

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

Отчет по лабораторной работе №3  
по курсу «Алгоритмы и структуры данных»  
Тема: Быстрая сортировка. Сортировка за линейное время  
Вариант 3

Выполнила:  
Блинова П. В.  
К3139 (номер группы)

Проверил:  
Афанасьев А. В.

Санкт-Петербург  
2024 г.

# Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка вставкой	3
Задача №2. Сортировка слиянием+	7
Задача №5. Представитель большинства	12
Дополнительные задачи	17
Задача №4. Бинарный поиск	17
Задача №3. Число инверсий	20
Задача №10. ★	24
Вывод	30

# Задачи по варианту

## Задача №1. Сортировка вставкой

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 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.

Листинг кода.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_1/tests/output.txt", 'w')

def merge(left, right):
    list_merged = list()
    left_i = right_i = 0

    while left_i < len(left) and right_i < len(right):
        if left_i < len(left) and right_i < len(right):
            if left[left_i] < right[right_i]:
                list_merged.append(left[left_i])
                left_i += 1
            else:
                list_merged.append(right[right_i])
```

```

        right_i += 1

    while left_i < len(left):
        list_merged.append(left[left_i])
        left_i += 1

    while right_i < len(right):
        list_merged.append(right[right_i])
        right_i += 1

    return list_merged

def merge_sort(list_arr):
    if len(list_arr) <= 1:
        return list_arr

    middle = len(list_arr) // 2
    left_list = merge_sort(list_arr[:middle])
    right_list = merge_sort(list_arr[middle:])
    return merge(left_list, right_list)

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_1/tests/input.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 2 * (10 ** 4):
        nums = list(map(int, list(file[1].split(' '))))
        if all([abs(n) <= 10 ** 9 for x in nums]):
            file_output.write(' '.join(map(str, merge_sort(nums))))
        else:
            print('Введите другое число')
    else:
        print('Неверный ввод данных')

file_output.close()

```

### Текстовое объяснение решения.

Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю вторую строку в файле по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. Записываю результат в переменную nums. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file\_output результат функции merge\_sort, делая с помощью map() все элементы строками, и объединяю результат в строку через join. В ином случае прошу ввести корректные данные. Закрываю файл.

#### Функция merge:

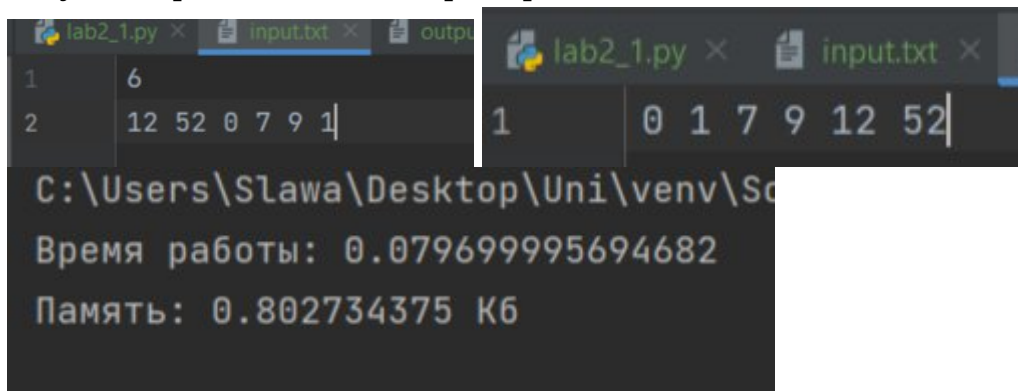
Функция отвечает за объединение двух отсортированных массивов в один отсортированный массив. list\_merged – это список, в который будут добавляться элементы из двух входящих списков left и right. Переменные left\_i и right\_i используются как индексы для отслеживания текущего элемента в каждом из списков. В цикле выполняется слияние: сравниваются текущие элементы из left и right (по индексам left\_i и right\_i), если элемент из left меньше, он добавляется в list\_merged, и индекс left\_i увеличивается. В противном случае элемент из right добавляется в list\_merged, и индекс right\_i увеличивается. Цикл продолжается, пока не будет достигнут конец одного из списков. После завершения основного цикла один из списков может всё ещё иметь некоторые

элементы. Эти циклы добавляют оставшиеся элементы из списка left или right в list\_merged. Функция возвращает объединённый отсортированный список.

Функция *merge\_sort*:

Функция *merge\_sort* выполняет сам алгоритм сортировки слиянием. Если длина массива *list\_arr* меньше или равна 1, он уже отсортирован, и функция возвращает его без изменений. Массив делится пополам, рассчитывается индекс *middle*. Затем рекурсивно вызывается функция *merge\_sort* для левой и правой половин. В результате вы получаете два отсортированных подмассива: *left\_list* и *right\_list*. Таким образом, сортировка слиянием — это алгоритм «разделяй и властвуй»: разделяет массив на две половины, рекурсивно сортирует обе половины и объединяет отсортированные половины в один отсортированный массив.

Результат работы кода на примерах из текста задачи:



The screenshot shows a code editor with two tabs: 'lab2\_1.py' and 'input.txt'. The 'input.txt' tab is active, displaying two lines of input data. Line 1 contains the number '6'. Line 2 contains the array '12 52 0 7 9 1'. Below the input, the output is shown as '0 1 7 9 12 52'. At the bottom of the editor, the execution time and memory usage are displayed: 'Время работы: 0.079699995694682' and 'Память: 0.802734375 Кб'.

Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab2.task_1.src.lab2_1 import merge_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = merge_sort(input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

Максимальные значения

```

import time
import tracemalloc
import random
from asd_itmo.lab2.task_1.src.lab2_1 import merge_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(2 * (10 ** 4))]

result = merge_sort(input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()

```

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений входных данных из текста задачи	0.01189	0.386
Пример из задачи	0.07969	0.8027
Верхняя граница диапазона значений входных данных из текста задачи	0.7594	1099.3925

Вывод по задаче:

В ходе работы над задачей мной был изучен способ сортировки вставкой. Была изучена эффективность этого метода.

## 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:  $l_f$ ,  $l_r$ ,  $v_f$ ,  $v_r$ , где  $l_f$  — индекс начала области слияния,  $l_r$  — индекс конца области слияния,  $v_f$  — значение первого элемента области слияния,  $v_r$  — значение последнего элемента области слияния.
  - Все индексы начинаются с единицы (то есть,  $1 \leq l_f \leq l_r \leq n$ ). **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.

Листинг кода.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_2/tests/output.txt", 'w')

def merge_output(arr, temp_arr, left, mid, right, res):
    left_i = left
    right_i = mid + 1
    merged_i = left

    while left_i <= mid and right_i <= right:
        if arr[left_i] <= arr[right_i]:
            temp_arr[merged_i] = arr[left_i]
            left_i += 1
        else:
            temp_arr[merged_i] = arr[right_i]
            right_i += 1
        merged_i += 1

    while left_i <= mid:
        temp_arr[merged_i] = arr[left_i]
        left_i += 1
        merged_i += 1

    while right_i <= right:
        temp_arr[merged_i] = arr[right_i]
        right_i += 1
        merged_i += 1

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

    if (right - left + 1) > 1:
        res.append(f"{left + 1} {right + 1} {arr[left]} {arr[right]}")
```

```

def merge_sort_output(list_arr, temp, left_i, right_i, res):
    if left_i < right_i:
        middle = (left_i + right_i) // 2

        merge_sort_output(list_arr, temp, left_i, middle, res)
        merge_sort_output(list_arr, temp, middle + 1, right_i, res)
        merge_output(list_arr, temp, left_i, middle, right_i, res)

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_2/tests/input.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 2 * (10 ** 4):
        nums = list(map(int, list(file[1].split(' '))))
        if all([abs(n) <= 10 ** 9 for x in nums]):
            res = list()
            merge_sort_output(nums, [0] * n, 0, n - 1, res)
            for num_line in res:
                file_output.write(num_line + "\n")
            file_output.write(" ".join(map(str, nums)))
        else:
            print('Введите другое число')
    else:
        print('Неверный ввод данных')

file_output.close()

```

### Текстовое объяснение решения.

Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю вторую строку в файле по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. Записываю результат в переменную nums. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file\_output результат функции merge\_sort, делая с помощью map() все элементы строками, и объединяю результат в строку через join. В ином случае прошу ввести корректные данные. Закрываю файл.

### Функция merge и merge\_sort:

Функции аналогичны первому заданию. Добавочным параметром является массив res, который выводится в файл output.txt. Если слияние происходит для больше, чем одного элемента (т.е. длина подмассива больше 1), записываем информацию о слиянии — индексы начала и конца, а также значения первого и последнего элемента.

Результат работы кода на примерах из текста задачи.



```
lab2_3.py x lab2_1.py x lab2_2.py x
1 10
2 1 8 2 1 4 7 3 2 3 6
```

```
lab2_1.py x lab2_2.py x
1 2 1 8
1 3 1 8
4 5 1 4
1 5 1 8
6 7 3 7
6 8 2 7
9 10 3 6
6 10 2 7
1 10 1 8
1 1 2 2 3 3 4 6 7 8
```

```
test_example (1) x
C:\Users\Slawa\Desktop\Uni\venv\Scr
Время работы: 0.16679998952895403
Память: 1.44921875 K6
```

Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab2.task_2.src.lab2_2 import merge_sort_output

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = merge_sort_output(input_arr, [0] * len(input_arr), 0, len(input_arr) - 1, list())
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' K6')
tracemalloc.stop()
```

```
test_min (1) x
C:\Users\Slawa\Desktop\Uni\venv\Scr
Время работы: 0.01479999627918005
Память: 0.439453125 K6
```

Максимальные значения

```
import time
import tracemalloc
import random
from asd_itmo.lab2.task_2.src.lab2_2 import merge_sort_output

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(10 ** 5)]
result = merge_sort_output(input_arr, [0] * len(input_arr), 0, len(input_arr)
```

```
- 1, list())
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

The screenshot shows a terminal window titled "Run: test\_max (1)". The command being executed is `C:\Users\Slawa\Desktop\Uni\venv\Sc`. The output displayed is:

```
Время работы: 6.019005299996934
Память: 13145.369140625 Кб
```

6

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений входных данных из текста задачи	0.0147	0.4394
Пример из задачи	0.166	1.449
Верхняя граница диапазона значений входных данных из текста задачи	6.019	13145.369

Вывод по задаче:

В ходе работы над задачей мной был изучен способ сортировки слиянием с выводом результата в файл.

## 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:

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

Листинг кода.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_5/tests/output.txt", 'w')

def find_majority(list_arr, left_i, right_i):
    if left_i == right_i:
        return list_arr[left_i]

    middle_arr = (left_i + right_i) // 2
    left = find_majority(list_arr, left_i, middle_arr)
    right = find_majority(list_arr, middle_arr + 1, right_i)

    counter_left, counter_right = 0, 0
    for i in range(left_i, right_i + 1):
        if list_arr[i] == right:
            counter_right += 1
        elif list_arr[i] == left:
            counter_left += 1
    return left if counter_left > counter_right else right

def majority(list_arr):
    result = find_majority(list_arr, 0, len(list_arr) - 1)
    if result:
        if list_arr.count(result) > n / 2:
            return 1
    return 0

with
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_5/tests/input_2.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 10 ** 5:
```

```

nums = list(map(int, list(file[1].split(' '))))
if all([abs(n) <= 10 ** 9 for x in nums]):
    file_output.write(str(majority(nums)))
else:
    print('Введите другое число')
else:
    print('Неверный ввод данных')

file_output.close()

```

### Текстовое объяснение решения.

Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю вторую строку в файле по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. Записываю результат в переменную nums. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file\_output результат функции majority. В ином случае прошу ввести корректные данные. Закрываю файл.

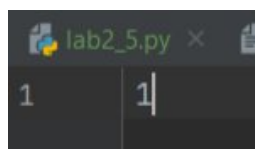
#### Функция find\_majority:

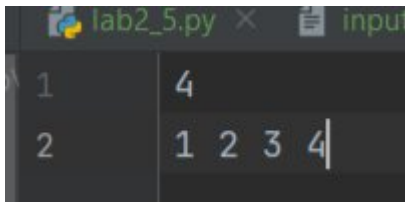
Эта функция рекурсивно ищет элемент, который может быть большинством в подмассиве. list\_arr — исходный массив, в котором будет происходить поиск. left\_i — начальный индекс подмассива для рекурсивного вызова. right\_i — конечный индекс подмассива для рекурсивного вызова. Если left\_i и right\_i совпадают, это означает, что мы достигли элемента, который не может быть разделен. В этом случае возвращается текущий элемент массива как потенциальный кандидат на большинство. Массив делится пополам и рекурсивно обрабатываются обе половины для поиска потенциальных кандидатов на большинство. Результаты сохраняются в переменных left и right. После того как мы получили два потенциальных кандидата (left) и (right), осуществляется подсчет их вхождений в текущем подмассиве, определяемом индексами left\_i и right\_i. Функция возвращает тот элемент, который встречается чаще в рассматриваемом подмассиве.

#### Функция majority:

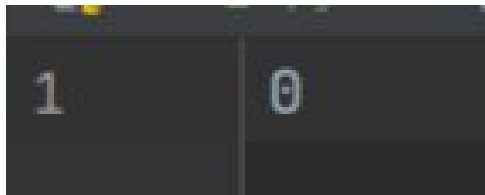
Эта функция иницирует процесс поиска и проверяет, является ли найденный элемент действительным большинством. Рекурсивно вызывается функция для обработки всего массива (list\_arr) от 0 до len(list\_arr) - 1. Если нашли элемент, затем проверяется, встречается ли он больше, чем (n/2) раз с использованием метода .count(). Если да, возвращается 1, что подразумевает наличие большинства. Если элемент не найден, или он не является большинством, возвращается 0. Таким образом, этот алгоритм имеет время выполнения ( $O(n \log n)$ ). Он делит подмассив на две части и затем использует линейный проход для подсчета, сколько раз каждый из кандидатов встречается в текущем диапазоне.

Результат работы кода на примерах из текста задачи.





1	4
2	1 2 3 4



1	0

```
C:\Users\Slawa\Desktop\Uni\venv\Scr
Время работы: 0.07530000220867805
Память: 0.908203125 Кб
```

Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab2.task_5.src.lab2_5 import majority

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = majority(input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

```
C:\Users\Slawa\Desktop\Uni\venv\Scrip
Время работы: 0.016099998902063817
Память: 0.38671875 Кб
```

Максимальные значения

```
import time
import tracemalloc
import random
from asd_itmo.lab2.task_5.src.lab2_5 import majority

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(10 ** 5)]

result = majority(input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

```
C:\Users\Slawa\Desktop\Uni\venv\Sc
Время работы: 2.0414853000000204
Память: 3708.51171875 Кб
```

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений входных данных из текста задачи	0.016	0.386
Пример из задачи	0.0753	0.9082
Верхняя граница диапазона значений входных данных из текста задачи	2.0414	3708.5117

Вывод по задаче:

В ходе работы над задачей мной был получен способ алгоритма проверки, содержится ли во входной последовательности элемент, который встречается больше половины раз, за время  $O(n \log n)$ .

# Дополнительные задачи

## 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

Листинг кода.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_4/tests/output.txt", 'w')

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

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_4/tests/input.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    nums = list(map(int, list(file[1].split(' '))))
    k = int(file[2])
    list_search = list(map(int, list(file[3].split(' '))))
    if 1 <= n <= 10 ** 5 and 1 <= k <= 10 ** 5:
        nums = list(map(int, list(file[1].split(' '))))
        if all([abs(n) <= 10 ** 9 for x in nums]) and all([abs(n) <= 10 ** 9
for x in list_search]):
            result = [binary_search(nums, number) for number in list_search]
            file_output.write(' '.join(map(str, result)))
        else:
            print('Введите другое число')
    else:
        print('Неверный ввод данных')
```

```
file_output.close()
```

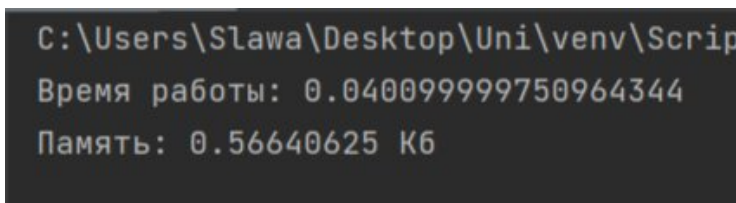
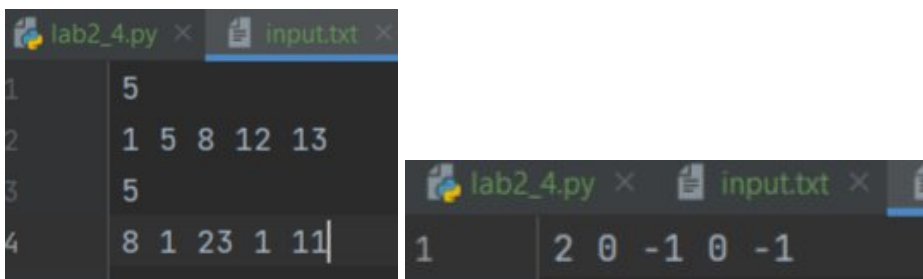
### Текстовое объяснение решения.

Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. В переменную nums числа, в переменную k — количество чисел, которые нужно найти. В переменную list\_search — числа, которые нужно найти. Дальше проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю переменную nums по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file\_output результат функции binary\_search делая с помощью map() все элементы строками, и объединяю результат в строку через join. В ином случае прошу ввести корректные данные. Закрываю файл.

Функция *binary\_search*:

Параметрами функции являются: list\_arr — отсортированный массив (или список), в котором мы ищем элемент, — number — значение, которое мы ищем в массиве. Создаются два индекса: left указывает на начало массива (0), а right указывает на конец массива (длина массива минус 1). Цикл продолжает работать до тех пор, пока индекс left не станет больше индекса right (это значит, что все элементы были проверены и число не найдено). Алгоритм работает по принципу "разделяй и властвуй" и на каждом шаге делит массив пополам, что позволяет значительно сократить количество элементов, которые необходимо проверить для нахождения искомого значения. Устанавливаются два указателя: один на начало (левый индекс) массива, другой — на конец (правый индекс). На каждой итерации вычисляется средний индекс массива. Если средний элемент равен искомому значению, поиск завершается с успехом. Если искомое значение меньше среднего элемента, поиск продолжается в левой половине массива. Если искомое значение больше среднего элемента, поиск продолжается в правой половине массива. Процесс повторяется, пока левый указатель не превысит правый.

Результат работы кода на примерах из текста задачи.



Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab2.task_4.src.lab2_4 import binary_search
```



```

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]
search_arr = [0]

result = [binary_search(input_arr, number) for number in search_arr]
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()

```

```

Время работы: 0.02040000254055485
Память: 0.423828125 Кб

```

Максимальные значения

```

import time
import tracemalloc
from asd_itmo.lab2.task_4.src.lab2_4 import binary_search

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [i for i in range(10 ** 5)]
search_arr = [i for i in range(10 ** 5, 0, -1)]

result = [binary_search(input_arr, number) for number in search_arr]
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()

```

```

C:\Users\Slawa\Desktop\Uni\venv\Sc
Время работы: 2.8562414000007266
Память: 10528.9921875 Кб

```

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений входных данных из текста задачи	0,02	0,4238
Пример из задачи	0.04	0.566
Верхняя граница диапазона значений входных данных из текста задачи	2,856	10528,99

Вывод по задаче:

В ходе работы над задачей мной был изучен способ бинарного поиска. Была изучена эффективность этого метода.

## 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.

Листинг кода.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_3/tests/output.txt", 'w')

def merge_count(arr, temp_arr, left, mid, right):
    inversion_counter = 0

    left_i = left
    right_i = mid + 1
    merged_i = left

    while left_i <= mid and right_i <= right:
        if arr[left_i] <= arr[right_i]:
            temp_arr[merged_i] = arr[left_i]
            left_i += 1
        else:
            temp_arr[merged_i] = arr[right_i]
            inversion_counter += (mid - left_i + 1)
            right_i += 1
        merged_i += 1

    while left_i <= mid:
        temp_arr[merged_i] = arr[left_i]
        left_i += 1
        merged_i += 1

    while right_i <= right:
        temp_arr[merged_i] = arr[right_i]
        right_i += 1
        merged_i += 1
```

```

        merged_i += 1

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

    return inversion_counter

def merge_sort_count(list_arr, temp, left_i, right_i):
    inversion_counter = 0

    if left_i < right_i:
        middle = (left_i + right_i) // 2

        inversion_counter += merge_sort_count(list_arr, temp, left_i, middle)
        inversion_counter += merge_sort_count(list_arr, temp, middle + 1,
right_i)
        inversion_counter += merge_count(list_arr, temp, left_i, middle,
right_i)
    return inversion_counter

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_3/tests/input.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 2 * (10 ** 4):
        nums = list(map(int, list(file[1].split(' '))))
        if all([abs(n) <= 10 ** 9 for x in nums]):
            file_output.write((str(merge_sort_count(nums, [0] * n, 0, n -
1))))
        else:
            print('Введите другое число')
    else:
        print('Неверный ввод данных')

file_output.close()

```

### Текстовое объяснение решения.

Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю вторую строку в файле по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. Записываю результат в переменную nums. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file\_output результат функции merge\_sort, делая с помощью map() все элементы строками, и объединяю результат в строку через join. В ином случае прошу ввести корректные данные. Закрываю файл.

Функция merge\_sort и merge:

Аналогично первому заданию, только эта функция подсчитывает количество инверсий в массиве.

Результат работы кода на примерах из текста задачи.

lab2_3.py	input.txt	output	lab2_3.py	input.txt
1	10		1	17
2	1 8 2 1 4 7 3 2 3 6			

```
test_example x
C:\Users\Slawa\Desktop\Uni\venv\Scr
Время работы: 0.07520000508520752
Память: 0.939453125 K6
```

Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab2.task_3.src.lab2_3 import merge_sort_count

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = merge_sort_count(input_arr, [0] * len(input_arr), 0, len(input_arr)
- 1)
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' K6')
tracemalloc.stop()
```

```
test_min x
C:\Users\Slawa\Desktop\Uni\venv\Scr
Время работы: 0.01930000144056976
Память: 0.384765625 K6
```

Максимальные значения

```
import time
import tracemalloc
import random
from asd_itmo.lab2.task_3.src.lab2_3 import merge_sort_count

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(2 * (10 **
4))]

result = merge_sort_count(input_arr, [0] * len(input_arr), 0, len(input_arr)
- 1)
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' K6')
tracemalloc.stop()
```

```
test_max x
C:\Users\Slawa\Desktop\Uni\venv\Sc
Время работы: 0.9053249999997206
Память: 917.103515625 K6
```

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений входных данных из текста задачи	0.0193	0.3847
Пример из задачи	0.0752	0.9394
Верхняя граница диапазона значений входных данных из текста задачи	0.905	917.103

Вывод по задаче:

В ходе работы над задачей мной был изучен способ подсчёта инверсий. Была изучена эффективность этого метода.

## Задача №10. ★

Текст задачи.

1. Реализуйте сортировку слиянием, учитывая, что можно сэкономить на отсортированных массивах, которые не нужно объединять. Проверьте  $A[q]$ , меньше он или равен  $A[q + 1]$ , и объедините их, только если  $A[q] > A[q + 1]$ , где  $q$  - середина при делении в Merge\_Sort.
2. В небольших массивах сортировки методом вставок и методом выбора могут работать быстрее сортировки слиянием. Сравните свои реализации сортировки методом вставок, методом выбора и сортировки слиянием и найдите порог, где сортировка слиянием работает быстрее двух других.

Листинг кода.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_10/tests/output.txt",
'w')

def merge(left, right):
    list_merged = list()
    left_i = right_i = 0

    while left_i < len(left) and right_i < len(right):
        if left_i < len(left) and right_i < len(right):
            if left[left_i] < right[right_i]:
                list_merged.append(left[left_i])
                left_i += 1
            else:
                list_merged.append(right[right_i])
                right_i += 1
```

```

while left_i < len(left):
    list_merged.append(left[left_i])
    left_i += 1

while right_i < len(right):
    list_merged.append(right[right_i])
    right_i += 1

return list_merged

def merge_sort(list_arr):
    if len(list_arr) <= 1:
        return list_arr

    middle = len(list_arr) // 2
    left_list = merge_sort(list_arr[:middle])
    right_list = merge_sort(list_arr[middle:])
    return merge(left_list, right_list)

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab2/task_10/tests/input.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 2 * (10 ** 4):
        nums = list(map(int, list(file[1].split(' '))))
        if all([abs(n) <= 10 ** 9 for x in nums]):
            file_output.write(' '.join(map(str, merge_sort(nums))))
        else:
            print('Введите другое число')
    else:
        print('Неверный ввод данных')

file_output.close()

```

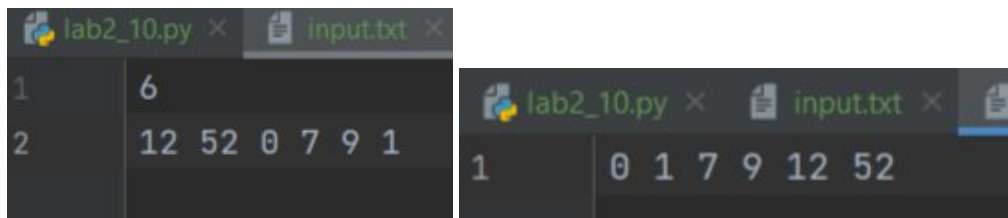
### Текстовое объяснение решения.

Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю вторую строку в файле по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. Записываю результат в переменную nums. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file\_output результат функции merge\_sort, делая с помощью map() все элементы строками, и объединяю результат в строку через join. В ином случае прошу ввести корректные данные. Закрываю файл.

Функция merge\_sort и merge:

Аналогично функциям из первого задания. Проверяю  $A[q]$ , меньше он или равен  $A[q + 1]$ , и объедините их, только если  $A[q] > A[q + 1]$ , где  $q$  - середина при делении в Merge\_Sort.

Результат работы кода на примерах из текста задачи.



```
C:\Users\Slawa\Desktop\Uni\venv\Scr
Время работы: 0.05640000745188445
Память: 0.806640625 Kб
```

Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab2.task_10.src.lab2_10 import merge_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = merge_sort(input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Kб')
tracemalloc.stop()
```

```
C:\Users\Slawa\Desktop\Uni\venv\Scr
Время работы: 0.023900007363408804
Память: 0.38671875 Kб
```

Максимальные значения

```
import time
import tracemalloc
import random
from asd_itmo.lab2.task_10.src.lab2_10 import merge_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(2 * (10 ** 4))]

result = merge_sort(input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Kб')
tracemalloc.stop()
```

```
C:\Users\Slawa\Desktop\Uni\venv\Scr
Время работы: 0.6166492000047583
Память: 1099.248046875 Кб
```

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений входных данных из текста задачи	0.0239	0.3867
Пример из задачи	0.0564	0.8066
Верхняя граница диапазона значений входных данных из текста задачи	0.6166	1099.248

Результат работы кода теста со сравнением времени и поиска порога:

```
import time
import random
from asd_itmo.lab2.task_10.src.lab2_10 import merge_sort

def insertion_sort(list_arr):
    for i in range(1, len(list_arr)):
        key = list_arr[i]
        j = i - 1
        while j >= 0 and key < list_arr[j]:
            list_arr[j + 1] = list_arr[j]
            j -= 1
        list_arr[j + 1] = key
    return list_arr

def selection_sort(A):
    for i in range(len(A) - 1):
        min_local = i
        for j in range(i + 1, len(A)):
            if A[j] < A[min_local]:
                min_local = j
        A[i], A[min_local] = A[min_local], A[i]
    return A

def measure_time(sort_func, arr):
    start_time = time.time()
    sort_func(arr)
    return time.time() - start_time
```



```
def find_threshold(max_size):
    for size in range(1, max_size + 1):
        arr = [random.randint(-10 ** 9, 10 ** 9) for _ in range(size)]

        insertion_time = measure_time(insertion_sort, arr.copy())
        selection_time = measure_time(selection_sort, arr.copy())
        merge_time = measure_time(merge_sort, arr.copy())

        print(
            f"Размер: {size} | Merge: {merge_time:.8f}s | Insertion: {insertion_time:.8f}s | Selection: "
            f"{selection_time:.8f}s")

        if merge_time < insertion_time and merge_time < selection_time:
            print(f"Merge sort быстрее, чем insertion sort и selection sort на массиве размером: {size}")
            break

find_threshold(1000)
```

```
test x
Размер: 89 | Merge: 0.00000000s | Insertion: 0.00098228s | Selection: 0.00000000s
Размер: 90 | Merge: 0.00000000s | Insertion: 0.00101566s | Selection: 0.00000000s
Размер: 91 | Merge: 0.00000000s | Insertion: 0.00100803s | Selection: 0.00000000s
Размер: 92 | Merge: 0.00099182s | Insertion: 0.00103331s | Selection: 0.00000000s
Размер: 93 | Merge: 0.00104403s | Insertion: 0.00000000s | Selection: 0.00000000s
Размер: 94 | Merge: 0.00098062s | Insertion: 0.00000000s | Selection: 0.00000000s
Размер: 95 | Merge: 0.00000000s | Insertion: 0.00000000s | Selection: 0.00106430s
Размер: 96 | Merge: 0.00000000s | Insertion: 0.00000000s | Selection: 0.00096154s
Размер: 97 | Merge: 0.00000000s | Insertion: 0.00100064s | Selection: 0.00000000s
Размер: 98 | Merge: 0.00000000s | Insertion: 0.00100017s | Selection: 0.00000000s
Размер: 99 | Merge: 0.00098252s | Insertion: 0.00000000s | Selection: 0.00000000s
Размер: 100 | Merge: 0.00000000s | Insertion: 0.00000000s | Selection: 0.00101709s
Размер: 101 | Merge: 0.00000000s | Insertion: 0.00099158s | Selection: 0.00000000s
Размер: 102 | Merge: 0.00100017s | Insertion: 0.00100017s | Selection: 0.00000000s
Размер: 103 | Merge: 0.00000000s | Insertion: 0.00000000s | Selection: 0.00100875s
Размер: 104 | Merge: 0.00000000s | Insertion: 0.00100017s | Selection: 0.00000000s
Размер: 105 | Merge: 0.00100017s | Insertion: 0.00100040s | Selection: 0.00000000s
Размер: 106 | Merge: 0.00000000s | Insertion: 0.00000000s | Selection: 0.00098324s
Размер: 107 | Merge: 0.00000000s | Insertion: 0.00101781s | Selection: 0.00000000s
Размер: 108 | Merge: 0.00000000s | Insertion: 0.00000000s | Selection: 0.00099158s
Размер: 109 | Merge: 0.00000000s | Insertion: 0.00100875s | Selection: 0.00000000s
Размер: 110 | Merge: 0.00100040s | Insertion: 0.00100017s | Selection: 0.00000000s
Размер: 111 | Merge: 0.00000000s | Insertion: 0.00000000s | Selection: 0.00099993s
Размер: 112 | Merge: 0.00000000s | Insertion: 0.00099182s | Selection: 0.00100899s
Merge sort быстрее, чем insertion sort и selection sort на массиве размером: 112
```

Вывод по задаче:

В ходе работы над задачей мной был изучен способ сортировки вставкой с оптимизацией. Был проведён анализ эффективности сортировок методом вставок, методом выбора и слиянием. Был найден порог, при котором сортировка слиянием становится более эффективной, чем сортировки методом вставок и выбором. На малых размерах массивов сортировка

вставками и выбором показывали лучшие результаты по времени выполнения, чем сортировка слиянием, благодаря своей менее сложной логике и меньшим накладным расходам. Выбор алгоритмов сортировки должен зависеть от условий задачи, таких как размер входных данных и спецификации производительности.

## Вывод

В ходе лабораторной работы был изучен метод сортировки слиянием и метод декомпозиции («Разделяй и властвуй»). Был проведён анализ работы алгоритмов на максимальных и минимальных значениях. Был изучен алгоритм бинарного поиска.