052234 243155882 153470
260650967 -22530812 872876825 119540170 714336900 712178628 -656036
-602484499 -809429893 -581186511 -10223140 -506082258 -468475213 15
-842480013 120686776 752183398 -949982452 -434612158 -948011170 -76
214381177 601932899 784806327 -300460415 -927198623 413485311 49928
```


	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00027195800794288516 секунд	14188 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.09492137498455122 секунд	2127532 байт

Вывод по задаче: мною был изучен алгоритм поиска максимального подмассива за линейное время, а также выяснилось, что чем больше массив, тем дольше будет время выполнения и больше затраты памяти.

Дополнительные задачи

Задача №3. Число инверсий

Текст задачи.

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 отсортированном массиве число инверсий равно 0, а в массиве, отсортированном 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.

- Пример:

input.txt	output.txt
10 1 8 2 1 4 7 3 2 3 6	17

Листинг кода.

```
import tracemalloc
import time
from lab2.utils import open_file, write_file
t_start = time.perf_counter()
tracemalloc.start()

def merge_and_count(arr, temp_arr, left, mid, right):
    i = left
    j = mid + 1
    k = left
    inv_count = 0

    while i <= mid and j <= right:
        if arr[i] <= arr[j]:
            temp_arr[k] = arr[i]
            i += 1
        else:
            temp_arr[k] = arr[j]
            inv_count += (mid - i + 1)
            j += 1
        k += 1

    while i <= mid:
```

```

        temp_arr[k] = arr[i]
        i += 1
        k += 1

    while j <= right:
        temp_arr[k] = arr[j]
        j += 1
        k += 1

    for i in range(left, right + 1):
        arr[i] = temp_arr[i]

    return inv_count

def merge_sort_and_count(arr, temp_arr, left, right):
    inv_count = 0
    if left < right:
        mid = (left + right) // 2

        inv_count += merge_sort_and_count(arr, temp_arr, left, mid)
        inv_count += merge_sort_and_count(arr, temp_arr, mid + 1, right)

        inv_count += merge_and_count(arr, temp_arr, left, mid, right)

    return inv_count

if __name__ == "__main__":
    n_str, m = open_file("../txtf/input.txt")
    n = int(n_str[0])
    if (1 <= n <= 10**5) and (all(abs(i) <= 10**9 for i in m)):
        temp_arr = [0] * n
        result = merge_sort_and_count(m, temp_arr, 0, n - 1)
        write_file(result, "../txtf/output.txt")
    else:
        print('Введите корректные данные')

    print("Время работы: %s секунд" % (time.perf_counter() - t_start))
    print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()

```

Текстовое объяснение решения.

- 1) Проверяем удовлетворяют ли входные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
- 2) Объявляем функцию `merge_and_count`, которая объединяет две отсортированные части массива `arr` и одновременно подсчитывает количество инверсий
 - `i` – текущий индекс левой половины массива; `j` – текущий индекс правой половины массива; `k` – текущий индекс массива `temp_arr`, куда записываются отсортированные элементы; `inv_count` – счетчик инверсий, который увеличивается, если найдено расположение, нарушающее порядок сортировки

- Пока индексы находятся в пределах левой и правой половины массива
 - Если $arr[i] \leq arr[j]$, то копируем элемент $arr[i]$ в $temp_arr[k]$, так как он не нарушает порядок сортировки
 - Если $arr[i] > arr[j]$, то копируем элемент $arr[j]$ в $temp_arr[k]$, увеличивая счетчик инверсий на $(mid - i + 1)$, так как все элементы от $arr[i]$ до $arr[mid]$ образуют инверсии с $arr[j]$
 - Индекс k увеличивается в обоих случаях
 - Копируем оставшиеся элементы из левой половины массива
 - Копируем оставшиеся элементы из правой половины массива
 - Копируем отсортированные элементы из $temp_arr$ обратно в arr
 - Возвращаем количество инверсий, найденное в текущем слиянии
- 3) Объявляем функцию `merge_sort_and_count`, которая рекурсивно разделяет массив и считает инверсии, используя алгоритм “Разделяй и властвуй”
 - 4) Создаём временный массив `temp_arr` для хранения промежуточных данных
 - 5) Вызываем `merge_sort_and_count` для массива `m`, чтобы получить количество инверсий, и сохраняем его в `result` и записываем его в `output.txt`

Результат работы кода на примере из текста задачи:

task3.pyinput.txt ×output.txt

110

21 8 2 1 4 7 3 2 3 6

task3.py input.txt output.txt ×	
1	17

Результат работы кода на максимальных и минимальных значениях:

task3.py input.txt × output.txt	
1	1
2	-1000000000

task3.py input.txt output.txt ×	
1	0

task3.py input.txt × output.txt	
1	100000
2	656021552 711656109 -431214859 383686683 262514561 264088286 -588632 104550595 496949743 166568588 667622669 848320524 358097262 337391 883393464 -907010870 -995898838 250500035 161067002 637741606 -3791 -514239473 156901103 -714323657 -251880534 -777988819 -180778417 29 -892249685 27971720 -324564075 101697844 142096959 549013309 376856 -840843649 -375163570 -715446685 -752702132 -637307477 997211532 35

```
task3.py  input.txt  output.txt x
1 2490224434
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00027833401691168547 секунд	13863 байт
Пример из задачи	0.0004927919944748282 секунд	14400 байт
Верхняя граница диапазона значений входных данных из текста задачи	5.460720582981594 секунд	10486703 байт

Вывод по задаче: мною был изучен алгоритм подсчёта инверсий в последовательности, используя алгоритм “Разделяй и властвуй”

Задача №4. Бинарный поиск

Текст задачи.

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 $a_0 < a_1 < \dots < a_{n-1}$ 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, \dots, 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	2 0 -1 0 -1
1 5 8 12 13	
5	
8 1 23 1 11	

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.

Листинг кода.

```
import tracemalloc
import time
from lab2.utils import open_file, write_file
t_start = time.perf_counter()
tracemalloc.start()

def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1

if __name__ == "__main__":
    data_n, data_k = open_file("../txtf/input.txt")

    n = int(data_n[0])
    a = data_n[1:]
    k = int(data_k[0])
    b = data_k[1:]

    if 1 <= n <= 10**5 and 1 <= k <= 10**5 and min(a) >= 1 and \
```

```

min(b) >= 1 and max(a) <= 10**9 and max(b) <= 10**9:
    write_file("", "../txtf/output.txt", mode="w")
    for i in b:
        result = binary_search(a, i)
        write_file(f"{result} ", "../txtf/output.txt", mode='a')
    else:
        print('Введите корректные данные')

print("Время работы: %s секунд" % (time.perf_counter() - t_start))
print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")
tracemalloc.stop()

```

Текстовое объяснение решения.

- 1) Проверяем удовлетворяют ли входные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
- 2) Определяем функцию `binary_search`, которая реализует бинарный поиск, который ищет целевое значение (`target`) в отсортированном массиве (`arr`). Если элемент найден, возвращается его индекс, иначе – 1
 - `left` – начальный индекс массива (0); `right` – последний индекс массива
 - Пока диапазон поиска допустим (левая граница не превышает правую), продолжаем поиск
 - `mid` – индекс среднего элемента, рассчитанный как целочисленное деление суммы границ
 - Если средний элемент равен `target`, возвращаем его индекс `mid`
 - Если средний элемент меньше `target`, сдвигаем левую границу вправо
 - Если средний элемент больше `target`, сдвигаем правую границу влево
 - Возврат -1, если элемент не найден
- 3) Для каждого элемента `i` из массива `b` вызываем `binary_search`, передавая массив `a` и текущий элемент `i` как целевое значение
- 4) Результат `result` записываем в `output.txt`

Результат работы кода на примере из текста задачи:


```
task4.py  input.txt  output.txt
1 5 1 5 8 12 13
2 5 8 1 23 1 11
```

```
task4.py  input.txt  output.txt
1 2 0 -1 0 -1
```

Результат работы кода на максимальных и минимальных значениях:

```
task4.py  input.txt  output.txt
1 1 1
2 1 1
```

```
task4.py  input.txt  output.txt x
1 0
```

```
task4.py  input.txt x  output.txt
1 10000 861821054 989435362 844639381 642149861 756467677 616298378 4
  485577072 688033919 255144142 418016634 584527426 693214448 9230779
  772770834 505461831 408669276 513614704 308464010 910761908 7811194
  416137039 43473190 241483510 427572504 584295914 48362188 952868930
  349626936 934805156 638076357 962007070 254784389 958178758 154939
  955352981 121834577 928879536 576411851 788482298 892395217 6506531
  725431297 432591329 164838021 244373676 553287265 236757033 9310799
  468739676 259928592 118025321 826549361 803306521 402848846 5220968
  11944247 590494539 702836902 456079627 558593214 167341367 71894245
  564377111 824349564 117651835 103723773 919647056 703568688 1748556
  955206251 420091644 352524307 57143844 555401459 929614659 47752273
```

```
task4.py  input.txt  output.txt x
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

	Время выполнения	Затраты памяти
Нижняя граница	0.0002555839600972831	14226 байт

диапазона значений входных данных из текста задачи	секунд	
Пример из задачи	0.00047691597137600183 секунд	14256 байт
Верхняя граница диапазона значений входных данных из текста задачи	8.347801583004184 секунд	15776801 байт

Вывод по задаче: мною был изучен алгоритм бинарного поиска.

Задача №5. Представитель большинства

Текст задачи.

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов a_1, a_2, \dots, 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 ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n положительных целых чисел, по модулю не превосходящих 10^9 , $0 \leq a_i \leq 10^9$.
- **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.
- Пример 1:

input.txt	output.txt
5 2 3 9 2 2	1

Число "2" встречается больше $5/2$ раз.

- Пример 2:

input.txt	output.txt
4 1 2 3 4	0

Нет элемента, встречающегося больше $n/2$ раз.

Листинг кода.

```
import tracemalloc
import time
from lab2.utils import open_file, write_file
t_start = time.perf_counter()
tracemalloc.start()

def count_occurrences(arr, num, left, right):
```

```

count = 0
for i in range(left, right):
    if arr[i] == num:
        count += 1
return count

def majority_element_rec(arr, left, right):

    if left == right - 1:
        return arr[left]

    mid = (left + right) // 2
    left_majority = majority_element_rec(arr, left, mid)
    right_majority = majority_element_rec(arr, mid, right)

    if left_majority == right_majority:
        return left_majority

    left_count = count_occurrences(arr, left_majority, left, right)
    right_count = count_occurrences(arr, right_majority, left, right)

    if left_count > (right - left) // 2:
        return left_majority
    elif right_count > (right - left) // 2:
        return right_majority
    else:
        return None

if __name__ == "__main__":
    n_list, arr = open_file("../txtf/input.txt")

    n = int(n_list[0])

    if 1 <= n <= 10**5 and (all(abs(i) <= 10**9 for i in arr)):
        write_file("", "../txtf/output.txt", mode="w")

        majority_element = majority_element_rec(arr, 0, n)
        if majority_element is not None:
            if count_occurrences(arr, majority_element, 0, n) > n // 2:
                write_file('1\n', "../txtf/output.txt", mode="a")
            else:
                write_file('0\n', "../txtf/output.txt", mode="a")
        else:
            write_file('0\n', "../txtf/output.txt", mode="a")
    else:
        print('Введите корректные данные')

    print("Время работы: %s секунд" % (time.perf_counter() - t_start))
    print("Затрачено памяти:", tracemalloc.get_traced_memory()[1], "байт")
    tracemalloc.stop()

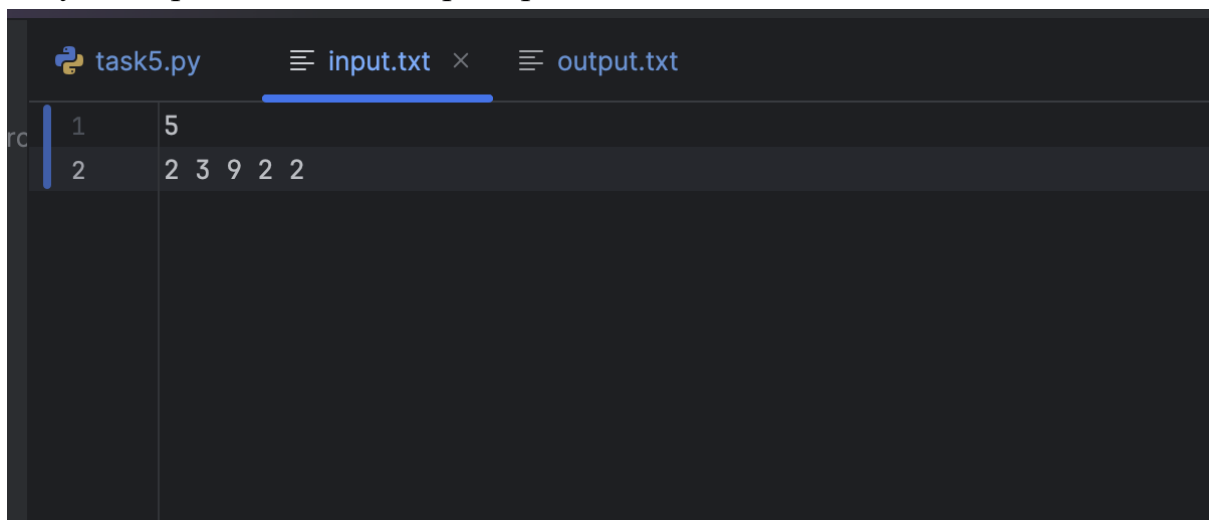
```

Текстовое объяснение решения.

- 1) Проверяем удовлетворяют ли входные данные условию задачи.
Если нет, то просим пользователя ввести корректные данные

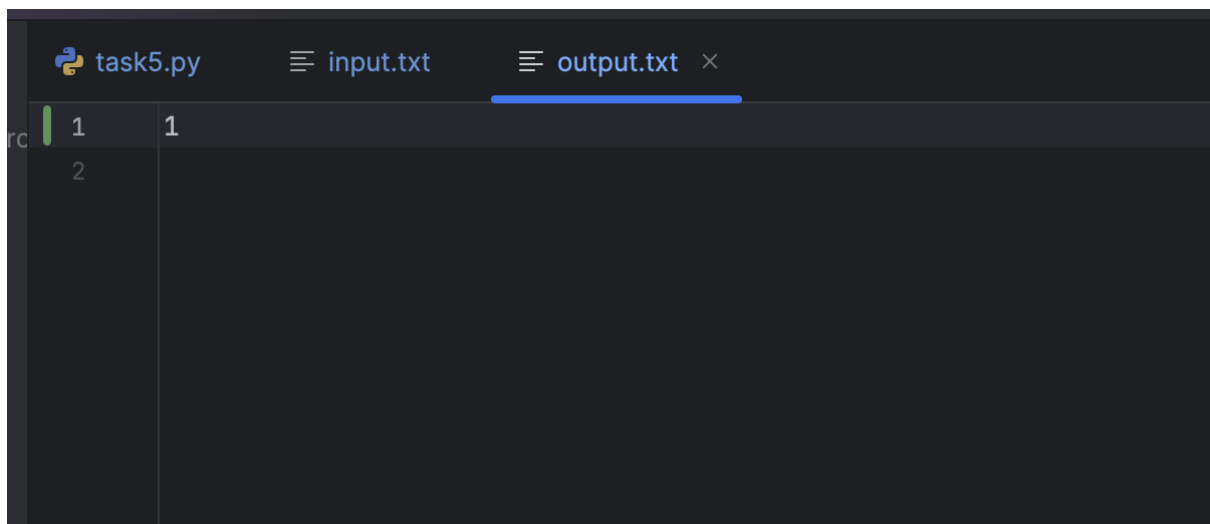
- 2) Объявляем функцию `count_occurrences`, которая подсчитывает, сколько раз число `num` встречается в массиве `arr` в пределах заданного диапазона индексов `[left, right)`
 - `count` – переменная для подсчета вхождений
 - Цикл `for` проходит по элементам массива в заданном диапазоне и увеличивает `count`, если элемент равен `num`
 - Возвращается итоговое значение `count`
- 3) Объявляем функцию `majority_element_rec`, которая рекурсивно определяет, есть ли элемент, встречающийся более чем в половине элементов массива `arr` в пределах диапазона `[left, right)`. Она использует метод “Разделяй и властвуй”
- 4) Вызывается функция `majority_element_rec` для поиска элемента большинства в массиве `arr`
- 5) Если `majority_element` не `None`, проверяем, действительно ли он встречается более чем в половине массива:
 - Если да, записываем 1 в `output.txt`
 - Если нет, записываем 0
- 6) Если `majority_element` оказался `None`, записываем 0

Результат работы кода на примере из текста задачи:



The screenshot shows a code editor with three tabs: `task5.py`, `input.txt`, and `output.txt`. The `input.txt` tab is active and displays the following content:

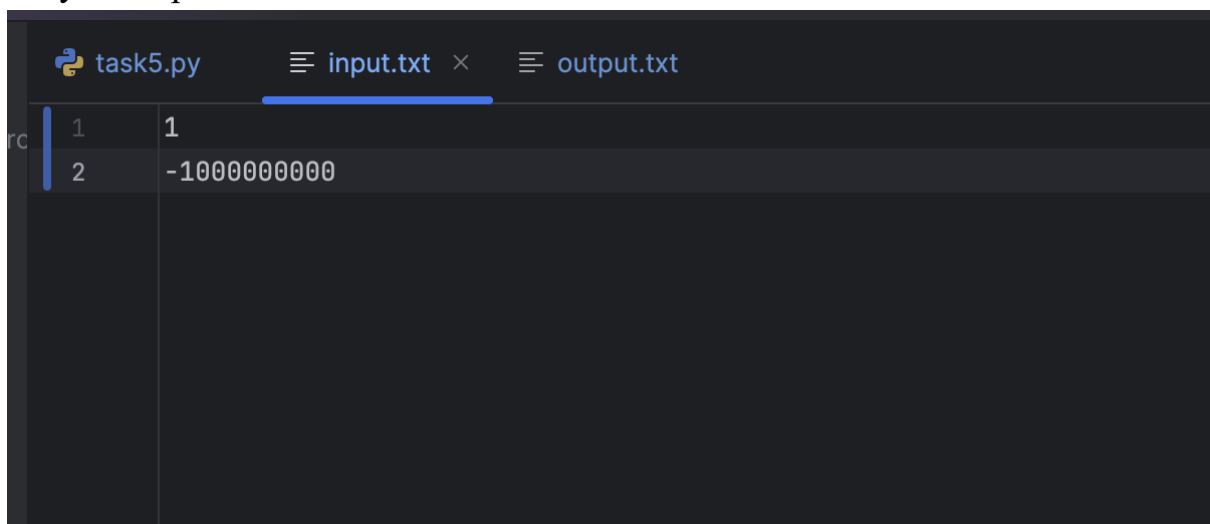
1	5
2	2 3 9 2 2



The screenshot shows a code editor with three tabs: 'task5.py', 'input.txt', and 'output.txt'. The 'output.txt' tab is active. The editor displays a single test case with input '1' and output '1'.

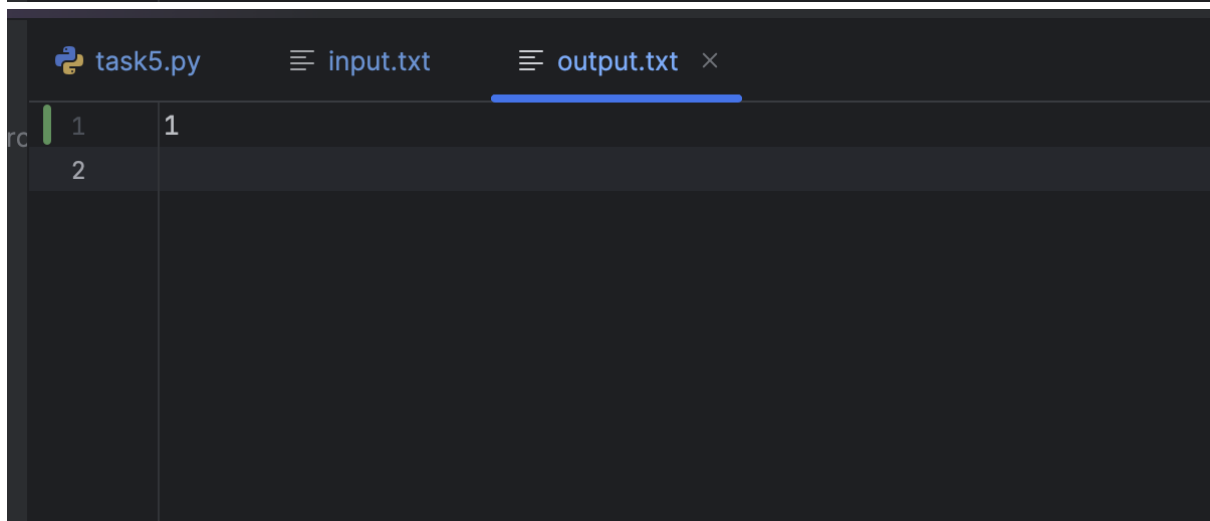
Input	Output
1	1

Результат работы кода на максимальных и минимальных значениях:



The screenshot shows a code editor with three tabs: 'task5.py', 'input.txt', and 'output.txt'. The 'output.txt' tab is active. The editor displays two test cases: input '1' with output '1', and input '2' with output '-10000000000'.

Input	Output
1	1
2	-10000000000



The screenshot shows a code editor with three tabs: 'task5.py', 'input.txt', and 'output.txt'. The 'output.txt' tab is active. The editor displays a single test case with input '1' and output '1'.

Input	Output
1	1

The top screenshot shows the 'input.txt' file with a list of integers. The first line is '100000'. The second line contains a long sequence of integers: '215137964 349566914 678554869 -366379263 -605999798 973491289 54213 89479081 218067383 -773526008 -160169433 889590368 -286996802 -93 526716684 780151965 -370150705 470108074 -904382160 -482068596 -29 -19402226 -132221341 -974679678 119828793 -549043597 311994233 639 977756706 -46813446 88090235 976665842 -599460888 -604336051 -5213 -664961951 -39173454 -276646423 677827180 -525168771 277983790 588 -404000341 -908632294 47445851 266643044 -822410309 37275292 67312 608949287 -500270946 -882264865 -792804888 -624383794 773963765 -6 -130797316 -988515161 272789084 -322501413 591231655 908796347 -172'. The bottom screenshot shows the 'output.txt' file with the number '0' on the first line and an empty second line.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00026537501253187656 секунд	14348 байт
Пример из задачи	0.0002447910374030471 секунд	14346 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.801410041982308 секунд	10487189 байт

Вывод по задаче: мною был изучен алгоритм проверки, содержится ли во входной последовательности элемент, который встречается больше половины раз, используя метод “Разделяй и властвуй”

Вывод

В ходе лабораторной работы были изучены алгоритм сортировки слиянием и метод “Разделяй и властвуй”. Был проведён анализ работы алгоритмов на максимальных и минимальных значениях.