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 1

Выполнил:

Аксянова А.Р

К3140

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[Содержание отчета 2](#_Toc180420328)

[Задачи 3](#_Toc180420329)

[Задача №1. Сортировка слиянием 3](#_Toc180420330)

[Задача №2. Число инверсий 6](#_Toc180420331)

[Задача №4. Бинарный поиск 8](#_Toc180420332)

[Задача №5 . Представитель большинства 9](#_Toc180420333)

[Задача №7 . Поиск максимального подмассива за линейное время 11](#_Toc180420334)

[Вывод 13](#_Toc180420335)

# Задачи

## Задача №1. Сортировка слиянием

1. Текст задачи

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.
* **Форматвыходногофайла(output.txt).**Однастрокавыходногофайла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
* Ограничение по времени. 2сек.
* Ограничение по памяти. 256 мб.

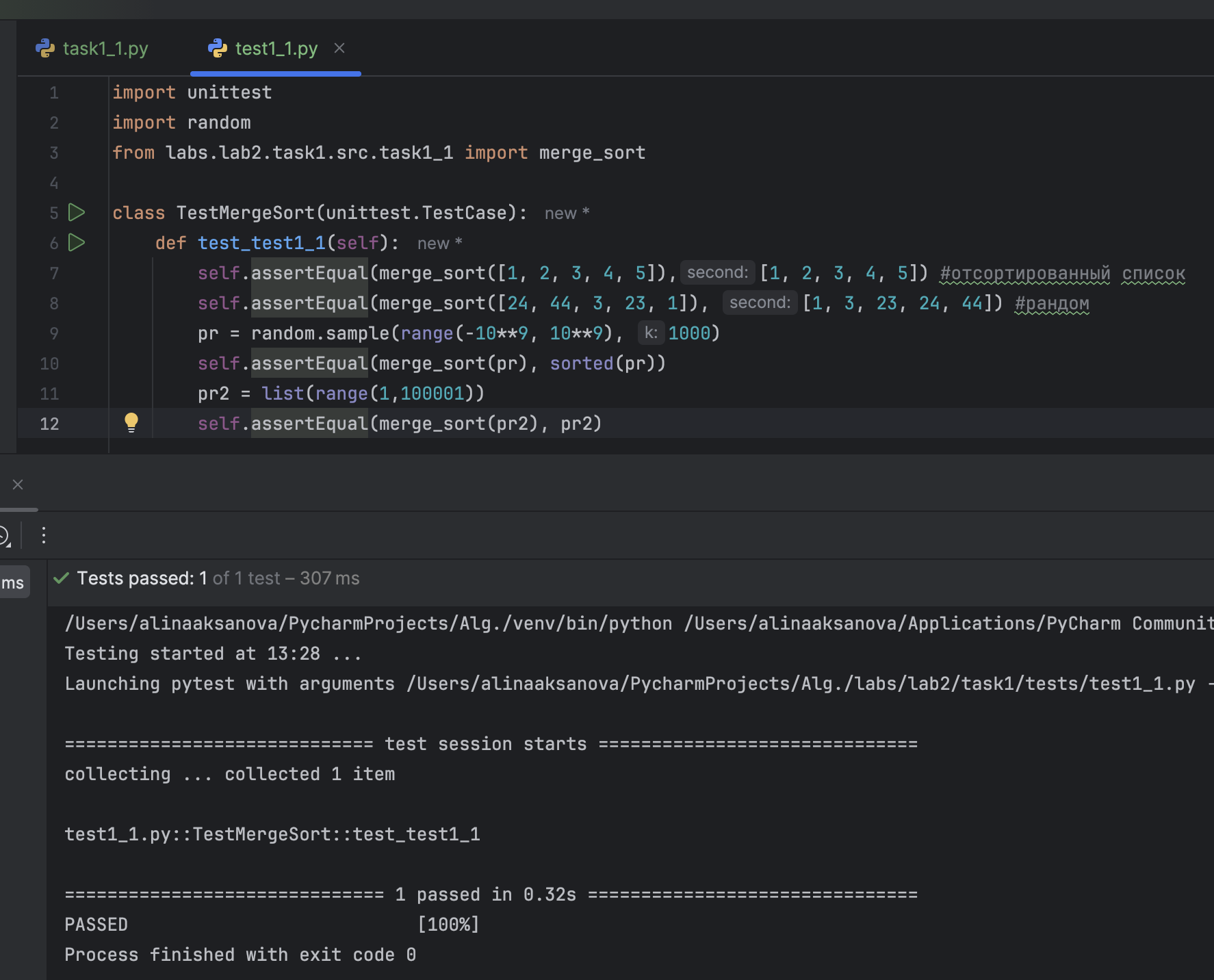
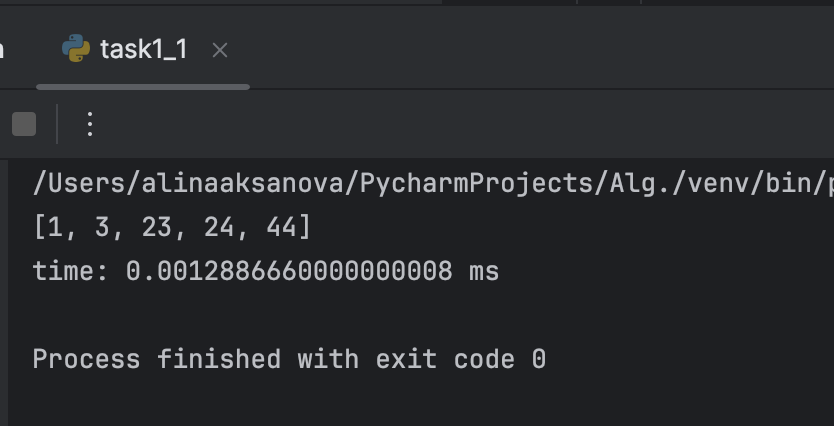
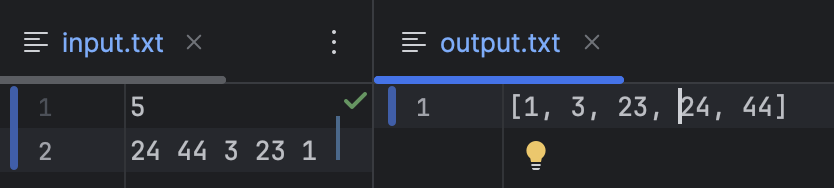
1. Для размера 1000, 104 , 105 чисел порядка 109 , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний. Сравните, например, с сортировкой вставкой на этих же данных.
2. Перепишите процедуру Merge так, чтобы в ней не использовались сигналь- ные значения. Сигналом 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.
3. Листинг кода

import time  
start = time.perf\_counter()  
  
k = open("../files/input.txt")  
n = int(k.readline())  
b = k.readline().split(" ")  
s = [int(l) for l in b ]  
  
def merge(list1, list2):  
 sorted\_list = []  
 l1\_index = l2\_index = 0  
 l1\_length, l2\_length = len(list1), len(list2)  
 for \_ in range(l1\_length + l2\_length):  
 if l1\_index < l1\_length and l2\_index < l2\_length:  
 if list1[l1\_index] <= list2[l2\_index]:  
 sorted\_list.append(list1[l1\_index])  
 l1\_index += 1  
 else:  
 sorted\_list.append(list2[l2\_index])  
 l2\_index += 1  
 elif l1\_index == l1\_length:  
 sorted\_list.append(list2[l2\_index])  
 l2\_index += 1  
 elif l2\_index == l2\_length:  
 sorted\_list.append(list1[l1\_index])  
 l1\_index += 1  
 return sorted\_list  
  
def merge\_sort(list):  
 if len(list) <= 1:  
 return list  
 mid = len(list) // 2  
 list1 = merge\_sort(list[:mid])  
 list2 = merge\_sort(list[mid:])  
 return merge(list1, list2)  
  
print(merge\_sort(s))  
otv = str(merge\_sort(s))  
y = open("../files/output.txt", "w")  
y.write(otv)  
  
stop = time.perf\_counter()  
print("time: %s ms" % (stop - start))

1. Текстовое объяснение решения.

Считываю данные из файла. Функция merge\_sort реализует алгоритм сортировки слиянием. Если длинна списка <= 1 , мы возвращаем список, в другом случае делим его пополам на 2 списка, к которым также применяем сортировку, а далее к этим спискам применяем функцию merge. Функция merge принимает два отсортированных списка и объединяет их в один. Функция сравнивает попарно по 1 элементу из каждого списка и добавляет сначала меньший элемент в список, а затем больший. С помощью индексов я отслеживаю текущие позиции в каждом из списков.

1. Результат работы кода на примерах:



1. Вывод по задаче:

В результате выполнения задачи я научилась писать алгоритм сортировки слиянием.

## Задача №2. Число инверсий

1. Текст задачи

Инверсией в последовательности чисел A называется такая ситуация, когда i < j, а Ai > Aj. Количество инверсий в последовательности в некотором ро- де определяет, насколько близка данная последовательность к отсортированной. Например, в сортированном массиве число инверсий равно 0, а в массиве, сор- тированном 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.

1. Листинг кода

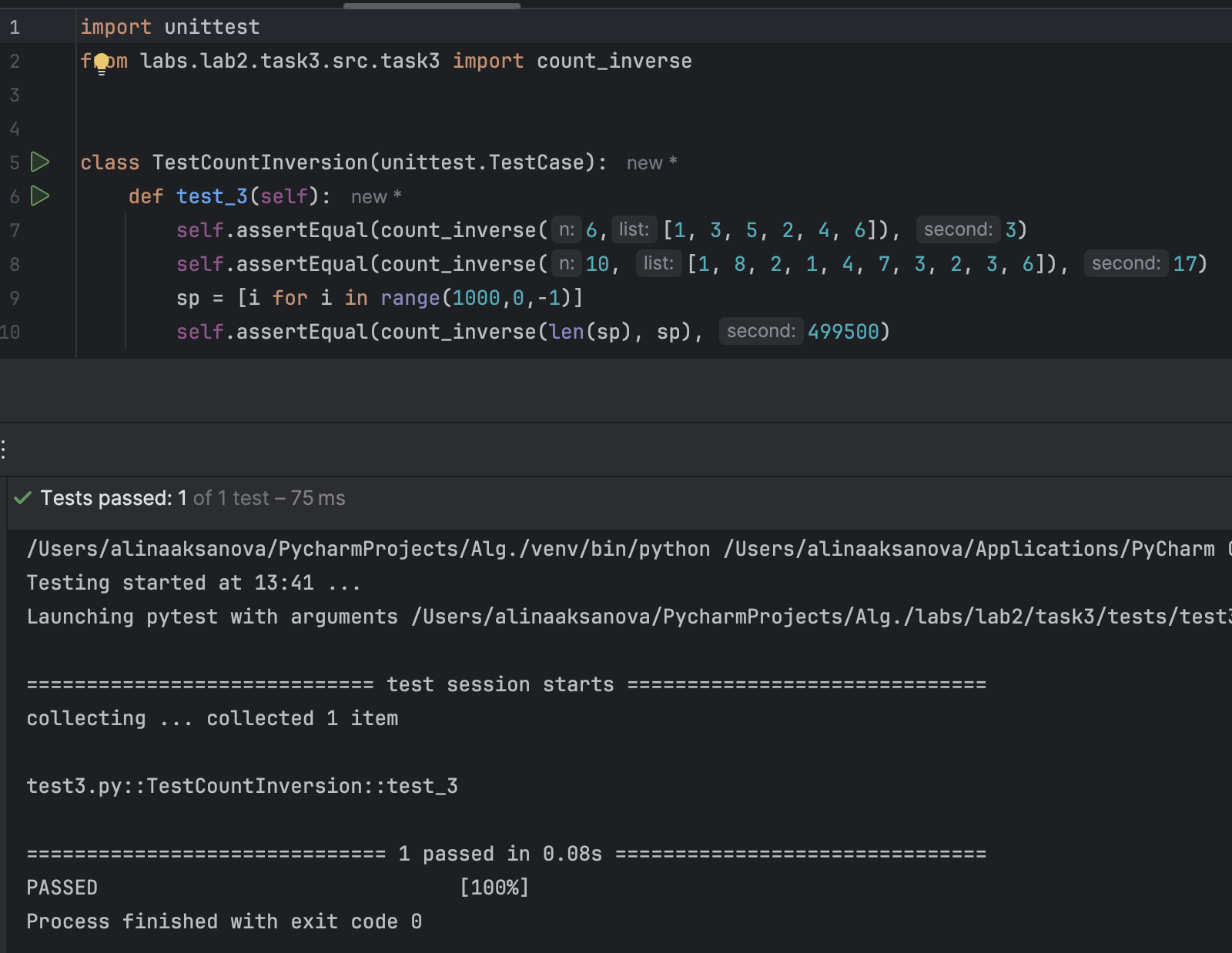
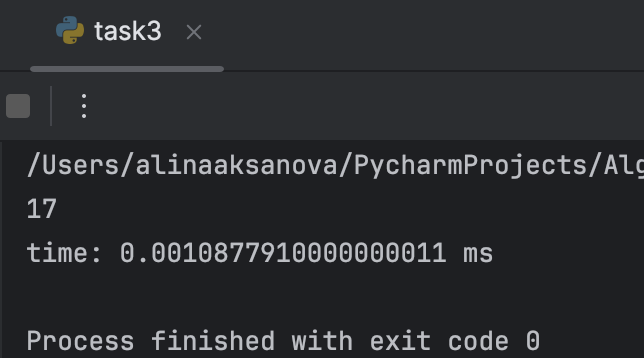
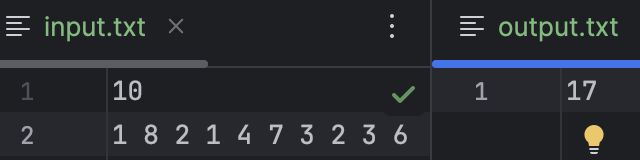
import time  
start = time.perf\_counter()  
  
k = open("../files/input.txt")  
n = int(k.readline())  
b = k.readline().split(" ")  
s = [int(l) for l in b ]  
  
def count\_inverse(n, list):  
 cnt = 0  
 ls = list  
 for i in range(len(ls)):  
 for j in range(len(ls)):  
 if ls[i]>ls[j] and i<j:  
 cnt+=1  
 return cnt  
  
otv = str(count\_inverse(n, s))  
print(otv)  
y = open("../files/output.txt", "w")  
y.write(otv)  
  
stop = time.perf\_counter()  
print("time: %s ms" % (stop - start))

1. Текстовое объяснение решения.

Считываю данные из файла. В функции прохожусь по элементам списка и сравниваю 2 из них. Если i-ый элемент больше j-го, и при этом i < j, то мы засчитываем инверсию, увеличивая счетчик.

В результате вывожу счетчик и записываю ответ в файл.

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я научилась писать алгоритм, определяющий число инверсий в списке.

## Задача №4. Бинарный поиск

1. 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 со- держится число n (1 ≤ n ≤ 105) — число элементов в массиве, и последо- вательность a0 < a1 < ... < an−1 из n **различных** положительных целых чисел в порядке возрастания, 1 ≤ ai ≤ 109 для всех 0 ≤ i < n. Следующая строка содержит число k, 1 ≤ k ≤ 105 и k положительных целых чисел b0,...bk−1,1≤bj ≤109 длявсех0≤j <k.

**Формат выходного файла (output.txt).** Для всех i от 0 до k − 1 вывести индекс0≤j ≤n−1,такойчтоai =bj или-1,еслитакогочиславмассиве нет.

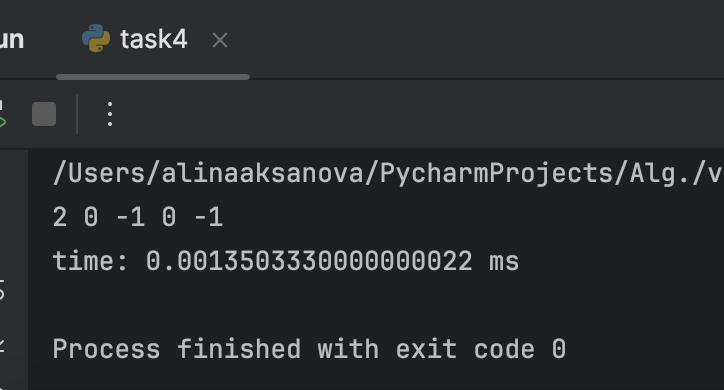
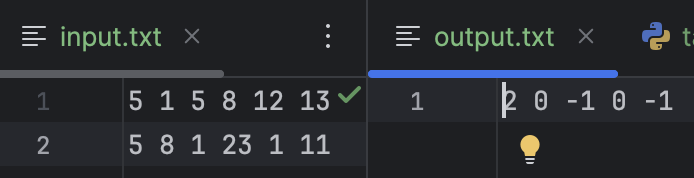
1. Листинг кода

import time  
  
start = time.perf\_counter()  
  
k = open("../files/input.txt")  
n, \*list = map(int, k.readline().split())  
k, \*searchs = map(int, k.readline().split())  
  
def binary\_search(list, search):  
 left, right = 0, len(list) - 1  
 while left<=right:  
 mid = left + (right - left) //2  
 if list[mid] == search:  
 return mid  
 elif list[mid] < search:  
 left = mid + 1  
 else:  
 right = mid - 1  
 return -1  
  
res = [binary\_search(list,search) for search in searchs ]  
res = " ".join(map(str,res))  
print(res)  
  
y = open("../files/output.txt", "w")  
y.write(res)  
  
stop = time.perf\_counter()  
print("time: %s ms" % (stop - start))

1. Текстовое объяснение решения.

Считываю данный из файла. Проводим поиск в 1 списке для всех элементов 2 списка поочередно. Делю отсортированный список на части и нахожу середину. Сравниваю середину с элементом, который мы ищем: если они равны то возвращаю значение середины, если середина меньше чем нужны элемент, продолжаем поиск в левой части элемента, в противном случае – в правой части. Если элемент не найден, записываю в строку -1. Если найден – записываю индекс этого элемента в 1 списке.

1. Результат работы кода на примерах из текста задачи:



1. Вывод по задаче:

В результате выполнения задачи я научилась писать алгоритм бинарного поиска элементов одного списка в другом.

## Задача №5 . Представитель большинства

1. Текст задачи

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.

Очевидно, время выполнения этого алгоритма квадратично. Ваша цель - использовать метод "Разделяй и властвуй"для разработки алгоритма проверки, содержится ли во входной последовательности элемент, который̆ 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.
* **Форматвыходногофайла(output.txt).**Выведите1,если во входной последовательности есть элемент, который встречается строго больше половины раз; в противном случае - 0.

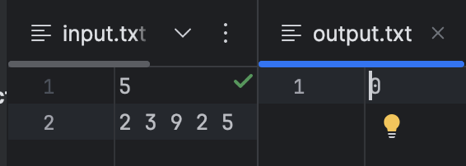
1. Листинг кода

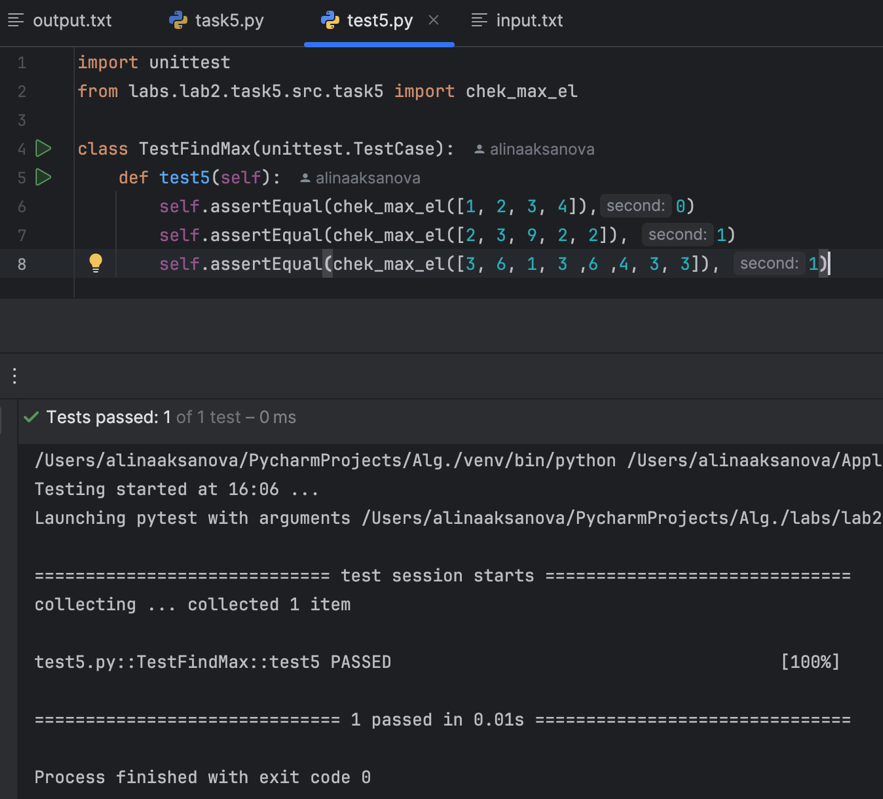
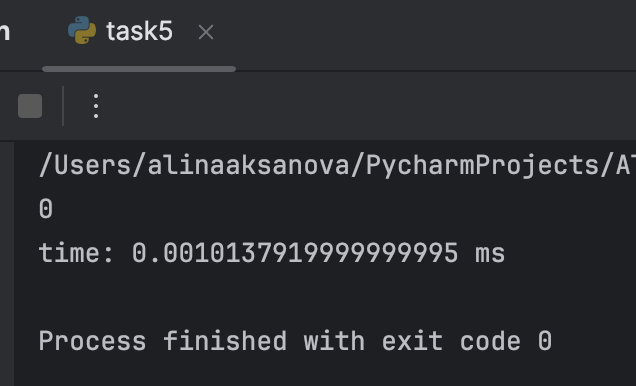
import time  
start = time.perf\_counter()  
  
k = open("../files/input.txt")  
n = int(k.readline())  
b = k.readline().split(" ")  
list1 = [int(l) for l in b ]  
  
def find\_max\_elem(list1, left, right):  
 if left == right:  
 return list1[left]  
  
 mid = (left + right) // 2  
 left\_elem = find\_max\_elem(list1, left, mid)  
 right\_elem = find\_max\_elem(list1, mid+1, right)  
  
 if left\_elem == right\_elem:  
 return left\_elem  
  
 left\_cnt = sum(1 for i in range(left, right + 1) if list1[i] == left\_elem)  
 right\_cnt = sum(1 for i in range(left, right +1 ) if list1[i] == right\_elem)  
  
 if left\_cnt > right\_cnt:  
 return left\_elem  
 else:  
 return right\_elem  
  
def if\_max(list1, elem):  
 if list1.count(elem) > n//2:  
 return True  
 else:  
 return False  
  
def chek\_max\_el(list1):  
 elem = find\_max\_elem(list1,0, len(list1)-1)  
 if if\_max(list1, elem):  
 return 1  
 else:  
 return 0  
  
otv = str(chek\_max\_el(list1))  
print(otv)  
  
y = open("../files/output.txt", "w")  
y.write(otv)  
  
stop = time.perf\_counter()  
print("time: %s ms" % (stop - start))

1. Текстовое объяснение решения.

Считываю данный из файла. функция find\_max\_elem ищет элемент, который может быть кандидатом на "максимальный". Нахожу середину диапазона и ищу элементы в левой и правой половинах. Если оба элемента равны, возвращаем один из них. Если они разные, считаю, сколько раз каждый из них встречается в общем диапазоне, и возвращаю элемент, который встречается чаще. Функция if\_max проверяет, является ли выбранный элемент большинством. Функция chek\_max\_elem проводит проверку с помощью двух прошлых функций и возвращает 1 если элемент большинство и 0 если нет.

1. Результат работы кода на примерах из текста задачи:





1. Вывод по задаче:

В результате выполнения задачи я научилась писать алгоритм поиска элемента, который является большинством.

## Задача №7 . Поиск максимального подмассива за линейное время

1. Текст задачи

Можно найти максимальный подмассив за линейное время, воспользовавшись следующими идеями. Начните с левого конца массива и двигайтесь вправо, отсле- живая найденный к данному моменту максимальный подмассив. Зная максималь- ный 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 ≤ i ≤ j + 1. Определите максимальный подмассив вида A[i..j + 1] за константное время, зная максимальный подмассив, заканчивающийся индексом j.

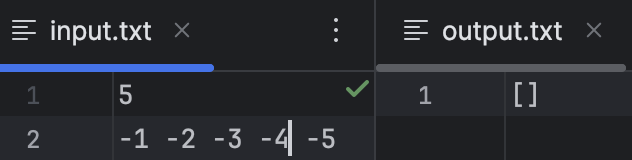
1. Листинг кода

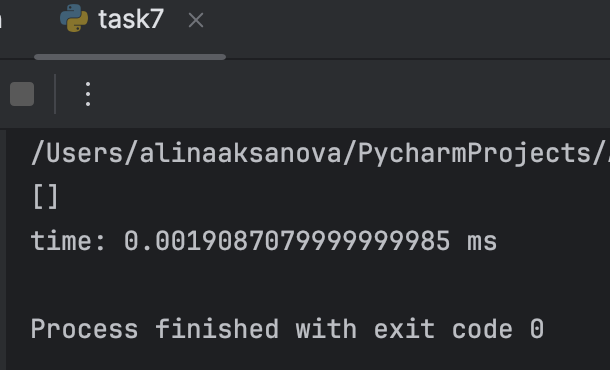
import time  
start = time.perf\_counter()  
  
k = open("../files/input.txt")  
n = int(k.readline())  
b = k.readline().split(" ")  
s = [int(l) for l in b ]  
  
def find\_max\_sum(list):  
 max\_sum = float("-inf")  
 summ, start\_index, end\_index = 0, 0, -1  
 temp\_start\_index = 0  
 for i in range(len(list)):  
 summ += list[i]  
 if summ > max\_sum:  
 max\_sum = summ  
 start\_index = temp\_start\_index  
 end\_index = i  
 if summ < 0:  
 summ = 0  
 temp\_start\_index = i + 1  
 if max\_sum < 0:  
 return []  
 return list[start\_index:end\_index+1]  
  
otv = str(find\_max\_sum(s))  
print(otv)  
  
y = open("../files/output.txt", "w")  
y.write(otv)  
  
stop = time.perf\_counter()  
print("time: %s ms" % (stop - start))

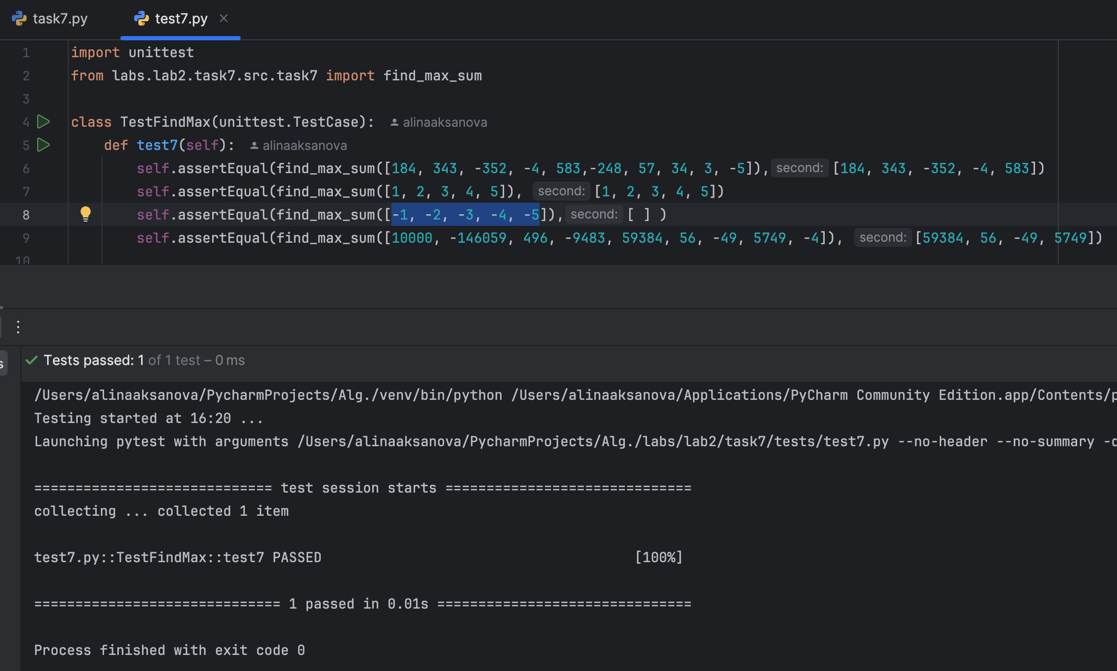
1. Текстовое объяснение решения.

Считываю данный из файла. Функция find\_max\_sum: прохожусь по всем элементам списка и добавляю их в сумму , если она больше чем максимальная, то обновляем значение максимума, если нет, просто переходим и рассчету суммы других элементов.

1. Результат работы кода на примерах из текста задачи:







1. Вывод по задаче:

В результате выполнения задачи я научилась писать алгоритм поиска максимального подмассива.

# Вывод

В результате лабораторной я написала несколько алгоритмов сортировки слиянием и изучила метод декомпозиции.