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 вставками, выбором, пузырьковая

Выполнил:
Нгуен Хыу Жанг
К3140

Проверила:
Афанасьев А.В

Санкт-Петербург
2024

Содержание

Задачи по варианту

Задание 1 : Сортировка вставкой

Используя код процедуры Insertion-sort, напишите программу и проверьте сортировку массива $A = \{31, 41, 59, 26, 41, 58\}$.

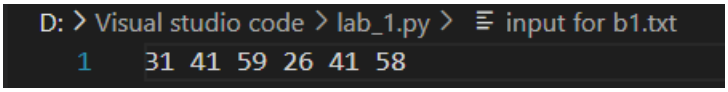
- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^3$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
- **Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

Выберите любой набор данных, подходящих по формату, и протестируйте алгоритм.

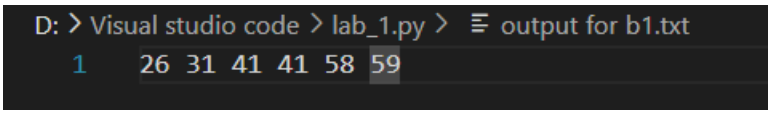
D: > Visual studio code > lab_1.py > b1.py > ...

```python
def insertion_sort(arr):

    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1

        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

with open('d:/Visual studio code/lab_1.py/input for b1.txt', 'r') as infile:
    arr = list(map(int, infile.readline().strip().split()))

insertion_sort(arr)

with open('d:/Visual studio code/lab_1.py/output for b1.txt', 'w') as outfile:
        outfile.write(' '.join(map(str, arr)))

```

Input :



Output :



1. Объявление функции: def insertion_sort(arr): — определяет функцию, принимающую массив arr.

2. Внешний цикл: for i in range(1, len(arr)): — проходит по всем элементам массива, начиная со второго.

3. Ключевой элемент: key = arr[i] — сохраняет текущий элемент, который нужно вставить на правильную позицию.

4. Внутренний цикл: while j >= 0 and key < arr[j]: — сдвигает элементы массива вправо, если они больше `key`, и продолжает, пока не найдёт правильную позицию для key.

5. Вставка: arr[j + 1] = key — помещает key на его место.

6. Чтение из файла: Код читает массив из файла input for b1.txt.

7. Запись в файл: Результат сортировки записывается в output for b1.txt.

Задание 2 : Сортировка вставкой +

Измените процедуру Insertion-sort для сортировки таким образом, чтобы в выходном файле отображалось в первой строке n чисел, которые обозначают новый индекс элемента массива после обработки.

- **Формат выходного файла (input.txt).**В первой строке выходного файла выведите n чисел. При этом i-ое число равно индексу, на который, в момент обработки его сортировкой вставками, был перемещен i-ый элемент исход- ного массива. Индексы нумеруются, начиная с единицы. Между любыми двумя числами должен стоять ровно один пробел.

Пример.

input.txt	output.txt
10 1 8 4 2 3 7 5 6 9 0	1 2 2 2 3 5 5 6 9 1 0 1 2 3 4 5 6 7 8 9

В примере сортировка вставками работает следующим образом:

- Первый элемент остается на своем месте, поэтому первое число в ответе — единица. Отсортированная часть массива: [1]
- Второй элемент больше первого, поэтому он тоже остается на своем месте, и второе число в ответе — двойка. [1 8]
- Четверка меньше восьмерки, поэтому занимает второе место. [1 4 8]
- Двойка занимает второе место. [1 2 4 8]
- Тройка занимает третье место. [1 2 3 4 8]
- Семерка занимает пятое место. [1 2 3 4 7 8]

- Пятерка занимает пятое место. [1 2 3 4 5 7 8]
- Шестерка занимает шестое место. [1 2 3 4 5 6 7 8]
- Девятка занимает девятое место. [1 2 3 4 5 6 7 8 9]
- Ноль занимает первое место. [0 1 2 3 4 5 6 7 8 9]

D: > Visual studio code > lab_1.py > b2.py > ...

```python
with open('d:/Visual studio code/lab_1.py/input for b2.txt', 'r') as input_file:
    data = input_file.read().split()
    numbers = list(map(int, data))

def insertion_sort(arr):
    sorted_list = []
    with open('d:/Visual studio code/lab_1.py/output for b2.txt', 'w') as output_file:
        for i in range(len(arr)):
            sorted_list.append(arr[i])
            sorted_list.sort()
            output_file.write(f'{sorted_list}\n')

insertion_sort(numbers)
```

Input :

D: > Visual studio code > lab_1.py > input for b2.txt

```
1 8 4 2 3 7 5 6 9 0
```

Output :

D: > Visual studio code > lab_1.py > output for b2.txt

```
[1]
[1, 8]
[1, 4, 8]
[1, 2, 4, 8]
[1, 2, 3, 4, 8]
[1, 2, 3, 4, 7, 8]
[1, 2, 3, 4, 5, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

1. Чтение данных из файла:

- with open('d:/Visual studio code/lab_1.py/input for b2.txt', 'r') as input_file: открывает файл для чтения.
- data = input_file.read().split(): считывает все содержимое файла и разбивает его на список строк.
- numbers = list(map(int, data)): преобразует строки в целые числа.

2. Определение функции сортировки:

- def insertion_sort(arr): определяет функцию, принимающую массив arr.
- sorted_list = []: инициализирует пустой список для хранения отсортированных элементов.

3. Запись результатов в файл:

- with open('d:/Visual studio code/lab_1.py/output for b2.txt', 'w') as output_file: открывает файл для записи.
- for i in range(len(arr)): перебирает каждый элемент массива.
- sorted_list.append(arr[i]): добавляет текущий элемент в sorted_list.
- sorted_list.sort(): сортирует sorted_list после добавления каждого элемента.
- output_file.write(f'{sorted_list}\n'): записывает отсортированный список в файл.

4. Вызов функции:

insertion_sort(numbers): вызывает функцию для сортировки чисел, считанных из файла.

Задание 3 : Сортировка вставкой по убыванию

Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедуры `Swap`.

Формат входного и выходного файла и ограничения - как в задаче 1.

Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?

D: > Visual studio code > lab_1.py > b3.py > ...

```python
def insertion_sort_recursive(arr, n):

    if n <= 1:
        return
    insertion_sort_recursive(arr, n - 1)
    key = arr[n - 1]
    j = n - 2
    while j >= 0 and arr[j] < key:
        arr[j + 1] = arr[j]
        j -= 1
    arr[j + 1] = key


with open('d:/Visual studio code/lab_1.py/input for b3.txt', 'r') as infile:
    data = list(map(int, infile.readline().strip().split()))

insertion_sort_recursive(data, len(data))


with open('d:/Visual studio code/lab_1.py/output for b3.txt', 'w') as outfile:
    outfile.write(' '.join(map(str, data)))

```

Input :

D: > Visual studio code > lab_1.py > input for b3.txt

```
31 41 59 26 41 58
```

Output :

D: > Visual studio code > lab_1.py > output for b3.txt

```
59 58 41 41 31 26
```

1. Определение функции insertion_sort_recursive:

- def insertion_sort_recursive(arr, n): Функция принимает массив arr и количество элементов n.
- if n <= 1: Если массив содержит 1 или 0 элементов, сортировка не требуется.

2. Рекурсивный вызов:

- insertion_sort_recursive(arr, n - 1): Вызывается функция для сортировки первых n-1 элементов.

3. Вставка элемента на правильную позицию:

- key = arr[n - 1]: Сохраняется значение последнего элемента.
- j = n - 2: Начинается с элемента перед key.
- Цикл while: Сравнивает key с предыдущими элементами и сдвигает их при необходимости.
- arr[j + 1] = key: Вставляет key на правильную позицию.

4. Чтение и запись в файл:

- Читает данные из файла input for b3.txt и преобразует их в список целых чисел.
- Вызывает функцию insertion_sort_recursive для сортировки списка.
- Записывает результат в файл output for b3.txt.

Задание 4 : Линейный поиск

Рассмотрим задачу поиска.

- **Формат входного файла**. Последовательность из n чисел $A = a_1, a_2, \ldots, a_n$ в первой строке, числа разделены пробелом, и значение V во второй строке. Ограничения: $0 \leq n \leq 10^3$, $-10^3 \leq a_i, V \leq 10^3$
- **Формат выходного файла**. Одно число - индекс i, такой, что $V = A[i]$, или значение -1, если V в отсутствует.
- Напишите код линейного поиска, при работе которого выполняется сканирование последовательности в поисках значения V.
- Если число встречается несколько раз, то выведите, сколько раз встречается число и все индексы i через запятую.
- Дополнительно: попробуйте найти свинью, как в лекции. Используйте во входном файле последовательность слов из лекции, и найдите соответствующий индекс.

D: > Visual studio code > lab_1.py > b4.py > ...

```python
def linear_search(arr, v):
    indices = [i for i, x in enumerate(arr) if x == v]

    if len(indices) > 0:
        return f"{len(indices)}: " + ', '.join(map(str, indices))
    else:
        return "-1"

with open('d:/Visual studio code/lab_1.py/input for b4.txt', 'r') as file:
    arr = list(map(int, file.readline().split()))
    v = int(file.readline().strip())

result = linear_search(arr, v)

with open('d:/Visual studio code/lab_1.py/output for b4.txt', 'w') as file:
    file.write(result)

```

Input :

D: > Visual studio code > lab_1.py > input for b4.txt

```
1 2 3 4 2 5
2
```

Output :

D: > Visual studio code > lab_1.py > output for b4.txt

```
2: 1, 4
```

1. Функция linear_search(arr, v):

- Принимает два аргумента: массив arr и значение v, которое нужно найти.
- Перебирает элементы массива один за другим с помощью цикла for.
- Для каждого элемента проверяется, совпадает ли он с v. Если совпадает, текущий индекс добавляется в список indices.
- Если список indices не пуст, возвращаются все индексы, где найдено значение v. Если таких индексов нет, возвращается -1.

2. Основные шаги работы программы:

- Чтение данных из файла: первая строка содержит массив чисел, вторая — значение v, которое необходимо найти.
- После выполнения поиска результат записывается в выходной файл. Если число встречается несколько раз, выводятся все индексы через запятую.

3. Особенности:

- Программа поддерживает случаи, когда число не встречается вовсе, возвращая -1.
- Обработка данных происходит за один проход по массиву, что делает алгоритм эффективным при больших объемах данных.

Задание 5 : Сортировка выбором

Рассмотрим сортировку элементов массива , которая выполняется следующим образом. Сначала определяется наименьший элемент массива , который ставится на место элемента $A[1]$. Затем производится поиск второго наименьшего элемента массива A, который ставится на место элемента $A[2]$. Этот процесс продолжается для первых $n - 1$ элементов массива A.

Напишите код этого алгоритма, также известного как сортировка выбором (selection sort). Определите время сортировки выбором в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

Формат входного и выходного файла и ограничения - как в задаче 1.

D: > Visual studio code > lab_1.py > b5.py > ...

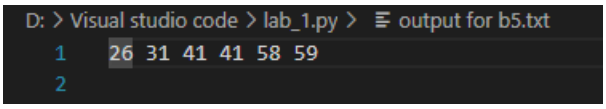
```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

def main():
    with open('d:/Visual studio code/lab_1.py/input for b5.txt', 'r') as infile:
        n = int(infile.readline().strip())
        arr = list(map(int, infile.readline().strip().split()))

    selection_sort(arr)

    with open('d:/Visual studio code/lab_1.py/output for b5.txt', 'w') as outfile:
        outfile.write(' '.join(map(str, arr)) + '\n')

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

Input :

D: > Visual studio code > lab_1.py > input for b5.txt

```
6
31 41 59 26 41 58
```

Output :



1. Функция selection_sort(arr)**:**

- Цель: Сортировка массива arr по возрастанию с помощью сортировки выбором.

- Алгоритм:

- Для каждого элемента i находим наименьший элемент в оставшейся части массива.
- Обмениваем его с элементом на позиции i.
- Повторяем процесс для каждого последующего элемента.

2. Функция main()**:**

- Цель: Обработка входных и выходных данных из файлов.

- Алгоритм:

- Чтение файла input for b5.txt, где первая строка содержит количество элементов, а вторая строка — массив чисел.
- Сортировка массива с помощью selection_sort().
- Запись отсортированного массива в файл output for b5.txt.

3. Основная программа :

- Запускается при условии, что файл выполняется напрямую: вызывает функцию main().

Задание 6 : Пузырьковая сортировка

Пузырьковая сортировка представляет собой популярный, но не очень эффективный алгоритм сортировки. В его основе лежит многократная перестановка соседних элементов, нарушающих порядок сортировки. Вот псевдокод этой сортировки:

```
Bubble_Sort(A):
  for i = 1 to A.length - 1
    for j = A.length downto i+1
      if A[j] < A[j-1]
        поменять A[j] и A[j-1] местами
```

Напишите код на Python и докажите корректность пузырьковой сортировки. Для доказательства корректоности процедуры вам необходимо доказать, что она завершается и что $A'[1] \leq A'[2] \leq ... \leq A'[n]$, где A' - выход процедуры Bubble_Sort, а n - длина массива A.

Определите время пузырьковой сортировки в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

Формат входного и выходного файла и ограничения - как в задаче 1.

D: > Visual studio code > lab_1.py > b6.py > ...

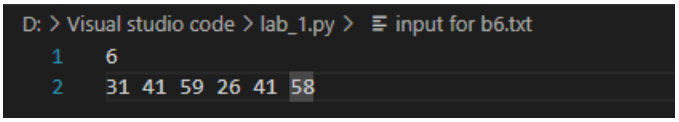
```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - 1, i, -1):
            if arr[j] < arr[j - 1]:
                arr[j], arr[j - 1] = arr[j - 1], arr[j]

def main():
    with open('d:/Visual studio code/lab_1.py/input for b6.txt', 'r') as infile:
        n = int(infile.readline().strip())
        arr = list(map(int, infile.readline().strip().split()))

    bubble_sort(arr)

    with open('d:/Visual studio code/lab_1.py/output for b6.txt', 'w') as outfile:
        outfile.write(' '.join(map(str, arr)) + '\n')

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

Input :

Output :

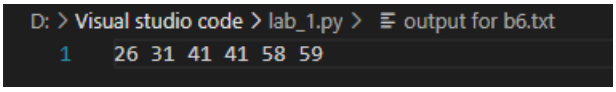
1. Функция bubble_sort(arr):

- Сортирует массив arr длиной n с помощью пузырьковой сортировки.
- Сравнивает соседние элементы массива и меняет их местами, если они расположены в неверном порядке (меньший элемент после большего).
- Внутренний цикл проходит по неотсортированной части массива, двигая большие элементы к концу.



2. Функция main():

- Читает входные данные из файла input for b6.txt.



- Первая строка файла содержит количество элементов n, а вторая строка содержит сам массив чисел.
- После чтения данных выполняется сортировка с использованием функции bubble_sort(arr).
- Записывает отсортированный массив в файл output for b6.txt.

3. Вывод:

- Код завершает работу, записав отсортированный массив в файл.

Задание 7 : Знакомство с жителями Сортлэнда

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со сво- ими подданными. Число жителей в графстве нечетно и составляет n, где n может быть достаточно велико, поэтому граф решил ограничиться знакомством с тремя представителями народонаселения: с самым бедным жителем, с жителем, облада- ющим средним достатком, и с самым богатым жителем.

Согласно традициям Сортлэнда, считается, что житель обладает средним до- статком, если при сортировке жителей по сумме денежных сбережений он оказы- вается ровно посередине. Известно, что каждый житель графства имеет уникаль- ный идентификационный номер, значение которого расположено в границах от единицы до n. Информация о размере денежных накоплений жителей хранится в массиве M таким образом, что сумма денежных накоплений жителя, обладающего

идентификационным номером i, содержится в ячейке $M[i]$. Помогите секретарю

графа мистеру Свопу вычислить идентификационные номера жителей, которые будут приглашены на встречу с графом.

- **Формат входного файла (input.txt).** Первая строка входного файла содер- жит число жителей n

($3 \le n \le 9999$, n нечетно). Вторая строка содержит описание массива M, состоящее из положительных вещественных чисел, разделенных пробелами. Гарантируется, что все элементы массива M различны, а их значения имеют точность не более двух знаков после запятой и не превышают 10^6.

D: > Visual studio code > lab_1.py > input for b7.txt

```
5
10.00  8.70  0.01  5.00  3.00

```

- **Формат выходного файла (output.txt).** В выходной файл выведите три целых положительных числа, разделенных пробелами — идентификационные номера беднейшего, среднего и самого богатого жителей Сортлэнда.

D: > Visual studio code > lab_1.py > output for b7.txt

```
3 4 1
```

- Пример:

input.txt	output.txt
5 10.00 8.70 0.01 5.00 3.00	3 4 1

Если отсортировать жителей по их достатку, получится следующий массив:

[0.01, 3][3.00, 5][5.00, 4][8.70, 2][10.00, 1]

Здесь каждый житель указан в квадратных скобках, первое число — его достаток, второе число — его идентификационный номер. Таким образом, самый бедный житель имеет номер 3, самый богатый — номер 1, а средний
— номер 4.

- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

D: > Visual studio code > lab_1.py > b7.py > ...

```python
def process_data(input_file, output_file):
    with open(input_file, 'r') as file:
        n = int(file.readline().strip())
        savings = list(map(float, file.readline().strip().split()))

        residents = [(savings[i], i + 1) for i in range(n)]

        residents.sort()

    poorest = residents[0][1]
    richest = residents[-1][1]
    median = residents[n // 2][1]


    with open(output_file, 'w') as file:
        file.write(f"{poorest} {median} {richest}\n")

process_data('d:/Visual studio code/lab_1.py/input for b7.txt', 'd:/Visual studio code/lab_1.py/output for b7.txt')

```

Input :

Output :

1. Чтение данных:

- Число n — количество жителей, и список их сбережений читаются из файла input_file. Сбережения преобразуются в числа с плавающей точкой.

```
with open(input_file, 'r') as file:
    n = int(file.readline().strip())
    savings = list(map(float, file.readline().strip().split()))
```

2. Создание списка жителей:

- Каждый житель представлен кортежем (сбережения, индекс), и все эти кортежи собираются в список residents.

```
residents = [(savings[i], i + 1) for i in range(n)]
```

3. Сортировка жителей по сбережениям:

- Жители сортируются по величине сбережений. Теперь первый элемент — самый бедный, последний — самый богатый.

```
residents.sort()
```

4. Поиск беднейшего, среднего и богатейшего жителя:

- Индексы жителей, представляющих эти категории, извлекаются из отсортированного списка.

```
poorest = residents[0][1]
richest = residents[-1][1]
median = residents[n // 2][1]
```

5. Запись результатов:

- Результат записывается в файл output_file.

```
with open(output_file, 'w') as file:
    file.write(f"{poorest} {median} {richest}\n")
```

Задание 8 : Секретарь Своп

Дан массив, состоящий из n целых чисел. Вам необходимо его отсортировать по неубыванию. Но делать это нужно так же, как это делает мистер Своп — то есть, каждое действие должно быть взаимной перестановкой пары элементов. Вам также придется записать все, что Вы делали, в файл, чтобы мистер Своп смог проверить Вашу работу.

- **Формат входного файла (input.txt).** В первой строке входного файла со- держится число n ($3 \leq n$

≤ 5000) — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 10^9. Числа могут совпадать друг с другом.

- **Формат выходного файла (output.txt).** В первых нескольких строках вы- ведите осуществленные Вами операции перестановки элементов. Каждая строка должна иметь следующий формат:

  Swap elements at indices X and Y.

  Здесь X и Y — различные индексы массива, элементы на которых нужно переставить ($1 \le X, Y \le n$). Мистер Своп любит порядок, поэтому сделайте так, чтобы $X < Y$.

  После того, как все нужные перестановки выведены, выведите следующую фразу:

  No more swaps needed.

- Пример:

input.txt	output.txt
5 3 1 4 2 2	Swap elements at indices 1 and 2. Swap elements at indices 2 and 4. Swap elements at indices 3 and 5. No more swaps needed.

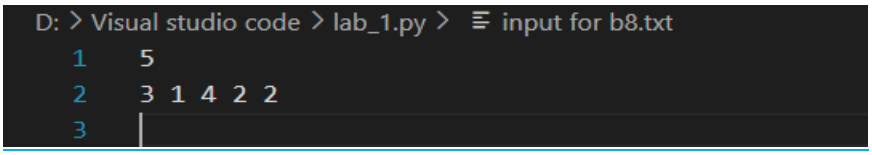
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 256 мб.

Семья секретаря Свопа занималась сортировками массивов, и именно с помо- щью перестановок пар элементов, как минимум с XII века, поэтому все Свопы владеют этим искусством в совершенстве. Мы не просим Вас произвести мини- мальную последовательность перестановок, приводящую к правильному ответу. Однако учтите, что для вывода слишком длинной последовательности у Вашего алгоритма может не хватить времени (или памяти — если выводимые строки хра- нятся в памяти перед выводом). Подумайте, что с этим можно сделать. Решение существует!

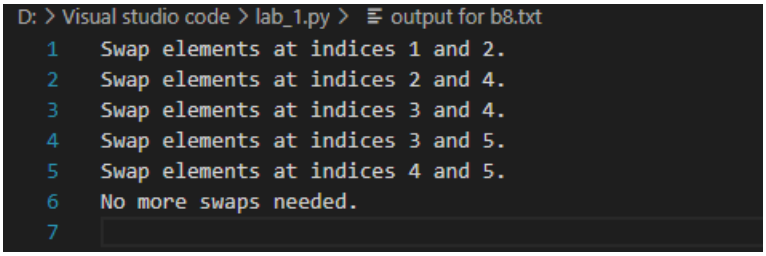
D: > Visual studio code > lab_1.py > b8.py > ...

```python
def swap_sort(arr):
    swaps = []
    n = len(arr)

    for i in range(n):
        for j in range(i + 1, n):
            if arr[i] > arr[j]:

                swaps.append(f"Swap elements at indices {i + 1} and {j + 1}.")
                arr[i], arr[j] = arr[j], arr[i]

    swaps.append("No more swaps needed.")

    return swaps

with open('d:/Visual studio code/lab_1.py/input for b8.txt', 'r') as infile:
    n = int(infile.readline().strip())
    arr = list(map(int, infile.readline().strip().split()))

swaps = swap_sort(arr)

with open( 'd:/Visual studio code/lab_1.py/output for b8.txt', 'w') as outfile:
    for swap in swaps:
        outfile.write(swap + '\n')

```

Input :



Output :



1. Определение функции swap_sort(arr):

- Функция принимает список arr и инициализирует пустой список swaps для хранения информации о произведенных обменах.

2. Основной цикл сортировки:

- Используются два вложенных цикла для перебора элементов массива:

- Внешний цикл проходит по каждому элементу i.
- Внутренний цикл проходит по элементам, расположенным после i, и сравнивает их.

- Если элемент arr[i] больше, чем arr[j], происходит обмен:

- Информация об обмене добавляется в список swaps.
- Элементы arr[i] и arr[j] меняются местами.

3. Завершение сортировки:

- После завершения обменов добавляется сообщение о завершении.

4. Чтение данных из файла:

- Код открывает файл input.txt, считывает количество элементов и сам массив.

5. Запись результатов в файл:

- Открывается файл output.txt, куда записываются все произведенные обмены.

Задание 9 : Сложение двоичных чисел

Расмотрим задачу сложения двух n-битовых двоичных целых чисел, храня- щихся в n-элементных массивах A и B. Сумму этих двух чисел необходимо занести в двоичной форме в $(n + 1)$-элементный массив C. Напишите скрипт для сложения этих двух чисел.

• **Формат входного файла (input.txt).** В одной строке содержится два n- битовых двоичных числа, записанные через пробел $(1 \leq n \leq 10^3)$

• **Формат выходного файла (output.txt).** Одна строка - двоичное число, которое является суммой двух чисел из входного файла.

• Оцените асимптотическое время выполнение вашего алгоритма.

D: > Visual studio code > lab_1.py > b9.py > ...

```python
def read_input_file(filename):
    with open(filename, 'r') as file:
        binary_numbers = file.readline().strip().split()
    return binary_numbers[0], binary_numbers[1]

def binary_addition(A, B):
    sum_binary = bin(int(A, 2) + int(B, 2))[2:]
    return sum_binary

def write_output_file(filename, result):
    with open(filename, 'w') as file:
        file.write(result)

def main():
    input_file = 'd:/Visual studio code/lab_1.py/input for b9.txt'
    output_file = 'd:/Visual studio code/lab_1.py/output for b9.txt'

    A, B = read_input_file(input_file)
    result = binary_addition(A, B)

    write_output_file(output_file, result)
if __name__ == "__main__":
    main()

```

Input :

D: > Visual studio code > lab_1.py > input for b9.txt

```
1101 1011

```

Output :

D: > Visual studio code > lab_1.py > output for b9.txt

```
11000
```

1. Чтение входного файла:

- Функция read_input_file(filename) открывает файл и считывает строку, содержащую два двоичных числа. Затем она возвращает их в виде двух строк.

2. Сложение двоичных чисел:

- Функция binary_addition(A, B) преобразует двоичные строки A и B в целые числа с помощью int(A, 2) и int(B, 2), затем складывает их. Результат преобразуется обратно в двоичную строку с помощью bin(), из которой удаляется префикс 0b.

3. Запись результата в выходной файл:

- Функция write_output_file(filename, result) открывает файл для записи и записывает результат сложения.

4. Основная функция:

- В функции main() задаются имена входного и выходного файлов, после чего вызываются функции для чтения, сложения и записи результата.

5. Запуск программы:

- Код в блоке if __name__ == "__main__": выполняет основную функцию, когда скрипт запускается.