**ADS Homework 7**

**Problem 7.1: Sorting in Linear Time**

**C**)

Pseudocode:

CountPreprocessing(arr, n, k)

for i=0 to k do:

A[i] = 0;

end for

for j=0 to n do:

A[arr[j]] = A[arr[j]] + 1;

end for

for i=1 to k do:

A[i] = A[i] + A[i-1];

end for

return A[ ];

// We have two loops one running up to n and another up to k giving us Θ(n+k) time complexity

end function

range(A, a, b) // this function runs in O(1) time complexity

return (A[b] - A[a-1]);

end function

**E**)

In the worst-case scenario, all elements of the array will end up in the same bucket. As such, the time complexity will be equal to the complexity of the algorithm used to solve each bucket, which in this case, is insertion sort which runs in O(n2) time complexity meaning bucket sort has O(n2) time complexity. An example could be the following array {0.12, 0.15, 0.11, 0.19, 0.14, 0.16, 0.13, 0.18} in which all the entries end up in the bucket B[1] all having 1 as their tenth place digit. So, the elements of this example will all end up in the bucket B[1] and insertion sort would be the only one used on this bucket hence giving the worst case time complexity of O(n2).

**Problem 7.2: Radix Sort**

**B**)

The time complexity of Radix Sort is O(k\*n) and its space complexity is O(k+n). Here n is the number of elements and k is the number of bits required to represent the largest element in the array. The explanation is as follows. The time complexity of the implementation of radix sort depends on the entries that end up in each bucket. In all(best, average and worst) cases, even if all entries are split into all buckets, the number of operations will always be equal to the number of operations when all the entries would have been in a single bucket as the sum of the elements of all buckets is always the same which means the time complexity depends on the number of digits of the largest number. Taking the case when all entries have k number of digits that differ from each other only by the least significant digit, all entries will become part of a single bucket and since all the other operations are carried out in O(n) except the recursion, there will be k splits into buckets giving the time complexity O(k\*n). This means the time complexity is linear. Moving on to the space complexity, each time the function is called, it uses two arrays one which has size n and another with size k and so the space complexity is O(k+n).