Problem 1:

1. This problem I choose to do merge sort because this code is stable and it meets the requirement of nlogn as its best and worst time condition.

**if**(left[i] <= right[j]){

a[k] = left[i];

i++;

}

**else**{

a[k] = right[j];

j++;

}

k++;

the code above makes sure that the left side is smaller than the right side. This means if left and right has the same number that means the number of that number appearing first will still appear first in the sorted array, thus making it stable.

2. The best time is nlogn in this code. To get this we look at

ExamSort(left,mid);

ExamSort(right,size-mid);

merge(left,right,a);

inside this code we will divide the array into two separate arrays and search from there until the size is 1. This will cause the log n time complexity.

3. With the same code the time complexity will be nlogn for the worst case. Using the code.

ExamSort(left,mid);

ExamSort(right,size-mid);

merge(left,right,a);

this is because we dont care about the array, only factor that affects the time complexity is the size of the array. Thus this code will have the same worst and best case.