CS112
Homework 2 Solutions

Q1.

```
package testhw;
import java.util.Random;

/*
 * A program to compare the average running time of Quick, Insertion and Merge sort algorithms
 */
public class CompareSorts{
	static int N = 100;
	static int [] arr = new int[N];
	static int [] aux = new int[N];

	public static void main(String args[]){
		initialize(arr);
		long sum=0;
		long start,stop;

		Random rand =new Random();

		for(int i=0; i<N; i++){
			aux[i]=rand.nextInt(500);
		}

		initialize(arr);
		sum=0;
		for(int i=1; i<=10;i++){
			start= System.nanoTime();
			QuickSort(arr);
			stop= System.nanoTime();
			sum=sum+start-stop;
		}
	System.out.println("Average running time of Quick Sort  "+ sum/10);
		sum=0;
		initialize(arr);
		for(int i=1; i<=10;i++){
			start = System.nanoTime();
			InsertionSort(arr);
			stop = System.nanoTime();
			sum = sum+start-stop;
		}
	System.out.println("Average running time of Insertion Sort  "+ sum/10);
		sum=0;
		initialize(arr);
```

```
		for(int i=1; i<=10;i++){
			start= System.nanoTime();
			MergeSort(arr);
			stop= System.nanoTime();
			sum=sum+start-stop;
		}
		System.out.println("Average running time of Merge Sort  "+ sum/10);

	}
	private static void initialize(int[] arr){

		for(int i=0; i<N; i++){
			arr[i]=aux[i];
		}


	}

	private static void InsertionSort(int[] arr){
		/*
		 * An implementation of Insertion Sort
		 */
	}

	private static void QuickSort(int[] arr){
		/*
		 * An implementation of Quick sort
		 */
	}

	private static void MergeSort(int[] arr){
	/*
	 * An implementation of merge sort
	 */
	}


}
```

Depending on the version of the sort algorithm implemented, we expect that the running time of quicksort <mergesort< insertion sort in general.

In the average case Quick sort is nlgn but quadratic in the worst case; mergesort is always nlgn while Insertion sort is always quadratic.

Q2

```
package testhw;
/*
 * Iterative MergeSort: to perform iterative merge sort on an array of int
 */

import java.util.Random;
public class IterativeMergeSort {
   static final int N=9;
   static int [] arr=new int[N];
   static int [] aux=new int[N];

   public static void main(String args[]){
            Random rand = new Random();
            for(int p=0; p<arr.length; p++){
                  arr[p]=rand.nextInt(4000);
                  System.out.println(arr[p]);
            }
            System.out.println("                  ");

            mergeSort(arr);   // do merge sort
            for(int p=0; p<arr.length; p++){
                 System.out.println(arr[p]);
            }
   }
   private static void mergeSort(int[] arr){
      /*
       * mergeSort method: performs iterative merge sort on arr
       */
            int l1=0, h1=0, l2=0, h2=0, num, cnt; // lower and upper indexes of the sub arrays to be /
                                                  //merged
            double size =(Math.log(N)/Math.log(2));
            for(int k=0;k<=(int)Math.ceil(size);k++){
               num =(int)Math.ceil(N/Math.pow(2,k+1));
               cnt=num;
               for(int i=0;i<N;i=i+(int)Math.pow(2,k+1)){
                     l1=i;
                     h1=i+(int)Math.pow(2,k)-1;
                     l2=h1+1;
                     h2=l2+(int)Math.pow(2,k)-1;
                     cnt--;
                     if(l2<N && h2>=N) h2=N-1; // to merge a case when the last sub array is smaller
                                              //than others
                     if(h2<N){          // to exclude the case when the second sub array is out of range
                        if(num%2==1 && cnt>=0){   // non-power of two: to exclude the last odd group
```

```
                                                    //from being merged
                        mergeOrderedSubArrays(arr,l1,h1,l2,h2);
                        }
                    else if(num%2==0) // to merge power of two sub arrays
                        mergeOrderedSubArrays(arr,l1,h1,l2,h2);
                }
            }
        }
    }
  private static void mergeOrderedSubArrays(int [] arr,int l1, int h1, int l2, int h2){
    /*
     * To merge two ordered sub arrays starting and ending at indices l1,h1 and l2,h2 respectively
     */
            for (int i=l1;i<=h1; i++){
               aux[i]=arr[i];
            }
            for (int i=l2;i<=h2; i++){
               aux[i]=arr[i];
            }
            int ind=l1,ind1=l1,ind2=l2;
            while(ind1<=h1 && ind2<=h2){
                if(aux[ind1]<= aux[ind2]){
                   arr[ind]=aux[ind1];
                   ind1++;
                }
                else{
                   arr[ind]=aux[ind2];
                   ind2++;
                }
                ind++;
            }

            if(ind1<=h1){
                for(int p=ind1; p<=h1; p++){
                   arr[ind]=aux[p];
                   ind++;
                }
            }
            else if(ind2<=h2){
                for(int p=ind2; p<=h2; p++){
                   arr[ind]=aux[p];
                   ind++;
                }
            }
  }
}
```

Q3

1. $O(N^2)$
   An example is when all the 15 elements are of equal value or when the middle element always partitions the remaining elements into (n-1) on the left and none on the right.

2. $O(N^2)$
   An example is when all the 15 elements are of equal value or when the median always partitions the remaining elements into (n-2) on the left and 1 element on the right.

3. To determine the optimal size of the base case, we can measure and compare the average running time of Quick sort and Insertion sort. We can choose the size for which the average running time of quick sort equals that of insertion sort.