

МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И  
МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ  
государственное бюджетное образовательное учреждение высшего  
образования ордена Трудового Красного Знамени  
«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и информационных технологий

Отчет по лабораторной работе №1  
Методы сортировки  
по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: студент группы БФИ1902

Шацкий Е.И

Проверил:

Мкртчян Г. М

Москва 2021 г.

## Здание на лабораторную работу №1

### Задание №1:

1. Создать Jupyter Notebook со следующим наименованием:  
Lab1\_Группа\_ФИО
2. Создать новую ячейку с помощью кнопки «Плюсик»
3. В созданной ячейке по указанной ниже форме заполните оглавление файла, заменив наименование группы и вписав свое ФИО, после чего создайте еще одну ячейку и напишите следующий код:
4. С помощью кнопки запустите выполнение всех ячеек.
5. После выполнения у вас должна отформатироваться ячейка с оглавлением и должен выполняться “Hello, World!”

### Ход работы:

На рисунке 1 показан результат выполненных действий, требуемых для первого задания лабораторной работы.

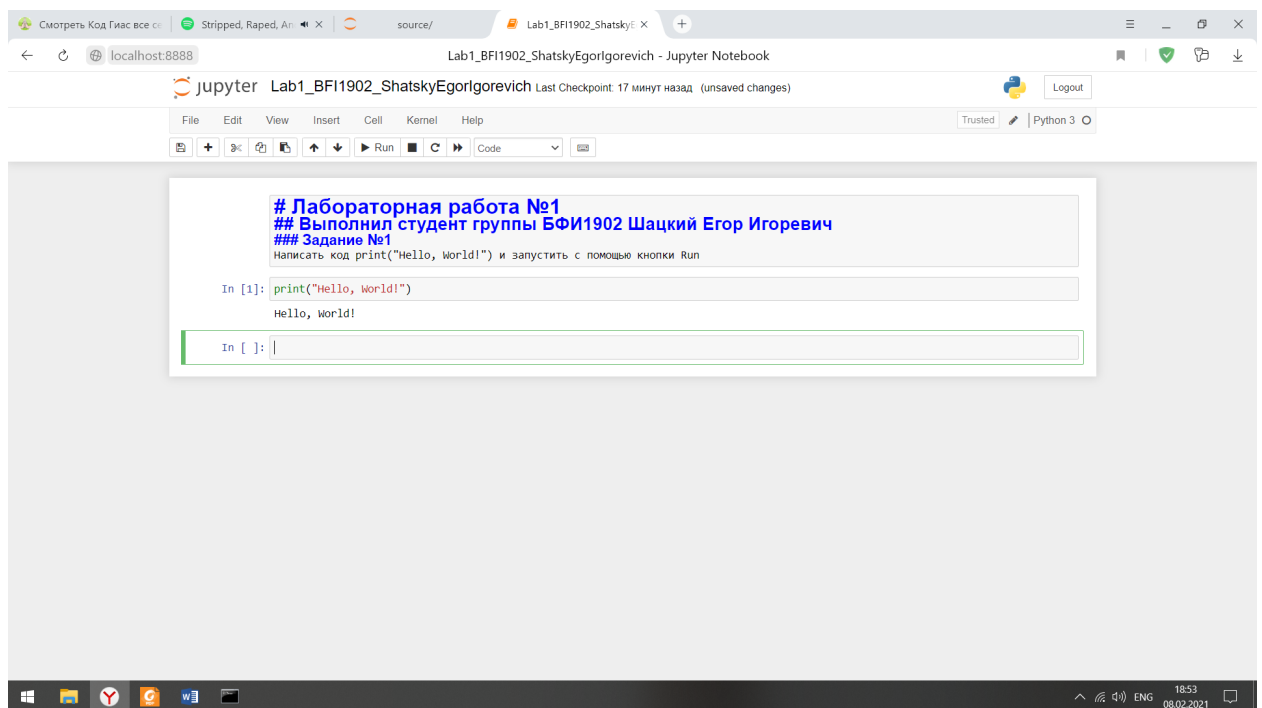


Рисунок 1 – Выполнение задания номер 1

## Вариант №20

### Задание №2

Написать генератор случайных матриц(многомерных), который принимает опциональные параметры  $m$ ,  $n$ ,  $\text{min\_limit}$ ,  $\text{max\_limit}$ , где  $m$  и  $n$  указывают размер матрицы, а  $\text{min\_lim}$  и  $\text{max\_lim}$  - минимальное и максимальное значение для генерируемого числа . По умолчанию при отсутствии параметров принимать следующие значения:

$m = 50$

$n = 50$

$\text{min\_limit} = -250$

$\text{max\_limit} = 1000 + (\text{номер своего варианта})$  в моем случае это 20

Результат выполнения задания №2 представлен на рисунке 2

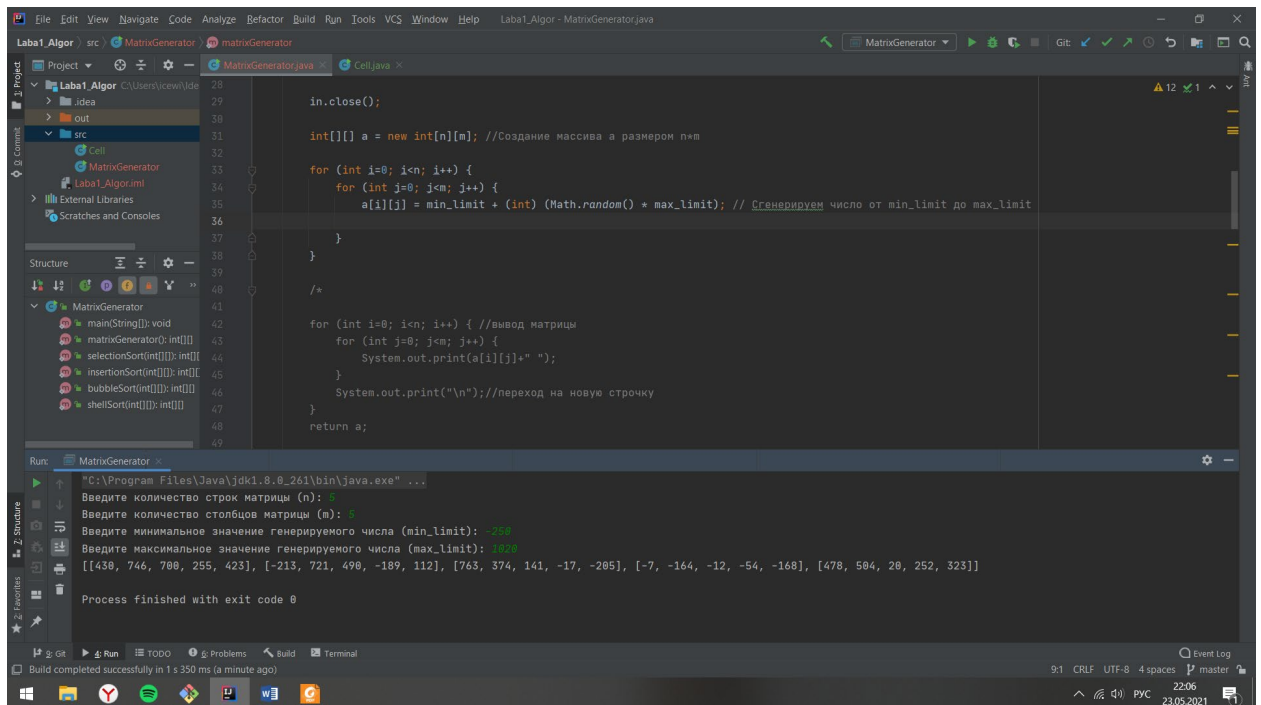


Рисунок 2 – генератор двумерной матрицы

### Задание №3:

Реализовать методы сортировки строк числовой матрицы в соответствии с заданием. Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки. Испытания проводить на сгенерированных

матрицах.

## Методы:

- Выбором
- Вставкой
- Обменом
- Шелла
- Турнирная
- Быстрая сортировка
- Пирамидальная

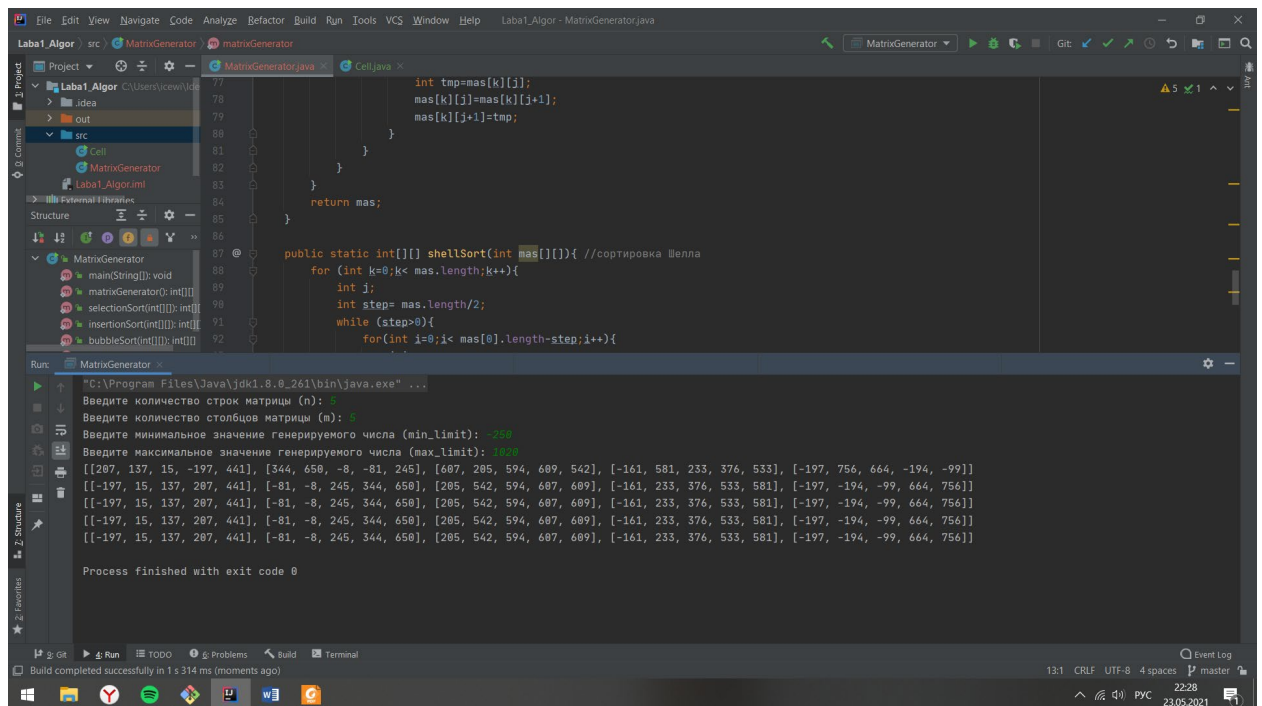


Рисунок 3 – результат выполнения сортировок

## Листинг программы

```
import java.util.Arrays;
import java.util.Scanner;

public class MatrixGenerator {
    public static void main(String[] args) {
        int mas[][]=matrixGenerator();
        System.out.println(Arrays.deepToString(mas));
        System.out.println(Arrays.deepToString(selectionSort(mas)));
        System.out.println(Arrays.deepToString(insertionSort(mas)));
        System.out.println(Arrays.deepToString(bubbleSort(mas)));
    }

    public static int[][] matrixGenerator() {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int m = scanner.nextInt();
        int min_limit = scanner.nextInt();
        int max_limit = scanner.nextInt();
        int mas[][] = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                mas[i][j] = (int) (Math.random() * (max_limit - min_limit + 1) + min_limit);
            }
        }
        return mas;
    }

    public static int[][] selectionSort(int mas[][]) {
        for (int k = 0; k < mas.length; k++) {
            int j = k;
            for (int i = k + 1; i < mas.length; i++) {
                if (mas[i][k] < mas[j][k]) {
                    j = i;
                }
            }
            int tmp = mas[k][j];
            mas[k][j] = mas[k][k];
            mas[k][k] = tmp;
        }
        return mas;
    }

    public static int[][] insertionSort(int mas[][]) {
        for (int k = 1; k < mas.length; k++) {
            int tmp = mas[k][0];
            int i = k - 1;
            while (i > 0 && mas[i][0] > tmp) {
                mas[i + 1][0] = mas[i][0];
                i--;
            }
            mas[i + 1][0] = tmp;
        }
        return mas;
    }

    public static int[][] bubbleSort(int mas[][]) {
        for (int k = 0; k < mas.length - 1; k++) {
            for (int i = 0; i < mas.length - k - 1; i++) {
                if (mas[i][0] > mas[i + 1][0]) {
                    int tmp = mas[i][0];
                    mas[i][0] = mas[i + 1][0];
                    mas[i + 1][0] = tmp;
                }
            }
        }
        return mas;
    }
}
```

```

        System.out.println(Arrays.deepToString(shellSort(mas)));
    }
    public static int[][] matrixGenerator(){
        Scanner in = new Scanner(System.in);

        System.out.print("Введите количество строк матрицы (n): ");
        int n = in.nextInt();
        System.out.print("Введите количество столбцов матрицы (m): ");
        int m = in.nextInt();
        System.out.print("Введите минимальное значение генерируемого числа (min_limit): ");
        int min_limit = in.nextInt();
        System.out.print("Введите максимальное значение генерируемого числа (max_limit): ");
        int max_limit = in.nextInt();

        in.close();

        int[][] a = new int[n][m]; //Создание массива a размером n*m

        for (int i=0; i<n; i++) {
            for (int j=0; j<m; j++) {
                a[i][j] = min_limit + (int) (Math.random() * max_limit); //
                //Сгенерируем число от min_limit до max_limit
            }
        }
        return a;
    }

    public static int[][] selectionSort(int mas[][]){ //Сортировка выбором
        for (int i=0; i< mas.length; i++){ // проходим по строкам
            for (int step=0; step<mas[0].length; step++){// номер текущего
                //шага, mas[0].length - кол-во столбцов
                int pos=step; // присваиваем номер текущего шага
                int tmp=mas[i][pos]; // присваиваем элемент массива на текущем
                //шаге
                for (int j=step+1; j<mas[0].length; j++){// цикл поиска
                    //минимального элемента в строке
                    if (mas[i][j]<tmp){
                        pos=j; // pos присваивается индексу минимального
                        //элемента
                    }
                    tmp=mas[i][j]; // переприсваиваем минимальный элемент
                }
                mas[i][pos]=mas[i][step]; // минимальному элементу
                //присваиваем текущий
                mas[i][step]=tmp; // текущему элементу присваиваем
                //минимальный
            }
        }
        return mas;
    }

    public static int[][] insertionSort(int mas[][]){ //сортировка вставкой
        for (int i=0; i<mas.length; i++){
            for (int step=1; step<mas[0].length; step++){
                int j=step;
                while (j>0 && mas[i][j-1]>mas[i][j]){
                    int tmp=mas[i][j-1];
                    mas[i][j-1]=mas[i][j];
                    mas[i][j]=tmp;
                }
            }
        }
    }

```

```

        j--;
    }
}
return mas;
}

public static int[][] bubbleSort(int mas[][]) { // сортировка пузырьком
    for (int i=0; i<mas.length; i++) {
        for (int k=0; k<mas.length; k++) {
            for (int j=0; j<mas[0].length-1; j++) {
                if (mas[k][j]>mas[k][j+1]) {
                    int tmp=mas[k][j];
                    mas[k][j]=mas[k][j+1];
                    mas[k][j+1]=tmp;
                }
            }
        }
    }
    return mas;
}

public static int[][] shellSort(int mas[][]) { //сортировка Шелла
    for (int k=0; k< mas.length; k++) {
        int j;
        int step= mas.length/2;
        while (step>0) {
            for(int i=0; i< mas[0].length-step; i++) {
                j=i;
                while ((j>=0) && (mas[k][j]>mas[k][j+step])) {
                    int tmp=mas[k][j];
                    mas[k][j]=mas[k][j+step];
                    mas[k][j+step]=tmp;
                    j=j-step;
                }
            }
            step=step/2;
        }
    }
    return mas;
}
}

```

На рисунках 4,5,6,7 представлены результаты для Пирамидальной, Турнирной и быстрой сортировке

```
In [2]: from random import randint
import numpy as np

n, m = 50, 50
min_limit = -250
max_limit = 1007

a = [[randint(min_limit, max_limit) for j in range(m)] for i in range(n)]
an = np.asarray(a)
print(an)

[[ 962  442 -178 ... -209  702  860]
 [ 811  211  567 ...  920  32  626]
 [ 679 -249  884 ...  213  993  609]
 ...
 [ 117 -127  330 ...  315  313  512]
 [ 564  806 -102 ...  16  806 1005]
 [ 263 120 -19 ...  50  885  19]]

In [3]: %time
import numpy as np

def heap_sort(alist):
    build_max_heap(alist)
    for i in range(len(alist) - 1, 0, -1):
        alist[0], alist[i] = alist[i], alist[0]
        max_heapify(alist, index=0, size=i)

def parent(i):
    return (i - 1)//2

def left(i):
    return 2*i + 1

def right(i):
    return 2*i + 2

def build_max_heap(alist):
    length = len(alist)
    start = parent(length - 1)
    while start >= 0:
        max_heapify(alist, index=start, size=length)
        start = start - 1

def max_heapify(alist, index, size):
    l = left(index)
```

Рисунок 4 – Генерация матрицы

```
def right(i):
    return 2*i + 2

def build_max_heap(alist):
    length = len(alist)
    start = parent(length - 1)
    while start >= 0:
        max_heapify(alist, index=start, size=length)
        start = start - 1

def max_heapify(alist, index, size):
    l = left(index)
    r = right(index)
    if (l < size and alist[l] > alist[index]):
        largest = l
    else:
        largest = index
    if (r < size and alist[r] > alist[largest]):
        largest = r
    if (largest != index):
        alist[largest], alist[index] = alist[index], alist[largest]
        max_heapify(alist, largest, size)

k=0
while k<=n-1:
    heap_sort(a[k])
    k=k+1

an = np.asarray(a)
print(an)

[[-217 -209 -204 ...  911  961  962]
 [-241 -206 -205 ...  953  968  980]
 [-249 -198 -128 ...  940  986  993]
 ...
 [-226 -217 -175 ...  943  988 1006]
 [-241 -235 -198 ...  972  973 1005]
 [-216 -197 -116 ...  895  983 1003]]
Wall time: 9.97 ms

In [6]: %time
import numpy as np

def quick_sort(data):
    less = []
    pivotlist = []
    more = []
    if len(data) <= 1:
```

Рисунок 5 –пирамидальная сортировка

Jupyter Lab1\_3sort Last Checkpoint: 5 минут назад (autosaved)

File Edit View Insert Cell Kernel Help Not Trusted Python 3

```
[[-216 -197 -116 ... 895 983 1003]]
Wall time: 9.97 ms

In [6]: %%time
import numpy as np

def quick_sort(data):
    less = []
    pivotlist = []
    more = []
    if len(data) <= 1:
        return data
    else:
        pivot = data[0]
        for i in data:
            if i < pivot:
                less.append(i)
            elif i > pivot:
                more.append(i)
            else:
                pivotlist.append(i)
        less = quick_sort(less)
        more = quick_sort(more)
        return less + pivotlist + more

k=0
while k<=n-1:
    quick_sort(a[k])
    k=k+1

an = np.asarray(a)
print(an)

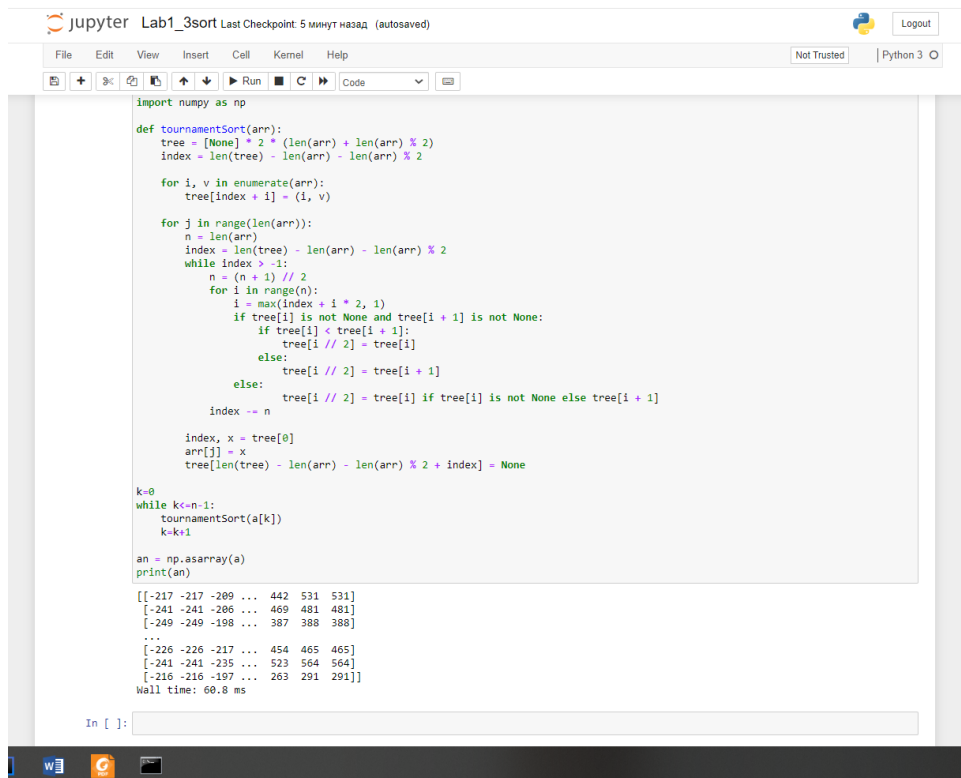
[[-217 -217 -209 ... 442 531 531]
 [-241 -241 -206 ... 469 481 481]
 [-249 -249 -198 ... 387 388 388]
 ...
 [-226 -226 -217 ... 454 465 465]
 [-241 -241 -235 ... 523 564 564]
 [-216 -216 -197 ... 263 291 291]]
Wall time: 5.98 ms

In [5]: %%time
import numpy as np

def tournamentSort(arr):
    tree = [None] * 2 * (len(arr) + len(arr) % 2)
    index = len(tree) - len(arr) - len(arr) % 2
```

Рисунок 6 –быстрая сортировка





```
import numpy as np

def tournamentSort(arr):
    tree = [None] * 2 * (len(arr) + len(arr) % 2)
    index = len(tree) - len(arr) - len(arr) % 2

    for i, v in enumerate(arr):
        tree[index + i] = (i, v)

    for j in range(len(arr)):
        n = len(arr)
        index = len(tree) - len(arr) - len(arr) % 2
        while index > -1:
            n = (n + 1) // 2
            for i in range(n):
                i = max(index + i * 2, 1)
                if tree[i] is not None and tree[i + 1] is not None:
                    if tree[i] < tree[i + 1]:
                        tree[i // 2] = tree[i]
                    else:
                        tree[i // 2] = tree[i + 1]
                else:
                    tree[i // 2] = tree[i] if tree[i] is not None else tree[i + 1]
            index -= n

        index, x = tree[0]
        arr[j] = x
        tree[len(tree) - len(arr) - len(arr) % 2 + index] = None

k=0
while k<=n-1:
    tournamentSort(a[k])
    k=k+1

an = np.asarray(a)
print(an)

[[-217 -217 -209 ... 442 531 531]
 [-241 -241 -286 ... 469 481 481]
 [-249 -249 -198 ... 387 388 388]
 ...
 [-226 -226 -217 ... 454 465 465]
 [-241 -241 -235 ... 523 564 564]
 [-216 -216 -197 ... 263 291 291]]
Wall time: 60.8 ms
```

Рисунок 7 – турнирная сортировка

from random import randint

import numpy as np

n, m = 50, 50

min\_limit = -250

max\_limit = 1007

a = [[randint(min\_limit, max\_limit) for j in range(m)] for i in range(n)]

an = np.asarray(a)

print(an)

%%time

import numpy as np

```

def heap_sort(alist):
    build_max_heap(alist)
    for i in range(len(alist) - 1, 0, -1):
        alist[0], alist[i] = alist[i], alist[0]
        max_heapify(alist, index=0, size=i)

def parent(i):
    return (i - 1)//2

def left(i):
    return 2*i + 1

def right(i):
    return 2*i + 2

def build_max_heap(alist):
    length = len(alist)
    start = parent(length - 1)
    while start >= 0:
        max_heapify(alist, index=start, size=length)
        start = start - 1

def max_heapify(alist, index, size):
    l = left(index)

```

```

    r = right(index)
    if (l < size and alist[l] > alist[index]):
        largest = l
    else:
        largest = index
    if (r < size and alist[r] > alist[largest]):
        largest = r
    if (largest != index):
        alist[largest], alist[index] = alist[index], alist[largest]
        max_heapify(alist, largest, size)

```

```

k=0
while k<=n-1:
    heap_sort(a[k])
    k=k+1

```

```

an = np.asarray(a)
print(an)

```

```

%%time
import numpy as np

```

```

def quick_sort(data):
    less = []
    pivotList = []

```

```

more = []
if len(data) <= 1:
    return data
else:
    pivot = data[0]
    for i in data:
        if i < pivot:
            less.append(i)
        elif i > pivot:
            more.append(i)
        else:
            pivotList.append(i)
    less = quick_sort(less)
    more = quick_sort(more)
    return less + pivotList + more

```

```

k=0

```

```

while k<=n-1:
    quick_sort(a[k])
    k=k+1

```

```

an = np.asarray(a)
print(an)

```

```

%%time

```

```
import numpy as np
```

```
def tournamentSort(arr):
```

```
    tree = [None] * 2 * (len(arr) + len(arr) % 2)
```

```
    index = len(tree) - len(arr) - len(arr) % 2
```

```
    for i, v in enumerate(arr):
```

```
        tree[index + i] = (i, v)
```

```
    for j in range(len(arr)):
```

```
        n = len(arr)
```

```
        index = len(tree) - len(arr) - len(arr) % 2
```

```
        while index > -1:
```

```
            n = (n + 1) // 2
```

```
            for i in range(n):
```

```
                i = max(index + i * 2, 1)
```

```
                if tree[i] is not None and tree[i + 1] is not None:
```

```
                    if tree[i] < tree[i + 1]:
```

```
                        tree[i // 2] = tree[i]
```

```
                    else:
```

```
                        tree[i // 2] = tree[i + 1]
```

```
                else:
```

```
                    tree[i // 2] = tree[i] if tree[i] is not None else tree[i + 1]
```

```
            index -= n
```

```
index, x = tree[0]
```

```
arr[j] = x
```

```
tree[len(tree) - len(arr) - len(arr) % 2 + index] = None
```

```
k=0
```

```
while k<=n-1:
```

```
    tournamentSort(a[k])
```

```
    k=k+1
```

```
an = np.asarray(a)
```

```
print(an)
```

#### **Задание №4:**

Создать публичный репозиторий на github, и запустить выполненное задание в **.ipynb** формате.

#### **Вывод:**

В данной лабораторной были изучены и реализованы основные методы сортировки.