Algorithm Lab Tasks

191-15-12960

1. Bubble Sort

```
package
com.company;

        class BubbleSort
        {
            void bubbleSort(int a[])
            {
                int n = a.length;
                for (int i = 0; i < n-1; i++)
                    for (int j = 0; j < n-i-1; j++)
                        if (a[j] > a[j+1])
                        {
                            int flag = a[j];
                            a[j] = a[j+1];
                            a[j+1] = flag;
                        }
            }
            void printArray(int x[])
            {
                int n = x.length;
                for (int i=0; i<n; ++i)
                    System.out.print(x[i] + " ");
                System.out.println();
            }
            public static void main(String args[])
            {
                BubbleSort ob = new BubbleSort();
                int array[] = {12,8,7,5,2};
                ob.bubbleSort(array);
                System.out.println("SORTED ARRAYS");
                ob.printArray(array);
            }
        }
```

Bubble Sort Algorithm:

Worst case performance O(n^2)
Best case performance O(n), Average case performance O(n^2)

2. Linear Search

```
package
com.company;

class LinearSearch
{
   public static int search(int a[], int x)
   {
      int n = a.length;
      for(int i = 0; i < n; i++)
      {
         if(a[i] == x)
            return i;
      }
      return -1;
   }

   public static void main(String args[])
   {
      int x[] = { 5,25,90,24 };
      int v = 90;

      int result = search(x, v);
      if(result == -1)
         System.out.print("Element is not available in array");
      else
         System.out.print("Element found at index " + result);
   }
}
```

Linear Search Algorithm:

Worst case performance O(n)
Best case performance O(1)
Average case performance O(n)

3. Insertion Sort

```
package
com.company;

        import java.util.Arrays;

        class InsertionSort {

          void insertionSort(int array[]) {
            int size = array.length;

            for (int step = 1; step < size; step++) {
              int key = array[step];
              int j = step - 1;

              while (j >= 0 && key < array[j]) {
                array[j + 1] = array[j];
                --j;
              }

              array[j + 1] = key;
            }
          }
          public static void main(String args[]) {
            int[] data = { 9, 5, 1, 4, 3 };
            InsertionSort is = new InsertionSort();
            is.insertionSort(data);
            System.out.println("Sorted Array in Ascending Order: ");
            System.out.println(Arrays.toString(data));
          }
        }
```

Insertion Sort Algorithm:

Worst case performance O(n2)
Best case performance O(n)
Average case performance O(n2)

4. Selection Sort

```
package
com.company;

                import java.util.Scanner;

                public class Selection
                {
                   public static void main(String args[])
                   {
                      int size, i, j, temp;
                      int arr[] = new int[50];
                      Scanner scan = new Scanner(System.in);

                      System.out.print("Enter Array Size : ");
                      size = scan.nextInt();

                      System.out.print("Enter Array Elements : ");
                      for(i=0; i<size; i++)
                      {
                         arr[i] = scan.nextInt();
                      }

                      System.out.print("Sorting Array using Selection Sort Technique..\n");
                      for(i=0; i<size; i++)
                      {
                         for(j=i+1; j<size; j++)
                         {
                            if(arr[i] > arr[j])
                            {
                               temp = arr[i];
                               arr[i] = arr[j];
                               arr[j] = temp;
                            }
                         }
                      }

                      System.out.print("Now the Array after Sorting is :\n");
                      for(i=0; i<size; i++)
```

```
        {
            System.out.print(arr[i]+ "  ");
        }
    }
}
```

Selection Sort Algorithm:

Worst case performance $O(n^2)$
Best case performance $O(n^2)$
Average case performance $O(n^2)$

5. Binary Search

```
class BinarySearchExample{
 public static void binarySearch (int arr[], int first, int last, int key){
   int mid = (first + last)/2;
   while( first <= last ){
     if (arr[mid] < key) {
       first = mid + 1;
     }else if ( arr[mid] == key ){
       System.out.println("Element is found at index: " + mid);
       break;
     }else{
        last = mid - 1;
     }
     mid = (first + last)/2;
   }
   if ( first > last ){
     System.out.println("Element is not found!");
   }
 }
 public static void main(String args[]){
       int arr[] = {10,20,30,40,50};
       int key = 30;
       int last=arr.length-1;
       binarySearch(arr,0,last,key);
}
 }
```

Binary search Algorithm:

Worst case performance O(log n)
Best case performance O(1)
Average case performance O(log n)

6. Merge Sort

```
public class MyMergeSort
{
    void merge(int arr[], int beg, int mid, int end)
{

    int l = mid - beg + 1;
    int r = end - mid;

    intLeftArray[] = new int [l];
    intRightArray[] = new int [r];

    for (int i=0; i<l; ++i)
    LeftArray[i] = arr[beg + i];

    for (int j=0; j<r; ++j)
    RightArray[j] = arr[mid + 1+ j];


    int i = 0, j = 0;
    int k = beg;
    while (i<l&&j<r)
{
    if (LeftArray[i] <= RightArray[j])
{
    arr[k] = LeftArray[i];
    i++;
}
    else
{
    arr[k] = RightArray[j];
    j++;
}
    k++;
}
    while (i<l)
{
    arr[k] = LeftArray[i];
```

```
        i++;
        k++;
    }
        while (j<r)
    {
        arr[k] = RightArray[j];
        j++;
        k++;
    }
    }
        void sort(int arr[], int beg, int end)
    {
        if (beg<end)
    {
        int mid = (beg+end)/2;
        sort(arr, beg, mid);
        sort(arr , mid+1, end);
        merge(arr, beg, mid, end);
    }
    }
        public static void main(String args[])
    {
        intarr[] = {90,23,101,45,65,23,67,89,34,23};
        MyMergeSort ob = new MyMergeSort();
        ob.sort(arr, 0, arr.length-1);

        System.out.println("\nSorted array");
        for(int i =0; i<arr.length;i++)
    {
        System.out.println(arr[i]+"");
    }
    }
    }
```

Time complexity of Merge Sort is O(n*Log n) in all the 3 cases

7. Quick Sort

```
public class QuickSort {
public static void main(String[] args) {
 int i;
 int[] arr={90,23,101,45,65,23,67,89,34,23};
 quickSort(arr, 0, 9);
 System.out.println("\n The sorted array is: \n");
 for(i=0;i<10;i++)
 System.out.println(arr[i]);
}
  public static int partition(int a[], int beg, int end)
{

 int left, right, temp, loc, flag;
 loc = left = beg;
 right = end;
 flag = 0;
 while(flag != 1)
 {
    while((a[loc] <= a[right]) && (loc!=right))
    right--;
    if(loc==right)
    flag =1;
    elseif(a[loc]>a[right])
    {
       temp = a[loc];
       a[loc] = a[right];
       a[right] = temp;
       loc = right;
    }
          if(flag!=1)
    {
       while((a[loc] >= a[left]) && (loc!=left))
       left++;
       if(loc==left)
       flag =1;
       elseif(a[loc] <a[left])
     {
          temp = a[loc];
```

```
                a[loc] = a[left];
                a[left] = temp;
                loc = left;
              }
            }
          }
          returnloc;
      }
      static void quickSort(int a[], int beg, int end)
      {

          int loc;
          if(beg<end)
          {
              loc = partition(a, beg, end);
              quickSort(a, beg, loc-1);
              quickSort(a, loc+1, end);
          }
      }
}
```

Quick Sort Algorithm:

Worst case performance $O(n^2)$
Best case performance O(n)
Average case performance O(n log n)

Thank You Sir