**Bubble Sort:**

1. Improve the BubbleSort implementation so that when the input array becomes sorted after some runs of outer for loop, the algorithm will stop. Call your new Java file BubbleSort1.java

private void bubbleSort() {  
 for(int i=0;i<arr.length;i++) {  
 int swap=0;  
   
 for(int j=0;j<arr.length-1;j++) {  
 if(arr[j]>arr[j+1]) {  
 swap++;  
 swap(j,j+1);   
 }  
   
 }  
 if(swap==0) {  
 break;  
 }  
   
   
   
 }  
}  
private int[] swap(int i,int j) {  
 int temp=arr[i];  
 arr[i]=arr[j];  
 arr[j]=temp;  
 return arr;  
}

1. Recall that in BubbleSort, at the end of the first pass through the outer loop, the largest element of the array is in its final sorted position. After the next pass, the next largest element is in its final sorted position. After the ith pass (i=0,1,2,…), the largest, second largest,…, i+1st largest elements are in their final sorted position. Use this observation to cut the running time of BubbleSort in half. Implement your solution in code, and prove that you have improved the running time in this way. Call your new Java file, which contains the improvements from this problem and the previous problem, BubbleSort2.java

private void bubbleSort() {  
 for(int i=0;i<arr.length;i++) {  
 int swap=0;  
   
 for(int j=0;j<arr.length-1-i;j++) {  
 if(arr[j]>arr[j+1]) {  
 swap++;  
 swap(j,j+1);   
 }  
   
 }  
 if(swap==0) {  
 break;  
 }  
   
   
   
 }  
}  
private int[] swap(int i,int j) {  
 int temp=arr[i];  
 arr[i]=arr[j];  
 arr[j]=temp;  
   
 return arr;  
}

1. When i test the bubbleSort 1 and 2 then I got the following result:

**742 ms -> BubbleSort2**

**809 ms -> BubbleSort1**

**1121 ms -> BubbleSort**

Running time for bubbleSort2 is low compare to bubbleSort and bubbleSort1.

1. In BubbleSort1 we are checking that if there is no Swap, which means the larger elements are already in the right and no swap is needed; more precisely no more sorting needed, array is sorted so break statement is executed.
2. In BubbleSort2 we are improving performance by cutting the loop size of nested loop. So the length of loop is len-I, which means last elements are already sorted, no need to iterate through last element.