Assignment 7

Done By: Nayan Man Singh Pradhan
Submitted: 23rd March 2020

Problem 7.1 (a)
Implemented in countingSort.cpp (make countingSort).

Problem 7.1 (b)
Implemented in bucketSort.cpp (make bucketSort).

Problem 7.1 (c)
Pseudocode for implementation that counts the number of integers in a given interval with pre-processing time $\theta(n+k)$ for building auxiliary storage.

```
Input -> n integers -> A[0, 1, 2, ... n] where range of A[j] is
{1,2, ... k}
Auxiliary Storage -> C[0, 1, 2, ... k]
```

```
k = largestElementOf (array)
A[n] -> input
C[k] -> create auxiliary storage
for i := 1 to k do
    C[i] = 0
for j := 1 to n do
    C[A[j]] = C[A[j]] + 1
for p := 2 to k do
    C[i] += C[i-1]
int a, b -> given interval such that b >= a
interval = C[b] - C[a]
```

Problem 7.1 (d)
I tried to implemented in words.cpp (make words), but could not be successful. I have attached "words.cpp" that shows my attempt.

Problem 7.1 (e)
Given any input sequence of length n, the worst-case for a BucketSort sorting algorithm is when all of n elements are in the same bucket. From the implementation of BucketSort in **Problem 7.1 (b)**, we can see that each individual buckets are sorted using another sorting algorithm (insertionSort in the implemented algorithm). Therefore, the worst-case time complexity for BucketSort is the worst-case time complexity for the insertionSort provided that all the elements fall under one single bucket.

Therefore, worst case of bucketSort $O(n^2)$.

Example of worst case: array = {0.19, 0.18, 0.17, 0.16, 0.15, 0.14, 0.13, 0.12, 0.11, 0.10}

Problem 7.2 (a)
I tired to implement RadixSort, but could not be successful. I have still attached two file “radixSort.cpp” and “oldRadixSort.cpp” in order to show my attempts. It would be helpful if you could help me/give feedback about my implementation.

Problem 7.3 (b)
I think the time complexity would be $O(n^2)$ by intuition.