Homework 8

Problem 8.1 Sorting in Linear Time

a) **Pseudocode:**

```
COUNTINTERVAL (A[], n , k, a, b) // based on Counting Sort algorithm
    new Count array of size k+1 // create a new array Count
    for i = 0 to k  // time complexity for this loop is O(k)
        Count[i] = 0 // initialize count array with 0s
    for i = 0 to n O(k) // time complexity for this loop is O(n)
        ++Count [A[i]]  // count the occurrences of each element in the array
    for i=0 to k  // time complexity for this loop is O(k)
        Count[i] = Count[i] + Count [i+1]  // find the the cumulative count
  return Count[b-1] - Count[a-1] // this function takes constant time, O(1)
// Overall time : Pre-processing part → O(k) + O(n) + O(k) = O (n+k);  O (n+k) + O(1) = O (n+k)
```

c) Worst case happens when the elements of the array are in close range for instance [1.23 , 1.6, 1.3, 1.75, 1.8]. In this case they are likely to be placed in the same bucket. Consequently, the time complexity will depend on the sorting algorithm used to sort the elements of a bucket. If insertion sort is used, then the time complexity becomes $O(n^2)$. If the elements are in reverse order, the complexity would become even worse.

Problem 8.2 Radix Sort:

b) Time Complexity:

Worst case occurs when all the elements of the array are in close range and therefore, placed in the same bucket. Then, the time depends on the recursion. Recursion is and it is called based on the number of digits present in maximum element. Consequently, the time complexity is **O(kn).**

Average case occurs when the elements of the array are splitted in the buckets, where half of the buckets have several elements, and the other half has just 1 element or does not have any at all. In the first half, recursion is called k times. In the second half, recursion is not called since

the elements are already sorted. Consequently, the time complexity is **O(k * n/2) + O(n) = O(kn).**

Best case occurs when each bucket has the same number of elements i.e. one element. This means the buckets are sorted and recursion is not performed. Consequently, the time complexity is **O(n).**

Space Complexity:

The array contains n elements. Recursion is performed k times - k is the number of digits present in maximum element. In every recursive call, stack space is added level by level. Consequently, the space complexity is **O(kn).**