САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Алгоритмы и структуры данных»

Тема: Лабораторная работа 2.

Вариант 21

Выполнил:

Ступичев М. Н.

К3139

Проверила:

…

Санкт-Петербург

2024 г.

# Содержание отчета

[Содержание отчета 2](#_Toc183360087)

[Задачи по варианту 3](#_Toc183360088)

[Задача №1. Сортировка слиянием. 3](#_Toc183360089)

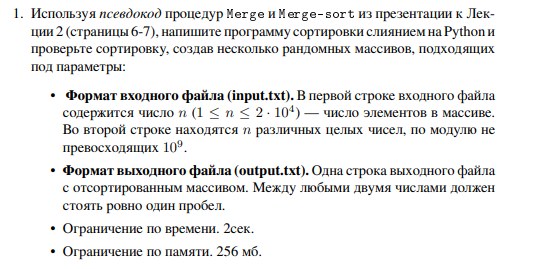
[Задача №5. Представитель большинства. 4](#_Toc183360090)

[Задача №8. Умножение многочленов. 6](#_Toc183360091)

[Вывод 7](#_Toc183360092)

# Задачи по варианту

## Задача №1. Сортировка слиянием.



Решение:

def merge\_sort(arr):

   if len(arr) <= 1:

       return arr

​

   mid = len(arr) // 2

   left\_half = arr[:mid]

   right\_half = arr[mid:]

​

   left\_half = merge\_sort(left\_half)

   right\_half = merge\_sort(right\_half)

​

   return merge(left\_half, right\_half)

​

def merge(left, right):

   merged = []

   i = j = 0

​

   while i < len(left) and j < len(right):

       if left[i] <= right[j]:

           merged.append(left[i])

           i += 1

       else:

           merged.append(right[j])

           j += 1

​

   merged.extend(left[i:])

   merged.extend(right[j:])

   return merged

​

​

with open('C:/Users/Михаил/PycharmProjects/algorithm\_and\_data\_structures/lab2/task1/tests/input', 'r') as f:

   n = int(f.readline())

   arr = list(map(int, f.readline().split()))

​

sorted\_arr = merge\_sort(arr)

​

with open('C:/Users/Михаил/PycharmProjects/algorithm\_and\_data\_structures/lab2/task1/tests/output', 'w') as f:

   f.write(' '.join(map(str, sorted\_arr)))

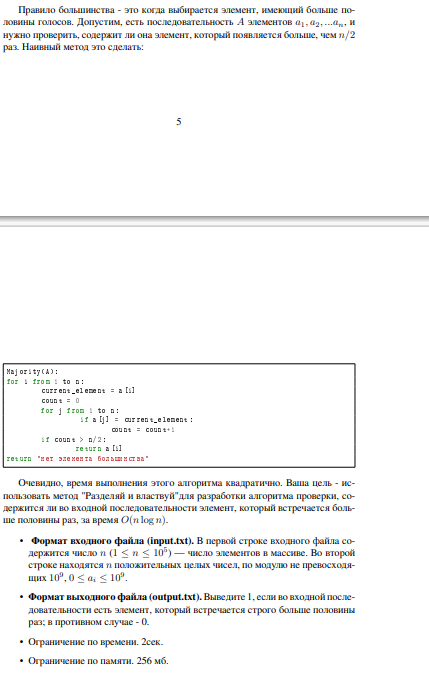
​Текстовое объяснение решения.

Этот код реализует алгоритм сортировки слиянием. Функция merge\_sort рекурсивно делит массив на две половины до тех пор, пока каждая половина не будет содержать один или ноль элементов (базовый случай рекурсии). Затем функция merge сливает две отсортированные половины в одну отсортированную последовательность, сравнивая элементы из каждой половины и добавляя меньший элемент в результирующий массив. После рекурсивного деления и слияния, весь исходный массив оказывается отсортированным. Код также включает чтение данных из входного файла и запись отсортированного массива в выходной файл.

Вывод по задаче:

Реализован стандартный алгоритм сортировки слиянием.

## Задача №5. Представитель большинства.



Решение:

def majority\_element(arr):

   if not arr:

       return None

   if len(arr) == 1:

       return arr[0]

​

   mid = len(arr) // 2

   left\_majority = majority\_element(arr[:mid])

   right\_majority = majority\_element(arr[mid:])

​

   if left\_majority == right\_majority:

       return left\_majority

​

   count\_left = arr.count(left\_majority) if left\_majority else 0

   count\_right = arr.count(right\_majority) if right\_majority else 0

​

   if count\_left > len(arr) // 2:

       return left\_majority

   elif count\_right > len(arr) // 2:

       return right\_majority

   else:

       return None

​

​

with open('C:/Users/Михаил/PycharmProjects/algorithm\_and\_data\_structures/lab2/task5/tests/input', 'r') as f:

   n = int(f.readline())

   arr = list(map(int, f.readline().split()))

​

ans = None

if majority\_element(arr) is not None:

   ans = 1

else:

   ans = 0

​

with open('C:/Users/Михаил/PycharmProjects/algorithm\_and\_data\_structures/lab2/task5/tests/output', 'w') as f:

   f.write(str(ans))

​​

​

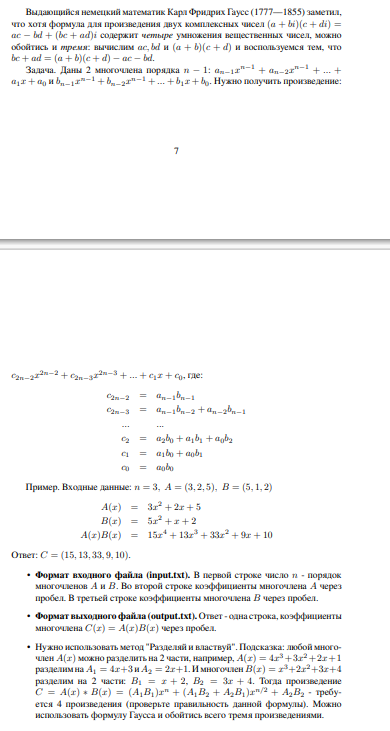
Текстовое объяснение решения.

Этот код рекурсивно ищет элемент большинства в массиве (элемент, встречающийся более чем в половине элементов массива). Функция majority\_element рекурсивно делит массив пополам, вызывая себя для каждой половины. Если элемент большинства найден в обеих половинах, то он и является элементом большинства для всего массива. Если элементы большинства в половинах разные, то подсчитывается количество вхождений каждого из них во всем массиве. Тот элемент, количество вхождений которого превышает половину длины массива, возвращается как результат. Если ни один из элементов не удовлетворяет этому условию, то возвращается None. Основной код считывает массив из файла, вызывает функцию majority\_element и записывает 1 в выходной файл, если элемент большинства найден, и 0 в противном случае.

Вывод по задаче:

Реализован алгоритм нахождения элемента в массиве, встречающегося более чем n/2 раз, если он там есть.

## Задача №8. Умножение многочленов.



Решение:

def multiply\_polynomials(a, b):

   n = len(a)

   if n == 1:

       return [a[0] \* b[0]]

​

   m = n // 2

   a0 = a[:m]

   a1 = a[m:]

   b0 = b[:m]

   b1 = b[m:]

​

   z0 = multiply\_polynomials(a0, b0)

   z2 = multiply\_polynomials(a1, b1)

   z1 = multiply\_polynomials(add\_polynomials(a0, a1), add\_polynomials(b0, b1))

​

   z1 = subtract\_polynomials(z1, z0)

   z1 = subtract\_polynomials(z1, z2)

​

   result = [0] \* (2 \* n - 1)

   for i in range(len(z0)):

       result[i] += z0[i]

   for i in range(len(z1)):

       result[i + m] += z1[i]

   for i in range(len(z2)):

       result[i + 2 \* m] += z2[i]

​

   return result

​

​

def add\_polynomials(a, b):

   max\_len = max(len(a), len(b))

   a += [0] \* (max\_len - len(a))

   b += [0] \* (max\_len - len(b))

   res = [0] \* max\_len

   for i in range(max\_len):

       res[i] = a[i] + b[i]

   return res

​

​

def subtract\_polynomials(a, b):

   max\_len = max(len(a), len(b))

   a += [0] \* (max\_len - len(a))

   b += [0] \* (max\_len - len(b))

   res = [0] \* max\_len

   for i in range(max\_len):

       res[i] = a[i] - b[i]

   return res

​

​

with open('C:/Users/Михаил/PycharmProjects/algorithm\_and\_data\_structures/lab2/task8/tests/input', 'r') as f:

   n = int(f.readline())

   koef\_first\_polynomial = list(map(int, f.readline().split()))

   koef\_second\_polynomial = list(map(int, f.readline().split()))

​

ans = multiply\_polynomials(koef\_first\_polynomial, koef\_second\_polynomial)

​

with open('C:/Users/Михаил/PycharmProjects/algorithm\_and\_data\_structures/lab2/task8/tests/output', 'w') as f:

   f.write(' '.join(map(str, ans)))

​​

Текстовое объяснение решения.

Этот код реализует алгоритм Карацубы для умножения многочленов. Функция multiply\_polynomials рекурсивно разбивает многочлены на две части, вычисляя три произведения: z0 (произведение низших степеней), z2 (произведение высших степеней) и z1 (произведение сумм низших и высших степеней). Затем, используя z0, z1 и z2, восстанавливает результат, используя формулу Карацубы. Функции add\_polynomials и subtract\_polynomials выполняют поразрядное сложение и вычитание многочленов соответственно. Код считывает коэффициенты двух многочленов из входного файла, вызывает функцию multiply\_polynomials для их умножения и записывает результат (коэффициенты результирующего многочлена) в выходной файл. Алгоритм Карацубы имеет асимптотическую сложность O(n^(log(n))), что быстрее, чем наивный алгоритм с O(n²).

Вывод по задаче:

Реализован алгоритм умножения двух многочленов одинаковой степени по методу “Разделяй и властвуй”.

# Вывод

Решены 3 задачи из 21 варианта.