МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по практике №1**

**по курсу**

**«ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ»**

Работу выполнил

Студент 36 группы

Корнилов К.А.

Преподаватель:

Лапина О.Н.

**Скалярное произведение векторов**

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

Оценка сложности алгоритма:

– размерность векторов.

– временная сложность алгоритма.

Внутри цикла выполняются две арифметические операции (+ и \*), количество итераций равно , также происходит присваивание значений переменным.

Тогда , где – константа для остальных операций.

Следовательно, временная сложность в худшем и лучшем случае будет , аналогично и пространственная сложность алгоритма .

def multiple\_vectors(vector1, vector2):  
 s = 0  
 for i in range(len(vector1)):  
 a = vector1[i] \* vector2[i]  
 s = s + a  
 return s

**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.

Оценка сложности алгоритма:

– размерность матриц.

– временная сложность алгоритма.

Чтобы вычислить произведение двух квадратных матрицы, метод «multiply\_matrix» выполняет умножений и сложений, поэтому временная сложность .

Полная формула временной сложности алгоритма выглядит следующим образом: .

Пространственная сложность:

def multiply\_matrix(matrix1, matrix2):  
 total\_matrix = []  
 lenght = len(matrix1)  
 for i in range(lenght):  
 row = []  
 for j in range(lenght):  
 b = [matrix2[o][j] for o in range(lenght)]  
 a = multiple\_vectors(matrix1[i], b)  
 row.append(a)  
 total\_matrix.append(row)  
 return total\_matrix

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

Пространственная сложность:

Сортировка вставкой.

Каждая строка внешнего цикла выполняется ровно n-1 раз. На каждом этапе присваивается значение ключу и j=i-1. Получаем стоимость 3 единицы.

Затем начинается внутренний цикл. В нем на каждой итерации выполняется перессылка переменных и вычисление нового j. В реализации на python они займут 3 единицы стоимости. Также на каждой итерации происходит 2 сравнения и один оператор и. Каждая из этих операций будет выполнена от максимум j раз.

В лучшем случае (когда массив уже отсортирован) время:

1 + (n-1) \* (7)+c

Таким образом время будет

def input\_sort(array):  
 length = len(array)  
 for i in range(2, length):  
 key = array[i]  
 j = i - 1  
 while j >= 0 and array[j] > key:  
 array[j + 1], array[j] = array[j], array[j + 1]  
 j = j - 1  
 array[j + 1] = key  
 return array

Сортировка пузырьком.

Пространственная сложность:

Пузырек использует два цикла для сортировки массива. Во внешнем цикле n-1 итерация, во внутреннем j. На каждой итерации внутреннего цикла также происходит сравнение и смена значений переменных, то есть 2 действия.

Тогда сложность:

Таким образом временная сложность в худшем случае.

В лучшем случае (массив отсортирован и нет перестановок) получаем .

def bubble(array):  
 length = len(array)  
 for i in range(length - 1):  
 for j in range(i + 1, length):  
 if array[i] < array[j]:  
 array[i], array[j] = array[j], array[i]  
  
 return array

Сортировка выбором.

Цикл имеет n-1 итерацию. На каждой итерации происходит смена переменных и инициализация первоначального значения mind\_index. Они имеют стоимость 2. Внутренний цикл имеет от i до n итераций. На каждой итерации происходит сравнение и присваивание значений => стоимость 2.

Тогда =

В лучшем случае, когда массив отсортирован, есть оператор присваивания и перестановки, следовательно сложность останется прежней

Пространственная сложность:

def choice\_sort(array):  
 length = len(array)  
 for i in range(length - 1):  
 min\_index = i  
 for k in range(i + 1, length):  
 if array[k] < array[min\_index]:  
 min\_index = k  
 array[i], array[min\_index] = array[min\_index], array[i]  
 return array

**Результаты работы программы:**

На рисунке 1 представлен результат работы программы – скалярное произведение векторов, перемножение квадратных матриц, сортировка пузырьком, вставкой, выбором.

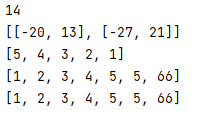


Рисунок 1 – Работа программы

**Текст программы:**

*# task1*def multiple\_vectors(vector1, vector2):  
 s = 0  
 for i in range(len(vector1)):  
 a = vector1[i] \* vector2[i]  
 s = s + a  
 return s  
  
  
 *print(multiple\_vectors([1, 2, 3], [1, 2, 3]))*def multiply\_matrix(matrix1, matrix2):  
 total\_matrix = []  
 lenght = len(matrix1)  
 for i in range(lenght):  
 row = []  
 for j in range(lenght):  
 b = [matrix2[o][j] for o in range(lenght)]  
 a = multiple\_vectors(matrix1[i], b)  
 row.append(a)  
 total\_matrix.append(row)  
 return total\_matrix  
  
  
 *print(multiply\_matrix([[2,-3],  
 [5,4]],  
 [[-7,5],  
 [2,-1],]))*def bubble(array):  
 length = len(array)  
 for i in range(length - 1):  
 for j in range(i + 1, length):  
 if array[i] < array[j]:  
 array[i], array[j] = array[j], array[i]  
  
 return array  
  
 *print(bubble([4,5,3,2,1]))*def choice\_sort(array):  
 length = len(array)  
 for i in range(length - 1):  
 min\_index = i  
 for k in range(i + 1, length):  
 if array[k] < array[min\_index]:  
 min\_index = k  
 array[i], array[min\_index] = array[min\_index], array[i]  
 return array  
  
  
print(choice\_sort([4, 5, 3, 2, 66, 5, 1]))  
  
  
def input\_sort(array):  
 length = len(array)  
 for i in range(2, length):  
 key = array[i]  
 j = i - 1  
 while j >= 0 and array[j] > key:  
 array[j + 1], array[j] = array[j], array[j + 1]  
 j = j - 1  
 array[j + 1] = key  
 return array  
  
  
print(input\_sort([4, 5, 3, 2, 66, 5, 1]))