Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Кубанский государственный университет»

Кафедра вычислительных технологий

**ОТЧЕТ**

о выполнении лабораторной работы № 1

по дисциплине «Оценка сложности алгоритмов»

Выполнил:

Воробьев А. Д.

Проверил: преподаватель

Лапина О. Н.

Краснодар

2024

**Скалярное произведение векторов**

Скалярное произведение (или скалярное умножение) векторов — это операция, результатом которой является число (скаляр).

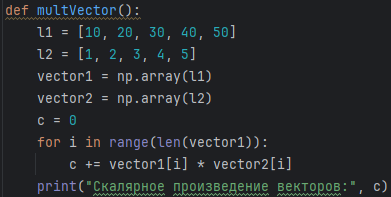


Рисунок 1. Скалярное перемножения векторов



Рисунок 2. Результат скалярного перемножения векторов

**Оценка сложности алгоритма:**

n – размерность векторов.

T(n) – временная сложность алгоритма.

Внутри цикла выполняются две арифметические операции (+ и \*), количество итераций равно n, выделение памяти для двух векторов соответствует n.

Тогда T(n) = n + 2n + c, где c – константа для остальных операций.

Следовательно, временная сложность в худшем и лучшем случае будет O(n), аналогично и пространственная сложность алгоритма O(n).

**Перемножение матриц (nxn)**

Перемножение двух квадратных матриц A и B размерности n × n, чтобы получить матрицу C размерности n × n, включает в себя выполнение операций умножения и сложения. Если A имеет размерность n × n и B также имеет размерность n × n, то C будет иметь размерность n × n.

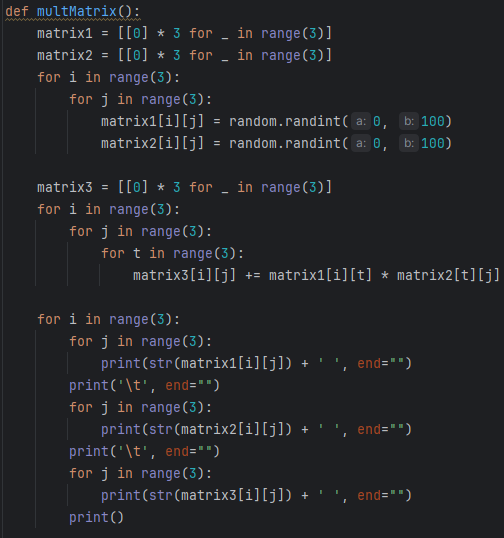


Рисунок 3. Умножение матриц

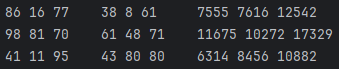


Рисунок 4. Матрицы: A, B, результирующая

**Оценка сложности алгоритма:**

– размерность матриц.

– временная сложность алгоритма.

Чтобы вычислить произведение двух квадратных матрицы, выполняется n3 умножений и n2(n-1) сложений, поэтому временная сложность Tn=O(n3)

В лучшем случае время O(n2.81), если используется алгоритм Штрассена (Применение его оправдано для плотных (содержащих мало нулей) матриц достаточно большого размера (от 45х45 и выше).

Пространственная сложность: O(n2)

**Сортировка массив (вставкой; выбором; пузырьковая).**

Сортировка вставкой.

Для определения функции сложности алгоритма сортировки вставками, мы должны проанализировать количество операций, которые выполняются в алгоритме в зависимости от размера входного массива n.

В этом алгоритме используется один внешний цикл, который выполняется n-1 раз (от второго элемента до последнего). В каждой итерации этого цикла есть внутренний цикл for, который будет иметь разное количество итераций (от 0 до n), в зависимости от элементов входного массива и отношений между ними.

Сама операция вставки производится на каждой итерации внутреннего цикла for.

Таким образом, суммарное количество операций осуществляемых алгоритмом вставки массива длиной n будут зависеть от специфики входных данных и выполняемых циклов.

В наихудшем случае, когда массив уже отсортирован в обратном порядке, вложенный цикл for будет выполнять n итераций на каждой итерации внешнего цикла.

Таким образом, функция сложности алгоритма сортировки вставками может быть оценена как O(n2) в худшем случае и O(n) в лучшем случае, где n - размер входного массива.

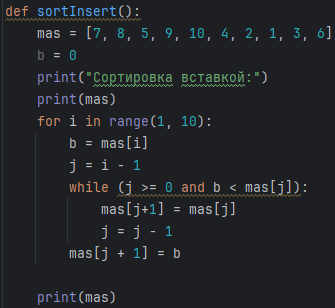


Рисунок 5. Сортировка вставками

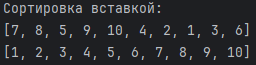


Рисунок 6. Результат сортировки вставками

**Сортировка выбором.**

Для определения функции сложности алгоритма сортировки выбором, мы должны проанализировать количество операций, которые выполняются в алгоритме в зависимости от размера входного массива n.

В этом алгоритме используется один внешний цикл, который выполняется n раз (для каждого элемента в массиве). В каждой итерации внешнего цикла есть вложенный цикл for, который будет иметь разное количество итераций от 0 до n-i, где i - это текущая итерация внешнего цикла.

Таким образом, на каждой итерации внешнего цикла выполняется вложенный цикл for, который также будет выполняться n раз.

Сама операция выбора минимального элемента и обмена его с текущим элементом выполняется во вложенном цикле.

Суммарное количество операций, выполняемых алгоритмом сортировки выбором, можно оценить как O(n2), так как имеется два вложенных цикла, каждый из которых будет выполняться n раз.

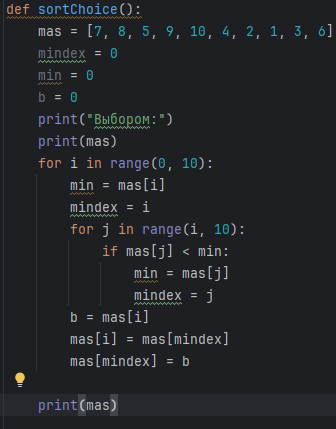


Рисунок 7. Сортировка выбором

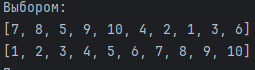


Рисунок 8. Результат сортировки выбором

**Сортировка пузырьком.**

Для определения функции сложности алгоритма сортировки пузырьком, нужно проанализировать количество операций, которые выполняются в алгоритме в зависимости от размера входного массива n.

В этом алгоритме используется два вложенных цикла for. Внешний цикл выполняется n - 1 раз, так как на каждой итерации наибольший элемент из неотсортированной части массива "всплывает" на правильное место. В каждой итерации внешнего цикла, внутренний цикл for будет иметь

меньшее количество итераций, так как каждый раз наибольший элемент уже будет находиться в конце.

Функция сложности алгоритма сортировки пузырьком составляет O(n2), где n - размер входного массива.

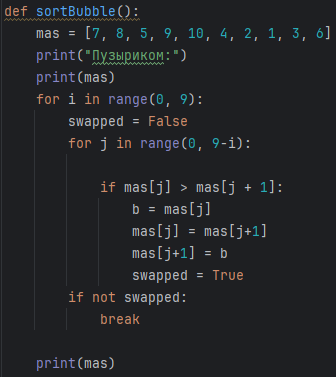


Рисунок 9. Сортировка пузырьком

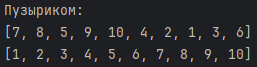


Рисунок 10. Результат сортировки пузырьком

**Листинг программы:**

import numpy as np  
import random  
  
def multVector():  
 l1 = [10, 20, 30, 40, 50]  
 l2 = [1, 2, 3, 4, 5]  
 vector1 = np.array(l1)  
 vector2 = np.array(l2)  
 c = 0  
 for i in range(len(vector1)):  
 c += vector1[i] \* vector2[i]  
 print("Скалярное произведение векторов:", c)  
  
def multMatrix():  
 matrix1 = [[0] \* 3 for \_ in range(3)]  
 matrix2 = [[0] \* 3 for \_ in range(3)]  
 for i in range(3):  
 for j in range(3):  
 matrix1[i][j] = random.randint(0, 100)  
 matrix2[i][j] = random.randint(0, 100)  
  
 matrix3 = [[0] \* 3 for \_ in range(3)]  
 for i in range(3):  
 for j in range(3):  
 for t in range(3):  
 matrix3[i][j] += matrix1[i][t] \* matrix2[t][j]  
  
 for i in range(3):  
 for j in range(3):  
 print(str(matrix1[i][j]) + ' ', end="")  
 print('\t', end="")  
 for j in range(3):  
 print(str(matrix2[i][j]) + ' ', end="")  
 print('\t', end="")  
 for j in range(3):  
 print(str(matrix3[i][j]) + ' ', end="")  
 print()  
  
def sortInsert():  
 mas = [7, 8, 5, 9, 10, 4, 2, 1, 3, 6]  
 b = 0  
 print("Сортировка вставкой:")  
 print(mas)  
 for i in range(1, 10):  
 b = mas[i]  
 j = i - 1  
 while (j >= 0 and b < mas[j]):  
 mas[j+1] = mas[j]  
 j = j - 1  
 mas[j + 1] = b  
  
 print(mas)  
  
def sortChoice():  
 mas = [7, 8, 5, 9, 10, 4, 2, 1, 3, 6]  
 mindex = 0  
 min = 0  
 b = 0  
 print("Выбором:")  
 print(mas)  
 for i in range(0, 10):  
 min = mas[i]  
 mindex = i  
 for j in range(i, 10):  
 if mas[j] < min:  
 min = mas[j]  
 mindex = j  
 b = mas[i]  
 mas[i] = mas[mindex]  
 mas[mindex] = b  
  
 print(mas)  
  
def sortBubble():  
 mas = [7, 8, 5, 9, 10, 4, 2, 1, 3, 6]  
 print("Пузыриком:")  
 print(mas)  
 for i in range(0, 9):  
 swapped = False  
 for j in range(0, 9-i):  
 if mas[j] > mas[j + 1]:  
 b = mas[j]  
 mas[j] = mas[j+1]  
 mas[j+1] = b  
 swapped = True  
 if not swapped:  
 break  
  
 print(mas)  
  
multVector()  
multMatrix()  
sortInsert()  
sortChoice()  
sortBubble()