**Lab 4 Solution**

1. **Determine whether InsertionSort, BubbleSort, SelectionSort from lesson 3 are stable sorting algorithms, and in each case, explain your answer.**

Sorting algorithm is said to be stable if two object with the same key should be in the same order in output array as they appear in the input array.

InsertionSort and BubbleSort are stable sorting algorithm.

1. InsertionSort: In insertion sort we are picking an element and placing them in correct location. Normally, if element is larger then we are placing it after current element, but we are not swapping element with same key.
2. BibbleSort: we are swapping element if and only if element is larger than current element. Idea is to place large element to the end. However, no swapping in case of equal key element, so it is stable.
3. SelectionSort: Selection sort is unstable, because we are swapping element if it is minimum than beginning element. However, in case of the same key element selection sort swap the element to beginning position. Hence, it is unstable (it can change the position of the element even they have same key)
4. **Solve the following problem with a recursive algorithm: Given a list with n elements, put the elements of the list in reverse order. Compute the running time of your algorithm (hint: count self-calls).**

public static List reverseList(List<Integer> list){  
 if(list.isEmpty()){  
 return list;  
 }  
 return *swap*(list,0,list.size()-1);  
  
}  
public static List swap(List<Integer> list,int firstIndex,int lastIndex){  
 int temp=0;  
 if(firstIndex>lastIndex){  
 return list;  
 }  
 temp=list.get(lastIndex);  
 list.set(lastIndex, list.get(firstIndex));  
 list.set(firstIndex,temp);  
 return *swap*(list,firstIndex+1,lastIndex-1);  
  
  
}

**Running Time complexity** for reverseList is Q(1) and Running time complexity for swap() function is Q(n).

1. **Merge Sort Plus:**

void mergeSort(int[] tempStorage, int lower, int upper) {  
 System.*out*.println("############## "+(upper-lower));  
 if(lower==upper){  
 return;  
 }  
  
  
 if((upper-lower)<=20) {  
 *insertionSort*(theArray,lower,upper);  
 }  
   
 else {  
 int mid = (lower+upper)/2;  
 mergeSort(tempStorage,lower,mid); //sort left half  
 mergeSort(tempStorage,mid+1, upper); //sort right half  
 merge(tempStorage,lower,mid+1,upper); //merge them  
 }  
 }  
  
  
public static int[] insertionSort(int[] anArray,int lower,int upper) {  
 //System.out.println("Result");  
 if(anArray == null || anArray.length <= 1) {  
 return anArray;  
 }  
 int temp = 0;  
 int j = 0;  
 for(int i = lower; i < upper; ++i) {  
 temp = anArray[i];  
 j=i;  
 while(j>0 && temp < anArray[j-1]){  
 anArray[j] = anArray[j-1];  
 j--;  
 }  
 anArray[j]=temp;  
 }  
 return anArray;  
}

Merging automatically takes care of sorting the elements however, we can use insertion sort below certain number of iteration; in our case, less than equal to 20 for small number of sort routine. This will improve the performance of merge sort. Although merge sort is O(nlogn) and insertion sort is O(n2), for small array insertion sort has faster performance.