**Федеральное агентство связи  
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования   
«Московский технический университет связи и информатики»**

Кафедра «Теории вероятностей и прикладной математики»

**Лабораторная работа №1**

По дисциплине «Структуры и алгоритмы обработки данных»   
по теме «Методы сортировки»  
Вариант №6

Выполнила: Гердт Таисия

Группа: БСТ1955

Руководитель: Кутейников Иван Алексеевич

## Задание

Реализовать пирамидальный метод сортировки строк числовой матрицы. Добавить реализацию быстрой сортировки (quicksort). Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки, используемой в выбранном языке программирования.

В качестве языка для реализации алгоритмов был выбран язык Python.

Код программы:

#функция преобразования бинарного дерева в кучу(сортировка)

def heapify(a, n, i):

mah = i

l = 2\*i+1

r = 2\*i+2

if l<n and a[i]<a[l]:

mah = l

if r<n and a[mah]<a[r]:

mah = r

if mah!=i:

a[i],a[mah]=a[mah],a[i]

heapify(a, n, mah)

#функция сортировки кучи

def heapSort(a):

n = len(a)

for i in range(n, -1, -1):

heapify(a, n, i)

for i in range (n-1, 0, -1):

a[i], a[0] = a[0], a[i]

heapify(a, i, 0)

#функция сортировки отельной части массива

def partition(arr, low=0, high=None):

if high == None: high = len(arr)

i = (low-1)

pivot = arr[high]

for j in range(low, high):

if arr[j] <= pivot:

i = i+1

arr[i], arr[j] = arr[j], arr[i]

arr[i+1], arr[high] = arr[high], arr[i+1]

return (i+1)

#функция быстрой сортировки («прыжками» в разные стороны)

def quickSort(arr, low=0, high=None):

if high == None: high = len(arr)-1

if len(arr) == 1:

return arr

if low < high:

pi = partition(arr, low, high)

quickSort(arr, low, pi-1)

quickSort(arr, pi+1, high)

#сортировка каждой строки массива

def matrixSort(a, func):

for i in range(len(a)):

func(a[i])

#основная функция

import random, time

mat = [[random.randint(0, 100000) for x in range(100)] for k in range(1000)]

copy = mat

#print("Generated array:")

#for row in mat: print(row)

t = time.time()

matrixSort(mat, heapSort)

t = time.time() - t

print(f"HeapSort execution time: {1000\*t} milliseconds")

#print("\nSorted array by heapSort:")

#for row in mat: print(row)

mat = copy

t = time.time()

matrixSort(mat, quickSort)

t = time.time() - t

print(f"QuickSort execution time: {1000\*t} milliseconds")

#print("\nSorted array by quickSort:")

#for row in mat: print(row)

mat = copy

t = time.time()

matrixSort(mat, sorted)

t = time.time() - t

print(f"Sorted execution time: {1000\*t} milliseconds")

#print("\nSorted array by Python sorted():")

#for row in mat: print(row)

Выполнение программы:

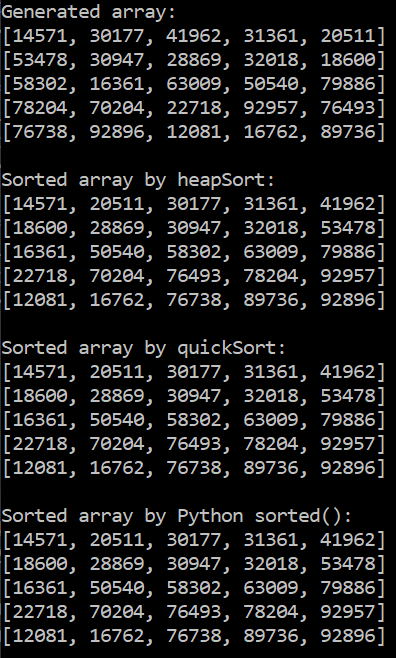


Рисунок 1. Выполнение программы для матрица размерности 5х5

Составим таблицу времени выполнения сортировок:

Таблица 1. Время выполнения сортировок

|  |  |  |  |
| --- | --- | --- | --- |
| **Сортировка\размер** | **10x10** | **100x100** | **1000x100** |
| **HeapSort** | 24.9324 | 517.6156 | 343.1122 |
| 15.9569 | 382.0088 | 432.8136 |
| 16.9826 | 445.8110 | 420.8791 |
| **Среднее** | 19.2906 | 448.4785 | 398.9350 |
| **QuickSort** | 22.9383 | 1207.7992 | 1077.1315 |
| 20.9448 | 1145.9339 | 1067.1463 |
| 18.9505 | 1470.0372 | 1096.0648 |
| **Среднее** | 20.9445 | 1274.5901 | 1080.1142 |
| **Sorted** | 0.9978 | 1.9660 | 1.9543 |
| 0.9983 | 0.9704 | 1.9953 |
| 0.9701 | 1.9953 | 1.9665 |
| **Среднее** | 0.9887 | 1.6439 | 1.9720 |

При быстрой сортировке матрицы размером 100х1000 достигается максимальная глубина рекурсии.

**Вывод:** Лучший результат выполнения сортировки массива показал встроенный метод sorted(), худший – QuickSort.