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 9

Выполнил:

Аль-Мошки Исмаил Абдулвахаб

К33391

Проверила:

Артамонова В.Е.

Санкт-Петербург

2022 г.

# Содержание отчета

[**Содержание отчета**](#_10udeakjagvs) **2**

[**Задачи по варианту**](#_gf7yxvsg0zb) **3**

[Задача №1. Сортировка слиянием](#_pgzbaj56cf0o)

[Задача №2. Число инверсий](#_pgzbaj56cf0o)

[Задача №3. Представитель большинства](#_pgzbaj56cf0o)

[**Вывод**](#_fu90fuyk873) **5**

# Задачи по варианту

**Задача №1.Сортировка слиянием**

1. Используя псевдокод процедур Merge и Merge-sort из презентации к Лек-

ции 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 ≤ n ≤ 2 · 104) — число элементов в массиве.

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

превосходящих 109.

• 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, 104, 105 чисел порядка 109, 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.

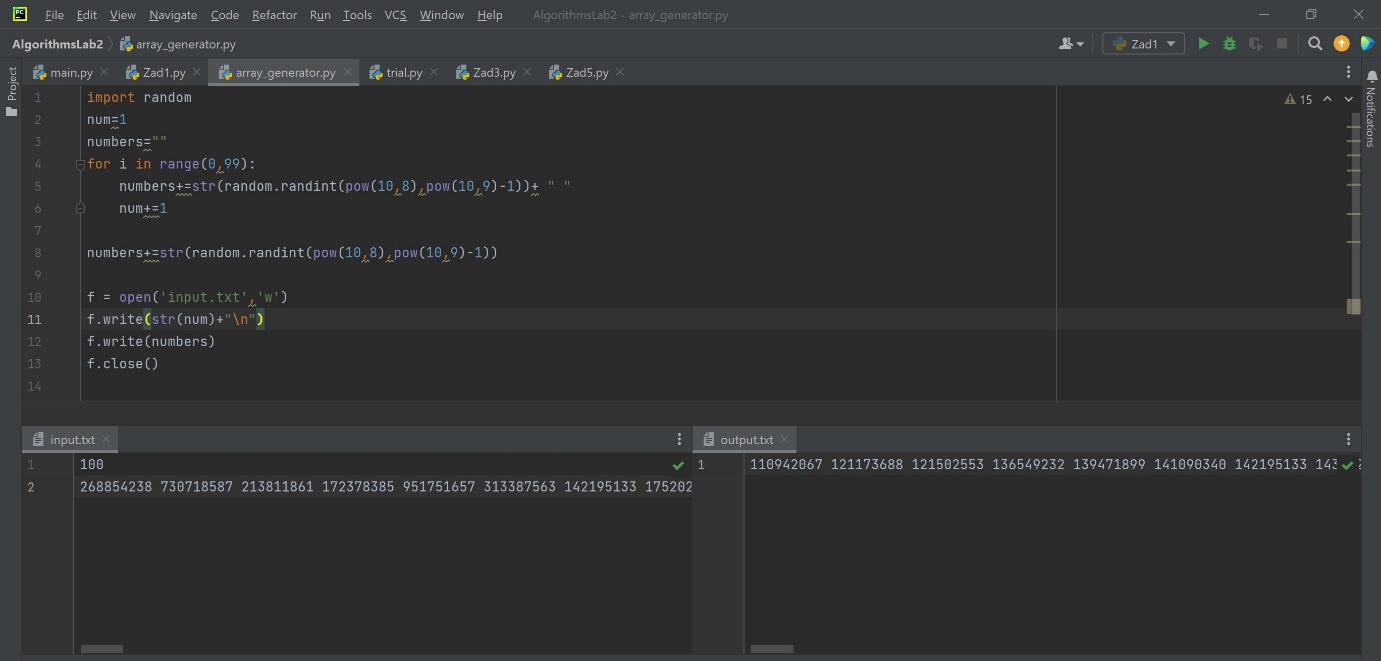
Lesting :

import tracemalloc,time  
  
  
  
def merge(a):  
  
  
 if(len(a)>1):  
 middle=len(a)//2  
 L\_subarray=a[:middle]  
 R\_subarray=a[middle:]  
  
 merge(L\_subarray)  
 merge(R\_subarray)  
  
 i=j=c=0  
  
 while i<len(L\_subarray) and j<len(R\_subarray):  
 if(L\_subarray[i]>=R\_subarray[j]):  
 a[c]=R\_subarray[j]  
 j+=1  
 else:  
 a[c]=L\_subarray[i]  
 i+=1  
 c+=1  
  
 while (i<len(L\_subarray)):  
 a[c]=L\_subarray[i]  
 i+=1  
 c+=1  
 while (j<len(R\_subarray)):  
 a[c]=R\_subarray[j]  
 j+=1  
 c+=1  
  
  
  
  
  
# read input  
f = open('input.txt', 'r')  
length = f.readline()  
massive = f.readline()  
a = massive.split()  
  
# type preperation  
for i in range(0, len(a)):  
 a[i] = int(a[i])  
  
# start timecount and memory monitoring  
tracemalloc.start()  
start\_t = time.perf\_counter()  
merge(a)  
# answer validation  
answer = ""  
for i in range(len(a) - 1):  
 answer += str(a[i]) + " "  
answer += str(a[-1])  
  
# answer written  
f = open('output.txt', 'w')  
f.write(answer)  
  
f.close()  
# Time and space Calculation  
sort\_time = time.perf\_counter() - start\_t  
sort\_space = tracemalloc.get\_tracemalloc\_memory()  
tracemalloc.stop()  
print("Time for data sorting in seconds: " + str(sort\_time))  
print("Space used in the sorting process in bytes: " + str(sort\_space))

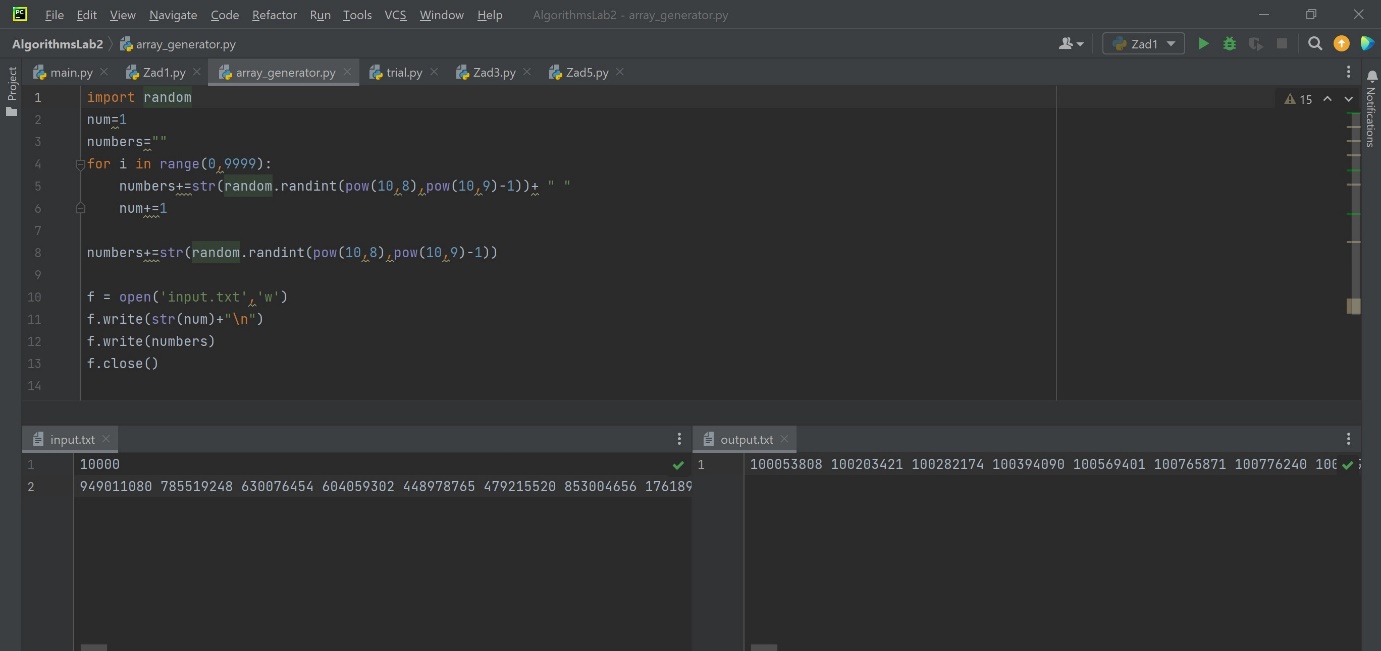
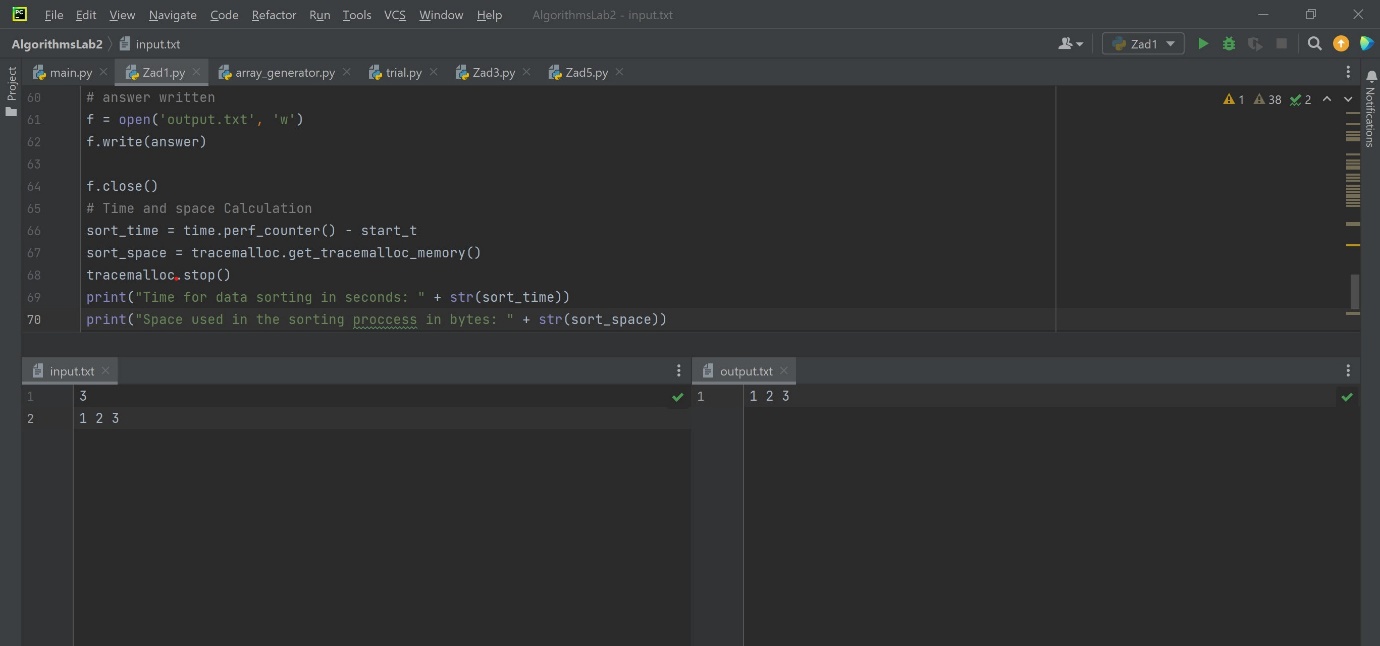
**Текстовое объяснение решения:**

рекурсивный алгоритм непрерывно разбивает массив пополам до тех пор, пока его нельзя будет разделить дальше. Это означает, что если массив станет пустым или в нем останется только один элемент, деление прекратится, т.е. это базовый вариант для остановки рекурсии. Если массив содержит несколько элементов, разделите массив на половины и рекурсивно вызовите сортировку слиянием для каждой из половин. Наконец, когда оба отверстия отсортированы, применяется операция слияния. Операция слияния - это процесс взятия двух отсортированных массивов меньшего размера и объединения их, чтобы в конечном итоге создать массив большего размера.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



Проверка задачи на (openedu, acmp и тд при наличии в задаче). (скрин)

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи |  | |
| Пример из задачи |  | |
| Верхняя граница диапазона значений входных данных из текста задачи |  | |

## Задача №2. Число инверсий

Инверсией в последовательности чисел A называется такая ситуация, когда i < j, а Ai > Aj . Количество инверсий в последовательности в некотором роде определяет, насколько близка данная последовательность к отсортированной. Например, в сортированном массиве число инверсий равно 0, а в массиве, сортированном наоборот - каждые два элемента будут составлять инверсию (всего n(n − 1)/2).

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

Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией

сортировки слиянием.

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

держится число n (1 ≤ n ≤ 105) — число элементов в массиве. Во второй

строке находятся n различных целых чисел, по модулю не превосходящих

109.

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

число инверсий в массиве.

• Ограничение по времени. 2сек.

• Ограничение по памяти. 256 мб.

• Пример:

input.txt output.txt

10 17

1 8 2 1 4 7 3 2 3 6

import tracemalloc,time  
def getInvCount(arr, n):  
 inv\_count = 0  
 for i in range(n):  
 for j in range(i + 1, n):  
 if (arr[i] > arr[j]):  
 inv\_count += 1  
  
 return inv\_count  
  
# read input  
f = open('input.txt', 'r')  
length = int(f.readline())  
massive = f.readline()  
arr = massive.split()  
  
# type preperation  
for i in range(0, len(arr)):  
 arr[i] = int(arr[i])  
  
# start timecount and memory monitoring  
tracemalloc.start()  
start\_t = time.perf\_counter()  
inversion\_count = getInvCount(arr, length)  
  
  
# answer written  
f = open('output.txt', 'w')  
f.write(str(inversion\_count))  
f.close()  
  
# Time and space Calculation  
sort\_time = time.perf\_counter() - start\_t  
sort\_space = tracemalloc.get\_tracemalloc\_memory()  
tracemalloc.stop()  
print("Time for data sorting in seconds: " + str(sort\_time))  
print("Space used in the sorting proccess in bytes: " + str(sort\_space))

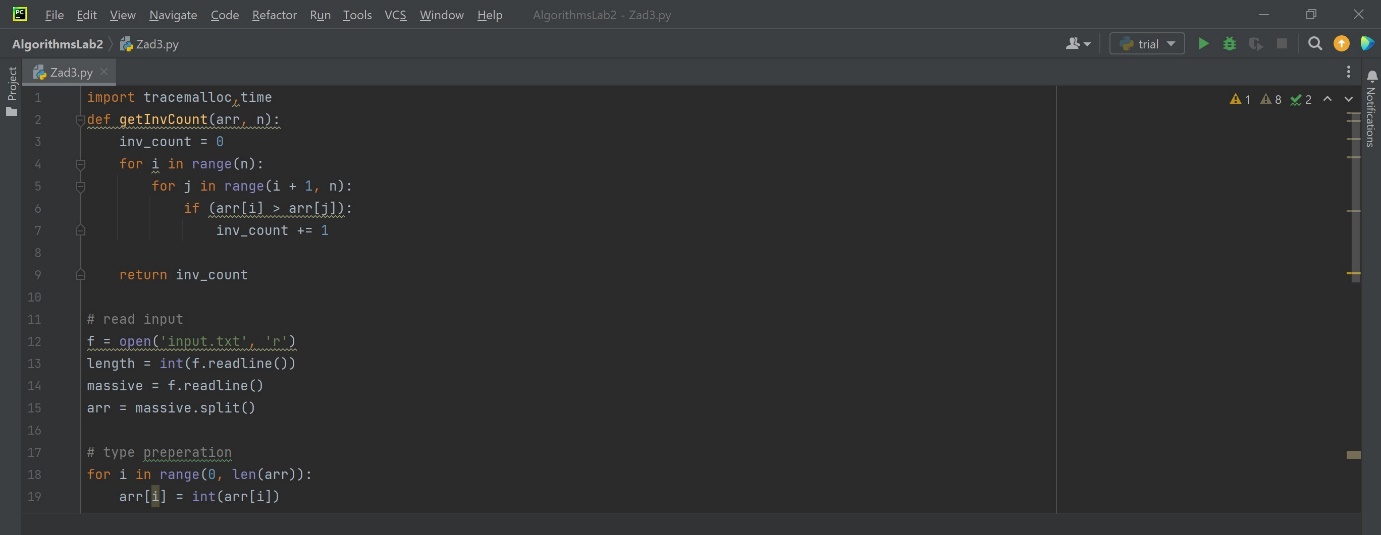
Текстовое объяснение решения.:

1-Пройдите по массиву от начала до конца

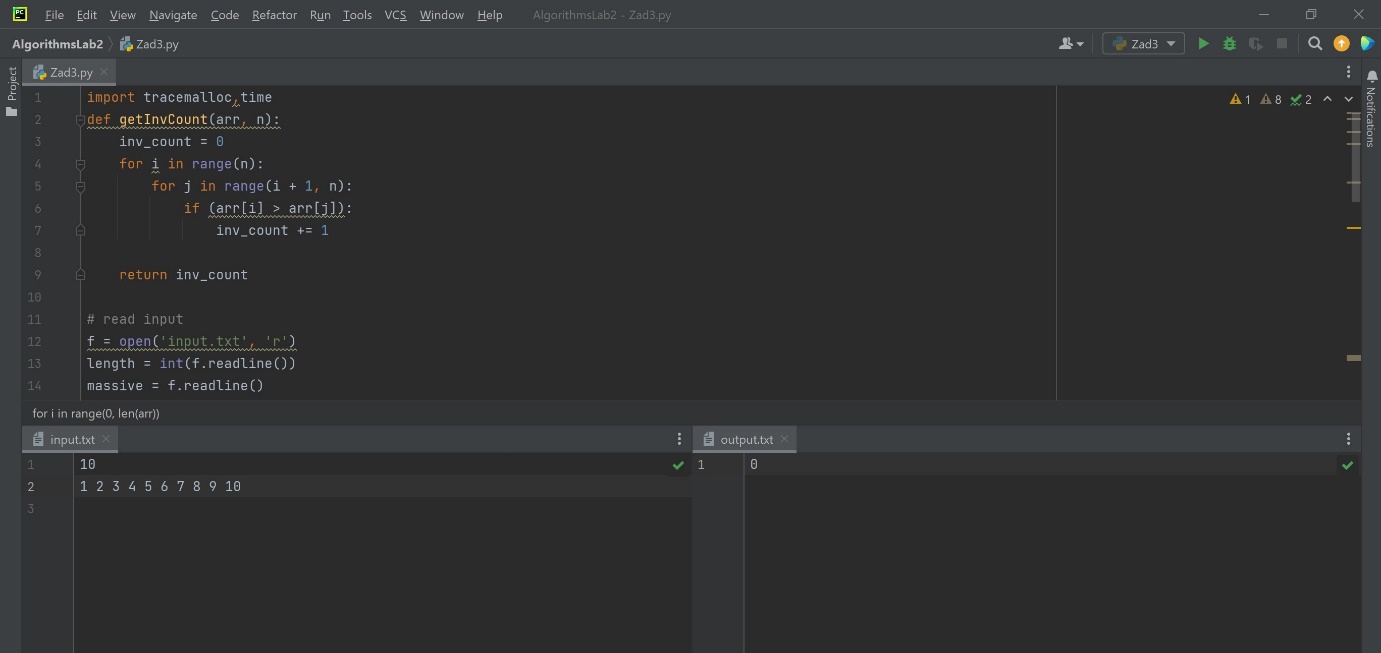
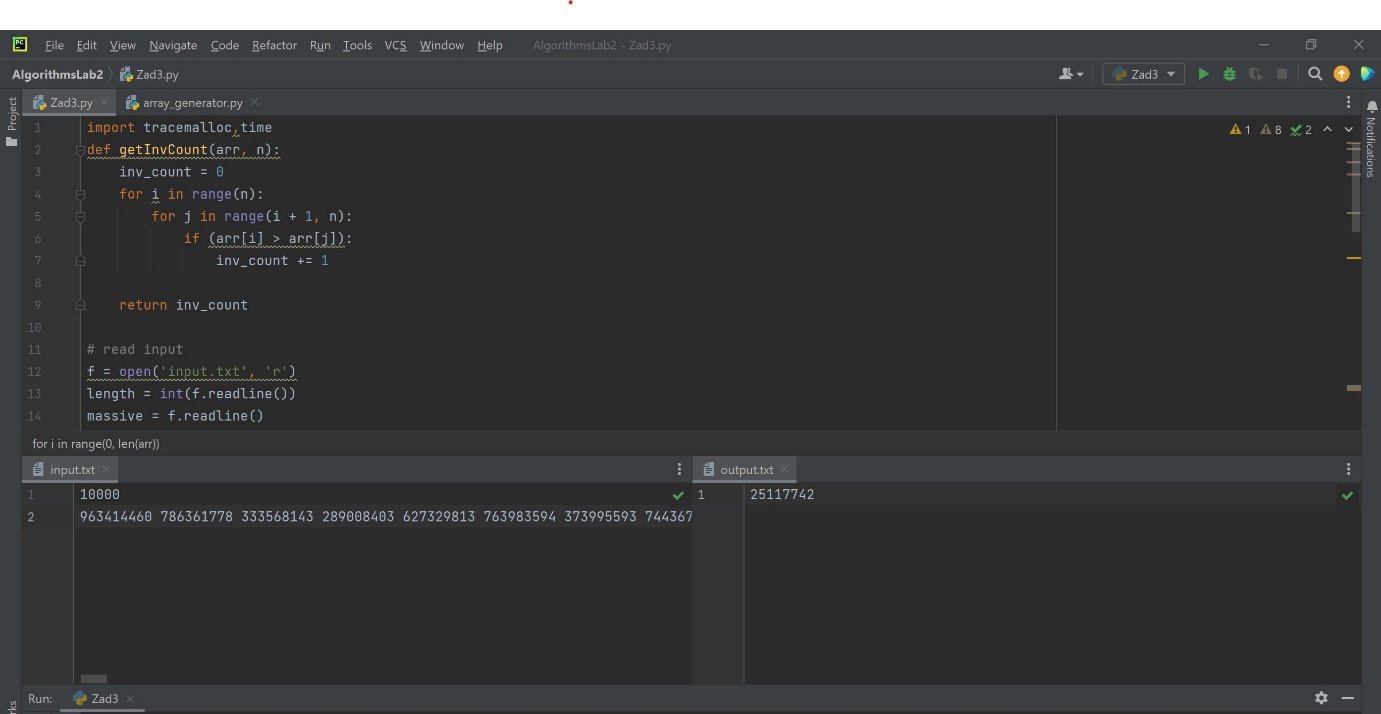
2-Для каждого элемента найдите количество элементов, меньших текущего числа, вплоть до этого индекса, используя другой цикл.

3-Суммируйте количество инверсий для каждого индекса.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи |  | |
| Пример из задачи |  | |
| Верхняя граница диапазона значений входных данных из текста задачи |  | |

## Задача №3.Представитель большинства

## Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов a1, a2,...an, и нужно проверить, содержит ли она элемент, который появляется больше, чем n/2 раз. Наивный метод это сделать:Bubble\_Sort(A):

## 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 "There is no majority"

## 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 ≤ n ≤ 105) — число элементов в массиве. Во второй строке находятся n положительных целых чисел, по модулю не превосходящих 109, 0 ≤ ai ≤ 109.

## • Формат выходного файла (output.txt). Выведите 1, если во входной после-довательности 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 1

## 2 3 9 2 2

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

## • Пример 2:

## input.txt output.txt

## 4 0

## 1 2 3 4

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

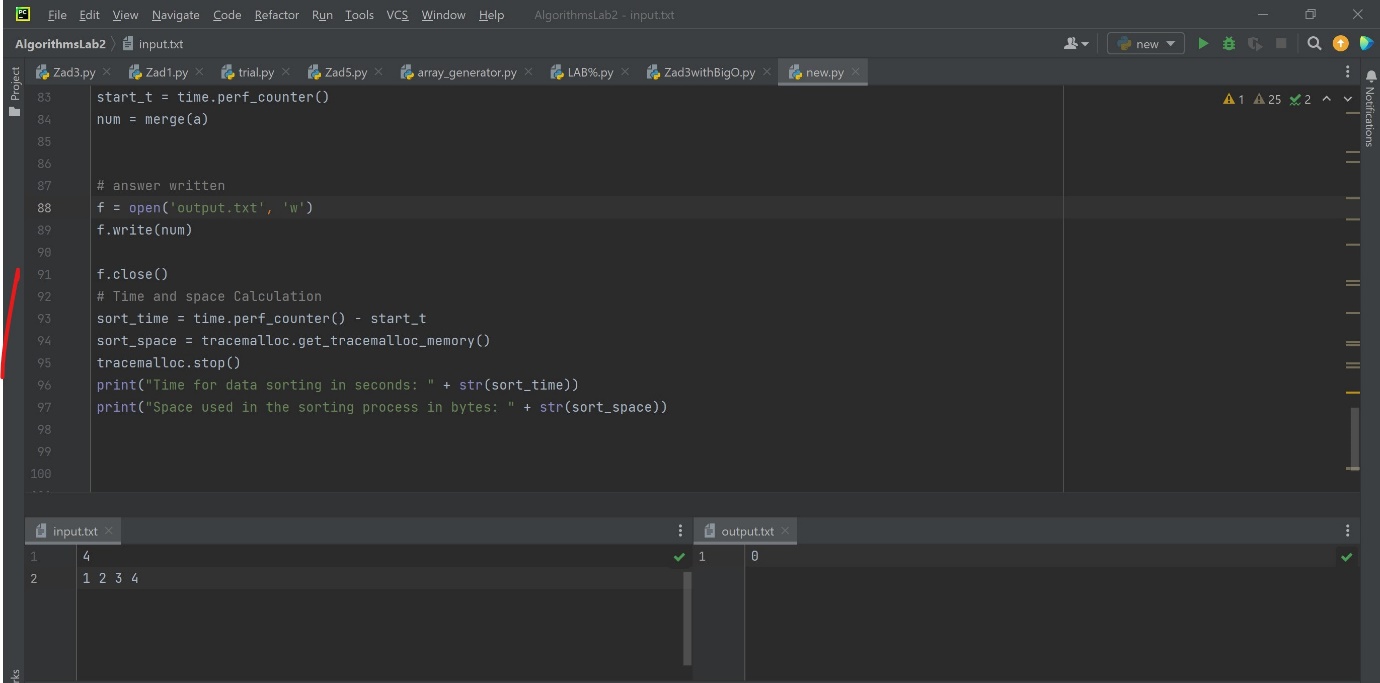
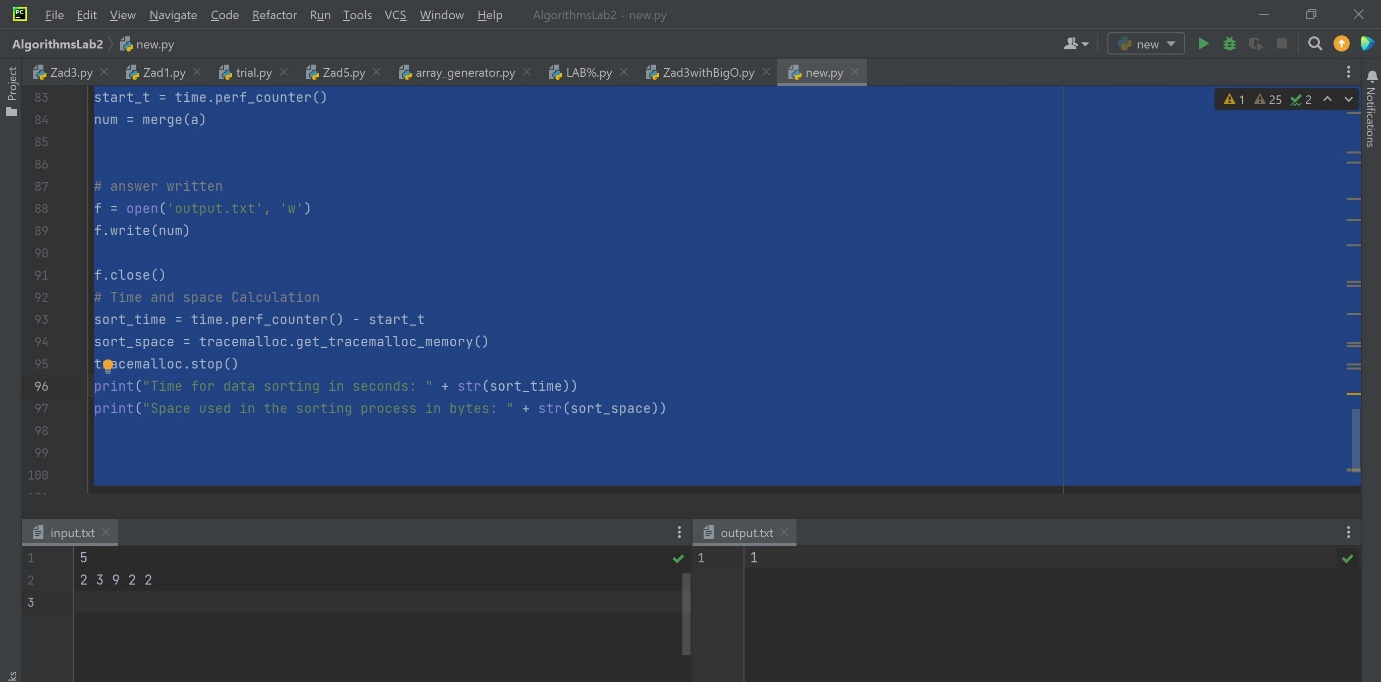
## 1.Листинг кода :

import tracemalloc, time  
  
  
# Python program for implementation of MergeSort  
def merge(arr):  
 if len(arr) > 1:  
 dec = {}  
 # Finding the mid of the array  
 mid = len(arr) // 2  
  
 # Dividing the array elements  
 L = arr[:mid]  
  
 # into 2 halves  
 R = arr[mid:]  
  
 # Sorting the first half  
 merge(L)  
  
 # Sorting the second half  
 merge(R)  
  
 # dictonary  
  
 i = j = k = 0  
  
 # Copy data to temp arrays L[] and R[]  
 while i < len(L) and j < len(R):  
 if L[i] <= R[j]:  
 arr[k] = L[i]  
 if (dec.get(arr[k]) != None):  
 dec[arr[k]] += 1  
 else:  
 dec[arr[k]] = 1  
 i += 1  
 else:  
 arr[k] = R[j]  
 if (dec.get(arr[k]) != None):  
 dec[arr[k]] += 1  
 else:  
 dec[arr[k]] = 1  
 j += 1  
 k += 1  
  
 # Checking if any element was left  
 while i < len(L):  
 arr[k] = L[i]  
 if (dec.get(arr[k]) != None):  
 dec[arr[k]] += 1  
 else:  
 dec[arr[k]] = 1  
 i += 1  
 k += 1  
  
 while j < len(R):  
 arr[k] = R[j]  
 if (dec.get(arr[k]) != None):  
 dec[arr[k]] += 1  
 else:  
 dec[arr[k]] = 1  
 j += 1  
 k += 1  
  
 for key in dec:  
 if dec.get(key) >= int(length)/2 :  
 return "1"  
 return "0"  
  
  
  
# read input  
f = open('input.txt', 'r')  
length = f.readline()  
massive = f.readline()  
a = massive.split()  
  
# type preperation  
for i in range(0, len(a)):  
 a[i] = int(a[i])  
  
# start timecount and memory monitoring  
tracemalloc.start()  
start\_t = time.perf\_counter()  
num = merge(a)  
  
  
# answer written  
f = open('output.txt', 'w')  
f.write(num)  
  
f.close()  
# Time and space Calculation  
sort\_time = time.perf\_counter() - start\_t  
sort\_space = tracemalloc.get\_tracemalloc\_memory()  
tracemalloc.stop()  
print("Time for data sorting in seconds: " + str(sort\_time))  
print("Space used in the sorting process in bytes: " + str(sort\_space))

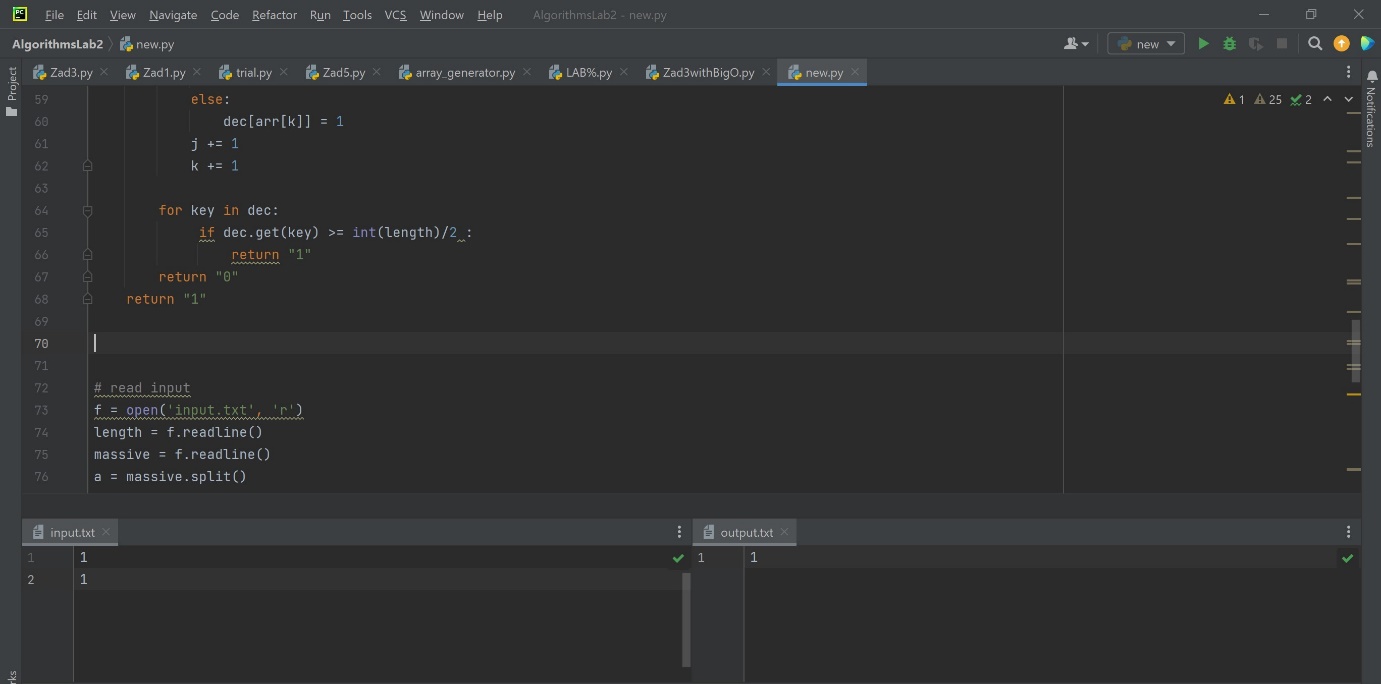
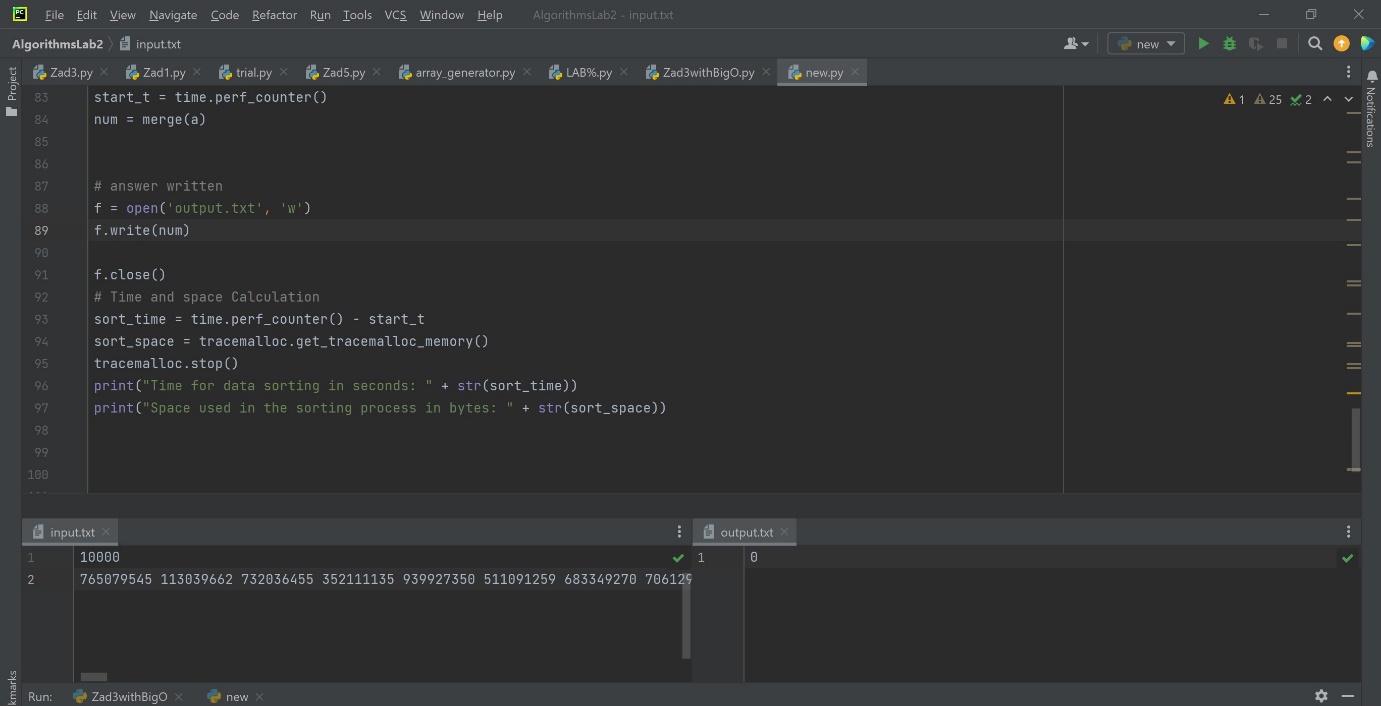
Текстовое объяснение решения.:

Мы включаем словарь count в сортировку слиянием, чтобы не получить квадратичную сложность. тогда с помощью словаря ситуация становится менее сложной, повторение числа уже сокращается за счет подсчета.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи |  | |
| Пример из задачи |  | |
| Пример из задачи |  | |
| Верхняя граница диапазона значений входных данных из текста задачи |  | |

# Вывод

Сортировка слиянием определяется как алгоритм сортировки, который работает путем разделения массива на более мелкие подмассивы, сортировки каждого подмассива, а затем объединения отсортированных подмассивов обратно вместе для формирования окончательного отсортированного массива.

Простыми словами, мы можем сказать, что процесс сортировки слиянием заключается в разделении массива на две половины, сортировке каждой половины, а затем объединении отсортированных половин обратно вместе. Этот процесс повторяется до тех пор, пока не будет отсортирован весь массив.

он обладает большей эффективностью и меньшей временной сложностью, чем уже рассмотренные алгоритмы. это может работать с большими сортами. Однако он использует сравнительно большое пространство по сравнению с другим алгоритмом.