Лабороторная работа №1

Выполнила студентка группы БСТ1904 Самарина Александра Задание 1

```
print("Hello, World!")
```

```
Hello, World!
```

Задание 2

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 . По умолчанию при отсутствии параметров принимать следующие значения: m = 50; n = 50; min_limit = -250; max_limit = 1000 +17

```
import random
def matrix ( n=50, m =50, min_limit = -250, max_limit = 1017):
    return [[random.randint(min_limit,max_limit) for _ in range(n)]
            for _ in range(m)]
```

Задание 3

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.

```
# Сортировка выбором
def selection_sort(arr):
    for i, e in enumerate(arr):
        mn = min(range(i, len(arr)), key=arr.__getitem__)
        arr[i], arr[mn] = arr[mn], e
    return arr

# Сортировка вставкой
def insertion_sort(arr):
    for i in range(1, len(arr)):
        j = i - i
        while j >= 0 and arr[i] < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = arr[i]
# Сортировка обменом
def bubble_sort(arr):
    for i in range(len(arr)):
        for j in range(0, len(arr)-i-1):
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
def shell_sort(arr):
    last_index = len(arr) - 1
    step = len(arr) // 2
    while step > 0:
```

```
        for i in range(step, last_index + 1, 1):
            j = i
            delta = j - step
            while delta >= 0 and arr[delta] > arr[j]:
                arr[delta], arr[j] = arr[j], arr[delta]
                j = delta
                delta = j - step
        step //= 2
# Быстрая сортировка
def quick_sort(arr, fst=0, lst=None):
    if lst == None:
        lst = len(arr) - 1

    if fst >= lst:
        return

    i, j = fst, lst
    pivot = arr[(lst + fst) // 2]

    while i <= j:
        while arr[i] < pivot: i += 1
        while arr[j] > pivot: j -= 1
        if i <= j:
            arr[i], arr[j] = arr[j], arr[i]
            i, j = i + 1, j - 1

    quick_sort(arr, fst, j)
    quick_sort(arr, i, lst)
# Турнирная сортировка
def tournament_sort(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) # на последней итерация index + i * 2 = 0
                if tree[i] != None and tree[i + 1] != None:
                    if tree[i][1] < tree[i + 1][1]:
                        tree[i // 2] = tree[i]
                    else:
                        tree[i // 2] = tree[i + 1]
                else:
                    tree[i // 2] = tree[i] if tree[i] != None else tree[i + 1]
```

```
            index -= n

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

# Процедура для преобразования в двоичную кучу поддерева с корневым узлом i, что является индексом в arr
[]. n - размер кучи
def heapify(arr, n, i):
    largest = i # Initialize largest as root
    l = 2 * i + 1   # left = 2*i + 1
    r = 2 * i + 2   # right = 2*i + 2

    # Проверяем существует ли левый дочерний элемент больший, чем корень

    if l < n and arr[i] < arr[l]:
        largest = l

    # Проверяем существует ли правый дочерний элемент больший, чем корень

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

    # Заменяем корень, если нужно
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i] # свап

        # Применяем heapify к корню.
        heapify(arr, n, largest)

# Основная функция для сортировки массива заданного размера
def heap_sort(arr):
    n = len(arr)

    # Построение max-heap.
    for i in range(n, -1, -1):
        heapify(arr, n, i)

    # Один за другим извлекаем элементы
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i] # свап
        heapify(arr, i, 0)
import random
import time

SORT_FUNCTIONS = {
    'Сортировка выбором': selection_sort,
    'Сортировка вставкой': insertion_sort,
    'Сортировка обменом': bubble_sort,
```

```
    'Сортировка Шелла': shell_sort,
    'Быстрая сортировка': quick_sort,
    'Турнирная сортировка': tournament_sort,
    'Пирамидальная сортировка': heap_sort,
    'Встроенная сортирока': sorted
}

def print_comparison(comparison):
    largest_name_len = len(max(comparison.keys(), key=len))
    largest_name_len += 6

    heading = 'Алгоритм'.ljust(largest_name_len) + 'Время'
    print(heading)
    print('-' * len(heading))
    for algo, time_taken in comparison.items():
        print(f'{algo:<{largest_name_len}}{time_taken}')

time_taken = {}
samples = matrix(50, 1000)
for algo_name, sorter in SORT_FUNCTIONS.items():
    samples_copy = samples.copy()
    start = time.perf_counter()
    for sample in samples_copy:
        sorter(sample)
    end = time.perf_counter()

    time_taken[algo_name] = (end - start) / len(samples)

time_taken_sorted = dict(sorted(time_taken.items(), key=lambda kv: kv[1]))
print_comparison(time_taken_sorted)
```

```
Алгоритм                        Время
-----------------------------------
Встроенная сортирока            1.0575999999673514e-06
Сортировка вставкой             3.552530000001752e-05
Сортировка Шелла                6.337619999999333e-05
Быстрая сортировка              8.521999999993568e-05
Сортировка выбором              0.00019010729999990872
Пирамидальная сортировка        0.0003813032000000476
Сортировка обменом              0.00046177349999993567
Турнирная сортировка            0.0038768464999998286
```

Вывод: в ходе лабораторной работы мы овладели методами работы в Jupyter Notebook и реализовали методы сортировки строк (выбором, вставкой..).