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

Выполнил:
Нгуен Хыу Жанг
К3140

Проверила:
Афанасьев А.В

Санкт-Петербург
2024 г

Содержание

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 рандомных массивов, подходящих под параметры:

   - **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \le n \le 2 \cdot 10^4$) — 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.

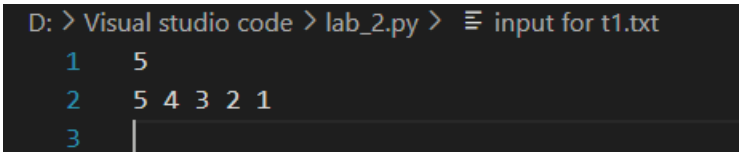
3. Перепишите процедуру `Merge` так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива L или R скопированы обратно в массив A, после чего в этот массив копируются элементы, оставшиеся в непустом массиве.

или перепишите процедуру `Merge` (и, соответственно, `Merge-sort`) так, чтобы в ней не использовались значения границ и середины - p, r и q.

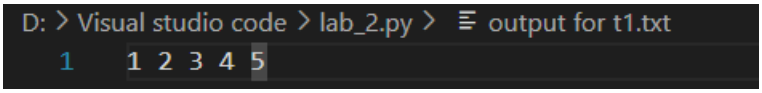
```
def merge(arr, left, mid, right):
    L = arr[left:mid+1]
    R = arr[mid+1:right+1]

    i = j = 0
    k = left

    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

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

    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

        merge_sort(arr, left, mid)
        merge_sort(arr, mid + 1, right)

        merge(arr, left, mid, right)

def main():
    with open('d:/Visual studio code/lab_2.py/input for t1.txt', 'r') as f:
        n = int(f.readline())
        arr = list(map(int, f.readline().split()))

    merge_sort(arr, 0, n - 1)

    with open('d:/Visual studio code/lab_2.py/output for t1.txt', 'w') as f:
        f.write(' '.join(map(str, arr)))

if __name__ == "__main__":
    main()
```

Input :



Output :



1. Функция merge(arr, left, mid, right):

- Она отвечает за слияние двух отсортированных частей массива.

- Параметры:

- arr — массив, который нужно сортировать.
- left, mid, right — индексы, которые задают границы для двух подмассивов: левый подмассив от left до mid, и правый подмассив от mid+1 до right.

- Алгоритм:

- Создаются два подмассива: L и R. L содержит элементы от left до mid, а R — от mid+1 до right.
- Далее в цикле поэлементно сливаются элементы из этих подмассивов обратно в основной массив arr. На каждом шаге сравниваются элементы из L и R, и меньший элемент помещается в массив arr. Этот процесс продолжается, пока один из подмассивов не будет полностью пройден.
- Если в одном из подмассивов еще остались элементы, они просто копируются в массив arr.

2. Функция merge_sort(arr, left, right):

- Это рекурсивная функция, которая разбивает массив на две части и затем рекурсивно сортирует каждую часть.

- Параметры:

- arr — массив для сортировки.
- left, right — индексы, которые задают границы части массива, которую нужно отсортировать.

- Алгоритм:

- Если массив содержит больше одного элемента (условие if left < right), то находится середина массива mid.
- Затем рекурсивно вызывается merge_sort для левой части (от left до mid) и для правой части (от mid+1 до right).
- После того как обе части отсортированы, вызывается функция merge для слияния этих частей.

3. Функция main():

- Эта функция организует ввод-вывод данных и запускает процесс сортировки.

- Сначала открывается файл input.txt, читается первое число n — количество элементов массива.

- Затем считывается сам массив arr.

- Вызывается функция merge_sort, которая сортирует массив.

- После сортировки массив записывается в файл output.txt.

4. Запуск программы:

- В блоке if __name__ == "__main__": вызывается функция main(), что означает, что программа начнет выполнение с этой функции, когда ее запускают.

Задание 2 : Сортировка слиянием+

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубы- вания *с помощью сортировки слиянием*.

Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим Вас, после каждого осуществленного слияния (то есть, когда соответству- ющий подмассив уже отсортирован!), выводить индексы граничных элементов и их значения.

- **Формат входного файла (input.txt).** В первой строке входного файла со- держится число n ($1 \le n \le 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
- **Формат выходного файла (output.txt).** Выходной файл состоит из несколь- ких строк.

– В **последней строке** выходного файла требуется вывести отсортиро- ванный в порядке неубывания массив, данный на входе. Между любыми двумя числами должен стоять ровно один пробел.

– Все предшествующие строки описывают осуществленные слияния, по одному на каждой строке. Каждая такая строка должна содержать по четыре числа: I_f , I_l, V_f , V_l, где I_f — индекс начала области слияния, I_l — индекс конца области слияния, V_f — значение первого элемента области слияния, V_l — значение последнего элемента области слияния.

– Все индексы начинаются с единицы (то есть, $1 \le I_f \le I_l \le n$). **Индексы области слияния должны описывать положение области слияния в исходном массиве!** Допускается не выводить информацию о слиянии для подмассива длиной 1, так как он отсортирован по опре- делению.

- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- **Приведем небольшой пример:** отсортируем массив [9, 7, 5, 8]. Рекурсивная часть сортировки слиянием (процедура SORT(A, L, R), где A — сортиру- емый массив, L — индекс начала области слияния, R — индекс конца области слияния) будет вызвана с A = [9, 7, 5, 8], L = 1, R = 4 и выполнит следующие действия:

– разделит область слияния [1, 4] на две части, [1, 2] и [3, 4];

– выполнит вызов SORT(A, L = 1, R = 2):

* разделит область слияния [1, 2] на две части, [1, 1] и [2, 2];

* получившиеся части имеют единичный размер, рекурсивные вы- зовы можно не делать;

* осуществит слияние, после чего A станет равным [7, 9, 5, 8];

* выведет описание слияния: $I_f = L = 1$, $I_l = R = 2$, $V_f = A_L = 7$, $V_l = A_R = 9$.

– выполнит вызов SORT(A, L = 3, R = 4):

* разделит область слияния [3, 4] на две части, [3, 3] и [4, 4];

* получившиеся части имеют единичный размер, рекурсивные вы- зовы можно не делать;

* осуществит слияние, после чего A станет равным [7, 9, 5, 8];

* выведет описание слияния: $I_f = L = 3$, $I_l = R = 4$, $V_f = A_L = 5$, $V_l = A_R = 8$.

– осуществит слияние, после чего A станет равным [5, 7, 8, 9];

– выведет описание слияния: $I_f = L = 1$, $I_l = R = 4$, $V_f = A_L = 5$, $V_l = A_R = 9$.

- Описания слияний могут идти в произвольном порядке, необязательно совпадающем с порядком их выполнения. Однако, с целью повышения производительности, рекомендуем выводить эти описания сразу, не храня их в памяти. Именно по этой причине отсортированный массив выводится в самом конце.

- Пример:

input.txt	output.txt
10 1 8 2 1 4 7 3 2 3 6	1 2 1 8 3 4 1 2 1 4 1 8 5 6 4 7 1 6 1 8 7 8 2 3 9 10 3 6 7 10 2 6 1 10 1 8 1 1 2 2 3 3 4 6 7 8

Любая корректная сортировка слиянием, делящая подмассивы на две части (необязательно равных!), будет зачтена, если успеет завершиться, уло- жившись в ограничения.

```python
import os

def merge(arr, left, mid, right, output):
    n1 = mid - left + 1
    n2 = right - mid
    L = arr[left:mid + 1]
    R = arr[mid + 1:right + 1]

    i = 0
    j = 0
    k = left

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

    output.write(f"{left + 1} {right + 1} {arr[left]} {arr[right]}\n")

def merge_sort(arr, left, right, output):
    if left < right:
        mid = (left + right) // 2

        merge_sort(arr, left, mid, output)
```

```
        merge_sort(arr, mid + 1, right, output)

        merge(arr, left, mid, right, output)

def main():
    input_file_path = 'd:/Visual studio code/lab_2.py/input for t2.txt'

    if not os.path.exists(input_file_path):
        print(f"File does not exist: {input_file_path}")
        return

    with open(input_file_path, 'r') as f:
        n = int(f.readline().strip())
        arr = list(map(int, f.readline().strip().split()))

    with open('d:/Visual studio code/lab_2.py/output for t2.txt', 'w') as output:
        merge_sort(arr, 0, n - 1, output)

        output.write(" ".join(map(str, arr)) + "\n")

if __name__ == "__main__":
    main()
```

Input :

```
D: > Visual studio code > lab_2.py > input for t2.txt
1   10
2   1 8 2 1 4 7 3 2 3 6
```

Output :

```
D: > Visual studio code > lab_2.py > output for t2.txt
1   1 2 1 8
2   1 3 1 8
3   4 5 1 4
4   1 5 1 8
5   6 7 3 7
6   6 8 2 7
7   9 10 3 6
8   6 10 2 7
9   1 10 1 8
10  1 1 2 2 3 3 4 6 7 8
```

1. Функция merge(arr, left, mid, right, output) :

Функция merge выполняет слияние двух отсортированных частей массива. Основные шаги:

- Аргументы:

- arr: массив, который нужно отсортировать.

- left, mid, right: индексы, указывающие на начало, середину и конец подмассива, который нужно слить.
- output: объект для записи выходных данных.

- Действия:

- n1 и n2: размеры левого и правого подмассивов (левая и правая части массива).
- L и R: подмассивы, которые представляют левую и правую часть соответственно.
- Затем мы начинаем процесс слияния этих подмассивов обратно в основной массив arr. Это делается через сравнение элементов обоих подмассивов:
- Если элемент из левого подмассива меньше либо равен элементу из правого подмассива, то он копируется в основной массив.
- Если элемент из правого подмассива меньше, то копируется он.
- Оставшиеся элементы, если таковые есть, копируются в основной массив после завершения основного цикла.
- **Запись результата слияния**: после слияния подмассивов в выходной файл записываются индексы начала (left + 1) и конца (right + 1) области слияния, а также значения первого и последнего элемента этой области.

2. Функция merge_sort(arr, left, right, output) :

- Функция merge_sort рекурсивно разбивает массив на подмассивы и вызывает функцию merge для их слияния.

- Если индекс left меньше right, то:

- Вычисляем средний индекс mid как среднюю точку между left и right.
- Рекурсивно вызываем merge_sort для левого подмассива (с индексами от left до mid).
- Рекурсивно вызываем merge_sort для правого подмассива (с индексами от mid + 1 до right).
- После того, как оба подмассива отсортированы, они сливаются с помощью функции merge.

3. Основная функция main() :

- Чтение данных:

- Открывается файл input.txt, из которого считывается число элементов массива n, а затем сам массив.

- Запись результатов:

- Открывается файл для записи output.txt, в который записываются промежуточные результаты слияний (индексы и значения) и отсортированный массив в последней строке.

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 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.

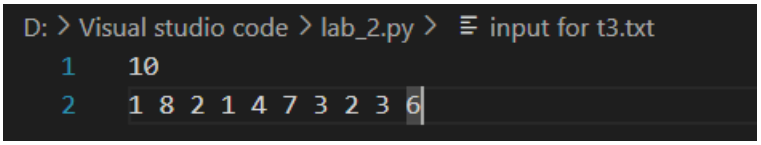
- **Формат входного файла (input.txt).** В первой строке входного файла со- держится число n ($1 \le n \le 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
- **Формат выходного файла (output.txt).** В выходной файл надо вывести число инверсий в массиве.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
10 1 8 2 1 4 7 3 2 3 6	17

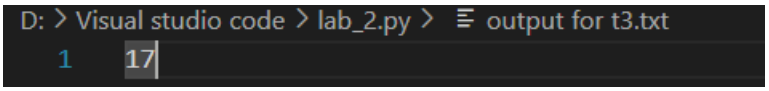
```
def merge_and_count(arr, temp_arr, left, mid, right):
    i = left
    j = mid + 1
    k = left
    inv_count = 0
    while i <= mid and j <= right:
        if arr[i] <= arr[j]:
            temp_arr[k] = arr[i]
            i += 1
        else:
            temp_arr[k] = arr[j]
            inv_count += (mid - i + 1)
            j += 1
        k += 1
    while i <= mid:
        temp_arr[k] = arr[i]
        i += 1
        k += 1
    while j <= right:
        temp_arr[k] = arr[j]
        j += 1
        k += 1
    for i in range(left, right + 1):
        arr[i] = temp_arr[i]
    return inv_count
def merge_sort_and_count(arr, temp_arr, left, right):
```

```
    inv_count = 0
    if left < right:
        mid = (left + right) // 2

        inv_count += merge_sort_and_count(arr, temp_arr, left, mid)
        inv_count += merge_sort_and_count(arr, temp_arr, mid + 1, right)

        inv_count += merge_and_count(arr, temp_arr, left, mid, right)

    return inv_count

def count_inversions(arr):
    temp_arr = [0] * len(arr)
    return merge_sort_and_count(arr, temp_arr, 0, len(arr) - 1)

with open('d:/Visual studio code/lab_2.py/input for t3.txt', "r") as f:
    n = int(f.readline().strip())
    arr = list(map(int, f.readline().strip().split()))

inv_count = count_inversions(arr)

with open('d:/Visual studio code/lab_2.py/output for t3.txt', "w") as f:
    f.write(str(inv_count))
```

Input :



Output :



1. Функция merge_and_count(arr, temp_arr, left, mid, right):

- Эта функция объединяет две отсортированные части массива, одновременно подсчитывая количество инверсий.

- Параметры:

- arr — исходный массив.
- temp_arr — временный массив для хранения промежуточных данных.
- left, mid, right — границы участков массива, которые нужно объединить.

- Логика:

- Два указателя, i и j, начинают с левой и правой части массива соответственно.

- Если элемент в левой части меньше или равен элементу в правой, он записывается во временный массив.
- Если элемент в правой части меньше, это означает инверсию, и количество инверсий увеличивается на число элементов, оставшихся в левой части, так как они все больше текущего элемента правой части.
- После обработки всех элементов обе части объединяются.

2. Функция merge_sort_and_count(arr, temp_arr, left, right):

- Эта функция рекурсивно делит массив на две части, сортирует их и подсчитывает инверсии.

- Логика:

- Разделение массива происходит до тех пор, пока каждая часть не будет содержать один элемент (базовый случай).
- Затем происходит слияние и подсчёт инверсий в каждой из частей и между ними.

3. Функция count_inversions(arr):

- Это основная функция, которая инициализирует временный массив и запускает процесс сортировки с подсчётом инверсий.

- Возвращает итоговое количество инверсий.

4. Работа с файлами:

- Открывается файл input.txt, из которого считываются данные: первое число — это размер массива, а затем сам массив.

- После подсчёта инверсий результат записывается в файл output.txt.

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 со- держится число n ($1 \leq n \leq 10^5$) — число элементов в массиве, и последо- вательность $a_0 < a_1 < ... < a_{n-1}$ 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 $b_0, ...b_{k-1}$, $1 \leq b_j \leq 10^9$ 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.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
5 1 5 8 12 13 5 8 1 23 1 11	2 0 -1 0 -1

В этом примере есть возрастающая последовательность из $a_0 = 1$, $a_1 = 5$, $a_2 = 8$, $a_3 = 12$ и $a_4 = 13$ длиной в $n = 5$ и пять чисел для поиска: 8 1 23 1 11. Видно, что $a_2 = 8$ и $a_0 = 1$, но чисел 23 и 11 нет в последовательности
a, поэтому они имеют индекс -1. В итоге ответ: 2 0 -1 0 -1.

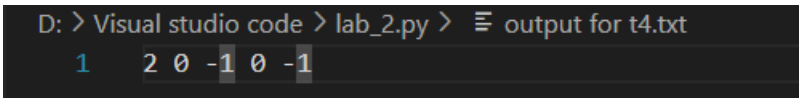
```python
def binary_search(arr, x):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            left = mid + 1
        else:
            right = mid - 1

    return -1

def main():
    with open('d:/Visual studio code/lab_2.py/input for t4.txt', 'r') as file:
        n = int(file.readline().strip())
        a = list(map(int, file.readline().strip().split()))
        k = int(file.readline().strip())
        b = list(map(int, file.readline().strip().split()))

    result = []
    for number in b:
        index = binary_search(a, number)
        result.append(index)

    with open('d:/Visual studio code/lab_2.py/output for t4.txt', 'w') as file:
        file.write(' '.join(map(str, result)) + '\n')

if __name__ == '__main__':
    main()
```

Input :

D: > Visual studio code > lab_2.py > input for t4.txt

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

Output :



1. Функция binary_search(arr, x):
- Это основная функция, которая реализует бинарный поиск. Входные параметры — отсортированный массив arr и искомый элемент x.

+ Инициализируются две переменные: left (левая граница поиска) и right (правая граница). Они указывают на начало и конец массива.

+ В цикле while, пока left <= right, выполняется поиск. Сначала вычисляется середина массива (mid), и проверяется элемент в этой позиции:

Если элемент в середине равен x, возвращается его индекс.

Если элемент в середине меньше x, левая граница сдвигается вправо (left = mid + 1), так как искомый элемент находится справа.

Если элемент больше x, правая граница сдвигается влево (right = mid - 1), так как искомый элемент находится слева.

Если элемент не найден, возвращается -1, указывающий, что такого элемента в массиве нет.

2. Функция main() :
- Это основная функция программы, которая выполняет ввод-вывод и вызывает бинарный поиск.

- Открывается файл input for t4.txt для чтения входных данных.
- Сначала читается число n, указывающее количество элементов в массиве a. Затем считывается сам массив a.
- Далее читается число k, указывающее количество чисел для поиска, и массив b, содержащий эти числа.
- Для каждого числа из массива b вызывается функция binary_search, результат (индекс или -1) сохраняется в список result.
- После завершения поиска, результаты записываются в файл output for t4.txt.

Задание 5 : Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше по- ловины голосов. Допустим, есть последовательность A элементов $a_1, a_2, \ldots a_n$, 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:
```

Очевидно, время выполнения этого алгоритма квадратично. Ваша цель - ис- пользовать метод "Разделяй и властвуй"для разработки алгоритма проверки, со- держится ли во входной последовательности элемент, который встречается боль- ше половины раз, за время $O(n \log n)$.

- **Формат входного файла (input.txt).** В первой строке входного файла со- держится число n ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n положительных целых чисел, по модулю не превосходя- щих 10^9, $0 \leq a_i \leq 10^9$.
- **Формат выходного файла (output.txt).** Выведите 1, если во входной после- довательности есть элемент, который встречается строго больше половины раз; в противном случае - 0.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример 1:

input.txt	output.txt
5 2 3 9 2 2	1

Число "2"встречается больше 5/2 раз.

- Пример 2:

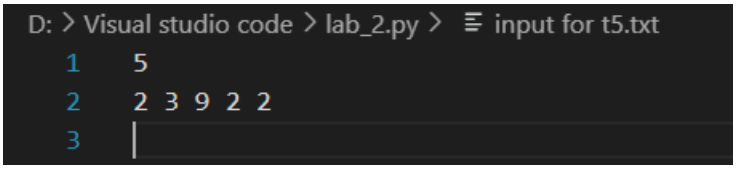
input.txt	output.txt
4 1 2 3 4	0

Нет элемента, встречающегося больше $n/2$ раз.

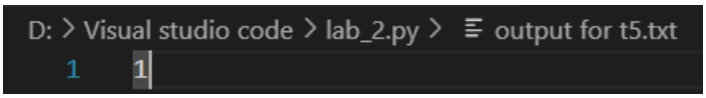
```python
def read_input(filename):
    with open(filename, 'r') as file:
        n = int(file.readline().strip())
        array = list(map(int, file.readline().strip().split()))
    return n, array

def count_occurrences(array, candidate):
    return sum(1 for x in array if x == candidate)

def majority_element(array, left, right):
    if left == right:
        return array[left]

    mid = (left + right) // 2
    left_candidate = majority_element(array, left, mid)
    right_candidate = majority_element(array, mid + 1, right)

    if left_candidate == right_candidate:
        return left_candidate

    left_count = count_occurrences(array, left_candidate)
    right_count = count_occurrences(array, right_candidate)

    return left_candidate if left_count > right_count else right_candidate
```

```python
def find_majority(n, array):
    candidate = majority_element(array, 0, n - 1)
    if count_occurrences(array, candidate) > n // 2:
        return 1
    return 0

def main():
    n, array = read_input('d:/Visual studio code/lab_2.py/input for t5.txt')
    result = find_majority(n, array)
    with open('d:/Visual studio code/lab_2.py/output for t5.txt', 'w') as file:
        file.write(str(result))

if __name__ == "__main__":
    main()
```

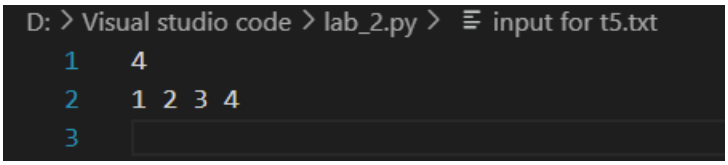
Input 1 :



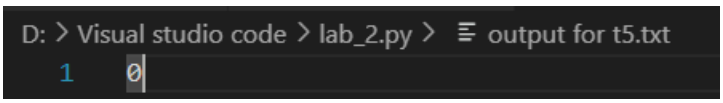
Output 1 :



Input 2 :



Output 2 :



1. Функция read_input(filename):

- Открывает файл filename для чтения.

- Читает первое значение, которое является числом n, обозначающим количество элементов.

- Читает вторую строку, которая содержит список из n целых чисел.

- Возвращает n и список чисел как кортеж.

2. Функция count_occurrences(array, candidate):

- Подсчитывает количество вхождений элемента candidate в массиве array.

- Возвращает это количество.

3. Функция majority_element(array, left, right):

- Это рекурсивная функция, которая находит кандидата на "представителя большинства" в массиве с помощью метода "Разделяй и властвуй".

- Если область поиска состоит из одного элемента (left == right), этот элемент возвращается.

- Массив делится на две части: левая и правая.

- Рекурсивно находится кандидат в обеих частях.

- Если кандидаты совпадают, возвращается этот кандидат.

- Иначе подсчитываются вхождения каждого кандидата, и возвращается тот, который встречается чаще.

4. Функция find_majority(n, array):

- Вызывает функцию majority_element, чтобы найти кандидата на "представителя большинства".

- Проверяет, встречается ли этот кандидат больше, чем n // 2 раз, используя функцию count_occurrences.

- Если да, возвращает 1, иначе — 0.

5. Функция main():

- Читает данные из файла input for t5.txt.

- Вычисляет, есть ли в массиве "представитель большинства".

- Записывает результат (1 или 0) в файл output for t5.txt.

6. Основная часть программы:

- Если файл запускается напрямую, выполняется функция main().

Задание 6 : Поиск максимальной прибыли

Используя *псевдокод* процедур Find Maximum Subarray и Find Max Crossing Subarray из презентации к Лекции 2 (страницы 25-26), напишите программу по- иска максимального подмассива.

Примените ваш алгоритм для ответа на следующий вопрос. Допустим, у нас есть данные по акциям какой-либо фирмы за последний месяц (год, или иной срок).

Проанализируйте этот срок и выдайте ответ, в какой из дней при покупке единицы акции данной фирмы, и в какой из дней продажи, вы бы получили максимальную прибыль? Выдайте дату покупки, дату продажи и максимальную прибыль.

Вы можете использовать любые данные для своего анализа. Например, я на- брала в Google

"акции" и мне поиск выдал акции Газпрома, тут - можно скачать информацию по стоимости акций за любой период. (Перейдя по ссылке, нажмите на вкладку "Настройки"→ "Скачать")

Соответственно, вам нужно только выбрать данные, посчитать ***изменение це- ны*** и применить алгоритм поиска максимального подмассива.

- **Формат входного файла** в данном случае на ваше усмотрение.
- **Формат выходного файла (output.txt).** Выведите название фирмы, рас- сматриваемый вами срок изменения акций, дату покупки и дату продажи единицы акции, чтобы получилась максимальная выгода; и сумма этой при- были.

```
def max_crossing_subarray(prices_diff, low, mid, high):
    left_sum = float('-inf')
    sum = 0
    max_left = mid

    for i in range(mid, low - 1, -1):
        sum += prices_diff[i]
        if sum > left_sum:
            left_sum = sum
            max_left = i

    right_sum = float('-inf')
    sum = 0
    max_right = mid + 1

    for j in range(mid + 1, high + 1):
        sum += prices_diff[j]
        if sum > right_sum:
            right_sum = sum
            max_right = j

    return max_left, max_right, left_sum + right_sum

def max_subarray(prices_diff, low, high):
    if low == high:
        return low, high, prices_diff[low]

    mid = (low + high) // 2
    left_low, left_high, left_sum = max_subarray(prices_diff, low, mid)
    right_low, right_high, right_sum = max_subarray(prices_diff, mid + 1, high)
    cross_low, cross_high, cross_sum = max_crossing_subarray(prices_diff, low, mid, high)

    if left_sum >= right_sum and left_sum >= cross_sum:
        return left_low, left_high, left_sum
    elif right_sum >= left_sum and right_sum >= cross_sum:
        return right_low, right_high, right_sum
    else:
        return cross_low, cross_high, cross_sum

def find_max_profit(prices):
```

```
    prices_diff = [prices[i] - prices[i - 1] for i in range(1, len(prices))]

    buy_day, sell_day, max_profit = max_subarray(prices_diff, 0, len(prices_diff) - 1)

    return buy_day + 1, sell_day + 1, max_profit

def main():
    with open('d:/Visual studio code/lab_2.py/input for t6.txt', 'r') as file:
        company_name = file.readline().strip()
        dates = file.readline().strip().split()
        prices = list(map(float, file.readline().strip().split()))

    buy_day, sell_day, max_profit = find_max_profit(prices)

    with open('d:/Visual studio code/lab_2.py/output for t6.txt', 'w') as file:
        file.write(f"Company: {company_name}\n")
        file.write(f"Buy on: {dates[buy_day]}\n")
        file.write(f"Sell on: {dates[sell_day]}\n")
        file.write(f"Max Profit: {max_profit}\n")

if __name__ == "__main__":
    main()
```

Input :

```
D: > Visual studio code > lab_2.py > input for t6.txt
1   Gazprom
2   2024-09-01 2024-09-02 2024-09-03 2024-09-04 2024-09-05
3   100 180 260 310 40
4
```

Output :

```
D: > Visual studio code > lab_2.py > output for t6.txt
1   Company: Gazprom
2   Buy on: 2024-09-02
3   Sell on: 2024-09-04
4   Max Profit: 210.0
5
```

1. Функция max_crossing_subarray(prices_diff, low, mid, high) **:**

- Эта функция находит максимальный подмассив, который пересекает середину массива, в пределах индексов low и high. Функция разделяет массив на две части — левую и правую — и ищет наибольшую сумму элементов в этих частях.

- Пояснение шагов:

- **Левая часть**: Ищем максимальную сумму, начиная с середины массива и идем влево (к элементу low). Переменная left_sum хранит максимальную сумму для левой части, а max_left — индекс начала этой суммы.
- **Правая часть**: Ищем максимальную сумму, начиная с элемента, следующего за серединой, и идем вправо (к элементу high). Переменная right_sum хранит максимальную сумму для правой части, а max_right — индекс конца этой суммы.
- **Результат**: Возвращаем индексы начала и конца максимального подмассива, а также сумму этих частей.

2. Функция max_subarray(prices_diff, low, high) **:**

- Эта функция рекурсивно находит максимальный подмассив в пределах индексов low и high. Если подмассив состоит из одного элемента (базовый случай), то возвращается сам элемент как максимальный подмассив.

- Пояснение шагов:

+ Если массив имеет один элемент (low == high), возвращаем его.

+ Иначе находим середину массива (mid) и рекурсивно вычисляем:

- Максимальный подмассив в левой половине.
- Максимальный подмассив в правой половине.
- Максимальный подмассив, пересекающий середину (используя функцию max_crossing_subarray).

+ Сравниваем три суммы и выбираем ту часть, которая дает наибольшую сумму.

3. Функция find_max_profit(prices) **:**

- Эта функция находит дни, на которые приходятся покупка и продажа акции, чтобы максимизировать прибыль.

- Пояснение шагов:

- Вычисляем разницу в ценах акций по дням (массив prices_diff), чтобы получить прибыль или убыток на каждый следующий день.
- Затем вызываем функцию max_subarray, чтобы найти дни покупки и продажи, а также максимальную прибыль. Поскольку индекс покупки и продажи был смещен на 1 (разница между днями начинается с 1-го индекса), к результатам прибавляется единица, чтобы вернуть правильные индексы дней.

4. Функция main() **:**

- Основная функция программы. Она считывает данные из файла и выводит результаты в новый файл.

- Пояснение шагов:

+ Открывается входной файл input for t6.txt:

- Первая строка содержит название компании.
- Вторая строка содержит даты.
- Третья строка содержит цены акций на соответствующие даты.

+ Вызывается функция find_max_profit, которая возвращает день покупки, день продажи и максимальную прибыль.

+ Открывается выходной файл output for t6.txt, и туда записываются результаты:

- Название компании.
- Дата покупки и продажи.
- Максимальная прибыль.

Задание 7 : Поиск максимального подмассива за линейное время

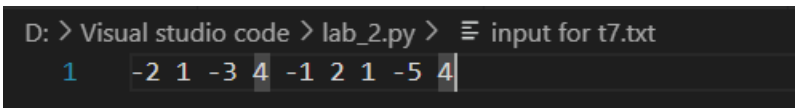
Можно найти максимальный подмассив за линейное время, воспользовавшись следующими идеями. Начните с левого конца массива и двигайтесь вправо, отсле- живая найденный к данному моменту максимальный подмассив. Зная максималь- ный подмассив массива $A[1..j]$, распространите ответ на поиск максимального подмассива,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

В этом случае у вас возможны 2 варианта тестирования: первый предполагает создание рандомного массива чисел, аналогично **задаче №1** (в этом случае фор- мат входного и выходного файла смотрите там). Второй вариант - взять любые данные по акциям какой-либо компании, аналогично **задаче №6**.

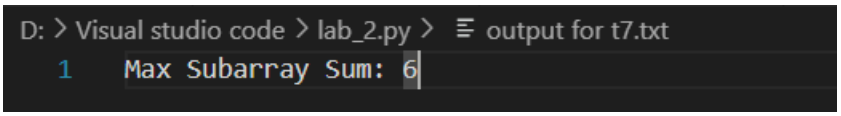
```python
def max_subarray_kadane(arr):
    max_current = arr[0]
    max_global = arr[0]

    for i in range(1, len(arr)):
        max_current = max(arr[i], max_current + arr[i])

        if max_current > max_global:
            max_global = max_current

    return max_global

def main():
    with open('d:/Visual studio code/lab_2.py/input for t7.txt', 'r') as file:
        arr = list(map(int, file.readline().split()))

    max_sum = max_subarray_kadane(arr)

    with open('d:/Visual studio code/lab_2.py/output for t7.txt', 'w') as file:
```

```
        file.write(f"Max Subarray Sum: {max_sum}\n")

if __name__ == "__main__":
    main()
```

Input :



Output :



1. Функция max_subarray_kadane(arr):

- Инициализируются две переменные: max_current и max_global, которые изначально равны первому элементу массива arr[0].

- max_current хранит текущую максимальную сумму подмассива, которая заканчивается на текущем индексе.
- max_global отслеживает максимальную сумму среди всех подмассивов, найденных на данный момент.

- Цикл for проходит по каждому элементу массива, начиная с индекса 1:

- На каждом шаге функция обновляет max_current, выбирая максимальное значение между текущим элементом массива и суммой текущего элемента с max_current.
- Если новое значение max_current превышает max_global, то обновляется max_global.

- В результате функция возвращает максимальную сумму подмассива — значение переменной max_global.

2. Функция main():

- Чтение данных из файла input for t7.txt. Строка из файла преобразуется в список целых чисел с помощью функции map() и split().

- Вызов функции max_subarray_kadane() для вычисления максимальной суммы подмассива.

- Запись результата в файл output for t7.txt.

3. Точка входа:

- Если код выполняется как основной модуль (то есть __name__ == "__main__"), вызывается функция main().

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 $n - 1$: $a_{n-1}x^{n-1} + a_{n-2}x^{n-1} + ... +$

$a_1x + a_0$ и $b_{n-1}x^{n-1} + b_{n-2}x^{n-1} + ... + b_1x + b_0$. Нужно получить произведение:

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

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

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

$A(x) = 3x^2 + 2x + 5$
$B(x) = 5x^2 + x + 2$
$A(x)B(x) = 15x^4 + 13x^3 + 33x^2 + 9x + 10$

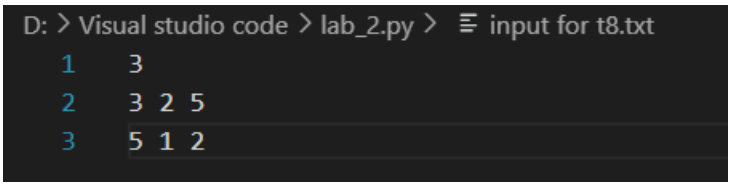
Ответ: $C = (15, 13, 33, 9, 10)$.

- **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 много- член $A(x)$ 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 $C = A(x) * B(x) = (A_1B_1)x^n + (A_1B_2 + A_2B_1)x^{n/2} + A_2B_2$ - требу-ется 4 произведения (проверьте правильность данной формулы). Можно использовать формулу Гаусса и обойтись всего тремя произведениями.

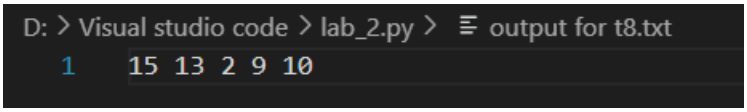
```
def poly_multiply(A, B):
    n = len(A)

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

    mid = n // 2
```

```
    A_low = A[:mid]
    A_high = A[mid:]
    B_low = B[:mid]
    B_high = B[mid:]

    z0 = poly_multiply(A_low, B_low)
    z2 = poly_multiply(A_high, B_high)

    A_sum = [a + b for a, b in zip(A_low, A_high)]
    B_sum = [a + b for a, b in zip(B_low, B_high)]

    z1 = poly_multiply(A_sum, B_sum)

    z1 = [z1[i] - z0[i] - z2[i] for i in range(len(z1))]

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

    for i in range(len(z0)):
        result[i] += z0[i]

    for i in range(len(z1)):
        result[i + mid] += z1[i]

    for i in range(len(z2)):
        result[i + 2 * mid] += z2[i]

    return result

with open('d:/Visual studio code/lab_2.py/input for t8.txt', 'r') as f:
    n = int(f.readline())
    A = list(map(int, f.readline().split()))
    B = list(map(int, f.readline().split()))

C = poly_multiply(A, B)

with open('d:/Visual studio code/lab_2.py/output for t8.txt', 'w') as f:
    f.write(' '.join(map(str, C)))
```

Input :



Output :



1. Функция poly_multiply(A, B) **:**

- Эта функция принимает на вход два многочлена *A* и *B*, представленные как списки коэффициентов. Возвращаемый результат — это список коэффициентов произведения многочленов $C(x) = A(x) * B(x)$.

- Основные шаги:

+ Базовый случай: Если длина многочленов (количество коэффициентов) равна 1, это означает, что оба многочлена являются константами, и их произведение просто равно произведению двух чисел.

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

+ Разделение многочленов: Если длина многочленов больше 1, то они делятся на две части:

A(low) и A(high) — нижняя и верхняя половины многочлена A

B(low) и B(high) — нижняя и верхняя половины многочлена B

```
mid = n // 2
A_low = A[:mid]
A_high = A[mid:]
B_low = B[:mid]
B_high = B[mid:]
```

+ Вычисление трёх произведений:

- z0=A(low)×B(low) — произведение нижних частей многочленов.
- z2=A(high)×B(high) — произведение верхних частей многочленов.
- z1 — результат применения формулы Гаусса. Для этого сначала нужно вычислить суммы A(sum)=A(low)+A(high) и B(sum)=B(low)+B(high), затем вычислить их произведение z1=A(sum)×B(sum). После этого от результата z1 вычитаются z0 и z2, чтобы получить нужное промежуточное значение.

```
z0 = poly_multiply(A_low, B_low)
z2 = poly_multiply(A_high, B_high)
A_sum = [a + b for a, b in zip(A_low, A_high)]
B_sum = [a + b for a, b in zip(B_low, B_high)]
z1 = poly_multiply(A_sum, B_sum)
z1 = [z1[i] - z0[i] - z2[i] for i in range(len(z1))]
```

+ Сборка результата: Используя результаты *z0*, *z1* и *z2*, собирается итоговый многочлен. Результат записывается в список *result*, который и возвращается.

```
    result = [0] * (2 * n - 1)
    for i in range(len(z0)):
    result[i] += z0[i]
    for i in range(len(z1)):
    result[i + mid] += z1[i]
    for i in range(len(z2)):
    result[i + 2 * mid] += z2[i]
    return result
```

2. Чтение входных данных :

- Открывается файл с входными данными. Первая строка содержит число n — порядок многочленов A и B. Далее считываются коэффициенты этих многочленов, которые преобразуются в списки.

```
with open('d:/Visual studio code/lab_2.py/input for t8.txt', 'r') as f:
n = int(f.readline())
A = list(map(int, f.readline().split()))
B = list(map(int, f.readline().split()))
```

3. Вызов функции и запись результата :

- Функция poly_multiply вызывается с аргументами A и B. Результат записывается в файл output.txt.

```
C = poly_multiply(A, B)
with open('d:/Visual studio code/lab_2.py/output for t8.txt', 'w') as f:
f.write(' '.join(map(str, C)))
```

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
```

Задачу умножения матриц достаточно легко разбить на подзадачи, поскольку произведение можно составлять из *блоков*. Разобьём каждую из матриц X и Y на четыре блока размера $\frac{n}{2} \times \frac{n}{2}$:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix},$$

Тогда их произведение выражается в терминах этих блоков по обычной формуле умножения матриц :

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Вычислив рекурсивно восемь произведений $AE, BG, AF, BH, CE, DG, CF, DH$ и просуммировав их за время $O(n^2)$, мы вычислим необходимое нам произведение матриц. Соответствующее рекуррентное соотношение на время работы алгоритма

$$T(n) = 8T(\frac{n}{2}) + O(n^2).$$

Какое получилось время у предыдущего рекурсивного алгоритма? Да, ничуть не лучше наивного. Однако его можно ускорить с помощью алгебраического трюка: для вычисления произведения XY достаточно перемножить ***семь*** пар матриц размера $\frac{n}{2} \times \frac{n}{2}$, после чего хитрым образом (*и как только Штрассен догадался?*) получить ответ:

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

где

$$\begin{aligned} P_1 &= A(F - H), & P_5 &= (A + D)(E + H), \\ P_2 &= (A + B)H, & P_6 &= (B - D)(G + H), \\ P_3 &= (C + D)E, & P_7 &= (A - C)(E + F), \\ & & P_4 &= D(G - E). \end{aligned}$$

- **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 соответсвенно сами значения матриц A и B.
- **Формат выхода**. Стандартный вывод или output.txt. Матрица $C = A \cdot B$.

```python
import numpy as np

def simple_matrix_multiplication(A, B):
    n = len(A)
    C = [[0 for _ in range(n)] for _ in range(n)]

    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]

    return C
def add_matrix(A, B):
    n = len(A)
    result = [[A[i][j] + B[i][j] for j in range(n)] for i in range(n)]
    return result

def sub_matrix(A, B):
    n = len(A)
    result = [[A[i][j] - B[i][j] for j in range(n)] for i in range(n)]
    return result

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

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

    mid = n // 2

    A11 = [[A[i][j] for j in range(mid)] for i in range(mid)]
    A12 = [[A[i][j] for j in range(mid, n)] for i in range(mid)]
    A21 = [[A[i][j] for j in range(mid)] for i in range(mid, n)]
    A22 = [[A[i][j] for j in range(mid, n)] for i in range(mid, n)]

    B11 = [[B[i][j] for j in range(mid)] for i in range(mid)]
    B12 = [[B[i][j] for j in range(mid, n)] for i in range(mid)]
    B21 = [[B[i][j] for j in range(mid)] for i in range(mid, n)]
    B22 = [[B[i][j] for j in range(mid, n)] for i in range(mid, n)]

    P1 = strassen_multiplication(add_matrix(A11, A22), add_matrix(B11, B22))
    P2 = strassen_multiplication(add_matrix(A21, A22), B11)
    P3 = strassen_multiplication(A11, sub_matrix(B12, B22))
    P4 = strassen_multiplication(A22, sub_matrix(B21, B11))
    P5 = strassen_multiplication(add_matrix(A11, A12), B22)
    P6 = strassen_multiplication(sub_matrix(A21, A11), add_matrix(B11, B12))
    P7 = strassen_multiplication(sub_matrix(A12, A22), add_matrix(B21, B22))

    C11 = add_matrix(sub_matrix(add_matrix(P1, P4), P5), P7)
    C12 = add_matrix(P3, P5)
```

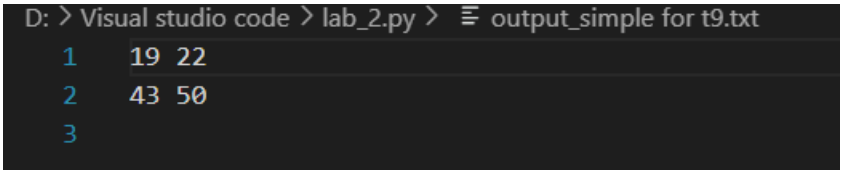
```
    C21 = add_matrix(P2, P4)
    C22 = add_matrix(sub_matrix(add_matrix(P1, P3), P2), P6)

    C = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(mid):
        for j in range(mid):
            C[i][j] = C11[i][j]
            C[i][j+mid] = C12[i][j]
            C[i+mid][j] = C21[i][j]
            C[i+mid][j+mid] = C22[i][j]

    return C
def read_input(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()

    n = int(lines[0].strip())
    matrix_A = []
    matrix_B = []

    for i in range(1, n+1):
        matrix_A.append(list(map(int, lines[i].strip().split())))
    for i in range(n+1, 2*n+1):
        matrix_B.append(list(map(int, lines[i].strip().split())))

    return n, matrix_A, matrix_B

def write_output(filename, matrix):
    with open(filename, 'w') as file:
        for row in matrix:
            file.write(' '.join(map(str, row)) + '\n')

if __name__ == '__main__':
    n, A, B = read_input('d:/Visual studio code/lab_2.py/input for t9.txt')

    result_simple = simple_matrix_multiplication(A, B)
    write_output('d:/Visual studio code/lab_2.py/output_simple for t9.txt', result_simple)

    result_strassen = strassen_multiplication(A, B)
    write_output('d:/Visual studio code/lab_2.py/output_strassen for t9.txt',
result_strassen)
```

Input :

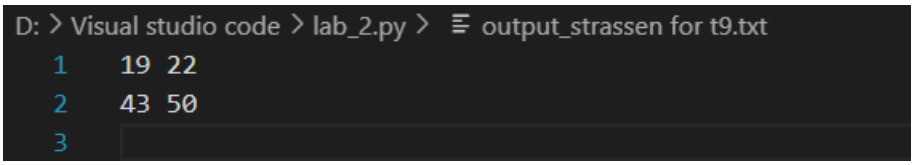
D: > Visual studio code > lab_2.py > input for t9.txt

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

Output_simple :



Output_strassen :



1. Функция simple_matrix_multiplication(A, B):

- Эта функция реализует стандартное умножение матриц. В ней используются три вложенных цикла для вычисления произведения двух квадратных матриц A и B, каждая размером n×n.

- Внешние два цикла перебирают строки и столбцы, а внутренний цикл перемножает соответствующие элементы строк и столбцов и складывает их, чтобы получить элементы результирующей матрицы C.

2. Функции add_matrix(A, B) **и** sub_matrix(A, B):

- Эти функции выполняют поэлементное сложение и вычитание матриц A и B соответственно. Для каждой строки и столбца обеих матриц вычисляется сумма или разность соответствующих элементов, и результат сохраняется в новой матрице.

3. Функция strassen_multiplication(A, B):

- Это основная функция для реализации алгоритма Штрассена. Она разбивает матрицы A и B на 4 подматрицы меньшего размера (размером n/2×n/2) и рекурсивно вычисляет семь вспомогательных матриц P1, P2, P3, P4, P5, P6 и P7, используя сложение и вычитание подматриц.

- После этого результаты вспомогательных матриц комбинируются для получения четырёх блоков результирующей матрицы C — C11, C12, C21, и C22. Эти блоки затем объединяются в одну результирующую матрицу.

4. Функция read_input(filename):

- Эта функция читает данные из файла. Первая строка файла содержит размер матриц *n*, а последующие строки содержат элементы матриц *A* и *B*. После прочтения данные возвращаются в виде двух матриц *A* и *B*, а также размера nnn.

5. Функция write_output(filename, matrix):

- Эта функция записывает результат умножения матриц в файл. Каждый элемент результирующей матрицы выводится в виде строки, где элементы разделены пробелами.

6. Основной блок программы:

- В основном блоке программы сначала читаются входные данные с помощью функции read_input. Затем производится умножение матриц двумя методами: простым умножением и умножением с использованием метода Штрассена. Результаты записываются в файлы с помощью функции write_output.

- Входные данные (файл input.txt): размер матриц nnn и элементы матриц A и B.

- Выходные данные (файл output_simple.txt и output_strassen.txt): результат умножения матриц методом стандартного перемножения и методом Штрассена.