

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: Сортировка слиянием, Метод декомпозиции  
Вариант 22

Выполнил:  
Чан Тхи Лиен  
К3140

Проверил:  
Афанасьев А.В.

Санкт-Петербург  
2024 г.

## Содержание отчета

<b>Содержание отчета</b>	<b>2</b>
<b>Задачи по варианту</b>	
Задание 1. Сортировка слиянием	<b>3</b>
Задание 5. Представитель большинства	<b>6</b>
Задание 9. Метод Штрассена для умножения матриц	<b>9</b>
<b>Дополнительные задачи</b>	
Задание 3. Число инверсий	<b>13</b>
Задание 4. Бинарный поиск	<b>15</b>
Задание 8. Умножение многочленов	<b>17</b>
<b>Вывод</b>	<b>19</b>

## Задачи по варианту

### Задание 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 случайных 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  $q$ .

```
> Task1 > Task1.py > ...
# Merge Sort in python
from time import perf_counter
def mergeSort(array):
    if len(array) == 1:
        return array
    else:
        r = len(array)//2
        L = array[:r]
        M = array[r:]

        mergeSort(L)
        mergeSort(M)
```

```

i = j = k = 0

while i < len(L) and j < len(M):
    if L[i] < M[j]:
        array[k] = L[i]
        i += 1
    else:
        array[k] = M[j]
        j += 1
    k += 1

while i < len(L):
    array[k] = L[i]
    i += 1
    k += 1

while j < len(M):
    array[k] = M[j]
    j += 1
    k += 1

return array

```

1. Импортировать библиотеку **time**.
2. Использую **def** для создания функции.
  - проверить количество элементов массива и рекурсивно разделить массив на две части.
  - сравнить элементы двух подмассивов, упорядочить и объединить два массива.
3. Использовать **with open** чтобы открыть файл и **readline** для чтения каждой строки в файле.

```

t1_start = perf_counter()
if __name__ == '__main__':

    with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task1\\input.txt', 'r') as infile:
        n = int(infile.readline())
        data = str(infile.readline())
        data = data.split(' ')
        arr = [int(i) for i in data]

    mergeSort(arr)
    result = ' '.join(str(i) for i in arr)
    with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task1\\output.txt', 'w') as outfile:
        outfile.write(result)

t1_stop = perf_counter()
print('Время работы: %s секунд' % (t1_stop - t1_start))

```

4. Метод **split()** - разделить строку на массив.
5. Метод **join()** - собирает строку из элементов списка.
6. Использовать **with open** чтобы открыть файл и **write** для записи результатов в файле.

## Результат

Лаб\_2 > Task1 > ≡ input.txt

```

1 7
2 19 34 5 17 9 43 10

```

Лаб\_2 > Task1 > ≡ output.txt

```

1 5 9 10 17 19 34 43

```

Время работы: 0.007456300008925609 секунд

7. Записать файл для тест.

```

Лаб_2 > Task1 > test_MergeSort.py > ...
1 import unittest
2 import random
3
4 > def merge_sort(arr): ...
33
34 class TestMergeSort(unittest.TestCase):
35     def test_sort_large_array(self):
36         large_array = [random.randint(1,10000) for _ in range(10000)]
37         sorted_array = sorted(large_array)
38         merge_sort(large_array)
39         self.assertEqual(large_array, sorted_array)
40
41 if __name__ == '__main__':
42     unittest.main()

```

## Задание 5. Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность  $A$  элементов  $a_1, a_2, \dots, a_n$ , и нужно проверить, содержит ли она элемент, который появляется больше, чем  $n/2$  раз. Наивный метод это сделать:

```
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 "нет элемента большинства"
```

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  $n$  ( $1 \leq n \leq 10^5$ ) — число элементов в массиве. Во второй строке находятся  $n$  положительных целых чисел, по модулю не превосходящих  $10^9$ ,  $0 \leq a_i \leq 10^9$ .
- **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.

Task5 > Task5.py > ...

```
from time import perf_counter

def find_mode(arr, left, right):
    if left > right:
        return None, 0
    if left == right:
        return arr[left], 1

    mid = (left + right) // 2
    left_mode, left_count = find_mode(arr, left, mid)
    right_mode, right_count = find_mode(arr, mid + 1, right)

    total_left = count_mode(arr, left_mode, left, right)
    total_right = count_mode(arr, right_mode, left, right)

    if total_left > total_right:
        return left_mode, total_left
    else:
        return right_mode, total_right

def count_mode(arr, element, left, right):
    count = 0
    for i in range(left, right + 1):
        if arr[i] == element:
            count += 1
    return count
```

1. Импортировать библиотеку **time**.
2. Создать 2 функции:
  - функция **“find\_mode”**:
    - Рекурсивно разделить массив на две половины, чтобы найти моду в каждой половине.
    - использовать функцию **“count\_mode”** подсчета количества появления моды.
    - Сравнить количество вхождений моды в каждой половине и найдите моду массива.

- функция “count\_mode”: Подсчитать количество вхождений элемента

```
def print_result(n):
    if n > len(array) // 2:
        return str(1)
    else:
        return str(0)
# result
t1_start = perf_counter()

if __name__ == "__main__":
    with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task5\\input.txt', 'r') as infile:
        num = int(infile.readline())
        data = str(infile.readline())
    data = data.split(' ')
    array = [int(i) for i in data]
    result = find_mode(array, 0, len(array)-1)
    n = result[1]
    with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task5\\output.txt', 'w') as outfile:
        outfile.write(print_result(n))

t1_stop = perf_counter()
print('Время работы: %s секунд' % (t1_stop - t1_start))
```

3. Сравнить количество вхождений моды с количеством элементов в массиве.
  - если количество вхождений больше половины, печать 1.
  - если нет, печать 0.
4. Использовать **with open** чтобы открыть файл и **readline** для чтения каждой строки в файле.
5. Метод **split()** - разделить строку на массив.
6. Метод **join()** - собирает строку из элементов списка.
7. Использовать **with open** чтобы открыть файл и **write** для записи результатов в файл.

## Результат

```
Лаб_2 > Task5 > ≡ input.txt
1 11
2 2 7 2 10 2 6 2 3 2 5 2
3
```

```
Лаб_2 > Task5 > ≡ output.txt
1 1
```

Время работы: 0.002512500010197982 секунд

```
Лаб_2 > Task5 > ≡ input.txt
1 11
2 2 7 2 10 2 6 2 3 2 5 4
3
```

```
Лаб_2 > Task5 > ≡ output.txt
1 0
```

Время работы: 0.005535399992368184 секунд



## Задание 9. Метод Штрассена для умножения матриц

**Умножение матриц. Простой метод.** Если есть квадратные матрицы  $X = (x_{ij})$  и  $Y = (y_{ij})$ , то их произведение  $Z = X \cdot Y \Rightarrow z_{ij} = \sum_{k=1}^n x_{ik} \cdot y_{kj}$ . Нужно вычислить  $n^2$  элементов матрицы, каждый из которых представляет собой сумму  $n$  значений.

```
Matrix_Multiply(X, Y)::
    n = X.rows
    Z - квадратная матрица размера n
    for i = 1 to n:
        for j = 1 to n:
            z[i,j] = 0
            for k = 1 to n:
                z[i,j] = z[i,j] + x[i,k]*y[k,j]
    return Z
```

- **Цель.** Применить метод Штрассена для умножения матриц и сравнить его с простым методом. *Найти размер матриц  $n$ , при котором метод Штрассена работает существенно быстрее простого метода.*
- **Формат входа.** Стандартный ввод или input.txt. Первая строка - размер **квадратных** матриц  $n$  для умножения. Следующие строки соответственно сами значения матриц  $A$  и  $B$ .
- **Формат выхода.** Стандартный вывод или output.txt. Матрица  $C = A \cdot B$ .

```
Лаб_2 > Task9 > Task9.py > next_size
1  from time import perf_counter
2  from math import *
3  def add_matrices(A, B):
4
5      return [[A[i][j] + B[i][j] for j in range(len(A[0]))] for i in range(len(A))]
6
7  def subtract_matrices(A, B):
8
9      return [[A[i][j] - B[i][j] for j in range(len(A[0]))] for i in range(len(A))]
```

1. Импортировать библиотеку **time**, **math**.
2. Создать две функции для умножения и вычитания двух матриц.

```

11 def strassen(A, B):
12     n = len(A)
13
14     if n == 1:
15         return [[A[0][0] * B[0][0]]]
16
17     mid = n // 2
18     A11 = [row[:mid] for row in A[:mid]]
19     A12 = [row[mid:] for row in A[:mid]]
20     A21 = [row[:mid] for row in A[mid:]]
21     A22 = [row[mid:] for row in A[mid:]]
22
23     B11 = [row[:mid] for row in B[:mid]]
24     B12 = [row[mid:] for row in B[:mid]]
25     B21 = [row[:mid] for row in B[mid:]]
26     B22 = [row[mid:] for row in B[mid:]]
27
28
29     P1 = strassen(A11, subtract_matrices(B12, B22))
30     P2 = strassen(add_matrices(A11, A12), B22)
31     P3 = strassen(add_matrices(A21, A22), B11)
32     P4 = strassen(A22, subtract_matrices(B21, B11))
33     P5 = strassen(add_matrices(A11, A22), add_matrices(B11, B22))
34     P6 = strassen(subtract_matrices(A12, A22), add_matrices(B21, B22))
35     P7 = strassen(subtract_matrices(A11, A21), add_matrices(B11, B12))
36
37
38     C11 = add_matrices(subtract_matrices(add_matrices(P5, P4), P2), P6)
39     C12 = add_matrices(P1, P2)
40     C21 = add_matrices(P3, P4)
41     C22 = subtract_matrices(add_matrices(P5, P1), add_matrices(P3, P7))
42
43     # result
44     C = []
45     for i in range(mid):
46         C.append(C11[i] + C12[i])
47     for i in range(mid):
48         C.append(C21[i] + C22[i])
49
50     return C

```

3. Создать функцию “strassen” для умножения двух матриц:
  - Рекурсивно разделить и умножить матриц и вычислить **P1 - P7**.
  - использовать **P1 - P7** для создания результата матрицы **C**.

```

52 def next_size(n):
53     num = floor(log2(n))
54     if n == pow(2, num):
55         size = n
56         return size
57     else:
58         size = int(pow(2, num+1))
59         return size
60
61 def pad_matrix(A, size):
62     padded = [[0] * size for _ in range(size)]
63     for i in range(len(A)):
64         for j in range(len(A[i])):
65             padded[i][j] = A[i][j]
66     return padded

```

4. Функция “next\_size” используется для проверки того, является ли степень матрицы степенью 2 или нет.
  - Если верно, не изменить степень матриц
  - Если нет, увеличить степень матрицы до ближайшей степени по основанию 2.
5. Функция “pad\_matrix” для добавления элемента 0 в недостающую позицию матриц A и B.

```

68 t1_start = perf_counter()
69
70 if __name__ == "__main__":
71     with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лабо - Copy\\Лабо_2\\Task9\\input.txt', 'r') as infile:
72         n = int(infile.readline())
73         data = infile.readline().split()
74         A_values = list(map(int, data[:n*n]))
75         A = [A_values[i*n:(i+1)*n] for i in range(n)]
76         B_values = list(map(int, data[n*n:]))
77         B = [B_values[i*n:(i+1)*n] for i in range(n)]
78
79         size = next_size(n)
80         A_padded = pad_matrix(A, size)
81         B_padded = pad_matrix(B, size)
82         C_padded = strassen(A_padded, B_padded)
83         result = [row[:n] for row in C_padded[:n]]
84         # print(result)
85
86         with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лабо - Copy\\Лабо_2\\Task9\\output.txt', 'w') as outfile:
87             for row in result:
88                 outfile.write(' '.join(map(str, row)) + '\n')
89
90     t1_stop = perf_counter()
91     print('Время работы: %s секунд' % (t1_stop - t1_start))

```

6. Использовать **with open** чтобы открыть файл и **write** для записать результаты в файле.
7. после получения результата мы удаляем 0 элементов, потому что мы добавили их в матрицы A и B

## Результат

```
Лаб_2 > Task9 > ≡ input.txt
```

```
1 3
2 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
```

```
Лаб_2 > Task9 > ≡ output.txt
```

```
1 30 36 42
2 66 81 96
3 102 126 150
4
```

```
Время работы: 0.002738000001045293 секунд
```

```
Лаб_2 > Task9 > ≡ input.txt
```

```
1 2
2 1 2 3 4 5 6 7 8
```

```
Лаб_2 > Task9 > ≡ output.txt
```

```
1 19 22
2 43 50
3
```

```
Время работы: 0.0021622000003844732 секунд
```

## Дополнительные задачи

### Задание 3. Число инверсий

Инверсией в последовательности чисел  $A$  называется такая ситуация, когда  $i < j$ , а  $A_i > A_j$ . 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 содержится число  $n$  ( $1 \leq n \leq 10^5$ ) — число элементов в массиве. Во второй строке находятся  $n$  различных целых чисел, по модулю не превосходящих  $10^9$ .
- **Формат выходного файла (output.txt).** В выходной файл надо вывести число инверсий в массиве.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

```
Task3 > Task3.py > ...
# Bubble sort in Python
#function
from time import perf_counter
def bubbleSort(array):
    if len(array)<=1:
        return 0

    num = 0

    for i in range(len(array)):
        for j in range(0, len(array) - i - 1):
            if array[j] > array[j + 1]:
                temp = array[j]
                array[j] = array[j+1]
                array[j+1] = temp
                num = num+1
    return str(num)
```

1. Импортировать библиотеку **time**.
2. Использовать функцию “**bubbleSort**”:

- Сравнит два соседних элемента.
- Если передний элемент больше заднего, поменять местами два элемента и увеличить **num** на 1.
- Если нет, оставить эти два элемента на месте.
- печать результат

```
#result
t1_start = perf_counter()
with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task3\\input.txt','r') as infile:
    n = int(infile.readline())
    data = str(infile.readline())

data = data.split(' ')
arr = [int(i) for i in data]

with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task3\\output.txt','w') as outfile:
    outfile.write(bubbleSort(arr))
t1_stop = perf_counter()
print('Время работы: %s секунд' % (t1_stop - t1_start))
```

3. Использовать **with open** чтобы открыть файл для чтения данных и записи результат.

```
Лаб_2 > Task3 > ≡ input.txt
1    10
2    1 8 2 1 4 7 3 2 3 6
```

```
Лаб_2 > Task3 > ≡ output.txt
1    17
```

```
Время работы: 0.0060917000082554296 секунд
```

## Задание 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  $a_0 < a_1 < \dots < a_{n-1}$  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  $b_0, \dots, b_{k-1}$ ,  $1 \leq b_j \leq 10^9$  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.

```
Лаб_2 > Task4 > Task4.py > ...
1  from time import perf_counter
2
3  def binary_search(arr, target):
4      left, right = 0, len(arr) - 1
5      while left <= right:
6          mid = left + (right - left) // 2
7          if arr[mid] == target:
8              return mid
9          elif arr[mid] < target:
10             left = mid + 1
11         else:
12             right = mid - 1
13     return -1
```

1. Импортировать библиотеку **time**.
2. Создать функцию “**binary\_search**” для поиска места элемента в массиве.
  - разделить массив на две части.
  - сравнить элемент, позицию которого нужно найти, со средним элементом массива.
  - Если равно, печать эту позицию.
  - Если больше, выполнить функцию со второй половиной массива.
  - Если меньше, выполнить функцию с первой половиной массива.
  - Если массив нет этого элемента, печать **-1**.

```

26 # result
27
28 t1_start = perf_counter()
29 if __name__ == '__main__':
30
31     with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task4\\input.txt', 'r') as infile:
32         n = int(infile.readline())
33         data1 = str(infile.readline()).split(' ')
34         k = int(infile.readline())
35         data2 = str(infile.readline()).split(' ')
36         list_to_search = [int(i) for i in data1]
37         list_to_find = [int(i) for i in data2]
38
39         array = find_positions(list_to_search, list_to_find)
40         result = ' '.join(str(i) for i in array)
41
42     with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task4\\output.txt', 'w') as outfile:
43         outfile.write(result)
44
45
46 t1_stop = perf_counter()
47 print('Время работы: %s секунд' % (t1_stop - t1_start))

```

3. Использовать **with open** чтобы открыть файл для чтения данных и записи результат.

Лаб\_2 > Task4 > ≡ input.txt

```

1 5
2 1 5 8 12 13
3 5
4 8 1 23 1 11

```

Лаб\_2 > Task4 > ≡ output.txt

```

1 2 0 -1 0 -1

```

Время работы: 0.0023622000007890165 секунд



## Задание 8. Умножение многочленов

Выдающийся немецкий математик Карл Фридрих Гаусс (1777—1855) заметил, что хотя формула для произведения двух комплексных чисел  $(a + bi)(c + di) = ac - bd + (bc + ad)i$  содержит *четыре* умножения вещественных чисел, можно обойтись и *тремя*: вычислим  $ac, bd$  и  $(a + b)(c + d)$  и воспользуемся тем, что  $bc + ad = (a + b)(c + d) - ac - bd$ .

Задача. Даны 2 многочлена порядка  $n - 1$ :  $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0$  и  $b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \dots + b_1x + b_0$ . Нужно получить произведение:

$c_{2n-2}x^{2n-2} + c_{2n-3}x^{2n-3} + \dots + c_1x + c_0$ , где:

$$\begin{aligned}c_{2n-2} &= a_{n-1}b_{n-1} \\c_{2n-3} &= a_{n-1}b_{n-2} + a_{n-2}b_{n-1} \\&\dots \dots \\c_2 &= a_2b_0 + a_1b_1 + a_0b_2 \\c_1 &= a_1b_0 + a_0b_1 \\c_0 &= a_0b_0\end{aligned}$$

Пример. Входные данные:  $n = 3$ ,  $A = (3, 2, 5)$ ,  $B = (5, 1, 2)$

$$\begin{aligned}A(x) &= 3x^2 + 2x + 5 \\B(x) &= 5x^2 + x + 2 \\A(x)B(x) &= 15x^4 + 13x^3 + 33x^2 + 9x + 10\end{aligned}$$

Ответ:  $C = (15, 13, 33, 9, 10)$ .

- **Формат входного файла (input.txt).** В первой строке число  $n$  - порядок многочленов  $A$  и  $B$ . Во второй строке коэффициенты многочлена  $A$  через пробел. В третьей строке коэффициенты многочлена  $B$  через пробел.
- **Формат выходного файла (output.txt).** Ответ - одна строка, коэффициенты многочлена  $C(x) = A(x)B(x)$  через пробел.
- Нужно использовать метод "Разделяй и властвуй". Подсказка: любой многочлен  $A(x)$  можно разделить на 2 части, например,  $A(x) = 4x^3 + 3x^2 + 2x + 1$  разделим на  $A_1 = 4x + 3$  и  $A_2 = 2x + 1$ . И многочлен  $B(x) = x^3 + 2x^2 + 3x + 4$  разделим на 2 части:  $B_1 = x + 2$ ,  $B_2 = 3x + 4$ . Тогда произведение  $C = A(x) * B(x) = (A_1B_1)x^n + (A_1B_2 + A_2B_1)x^{n/2} + A_2B_2$  - требуется 4 произведения (проверьте правильность данной формулы). Можно использовать формулу Гаусса и обойтись всего тремя произведениями.

```

Task8 > Task8.py > ...
def add_polynomials(A, B):

    return [a + b for a, b in zip(A, B)] + A[len(B):] + B[len(A):]

def multiplication(A, B):
    n = len(A)

    if n == 1:
        return [A[0] * B[0]]

    m = n // 2

    A1, A0 = A[:m], A[m:]
    B1, B0 = B[:m], B[m:]

    P1 = multiplication(A1, B1)
    P2 = multiplication(A0, B0)
    P3 = multiplication(add_polynomials(A1, A0), add_polynomials(B1, B0))

    result = [0] * (2 * n - 1)

    for i in range(len(P1)):
        result[i] += P1[i]

    for i in range(len(P2)):
        result[i + 2 * m] += P2[i]

    for i in range(len(P3)):
        result[i + m] += P3[i] - (P1[i] if i < len(P1) else 0) - (P2[i] if i < len(P2) else 0)

    return result

```

1. Создать функцию “**add\_polynomials**” для сложения двух многочленов.
  - Использовать функцию **zip**, чтобы объединить коэффициенты одинаковой степени из многочленов A и B.
  - Если многочлены разной длины, объединить оставшиеся коэффициенты из более длинного многочлена.
2. Создать функцию “**multiplication**” для умножения двух многочленов.
  - Если длина многочлена равна 1, умножить два коэффициента многочлена вместе.
  - Если больше 1, разделить каждый многочлен на две части.
  - Рекурсивно выполнить три умножения.
  - объединить результаты в список коэффициентов результирующего многочлена.

```
# result
if __name__ == "__main__":

    with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task8\\input.txt','r') as infile:
        n = int(infile.readline())
        data_a = str(infile.readline())
        data_b = str(infile.readline())

        A = [int(i) for i in data_a.split()]
        B = [int(i) for i in data_b.split()]

        arr = multiplication(A, B)
        result = ' '.join(str(i) for i in arr)

    with open('D:\\Lab CTDL - GT\\Чан Тхи Лиен_Лаб0 - Copy\\Лаб_2\\Task8\\output.txt', 'w') as outfile:
        outfile.write(result)
```

3. Использовать **with open** чтобы открыть файл для чтения данных и записи результат.

```
Лаб_2 > Task8 > ≡ input.txt
1      4
2      3 2 4 1
3      2 7 1 6
```

```
Лаб_2 > Task8 > ≡ output.txt
1      6 25 25 50 23 25 6
```

## Выход

- Знать, как использовать метод “**MergeSort**” для сортировки ряда чисел.
- Использовать методы “**декомпозиция**” для более простого решения заданий.
- Знакомство с умножении многочленов и матриц.