Josh Barber

Problem 1

Pseudo code:

```
k = max A value
for i=0 to k: B(i)=0
for j=0 to A.length: B[A[j]] = 1
for m=1 to k: B[m] = B[m] + B[m-1]

# values = B[b] - B[a-1]
```

Analysis:

Knowing array A max value to be k, we create B array in size of k in O(k) time. We then iterate over A and place a 1 value in B's array where the index of B is equal to the A array's indexed value in O(n) time. Then iterate over B from left to right while adding all the 1 values in O(k) time. Any calculation of how many values are between param 'a' and 'b' is done in O(1) time.

Josh Barber

Problem 3

Pseudo code:

```
CountingSort(A, k):
    B = [ ]
    C = [ ]
    for i=0 to 10: C[i] = 0

    for j=0 to A.length:
        d = A[j]/10^k mod 10
        C[d] = C[d] + 1
    for i=0 to 10: C[i] = C[i] + C[i-1]
    for j = A.length to 0
        d = A[j]/10^k mod 10
        C[d] = C[d] - 1
        B[C[d]] = A[j]

return B

RadixSort(A, d):
    r = # of digits in max number of A
    for i=0 to r: A = CountingSort(A, i)

    return A
```

Analysis:

Using RadixSort we just need a stable sorting algorithm. In the pseudo code, counting sort is used. In the counting sort, we initialize an array of size 10 so we can sort digits from 0-9 from values in the array that needs sorting from ones to tens to hundreds, etc. until the array that needs sorting is fully sorted. RadixSort is the initiater of shifting from the ones to tens to hundreds.