Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский Томский политехнический Университет»

****

самостоятельная работа № 1

арифметические операции PYthon

курс:

# «Программирование и алгоритмизация на Python»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Исполнитель:** |  | | | | |
| студент группы | 0В21ДзебанА.А |  | ФИО студента |  | 11.12.24 |
|  |  |  |  |  |  |
| **Руководитель:** |  | | | | |
| преподаватель |  |  | Полищук В.Ю. |  |  |
|  |  |  |  |  |  |

Томск – 2024

**Цель работы:**

Обрести навыки работы с циклами и последовательностями.

**Задание:**

Реализовать следующие алгоритмы сортировки:

* Сортировка выбором;
* Сортировка вставками;
* Сортировка “Методом пузырька”;
* Сортировка Шелла;
* Быстрая сортировка.

Программа должна сгенерировать список случайных неотсортированных чисел и вывести его на экран.

Затем этот список должен быть отсортирован каждым из пяти перечисленных выше алгоритмов сортировки. Необходимо также зафиксировать время работы каждого алгоритма сортировки и при выводе отсортированного массива на экран выводить время его сортировки.

**Ход работы:**

1. Импорт библиотек – random для генерации случайных последовательностей, time для подсчета времени выполнения алгоритма:

import random

import time

1. Для генерации массива случайных целых величин была реализована функция generate\_array, создающая массив длиной length, с минимальным значением min\_value и максимальным max\_value

def generate\_array(length, min\_value, max\_value):

    input\_array = []

    for \_ in range(length):

        input\_array.append(random.randint(min\_value, max\_value))

    return input\_array

1. Реализуем последовательно алгоритмы сортировки:
   1. Выбором

def selection\_sort(arr):

    start\_time = time.time()

    n = len(arr)

    for i in range(n):

        min\_index = i

        for j in range(i + 1, n):

            if arr[j] < arr[min\_index]:

                min\_index = j

        arr[i], arr[min\_index] = arr[min\_index], arr[i]

    end\_time = time.time()

    return arr, end\_time - start\_time

* 1. Вставками

def insertion\_sort(arr):

    start\_time = time.time()

    for i in range(1, len(arr)):

        key = arr[i]

        j = i - 1

        while j >= 0 and arr[j] > key:

            arr[j + 1] = arr[j]

            j -= 1

        arr[j + 1] = key

    end\_time = time.time()

    return arr, end\_time - start\_time

* 1. Пузырьком

def bubble\_sort(arr):

    start\_time = time.time()

    n = len(arr)

    for i in range(n):

        for j in range(0, n - i - 1):

            if arr[j] > arr[j + 1]:

                arr[j], arr[j + 1] = arr[j + 1], arr[j]

    end\_time = time.time()

    return arr, end\_time - start\_time

* 1. Шелла

def shell\_sort(arr):

    start\_time = time.time()

    n = len(arr)

    gap = n // 2

    while gap > 0:

        for i in range(gap, n):

            temp = arr[i]

            j = i

            while j >= gap and arr[j - gap] > temp:

                arr[j] = arr[j - gap]

                j -= gap

            arr[j] = temp

        gap //= 2

    end\_time = time.time()

    return arr, end\_time - start\_time

* 1. Быстрая сортировка

def quick\_sort(arr):

    start\_time = time.time()

    def \_quick\_sort(items):

        if len(items) <= 1:

            return items

        pivot = items[len(items) // 2]

        left = [x for x in items if x < pivot]

        middle = [x for x in items if x == pivot]

        right = [x for x in items if x > pivot]

        return \_quick\_sort(left) + middle + \_quick\_sort(right)

    sorted\_arr = \_quick\_sort(arr)

    end\_time = time.time()

    return sorted\_arr, end\_time - start\_time

1. Вводим параметры массива с клавиатуры:

length = int(input("Введите длину массива: \n"))

min\_size = int(input("Введите минимально вомзожное значение в массиве: \n"))

max\_size = int(input("Ввведите максимально возможное значение в массиве: \n"))

inp\_arr = generate\_array(length=length, min\_value=min\_size, max\_value=max\_size)

1. Создадим словарь, содержащий названия алгоритма и функцию, описывающую его поведение.

algorithms = {

    "Сортировка выбором": selection\_sort,

    "Сортировка вставками": insertion\_sort,

    "Пузырьковая сортировка": bubble\_sort,

    "Сортировка Шелла": shell\_sort,

    "Быстрая сортировка": quick\_sort

}

1. Реализован цикл, проходящий по всем парам “ключ” -> “значение”, получающий отсортированный массив и время обработки для каждого предоставленного способа.

for key, algorithm in algorithms.items():

    output\_array, time\_taken = algorithm(inp\_arr.copy())

    print(f"Выход для алгоритма \"{key}\":", output\_array,f". Время выполнения: {time\_taken}")

Для массива длиной тысяча, где значения распределены от 1 до 1000 получились следующие значения:

|  |  |
| --- | --- |
| Алгоритм | Время выполнения |
| Сортировка выбором | 0.018064… |
| Сортировка вставками | 0.017054… |
| Пузырьковая сортировка | 0.036598… |
| Сортировка Шелла | 0.000998… |
| Быстрая сортировка | 0.001510… |

Ожидаемо, сортировка пузырьком оказалась самой медленной (O(n2)), а самые быстрые – методом Шелла и быстрая сортировка.

**Вывод:** были изучены основные методы сортировки, а так же изучены способы задания циклов.