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:
Макунина А.А.
К32421

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №1. 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.

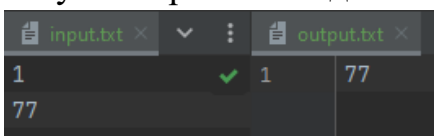
```
def Merge_sorting(array, left, right):
    if left < right:
        middle = (left + right) // 2  # делим без
остатка
        Merge_sorting(array, left, middle)
        Merge_sorting(array, middle + 1, right)
        Merge(array, left, middle, right)
        return array
    else:
        return array
```

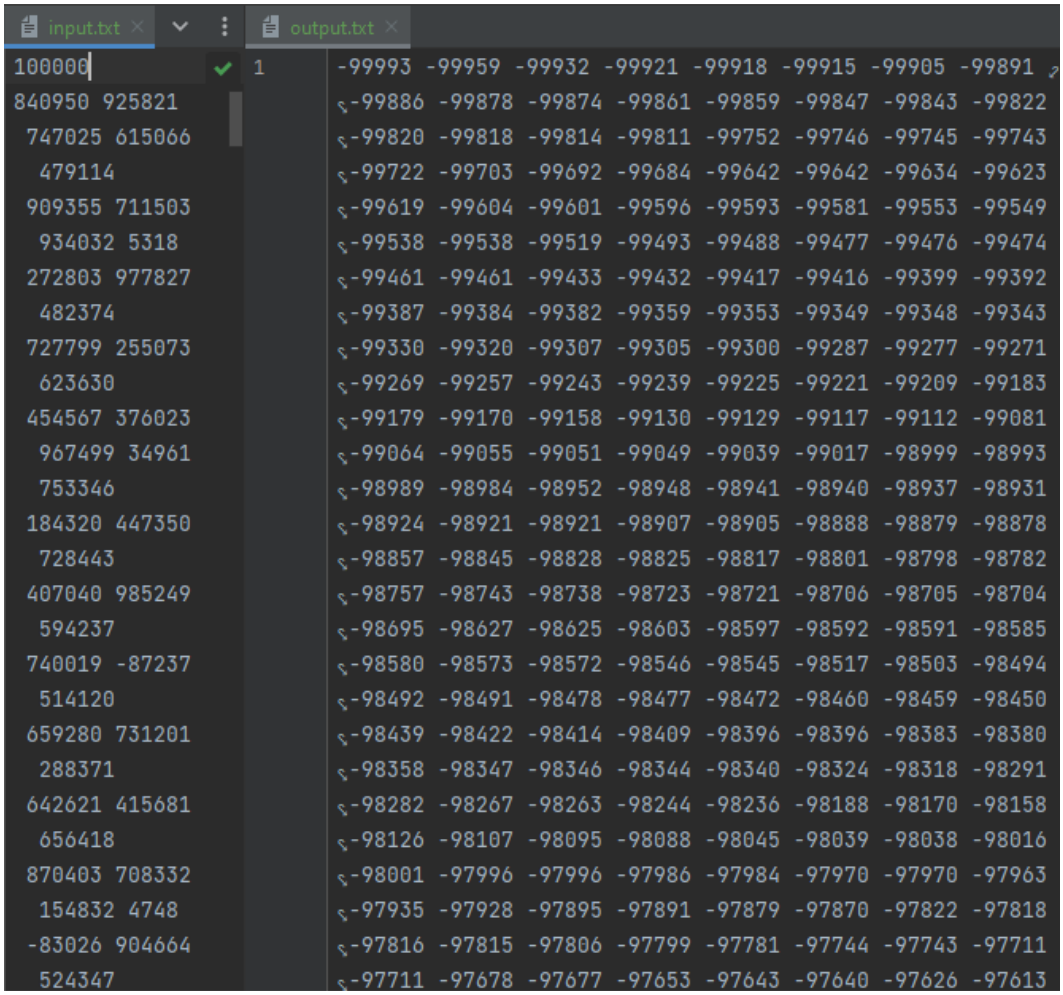
```
def Merge(arr, left, middle, right):
    left_array, right_array = arr[left:middle + 1],
arr[middle + 1:right + 1:]  # срез
    left_array.append(10 ** 10)
    right_array.append(10 ** 10)
    count_left, count_right = 0, 0
    for i in range(left, right + 1):
        if left_array[count_left] <
right_array[count_right]:
            arr[i] = left_array[count_left]
            count_left += 1
        else:
            arr[i] = right_array[count_right]
            count_right += 1
    return arr


f = open('input.txt')
n = int(f.readline())
array = list(map(int, f.readline().split()))
f.close()
w = open("output.txt", "w")
w.write(' '.join(list(map(str, Merge_sorting(array,
0, n - 1)))))
w.close()
```

Здесь реализована тривиальная сортировка слиянием с помощью псевдокода: Merge-sorting - делим массив на две части, потом ещё на две части и т.д. Когда у нас массив разобьётся на множество массивов, состоящих из одного элемента, мы переходим к Merge, где массивы делают только две вещи: сортируются и объединяются друг с другом до бесконечности (нет, до 10^{10}).

Результат работы кода на примерах:

Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Максимальные затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0012251999999999957	0.01746654510498047
Пример из задачи	0.016315199999999988	0.07828044891357422
Верхняя граница диапазона значений входных данных из текста задачи	1.9067169000000002	10.106904029846191

Вывод по задаче: метод слияния интересный, задание мной выполнено корректно, так как он прошёл тест даже на противный массив с 100000 числами.

Задача №4. 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.

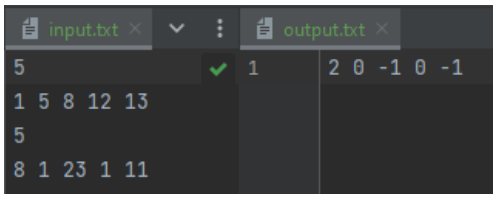
- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве, и последовательность $a_0 < a_1 < ... < a_{n-1}$ из n **различных** положительных целых чисел в порядке возрастания, $1 \leq a_i \leq 10^9$ для всех $0 \leq i < n$. 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.

```
def Binary_Search(array, x):
    left = 0
    right = n
    while right - left != 1:
        middle = left + (right - left) // 2
        if x < array[middle]:
            right = middle
        else:
            left = middle
    if x == array[left]:
        return left
    else:
        return -1


f = open('input.txt')
n = int(f.readline())
array = list(map(int, f.readline().split()))
k = int(f.readline())
x = list(map(int, f.readline().split()))
f.close()
w = open("output.txt", "w")
for i in range(k):
```

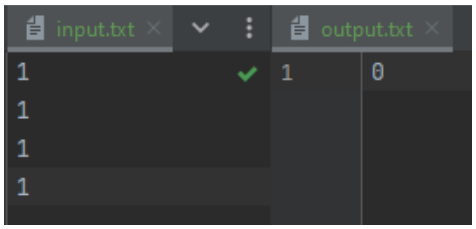
```
    w.write(str(Binary_Search(array, x[i])) + ' ')
w.close()
```

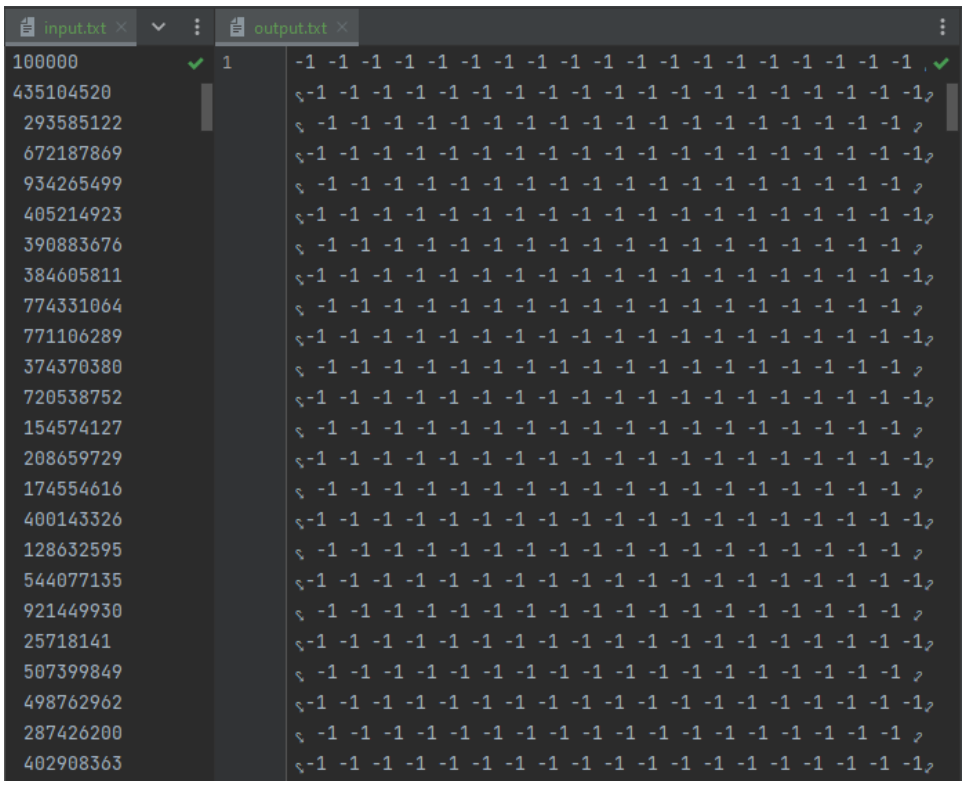
Берем средний элемент массива, сравниваем с числом x, выбираем половинку подмассива, где проводится поиск. Новые итерацим цикла с обновлением переменных left и right до массива длины 1: если элемент в этом массиве равен искомому, то выводится его индекс, а в ином случае -1.

Результат работы кода на примерах:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Максимальные затраты памяти, мб

Нижняя граница диапазона значений входных данных из текста задачи	0.0011580999999999952	0.01737689971923828
Пример из задачи	0.0009902999999999995	0.01748371124267578
Верхняя граница диапазона значений входных данных из текста задачи	1.9990614	13.163354873657227

Вывод по задаче: бинарный поиск – нормальный алгоритм поиска, но честно не по душе!

Задача №5. 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.

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.

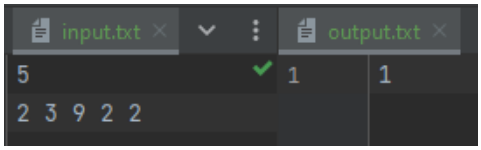
• Ограничение по времени. 2сек.

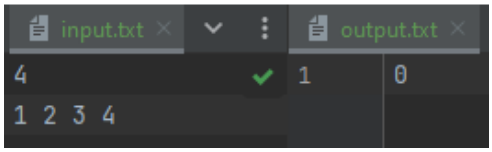
• Ограничение по памяти. 256 мб.

Воспользуемся алгоритмом сортировки «слияние». Мы ищем максимум по числу вхождений каждого числа при сортировке слиянием двух массивов. Далее половинчатая длинная массива сравнивается с максимальным значением и записывается результат: нуль или единица.

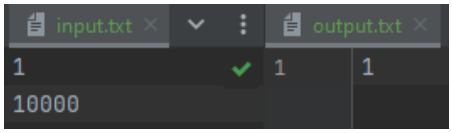
```
def Merge_sorting(array, left, right):
    if left < right:
        middle = (left + right) // 2
        Merge_sorting(array, left, middle)
        Merge_sorting(array, middle + 1, right)
        return Merge(array, left, middle, right)
    else:
        return 1

def Merge(arr, left, middle, right):
    left_array, right_array = arr[left:middle + 1:],
arr[middle + 1:right + 1:]
    left_array.append(10 ** 10)
    right_array.append(10 ** 10)
    count_left, count_right, max, now = 0, 0, 0, 0
    for i in range(left, right + 1):
        if left_array[count_left] <
right_array[count_right]:
            arr[i] = left_array[count_left]
            count_left += 1
        else:
            arr[i] = right_array[count_right]
            count_right += 1
        if i == left or arr[i] == arr[i - 1]:
            now += 1
        else:
            max = now if now > max else max
            now = 1
    return max > len(arr) / 2


f = open("input.txt")
n = int(f.readline())
array = list(map(int, f.readline().split()))
f.close()
w = open("output.txt", "w")
w.write(str(int(Merge_sorting(array, 0, len(array) -
1))))
w.close()
```

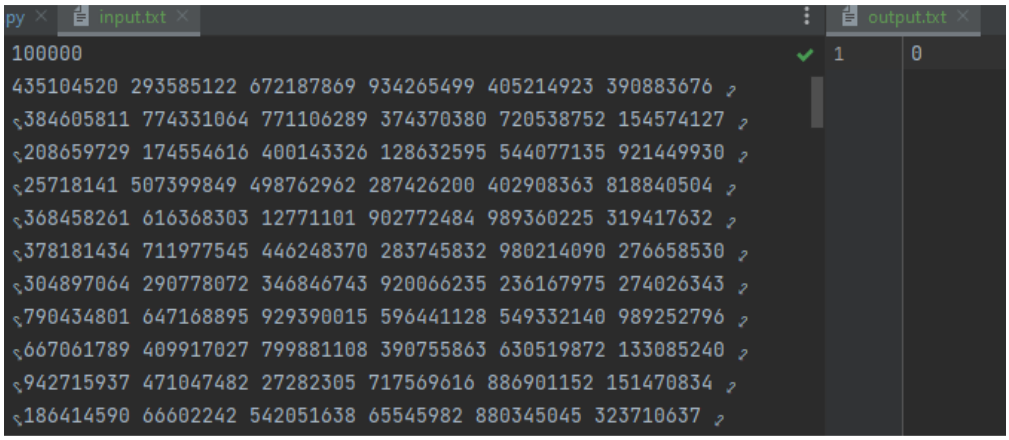
Результат работы кода на примерах из задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Максимальные затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.009520199999999993	0.1367044448852539
Пример из задачи	0.008975899999999995	0.13671207427978516
Пример из задачи	0.009660299999999997	0.13670825958251953
Верхняя граница диапазона значений входных данных из текста задачи	2.4769924	9.864349365234375

Вывод по задаче: задача реализована успешно, но тест с 100000 элементами в массиве очень тяжело укладывается в «дедлайн» задачи. Умнее реализации алгоритма я не придумала – «с пивом потянет».

Вывод

В ходе выполнения данной лабораторной работы я научилась работать с алгоритмами сортировки слиянием, бинарного поиска и методу декомпозиции «разделяй и властвуй». В целом лабораторная работа неплохая, алгоритмы интересные для изучения.