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 20

Выполнил:

Смирнов Георгий Валерьевич

К3239

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# 

# 

# 

# 

# 

# 

[**Задачи по варианту 3**](#_3rdcrjn)

[**Задача. №1 Сортировка слиянием. 3**](#_26in1rg)

[**Задача №5 Представитель большинства. 7**](#_lnxbz9)

[**Задача №7 Поиск максимального подмассива за линейное время. 10**](#_35nkun2)

[**Вывод 13**](#_1ksv4uv)

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

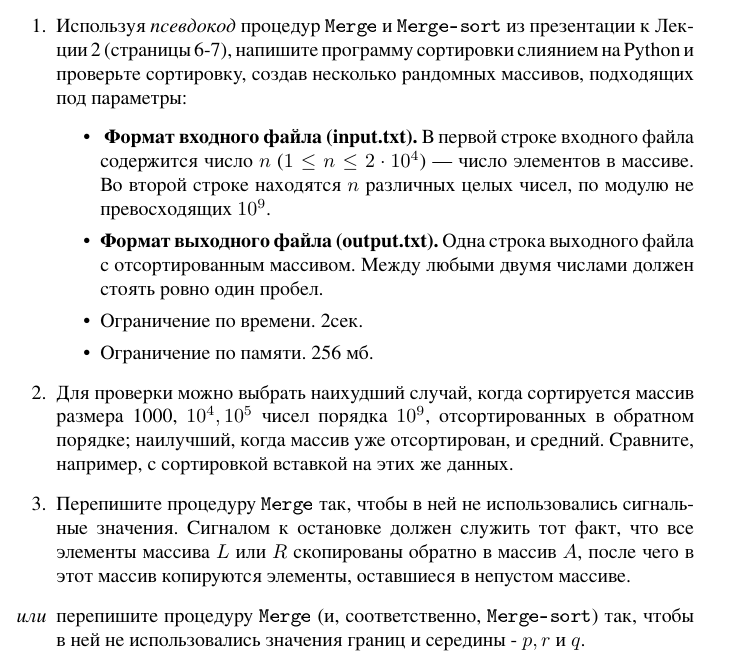
# 

# 

# 

# Задачи по варианту

## Задача. №1 Сортировка слиянием.



**Код программы**

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

# Создаем временные массивы

L = arr[left:mid+1]

R = arr[mid+1:right+1]

i = j = 0

k = left

# Слияние временных массивов обратно в arr

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

# Копируем оставшиеся элементы L, если есть

while i < len(L):

arr[k] = L[i]

i += 1

k += 1

# Копируем оставшиеся элементы R, если есть

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

# Сортируем первую и вторую половины

merge\_sort(arr, left, mid)

merge\_sort(arr, mid + 1, right)

# Сливаем отсортированные половины

merge(arr, left, mid, right)

# Чтение из файла input.txt

with open('input.txt', 'r') as file:

n = int(file.readline().strip())

array = list(map(int, file.readline().strip().split()))

# Сортировка массива

merge\_sort(array, 0, n - 1)

# Запись в файл output.txt

with open('output.txt', 'w') as file:

file.write(' '.join(map(str, array)))

Результат работы кода на примерах из текста задачи:

1)

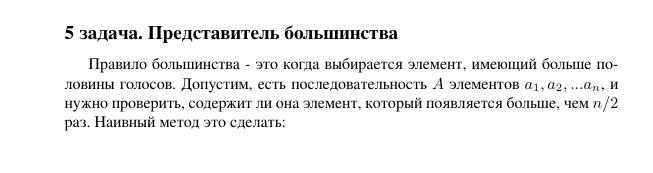


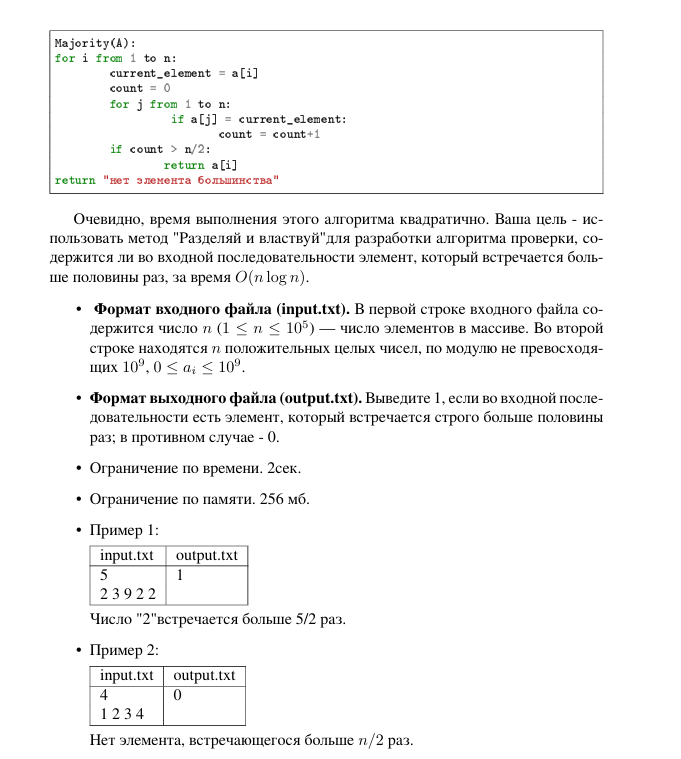


|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.000984200 | 0.001284 |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

## Задача №5 Представитель большинства.





**Код программы**

import time

import tracemalloc

def find\_majority\_element(arr):

def majority\_element\_rec(start, end):

if start == end:

return arr[start]

mid = (end - start) // 2 + start

left = majority\_element\_rec(start, mid)

right = majority\_element\_rec(mid + 1, end)

if left == right:

return left

left\_count = sum(1 for i in range(start, end + 1) if arr[i] == left)

right\_count = sum(1 for i in range(start, end + 1) if arr[i] == right)

return left if left\_count > right\_count else right

candidate = majority\_element\_rec(0, len(arr) - 1)

count = sum(1 for x in arr if x == candidate)

return candidate if count > len(arr) // 2 else None

def main():

# Начало отслеживания времени и памяти

start\_time = time.perf\_counter()

tracemalloc.start()

# Чтение данных из файла input.txt

with open('input.txt', 'r') as file:

n = int(file.readline().strip())

array = list(map(int, file.readline().strip().split()))

# Поиск элемента большинства

majority\_element = find\_majority\_element(array)

# Запись результата в файл output.txt

with open('output.txt', 'w') as file:

file.write('1' if majority\_element is not None else '0')

# Подсчет времени и памяти

end\_time = time.perf\_counter()

current, peak = tracemalloc.get\_traced\_memory()

tracemalloc.stop()

print(f"Время выполнения: {end\_time - start\_time:.9f} секунд")

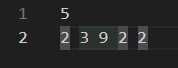
print(f"Использование памяти: {current / 10\*\*6:.6f} MB; Пиковое использование: {peak / 10\*\*6:.6f} MB")

if \_\_name\_\_ == "\_\_main\_\_":

main()

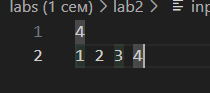
Результат работы кода на примерах из текста задачи:

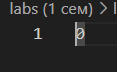
1)





2)

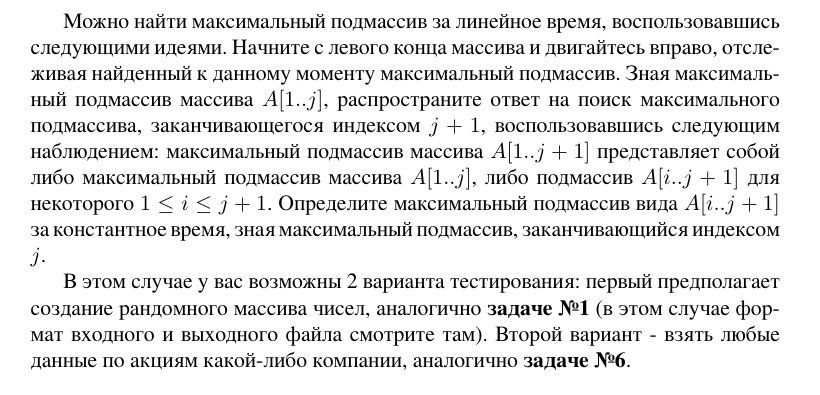




|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.005122000 | 0.001524 MB |
|  | 0.004328200 | 0.001524 |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

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



**Код программы**

ddef max\_subarray(arr):

max\_ending\_here = max\_so\_far = arr[0]

for x in arr[1:]:

max\_ending\_here = max(x, max\_ending\_here + x)

max\_so\_far = max(max\_so\_far, max\_ending\_here)

return max\_so\_far

def main():

# Чтение данных из файла input.txt

with open('input.txt', 'r') as file:

n = int(file.readline().strip())

array = list(map(int, file.readline().strip().split()))

# Поиск максимального подмассива

result = max\_subarray(array)

# Запись результата в файл output.txt

with open('output.txt', 'w') as file:

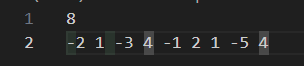
file.write(str(result))

if \_\_name\_\_ == "\_\_main\_\_":

main()

Результат работы кода на примерах из текста задачи:

1)





# Вывод

В ходе данной лабораторной работы я научился решать задачи. Написанные программы были протестированы, а также были измерены потребляемый ими объём памяти и время работы. Все программы работаю корректно и укладываются в установленные ограничения по времени и памяти на примерах из задач.