Problem 1B

1.
This is a stable sort because the splits and the merges in the merge sort preserve the order of equal elements.

2.
The the algorithm that I ran was merge sort, which has a best case time complexity of O(NlogN). The algorithm first splits the array in two smaller arrays, and recursively splits those arrays, until there is only one element, and this takes O(Log N) time :

```
if len(array) > 1:
    mid = len(array)//2        //This splits the array in half
    left = array[:mid]         // this makes the left array
    right = array[mid:]        // this makes the right array

    mergeSort(left)            //this recursively runs mergeSort on the left array
    mergeSort(right)           //this recursively runs mergeSort on the right array
```

After the array cannot be split more, the elements are merged, this process takes O(N) time, because each element has to be put back into the array at its appropriate index.

```
    i = 0
    j = 0
    k = 0

    while i < len(left) and j < len(right):
      if left[i] < right[i]:
        array[k] = left[i]              //this adds to the final array
        i+=1
      else:
        array[k] = right[j]             //this adds to the final array
        j+=1
      k+=1

    while i < len(left):
      array[k] = left[i]                //this adds to the final array
      i+=1
      k+=1

    while j < len(right):
      array[k] = right[j]               //this adds to the final array
      j+=1
      k+=1
```

So together, the best case time complexity is O(NlogN)

2. The worst case time complexity is also O(NlogN) because regardless of the order of the array, the mergeSort will split the arrays into other smaller arrays until there is only one elements in O(logN) time, and will still take O(N) time to reconstruct the new array. So in total, the worst case time complexity is O(NlogN).

```
if len(array) > 1:
    mid = len(array)//2        //This splits the array in half
    left = array[:mid]         // this makes the left array
    right = array[mid:]        // this makes the right array

    mergeSort(left)            //this recursively runs mergeSort on the left array
    mergeSort(right)           //this recursively runs mergeSort on the right array
```

After the array cannot be split more, the elements are merged, this process takes O(N) time, because each element has to be put back into the array at its appropriate index.

```
    i = 0
    j = 0
    k = 0

    while i < len(left) and j < len(right):
        if left[i] < right[i]:
            array[k] = left[i]         //this adds to the final array
            i+=1
        else:
            array[k] = right[j]        //this adds to the final array
            j+=1
        k+=1

    while i < len(left):
        array[k] = left[i]             //this adds to the final array
        i+=1
        k+=1

    while j < len(right):
        array[k] = right[j]            //this adds to the final array
        j+=1
        k+=1
```