Assignment 1

Sorting Techniques

Name:mostafa ahmed saad

Id:5672

Name:ahmed reda

Id:5325

Name:ahmed ibrahem el adham

Id:5448

- Program description:

  We implement 6 different sorting techniques 3 of them have complexity n^2 and the other 3 have n*log(n) we generate array of size [10,100,1000,10000,100000]
  and for each array size run the 6 different algorithms and compare the running time performance of your algorithms against each other and plot the relationship between the execution time of the sorting algorithm versus the input size.

- Pseudo code for each algorithm:
  1. Quick sort:

```
Quicksort(A,p,r) {
    if (p < r) {
       q <- Partition(A,p,r)
       Quicksort(A,p,q)
       Quicksort(A,q+1,r)
    }
}
```

```
Partition(A,p,r)
    x <- A[p]
    i <- p-1
    j <- r+1
    while (True) {
        repeat
            j <- j-1
        until (A[j] <= x)
        repeat
            i <- i+1
        until (A[i] >= x)
        if (i<-="> A[j]
        else
            return(j)
    }
}
```

2. Merge Sort:

```
public static void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = (l+r)/2;

        mergeSort(arr, l, m);
        mergeSort(arr , m+1, r);

        merge(arr, l, m, r);
    }
}

public static void merge(int arr[], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[] = new int [n1];
    int R[] = new int [n2];

    for (int i=0; i<n1; ++i)
        L[i] = arr[l + i];
    for (int j=0; j<n2; ++j)
        R[j] = arr[m + 1+ j];

    int i = 0, j = 0;

    int k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
```

```
        while (i < n1)
        {
            arr[k] = L[i];
            i++;
            k++;
        }

        while (j < n2)
        {
            arr[k] = R[j];
            j++;
            k++;
        }
    }
```

3. Heap Sort:

```
public static void heapSort(int arr[])
    {
        int n = arr.length;

        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        for (int i=n-1; i>=0; i--)
        {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
            heapify(arr, i, 0);
        }
    }
public static void heapify(int arr[], int n, int i)
    {
        int largest = i;
        int l = 2*i + 1;
        int r = 2*i + 2;

        if (l < n && arr[l] > arr[largest])
            largest = l;

        if (r < n && arr[r] > arr[largest])
            largest = r;

        if (largest != i)
```

```
        {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            heapify(arr, n, largest);
        }
    }
```

4. Bubble Sort:

```
public static void bubbleSort(int[] arr)
    {
        int n = arr.length;
        for(int k =0;k<n-1;k++){
            for(int j =0;j<n-k-1;j++)
            {
               if(arr[j]>arr[j+1])
                {
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }

    }
```

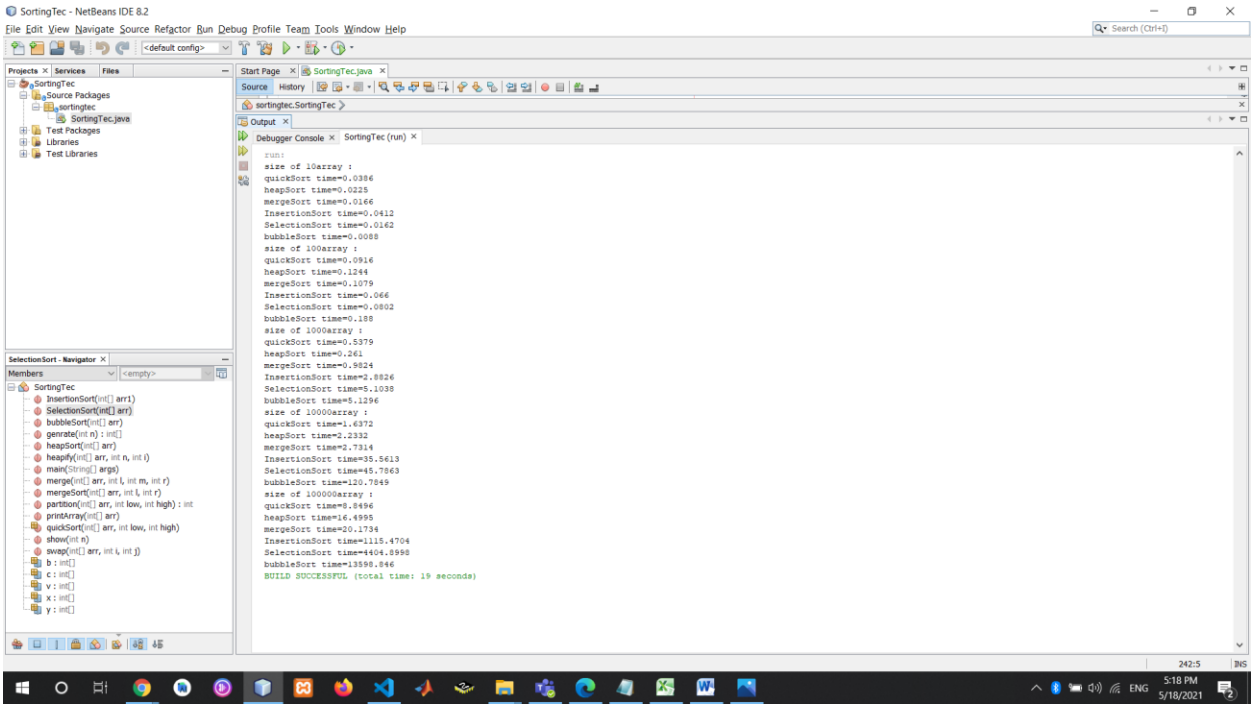
5. Insertion Sort:

```
public static void InsertionSort(int arr1[])
    {
        int n = arr1.length;
        for (int i = 1; i < n; ++i) {
            int key = arr1[i];
            int j = i - 1;
            while (j >= 0 && arr1[j] > key) {
                arr1[j + 1] = arr1[j];
                j = j - 1;
            }
            arr1[j + 1] = key;
        }

    }
```

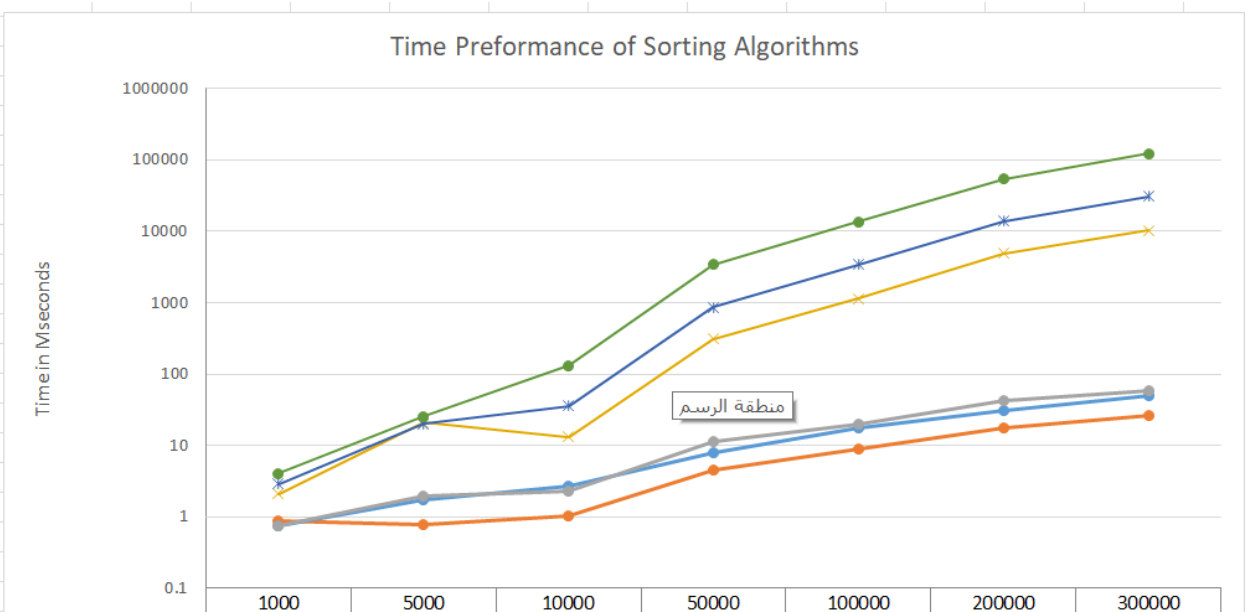
6. Selection Sort:

```
public static  void SelectionSort(int arr[])
 {
    int n = arr.length;



    for (int i = 0; i < n-1; i++)
    {

        int min_idx = i;
        for (int j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;



        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
 }
```

- Sample Run:



- Graph :



	1000	5000	10000	50000	100000	200000	300000
heapSort	0.892699	0.769799	1.039	4.4288	9.005	17.8383	26.57
quickSort	0.745699	1.698201	2.69	7.7849	17.29	30.826	50.13
meargeSort	0.7533	1.918001	2.285	11.12	20.14	42.91	57.63
InsertonSort	2.0942	21.0006	13.1667	308.277	1139.82	4881.44	10283.53
SelectionSort	2.904099	19.995	35.49	856.929	3401.79	13744.62	30699.15
bubbleSort	4.0176	25.71	131.9	3430.843001	13609.6322	54058.93	122107.96