

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

Содержание отчета	2
Задачи по варианту	
Задание 1. Сортировка слиянием	3
Задание 5. Представитель большинства	6
Задание 9. Метод Штрассена для умножения матриц	9
Дополнительные задачи	
Задание 3. Число инверсий	13
Задание 4. Бинарный поиск	15
Задание 8. Умножение многочленов	18
Вывод	20

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 .

```
1  # Merge Sort in python
2  from time import perf_counter
3  def mergeSort(array): 5 usages 1 lien
4      if len(array) == 1:
5          return array
6      else:
7
8          r = len(array)//2
9          L = array[:r]
10         M = array[r:]
11
12         mergeSort(L)
13         mergeSort(M)
```

```

5         i = j = k = 0
6
7         while i < len(L) and j < len(M):
8             if L[i] < M[j]:
9                 array[k] = L[i]
10                i += 1
11            else:
12                array[k] = M[j]
13                j += 1
14            k += 1
15
16        while i < len(L):
17            array[k] = L[i]
18            i += 1
19            k += 1
20
21        while j < len(M):
22            array[k] = M[j]
23            j += 1
24            k += 1
25        return array
26

```

1. Импортировать библиотеку **time**.
2. Используя **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.


```

37     t1_start = perf_counter()
38     if __name__ == '__main__':
39
40         with open('input.txt', 'r') as infile:
41             n = int(infile.readline())
42             data = str(infile.readline())
43             data = data.split(' ')
44             arr = [int(i) for i in data]
45
46             mergeSort(arr)
47             result = ' '.join(str(i) for i in arr)
48             with open('output.txt', 'w') as outfile:
49                 outfile.write(result)
50
51     t1_stop = perf_counter()
52     print('Время работы: %s секунд' % (t1_stop - t1_start))
53

```

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 запись результаты в файле.

Результат

1	7	
2	19 34 5 17 9 43 11	
		
		1 5 9 11 17 19 34 43

Время работы: 0.0015571000003546942 секунд

7. Записать файл для тест.

```

1     import unittest
2     import random
3     from Ла6_2.Task1.Task1 import mergeSort
4
5     class TestMergeSort(unittest.TestCase):
6         def test_sort_large_array(self):
7             large_array = [random.randint(1,10000) for _ in range(10000)]
8             sorted_array = sorted(large_array)
9             mergeSort(large_array)
10            self.assertEqual(large_array, sorted_array)
11
12     if __name__ == '__main__':
13         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.

```

1  from time import perf_counter
2
3  def find_mode(arr, left, right): 3 usages 1 lien
4      if left > right:
5          return None, 0
6      if left == right:
7          return arr[left], 1
8
9      mid = (left + right) // 2
10     left_mode, left_count = find_mode(arr, left, mid)
11     right_mode, right_count = find_mode(arr, mid + 1, right)
12
13     total_left = count_mode(arr, left_mode, left, right)
14     total_right = count_mode(arr, right_mode, left, right)
15
16     if total_left > total_right:
17         return left_mode, total_left
18     else:
19         return right_mode, total_right
20
21 def count_mode(arr, element, left, right): 2 usages 1 lien
22     count = 0
23     for i in range(left, right + 1):
24         if arr[i] == element:
25             count += 1
26     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

```

29 def print_result(array, n): 1 usage 1 lien
30     if n>len(array)//2:
31         return str(1)
32     else:
33         return str(0)
34 # result
35 t1_start = perf_counter()
36
37 if __name__ == "__main__":
38     with open('input.txt', 'r') as infile:
39         num = int(infile.readline())
40         data = str(infile.readline())
41         data = data.split(' ')
42         array = [int(i) for i in data]
43         result = find_mode(array, left=0, len(array)-1)
44         n = result[1]
45         with open('output.txt', 'w') as outfile:
46             outfile.write(print_result(array, n))
47
48     t1_stop = perf_counter()
49     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 запись результаты в файле.

Результат

1	11		
2	2 7 2 10 2 6 2 3 2 5 2	1	1

Время работы: 0.0012900999990961282 секунд

1	11		
2	2 7 2 10 2 6 2 3 2 5 4	1	0

Время работы: 0.00121839999988337746 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 $C = A \cdot B$.

```
1  from time import perf_counter
2  from math import *
3  def add_matrices(A, B): 12 usages 1 lien
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): 6 usages 1 lien
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): 8 usages  ⓘ lien
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): 1 usage 1 lien
    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): 2 usages 1 lien
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
67
68  t1_start = perf_counter()

```

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('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('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 запись результаты в файле.
7. после получения результата мы удаляем 0 элементов, потому что мы добавили их в матрицы A и B

Результат

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

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

Время работы: 0.001648699999350356 секунд

1	2
2	1 2 3 4 5 6 7 8

1	19 22
2	43 50

Время работы: 0.0012888000001112232 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 отсортированном массиве число инверсий равно 0, а в массиве, отсортированном 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.

```
1  # Bubble sort in Python
2  #function
3  from time import perf_counter
4  def bubbleSort(array): 1 usage  1 lien
5      if len(array)<=1:
6          return 0
7
8      num = 0
9
10     for i in range(len(array)):
11
12         for j in range(0, len(array) - i - 1):
13
14             if array[j] > array[j + 1]:
15                 temp = array[j]
16                 array[j] = array[j+1]
17                 array[j+1] = temp
18                 num = num+1
19     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

```

21     #result
22     t1_start = perf_counter()
23     with open('input.txt','r') as infile:
24         n = int(infile.readline())
25         data = str(infile.readline())
26
27     data = data.split(' ')
28     arr = [int(i) for i in data]
29
30
31     with open('output.txt', 'w') as outfile:
32         outfile.write(bubbleSort(arr))
33     t1_stop = perf_counter()
34     print('Время работы: %s секунд' % (t1_stop - t1_start))

```

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

1	10	
2	1 8 2 1 4 7 3 2 3 6	1 17

Время работы: 0.001406800001859665 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.

```
1  from time import perf_counter
2
3  def binary_search(arr, target): 1 usage 2 lien
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**.

3. Создать функцию “**find_postions**” для поиска позиции каждого элемента

```
15 def find_positions(list_to_search, list_to_find): 1 usage 2 lien
16
17     sorted_list = sorted(list_to_search)
18     positions = []
19
20     for item in list_to_find:
21         pos = binary_search(sorted_list, item)
22         positions.append(pos)
23
24     return positions
25
```

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

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

    with open('input.txt', 'r') as infile:
        n = int(infile.readline())
        data1 = str(infile.readline()).split(' ')
        k = int(infile.readline())
        data2 = str(infile.readline()).split(' ')
        list_to_search = [int(i) for i in data1]
        list_to_find = [int(i) for i in data2]

        array = find_positions(list_to_search, list_to_find)
        result = ' '.join(str(i) for i in array)

    with open('output.txt', 'w') as outfile:
        outfile.write(result)

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


Результат

1	5				
2	1	5	8	12	13
3	5				
4	8	1	23	1	11
		1	2	0	-1

Время работы: 0.0011126000026706606 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.

```

1  def add_polynomials(A, B): 2 usages  ⓘ lien
2
3      return [a + b for a, b in zip(A, B)] + A[len(B):] + B[len(A):]
4
5
6  def multiplication(A, B): 4 usages  ⓘ lien
7      n = len(A)
8
9      if n == 1:
10         return [A[0] * B[0]]
11
12     m = n // 2
13
14     A1, A0 = A[:m], A[m:]
15     B1, B0 = B[:m], B[m:]
16
17     P1 = multiplication(A1, B1)
18     P2 = multiplication(A0, B0)
19     P3 = multiplication(add_polynomials(A1, A0), add_polynomials(B1, B0))
20
21     result = [0] * (2 * n - 1)
22
23     for i in range(len(P1)):
24         result[i] += P1[i]
25
26     for i in range(len(P2)):
27         result[i + 2 * m] += P2[i]
28
29     for i in range(len(P3)):
30         result[i + m] += P3[i] - (P1[i] if i < len(P1) else 0) - (P2[i] if i < len(P2) else 0)
31
32     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.

```

35 ▶ if __name__ == "__main__":
36
37     with open('input.txt', 'r') as infile:
38         n = int(infile.readline())
39         data_a = str(infile.readline())
40         data_b = str(infile.readline())
41
42     A = [int(i) for i in data_a.split()]
43     B = [int(i) for i in data_b.split()]
44
45     arr = multiplication(A, B)
46     result = ' '.join(str(i) for i in arr)
47
48     with open('output.txt', 'w') as outfile:
49         outfile.write(result)
50

```

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

1	4	
2	3 2 4 1	
3	2 7 1 6	
		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.