• I certify that every answer in this assignment is the result of my own work; that I have neither copied off the Internet nor from any one else's work; and I have not shared my answers or attempts at answers with anyone else.

1: . QUICKSORTLOWEST-K(A, p,r, k) is almost like QUICKSORT(A, p,r) (the algorithm we have seen in class). There are two differences. First, it has an extra input k, a positive integer. Second, it only needs to sort (in non-decreasing order) the lowest k elements, leaving the rest of the input in any order.
Assume that the initial call is QUICKSORTLOWEST-K(A, 1, n, k) and $1 \leq k \leq n$
While it is possible to ignore k and let the algorithm execute QUICKSORT, your algorithm should do better.
Write pseudocode for QUICKSORTLOWEST-K(A, p, q, k) with comments that explain your strategy. (Use the notation we have used in class and is in the text. Make use of the PARTITION algorithm we have seen in class.)
Explain why your algorithm is better.

```
QUICKSORTLOWEST-K(A, p, r, k)
    if p < r then
        q ← PARTITION(A, p, r)
        if q≥ k then
            QUICKSORTLOWEST-K(A,p,q,k)
| > if k is less than or equal to the pivot q, then there is no need to look at the split beyond q
| > Therefore the recursive call is only called once for the section before the partition
        else then
            QUICKSORTLOWEST-K(A,p,q-1,k)
            QUICKSORTLOWEST-K(A, q+1, r, k)
```

| >if k is greater than the pivot, than anything before the pivot must be organised and the second
| >half must be organised until a pivot is less than or equal to k

This algorithm guarantees that everything at k and lower is sorted. It also does better because a significant amount of recursive calls will not be done because of the if statement. Quick sort will not be called for any unnecessary segments.

2: What is the smallest change you would make to COUNTINGSORT (the algorithm we have seen in class) to reverse the sorting order (i.e., to make it sort in non-increasing order) without impacting either stability or execution time?
(a) Explain why your changed algorithm would work.
(b) Explain why your changed algorithm would preserve stability
(c)Explain why your changed algorithm would preserve execution time.

My changes to the code for Counting sort are below in red:

```
COUNTINGSORT(A, B, k)
  n ← A.length
  for i ← 0 to k do
    C[i] ← 0
```

```
  for j ← 1 to n do
    C[A[j]] ← C[A[j]] + 1
  for i ← 1 to k do
    C[i] ← C[i] + C[i-1]
for j ← 1 to n
  B[n+1 - C[A[j]]] ← A[j]
    C[A[j]] ← C[A[j]] - 1
```

(a & b) My Algorithm changes the placement of the item in B to where the element should be in decreasing order (the size of the array - its place in increasing order). However, setting this does not preserve stability so we must reverse the last for loop to account for this (from 1 to n instead of n to 1).
(c) Execution time would still be preserved because none of the execution times for any of the for loops have been changed (the last loop still runs to n).