**Problem01Sorting**

1)

This sorting algorithm is known as Merge sort. This is the correct answer because merge sort is stable and runs on O(N) and O(lgn). This is because the function sort() has a run time of O(N)

**void** sort (**int** arr[], **int** l, **int** r)

{

**if** (l<r)

{

**int** m = (l+r)/2;

sort(arr, l,m);

sort(arr, m+1,r);

merge(arr, l,m,r);

}

and merge has a sort time of O(lgN)

**while** (i<n1 && j<n2)

{

**if** (L[i] < R[i])

{ arr[k] = L[i];

i++;

}

**else**

{

arr[k] = R[i];

j++;

}

k++;

}

**if** (i<n1)

{

arr[k] = L[i];

i++;

k++;

}

**if** (j<n2)

{

arr[k] = R[j];

j++;

k++;

}

. To find the complete run time we must add these two together which makes merge sort run O(NlgN).

2)

The best case time complexity of Merge Sort is O(NlgN). This is because regardless of if the array is sorted the algorithm will still do the same exact steps. It will split everything and then find which one is smaller and then add it to the array.

3)

The worst case time complexity of Merge Sort is O(NlgN). Merge Sort has the same time complexity in all its cases; best , worse, and average. The array will be broken down and then build back up.