Федеральное агентство связи

Ордена Трудового Красного Знамени

федеральное государственное бюджетное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Математическая кибернетика и информационные технологии»

Лабораторные работы по дисциплине

«Структуры и алгоритмы обработки данных»

Лабораторная работа №1

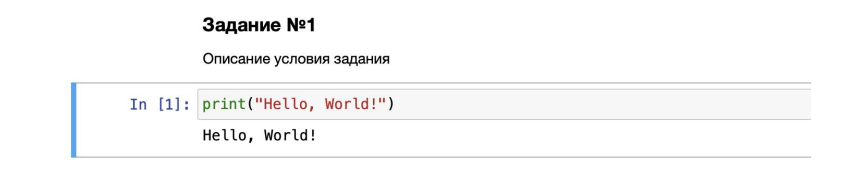
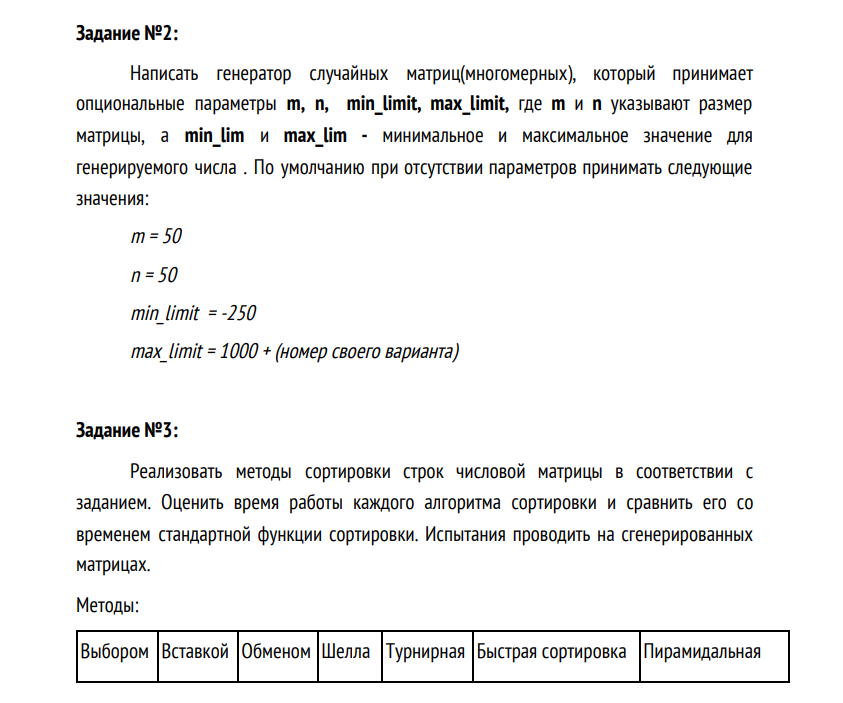
«Методы сортировки»

Выполнила студентка группы БСТ2002

Сергеева А.А.

Проверил: Аршинов Е.А.

Москва 2022

**Алгоритмы работы сортировок**

Во всех сортировках я буду говорить об одномерном массиве. Для того чтобы отсортировать двумерный массив, нужно просто применить сортировку для одномерного массива к каждой из его строк.

**Сортировка выбором**

В массиве мы устанавливаем границу, которую передвигаем во внешнем цикле. Слева от этой границы находится отсортированный массив, а справа – элементы, среди которых мы ищем новый минимальный/максимальный элемент (сама граница входит в неотсортированный массив). Найдя новый минимум, мы меняем его местами с текущим элементом границы, и таким образом отсортированный массив в левой части вырастает на один элемент. Далее мы передвигаем границу и ищем новый минимум.

Данный алгоритм имеет сложность n\*n, так как использует цикл, работающий n раз, в котором находится еще один цикл, также работающий n раз. (В случае с матрицей, сложность данного алгоритма нужно умножить на количество строк в матрице, так как сложность n\*n будет выполнена m раз.)

**Сортировка вставкой**

В данном алгоритме отсортированный массив также формируется в левой части (в окне). Мы берем “окно”, которое изначально равно одному элементу, и в котором все элементы отсортированы. Для того, чтобы внести элемент в это окно, где все элементы уже отсортированы, нужно найти для него место, сравнивая со всеми элементами “окна” начиная с первого. Когда элемент, следующий за “окном”, поставлен на нужное место, размер “окна” увеличивается, и алгоритм повторяется.

Данный алгоритм также имеет сложность n\*n так как имеет два вложенных цикла на n шагов каждый. (Фактически вложенный цикл имеет не n, а n/2 шагов, так как имеет шагов за весь алгоритм. Но это не значительно при сравнительно больших значения n.

**Сортировка обменом (пузырьком)**

Отсортированный массив формируется в правой части, так как за каждый проход наибольший элемент “всплывает” наверх. После того, как один элемент поднят наверх, новый элемент ищется в оставшимся массиве.

Алгоритм также имеет сложность n\*n так как имеет два вложенных цикла.

**Сортировка Шелла**

В этой сортировки сравниваются элементы на расстоянии d, изначально равном половине длины массива. На каждой новой итерации d уменьшается вдвое. При сортировке по возрастанию/убыванию, такой ход помогает перекинуть минимальные/максимальные элементы из начала в конец списка за гораздо меньшее число шагов, чем если бы их пришлось тащить через весь массив. Когда d станет меньше единицы, массив скорее всего еще не будет отсортирован, и процесс сортировки нужно будет закончить другим способом.

Алгоритм также имеет сложность n\*

**Турнирная сортировка**

Одна функция разбивает весь массив на пары и выводит “победителей” в следующий массив. Далее эта же функция вызывает внутри себя саму себя же, но уже от массива “победителей” предыдущего “раунда”. И так, пока не останется один “победитель”.

Эта функция имеет сложность так как количество элементов массива на каждом шаге сокращается в два раза.

Следующая функция вызывает предыдущую, получает минимальный/максимальный элемент, записывает его в результирующий массив, удаляет из рабочего массива и запускает заново, но уже от уменьшенного массива, чтобы найти следующий минимум/максимум в оставшемся массиве.

Эта функция имеет сложность n так как должна найти n минимумов/максимумов, а значит и проработать n раз.

Таким образом общая сложность алгоритма равняется n\*.

**Быстрая сортировка**

На каждой итерации мы выбираем опорный элемент. Далее мы делим все элементы на три группы: те, что меньше опорного элемента, больше него или равны ему. После, для каждой из групп мы запускаем тот же алгоритм, и так, пока группа не будет содержать 1 или 0 элементов.

Этот алгоритм будет иметь сложность n\*, так как на каждом шаге массив будет делиться на две +- равные части, а составление этих частей будет происходить за n шагов (n раз сравнить является ли текущий элемент больше/меньше/равным опорному элементу).

**Пирамидальная сортировка (сортировка кучей)**

Данный алгоритм работает со структурой данных двоичная куча. Для этой структуры характерно то, что “родительский” элемент имеет два “ребенка”, каждый из которых больше родителя.

Одна из функций, heapSortSub, нужна для “просеивания элемента вниз”. Она запускается от родителя, ищет его детей, находит среди них (родителя и детей) минимум и ставит его не место родителя. Далее, если эта функция поменяла родителя с каким-то из его детей, то она запускается того элемента, в который положила неподходящий.

Эта функция работает за , так как в куче каждый уровень имеет в два раза больше элементов, чем вышележащий. Таким образом, чтобы протащить элемент с вершины кучи нужно действий.

Следующая функция принимает в себя массив и первым делом преобразует его в кучу. Далее она формирует отсортированный массив в левой части. На каждой итерации функция меняет последний элемент в куче с первым, который является минимальным на данный момент. Так, в левой части добавляется один элемент, а в правая часть, являющейся кучей, уменьшается на 1. Элемент из конца, с которым поменяли минимум, просевается вниз при помощи функции, описанной выше, а новый минимум вновь встает в начало кучи.

Данная функция работает за n шагов, так как нужно переместить n элементов.

Таким образом общая сложность алгоритма равняется n\*.

**Код программы**

import random  
  
# Задание 1  
import time  
  
print("Hello, world")  
  
# Задание 2  
  
  
def matrixGenerator(m=50, n=50, min\_limit=-250, max\_limit=1013):  
 a = [[random.randint(min\_limit, max\_limit) for j in range(m)] for i in range(n)]  
 print(a)  
 return a  
  
  
def swapM(mat, n, a, b):  
 mat[n][a], mat[n][b] = mat[n][b], mat[n][a]  
  
  
def swap(mass, a, b):  
 mass[a], mass[b] = mass[b], mass[a]  
  
  
# Сортировка выбором (сложность: O(m\*n^2)))  
  
def selectionSort(matrix, n, m):  
 for matrixString in range(m):  
 for i in range(n-1):  
 minimum = i  
 for j in range(i+1, n):  
 if matrix[matrixString][j] < matrix[matrixString][minimum]:  
 swapM(matrix, matrixString, j, minimum)  
 return matrix  
  
# Сортировка вставкой(сложность: O(m\*n^2)))  
  
  
def insertionSort(matrix, n, m):  
 for matrixString in range(m):  
 for i in range(1, n):  
 for j in range(0, i-1):  
 if matrix[matrixString][j] > matrix[matrixString][i]:  
 swapM(matrix, matrixString, j, i)  
 return matrix  
  
# Сортировка обменом (пузырьком) (сложность: O(m\*n^2)))  
  
  
def bubbleSort(matrix, n, m):  
 for matrixString in range(m):  
 for i in range(0, n):  
 r = False  
 for j in range(0, n-i-1):  
 if matrix[matrixString][j] > matrix[matrixString][j+1]:  
 swapM(matrix, matrixString, j, j+1)  
 r = True  
 if r == False:  
 return matrix  
 return matrix  
  
# Сортировка Шелла (сложность: O(m\*n\*(log(n)^2)))  
  
  
def shellSort(matrix, n, m):  
 for matrixString in range(m):  
 d = n // 2  
 while d >= 1:  
 for i in range(0, n-d):  
 if matrix[matrixString][i] > matrix[matrixString][i+d]:  
 swapM(matrix, matrixString, i, i+d)  
 d = d // 2  
 for i in range(0, n):  
 r = False  
 for j in range(0, n - i - 1):  
 if matrix[matrixString][j] > matrix[matrixString][j + 1]:  
 swapM(matrix, matrixString, j, j + 1)  
 r = True  
 if r == False:  
 return matrix  
 return matrix  
  
# Турнирная сортировка (сложность: O(m\*n\*log(n))))  
  
  
def tournamentSortMatrix(matrix, n, m):  
 for matrixString in range(m):  
 matrix[matrixString] = tournamentSort(matrix[matrixString])  
 return matrix  
  
  
def tournamentSort(mass):  
 final\_list = []  
 for i in range(len(mass)):  
 final\_list.append(tournamentSortSub(mass))  
 mass.remove(final\_list[-1])  
 return final\_list  
  
  
def tournamentSortSub(mass):  
 if len(mass) == 2:  
 return min(mass)  
 if len(mass) % 2 == 1:  
 mass.append(2000)  
 new\_list = [min(mass[i], mass[i+1]) for i in range(0, len(mass)-1, 2)]  
 #print(new\_list)  
 return tournamentSortSub(new\_list)  
  
# Быстрая сортировка (сложность: O(m\*n\*log(n))))  
  
  
def quickSortMatrix(matrix, n, m):  
 for matrixString in range(m):  
 matrix[matrixString] = quickSort(matrix[matrixString])  
 return matrix  
  
  
def quickSort(mass):  
 if len(mass) <= 1:  
 return mass  
 pivot = mass[0]  
 less = [i for i in mass if i < pivot]  
 equally = [i for i in mass if i == pivot]  
 more = [i for i in mass if i > pivot]  
 return quickSort(less) + equally + quickSort(more)  
  
  
# Пирамидальная сортировка (сложность: O(m\*n\*log(n))))  
  
  
def heapSortMatrix(matrix, n, m):  
 for matrixString in range(m):  
 matrix[matrixString] = heapSort(matrix[matrixString])  
 return matrix  
  
  
def heapSort(mass):  
 # делает из массива кучу  
 for j in range(len(mass) - 1, -1, -1):  
 heapSortSub(mass, len(mass), j)  
 #print("Весь массив стал обладать свойствами кучи")  
 #print(mass)  
 #print('')  
  
 for i in range(len(mass)):  
 # просеиваем вниз, одновременно уменьшая размер кучи в массиве  
 # и перенося отсортированные элементы в правую часть  
 #print("Меняем первый элемент с", mass[len(mass)-i-1])  
 swap(mass, 0, len(mass)-i-1)  
 #print("Куча заканчивается на {}-ом элементе, а дальше идет отсортированный массив".format(len(mass)-i-1))  
 heapSortSub(mass, len(mass) - i - 1, 0)  
 # print(mass)  
 # print('')  
  
 return mass  
  
  
def heapSortSub(mass, n, i):  
 largest = i  
 left = i \* 2 + 1  
 right = i \* 2 + 2  
 if (left < n) and (mass[left] > mass[largest]):  
 largest = left  
 if (right < n) and (mass[right] > mass[largest]):  
 largest = right  
 if largest != i:  
 mass[largest], mass[i] = mass[i], mass[largest]  
 heapSortSub(mass, n, largest)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 try:  
 m = int(input())  
 except ValueError:  
 m = 50  
 try:  
 n = int(input())  
 except ValueError:  
 n = 50  
 try:  
 min\_limit = int(input())  
 except ValueError:  
 min\_limit = -250  
 try:  
 max\_limit = int(input())  
 except ValueError:  
 max\_limit = 1013  
 A = matrixGenerator(m, n, min\_limit, max\_limit)  
 a = time.time()  
 print(selectionSort(A, m, n)) # сортировка выбором  
 print(insertionSort(A, m, n)) # сортировка вставкой  
 print(bubbleSort(A, m, n)) # сортировка обменом (пузырьком)  
 print(shellSort(A, m, n)) # сортировка Шелла  
 print(quickSortMatrix(A, m, n)) # быстрая сортировка  
 print(tournamentSortMatrix(A, m, n)) # турнирная сортировка  
 print(heapSortMatrix(A, m, n)) # пирамидальная сортировка  
 b = time.time() - a  
 print('%8.8f'%b)

**Вывод**

В данной работе мы научились разным видам сортировки.