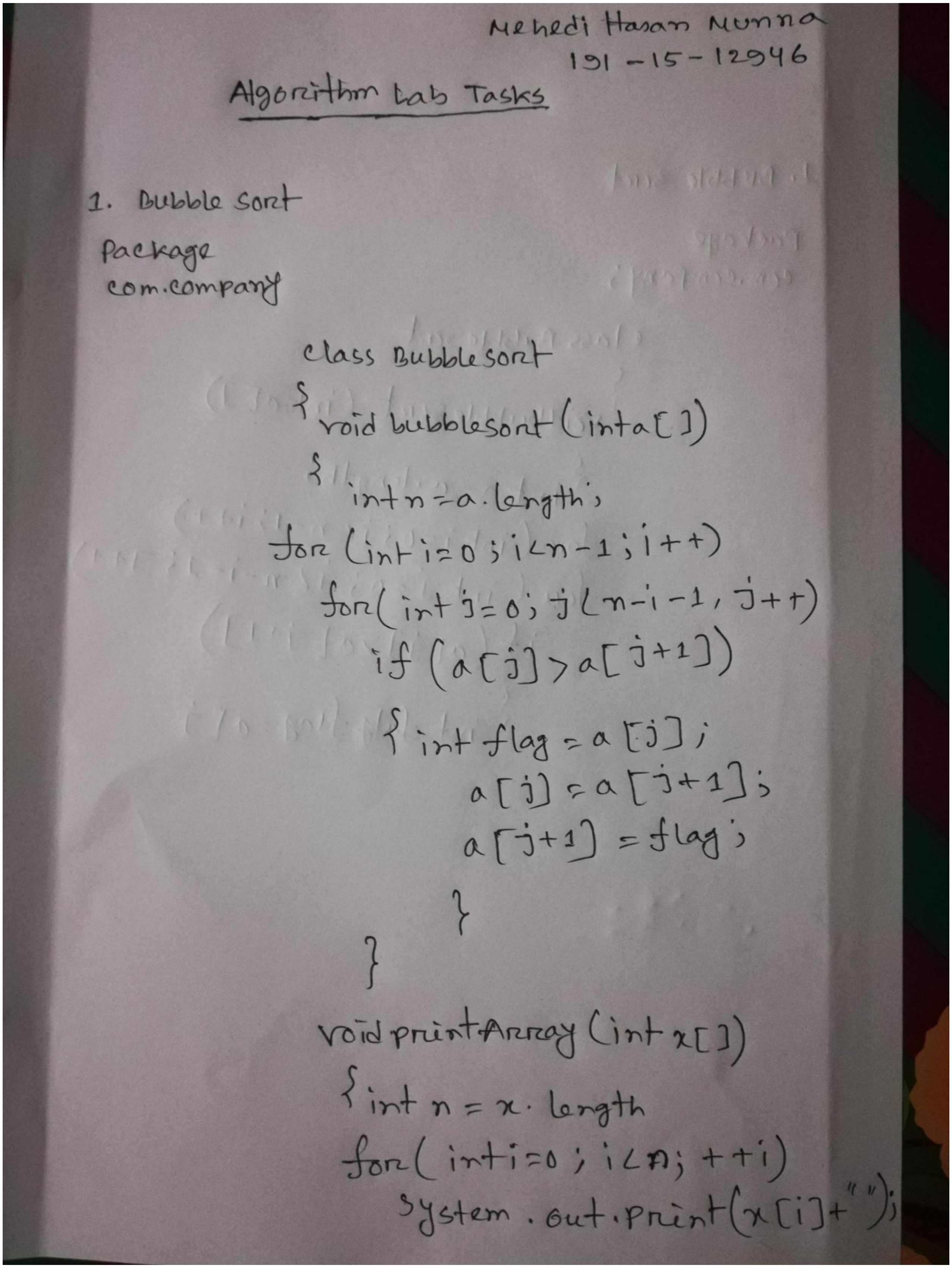


Mehedi Hasan Munna
191-15-12946

Algorithm Lab Tasks

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.print Array (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 (inta[ ], intx)
        { 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 r=90;
        int result = search (x,r);
         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 insertion sort (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]){
```

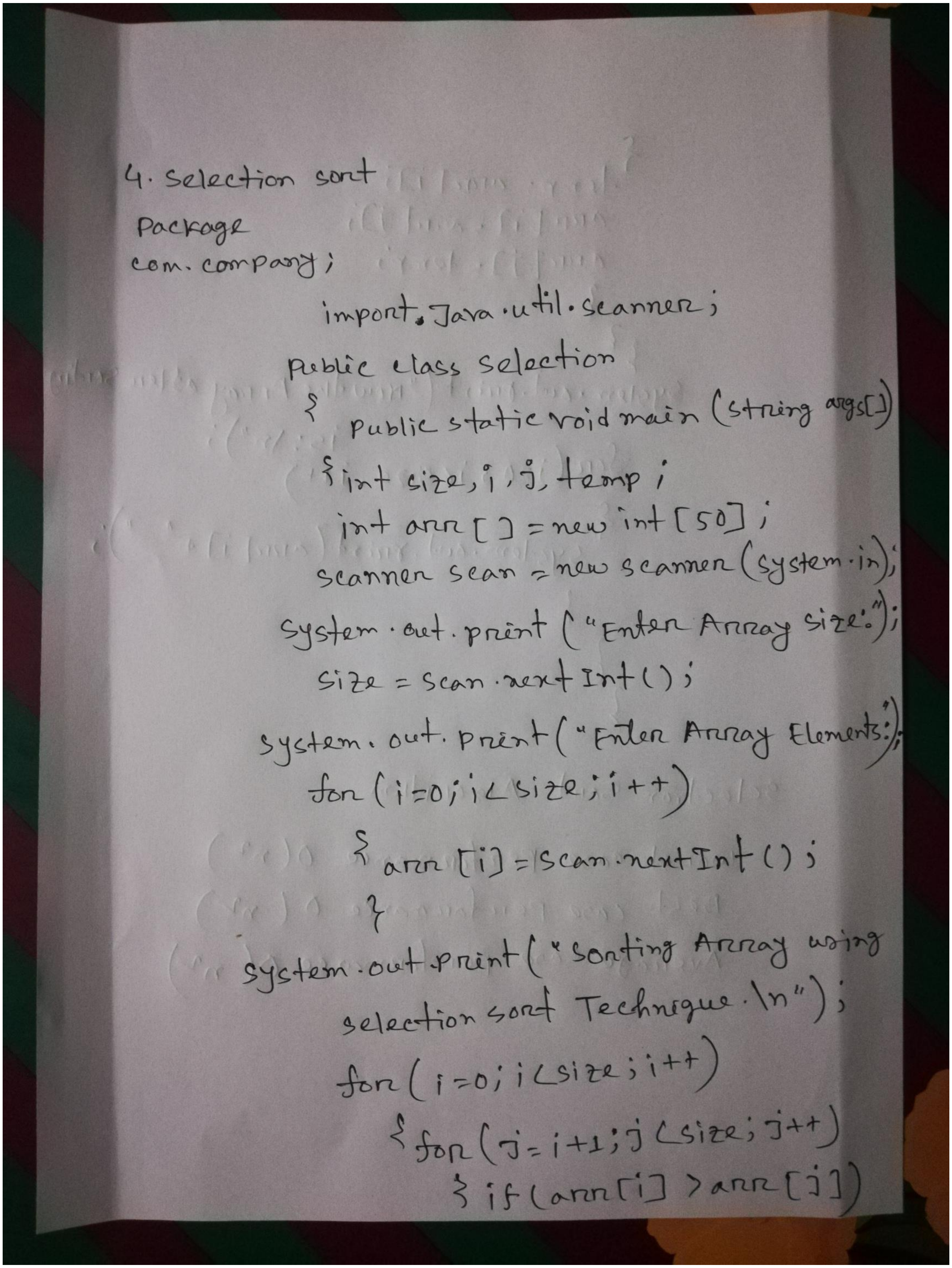
```
array [j+1] = array [j];
      --j;
   }
  array[ j+1] = key;
   }
  }
Public static void main (String args []){
      int [] data = {9,5,1,4,3}
   Insertionsort is = new Insertion Sort();
      is.insertion Sort (data);
   System.out.println("sorted Array in
                Ascending order :");
   system.out.println(Array.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 ("Elements 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.lenght -1;
   binarysearch(arr, 0, last, key);
 }
}
```

Binary search Algorithm:

worst case performance O(logn)

Best case performance O(1)

Average case performance O(logn)

6. Merge Sort

```
Public class My MergeSort
  { void merge(int arr[], int beg, int mid,
                                 int end)
  { int 1 = mid - beg + 1;
     int r = end - mid;
   int Left Array[] = new int[1];
   int Right Array[] = new int[r];
 for (int i = 0; i<1; ++i)
   LeftArray[i] = arr[beg + i];
   for (int j = 0; j<r; ++j)
   Right Array[j] = arr[mid+1+j];
   int i = 0, j = 0
     int k = beg;
   while(i<1 && j<r)
```

```
{ if (leftArray [i] <= RightArray [j])
    { arr[k] = leftArray [i];
        i++;
    }
  else
    { arr[k] = RightArray [j];
          j++;
      }
      k++;
    }
    {
      arr [k] = leftArray [i];
      i++;
      k++;
    }
    while (j<r)
    { arr[k] = Right Array [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};
 MyMerge sort ob = new My Merge sort();
     ob.sort(arr, 0, arr.length-1);
 system.out.println("\n sorted array");
 for(int i=0; i<arr.length; i++)
 {system.out.print(arr[i]+" ");
   }
   }
   }
```

Time complexity of Merge sort is (n* logn)

in all the 3 cases)

7. Quick sort

```
Public class Quicksort{
 Public static void main( Strin [] 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;
    else if(a[loc]<a[left])
          {temp=a[loc];
           a[loc]=a[left];
            a[left]=temp;
             loc =left;
              }
            }
          }
         return loc;
        }
  Static void quicsort(int a[],int beg; int
                                           end)
```

```
{int loc;
 if(beg<end)
   {loc=partition(a,beg,end);
    quicksort(a,beg, loc-1);
    quick sort(a, loc+1, end);

   }

  }
}
```

Quick sort Algorith:

worst case performace $O(n^2)$

Best case performace $O(n)$

Average case performance $O(n \log n)$