

Федеральное агентство связи  
Ордена Трудового Красного Знамени федеральное государственное  
бюджетное образовательное учреждение высшего образования  
«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №1  
по дисциплине «Структура и алгоритмы обработки данных»  
«Методы сортировки»

Выполнил студент  
группы БФИ1902  
Кочеринский Н.В.  
Проверил: Мкртчян Г.М.

Москва 2021

# Оглавление

1 Задание на лабораторную работу.....	3
2 Решение лабораторной работы.....	3
2.1 Задание 1.....	3
2.2 Задание 2.....	3
2.3 Задание 3.....	5

1 Задание на лабораторную работу.

А) Написать программу, которая выводит надпись «Hello World!».

Б) Написать генератор случайных матриц(многомерных), который принимает опциональные параметры  $m$ ,  $n$ ,  $min\_limit$ ,  $max\_limit$ , где  $m$  и  $n$  указывают размер матрицы, а  $min\_lim$  и  $max\_lim$  - минимальное и максимальное значение для генерируемого числа .

В) Реализовать методы сортировки строк числовой матрицы в соответствии с заданием. Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки. Испытания проводить на сгенерированных матрицах.

Г) Создать публичный репозиторий на github.

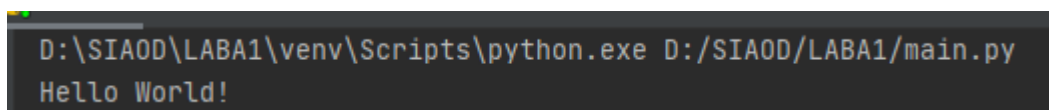
## 2 Решение лабораторной работы

### 2.1 Задание 1.

По плану лабораторной работы необходимо создать программу, которая выводит в командную строку надпись «Hello World!». На листинге 1 представлена код программы. На рисунке 1 представлен результат работы программы.

Листинг 1

```
Print("Hello World!")
```



```
D:\SIAOD\LABA1\venv\Scripts\python.exe D:/SIAOD/LABA1/main.py
Hello World!
```

Рисунок 1 - Результат работы программы.

### 2.2 Задание 2.

Далее необходимо реализовать генератор случайных матриц(многомерных), который принимает опциональные параметры  $m$ ,  $n$ ,

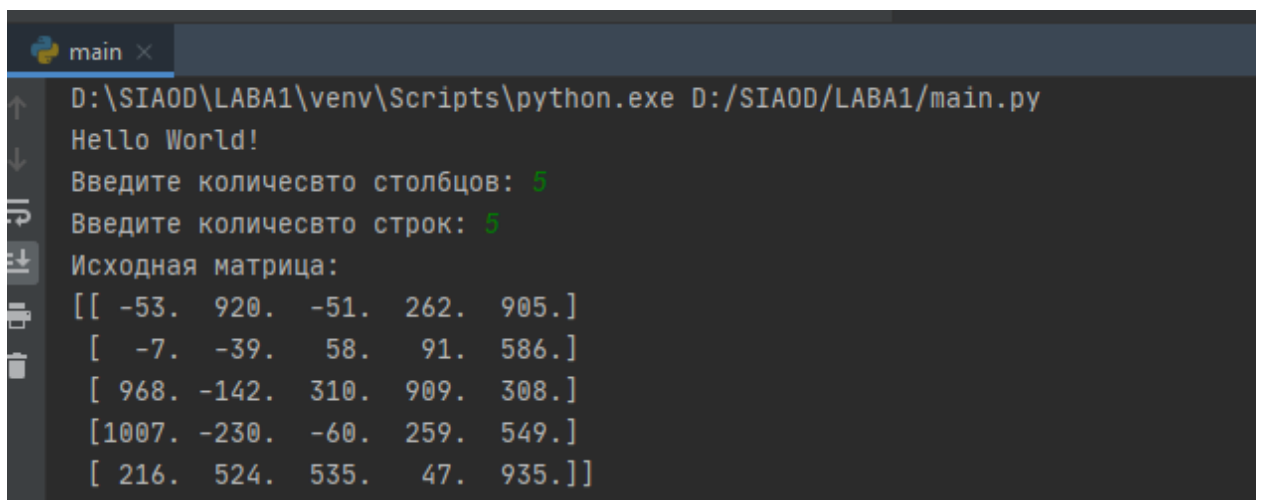
min\_limit, max\_limit, где m и n указывают размер матрицы, а min\_lim и max\_lim - минимальное и максимальное значение для генерируемого числа. На листинге 2 представлен код программы. На рисунке 2 представлен результат работы программы.

#### Листинг 2

```
import random
import numpy
import random

print("Hello World!")

m = input("Введите количесвто столбцов: ")
n = input("Введите количесвто строк: ")
m = int(m)
n = int(n)
if m == 0 and n == 0:
    m = 50
    n = 50
min_limit = -250
max_limit = 1012
# Создание матрицы
mas = numpy.zeros((m, n))
# Генерация матрицы 50 на 50
for i in range(m):
    for j in range(n):
        mas[i][j] = random.randint(int(min_limit), int(max_limit))
#Генерируем матрицу
```



```
main x
D:\SIA0D\LABA1\venv\Scripts\python.exe D:/SIA0D/LABA1/main.py
Hello World!
Введите количесвто столбцов: 5
Введите количесвто строк: 5
Исходная матрица:
[[ -53.  920.  -51.  262.  905.]
 [  -7.  -39.   58.   91.  586.]
 [ 968. -142.  310.  909.  308.]
 [1007. -230.  -60.  259.  549.]
 [ 216.  524.  535.   47.  935.]]
```

Рисунок 2 – Результат работы программы.

### 2.3 Задание 3.

Затем, необходимо реализовать методы сортировки строк числовой матрицы в соответствии с заданием. Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки. Испытания проводить на сгенерированных матрицах. На листинге 3 представлен код программы. На рисунке 3 представлен результат работы программы.

#### Листинг 3

```
import random
import numpy as np
import random

print("Hello World!")

m = input("Введите количесвто столбцов: ")
n = input("Введите количесвто строк: ")
m = int(m)
n = int(n)
if m == 0 and n == 0:
    m = 50
    n = 50
min_limit = -250
max_limit = 1012
# Создание матрицы
mas = np.zeros((m, n))
# Генерация матрицы 50 на 50
for i in range(m):
    for j in range(n):
        mas[i][j] = random.randint(int(min_limit), int(max_limit)) #Генерируем
матрицу

# Сортировка выбором
#
def SelectionSort(arr):
    new_array = arr.copy()
    for i in range(m):
        for j in range(n - 1):
            min = j
            for h in range(j + 1, n):
                if new_array[i][h] < new_array[i][min]:
                    min = h
```

```

        temp = new_array[i][j]
        new_array[i][j] = new_array[i][min]
        new_array[i][min] = temp
    return new_array

```

# Сортировка вставкой

```

def InsertionSort(arr):
    array = arr.copy()
    for i in range(len(array)):
        for j in range(len(array[i])):
            temp = array[i][j]
            index = j
            while (temp < array[i][index - 1]) and (index > 0):
                array[i][index] = array[i][index - 1]
                index -= 1
            array[i][index] = temp
    return array

```

# Сортировка обменом

```

def BubbleSort(arr):
    array = arr.copy()
    for i in range(len(array)):
        for j in range(len(array[i])):
            for h in range(len(array[i]) - j - 1):
                if array[i][h + 1] < array[i][h]:
                    temp = array[i][h]
                    array[i][h] = array[i][h + 1]
                    array[i][h + 1] = temp
    return array

```

# Сортировка Шелла¶

```

def ShellSort(arr):
    array = arr.copy()
    for i in range(len(array)):
        d = int(len(array[i]) / 2)
        while d > 0:
            for j in range(len(array[i])):
                for h in range(int(j + d), len(array[i]), d):
                    if array[i][j] > array[i][h]:
                        temp = array[i][j]
                        array[i][j] = array[i][h]
                        array[i][h] = temp
            d = int(d / 2)
    return array

```

```
    d = int(d / 2)
return array
```

# Сортировка Быстрая

```
def QuickSort(array):
    arr = array.copy()
    for i in range(len(arr)):
        quickSort(0, len(arr[i]) - 1, arr, i)
    return arr
```

```
def quickSort(_first, _last, array, row):
```

```
    first = int(_first)
    last = int(_last)
    middle = int((first + last) / 2)
```

```
    while first < last:
```

```
        while array[row][first] < array[row][middle]:
```

```
            first += 1
```

```
        while array[row][last] > array[row][middle]:
```

```
            last -= 1
```

```
        if first <= last:
```

```
            array[row][first], array[row][last] = array[row][last], array[row][first]
```

```
            first += 1
```

```
            last -= 1
```

```
    if _first < last:
```

```
        quickSort(_first, last, array, row)
```

```
    if first < _last:
```

```
        quickSort(first, _last, array, row)
```

# Сортировка Пирамидальная

```
def heapify(arr, n, i):
```

```
    largest = i
```

```
    l = 2 * i + 1
```

```
    r = 2 * i + 2
```

```
    if l < n and arr[i] < arr[l]:
```

```
        largest = l
```

```

if r < n and arr[largest] < arr[r]:
    largest = r

if largest != i:
    arr[i], arr[largest] = arr[largest], arr[i]

    heapify(arr, n, largest)

```

```

def HeapSort(array):
    arr = array.copy()
    for i in range(len(arr)):
        n = len(arr[i])

        for j in range(n, -1, -1):
            heapify(arr[i], n, j)

        for j in range(n - 1, 0, -1):
            arr[i][j], arr[i][0] = arr[i][0], arr[i][j]
            heapify(arr[i], j, 0)
    return arr

```

```

def tournamentSort(arr):
    tree = [None] * 2 * (len(arr) + len(arr) % 2)
    index = len(tree) - len(arr) - len(arr) % 2

    for i, v in enumerate(arr):
        tree[index + i] = (i, v)

    for j in range(len(arr)):
        n = len(arr)
        index = len(tree) - len(arr) - len(arr) % 2
        while index > -1:
            n = (n + 1) // 2
            for i in range(n):
                i = max(index + i * 2, 1)
                if tree[i] is not None and tree[i + 1] is not None:
                    if tree[i] < tree[i + 1]:
                        tree[i // 2] = tree[i]
                    else:
                        tree[i // 2] = tree[i + 1]
                else:
                    tree[i // 2] = tree[i] if tree[i] is not None else tree[i + 1]
            index -= n

```



```
index, x = tree[0]
arr[j] = x
tree[len(tree) - len(arr) - len(arr) % 2 + index] = None
```

```
def tour(mas):
    k = 0
    while k <= n - 1:
        tournamentSort(mas[k])
        k = k + 1
```

```
an = np.asarray(mas)
return an
```

```
print("Исходная матрица: ")
print(mas)
```

```
print("Сортировка выбором: ") #O(n^2)
print(SelectionSort(mas))
```

```
print("Сортировка вставкой: ") #O(n^2)
print(InsertionSort(mas))
```

```
print("Сортировка обменом: ") #O(n^2)
print(BubbleSort(mas))
```

```
print("Сортировки Шелла: ") #O(n*log2n)
print(ShellSort(mas))
```

```
print("Быстрая сортировка: ") #O(n*log2n)
print(QuickSort(mas))
```

```
print("Пирамидальная сортировка: ") #O(n*log2n)
print(HeapSort(mas))
```

```
print("Турнирная сортировка: ") #O(n*log2n)
print(tour(mas))
```

```
main
Введите количество столбцов: 4
Введите количество строк: 5
Исходная матрица:
[[ 507.  218.  135.  797.]
 [ 250.  833.  355.  138.]
 [-159. -154.  413.  106.]
 [ 524.  247.  450.   28.]]
Сортировка выбором:
[[ 135.  218.  507.  797.]
 [ 138.  250.  355.  833.]
 [-159. -154.  106.  413.]
 [  28.  247.  450.  524.]]
Сортировка вставкой:
[[ 135.  218.  507.  797.]
 [ 138.  250.  355.  833.]
 [-159. -154.  106.  413.]
 [  28.  247.  450.  524.]]
Сортировка обменом:
[[ 135.  218.  507.  797.]
 [ 138.  250.  355.  833.]
 [-159. -154.  106.  413.]
 [  28.  247.  450.  524.]]
Сортировки Шелла:
[[ 135.  218.  507.  797.]
 [ 138.  250.  355.  833.]
 [-159. -154.  106.  413.]
 [  28.  247.  450.  524.]]
Быстрая сортировка:
[[ 135.  218.  507.  797.]
 [ 138.  250.  355.  833.]
 [-159. -154.  106.  413.]
 [  28.  247.  450.  524.]]
Пирамидальная сортировка:
[[ 135.  218.  507.  797.]
 [ 138.  250.  355.  833.]
 [-159. -154.  106.  413.]
 [  28.  247.  450.  524.]]
Турнирная сортировка:
[[ 507.  218.  135.  797.]
 [ 250.  833.  355.  138.]
 [-159. -154.  413.  106.]
 [ 524.  247.  450.   28.]]
```

Рисунок 3 – Результат работы программы.

Вывод: в данной лабораторной работе были изучены и применены на практике различные методы сортировки.