ADS Homework 8

Name - Tewodros Adane
Email - tadane@jacobs-university.de

Problem 8.1 – Sorting in Linear Time

a) Counting Sort is implemented in the file "*CountingSort.cpp*" , and the usage is in "*main.cpp*".

b) Bucket Sort is implemented in the file "*BucketSort.cpp*" , and the usage is in "*main.cpp*".

c) Pseudocode of an algorithm to find the number of elements between **a** and **b** in O(1) time, given **n** numbers in range **0** to **k**:

```
COUNT-ELEMENTS(Array, a, b)
     // pre-processing part
     Let count be a new stack                  // O(1)
     FOR i = 0 TO Array.length – 1             // n-1 times
          IF Array[i] > a and Array[i] < b     // O(1)*(n-1)
               push Array[i] into count        // O(1)*(n-1)

     // counting part
     return count.size                         // O(1)
```

- In the pre-processing part, the algorithm will only go through the array once and the time complexity will be Θ(n).

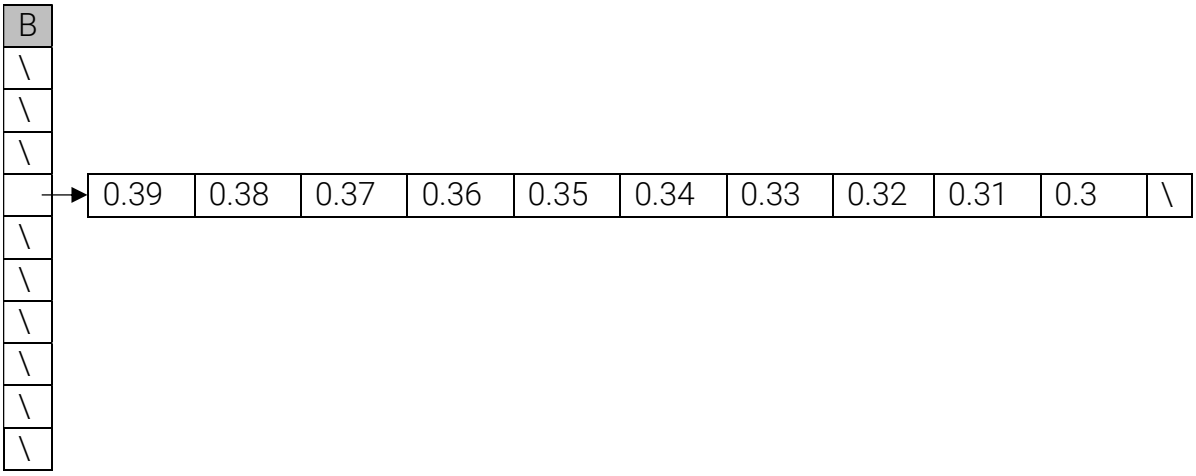
- In the counting part the function will return the size of the stack which is a constant time operation therefore it will only take O(1) time.

d) The algorithm to sort Strings in Θ(n) is implemented in the file "*StringSort.cpp*" , and the usage is in "*main.cpp*".

e) The worst-case time complexity for Bucket Sort will happen when all **n** elements fall in one bucket, and which will have the same time complexity as sorting with insertion sort.

Example:

Sorting the following numbers {0.39, 0.38, 0.37, 0.36, 0.35, 0.34, 0.33, 0.32, 0.31, 0.3}



The numbers will distribute to the buckets like in the figure above. Therefore, when the algorithm iterates through each bucket and sorts them using insertion sort the worst-case time complexity will be $O(n^2)$.

Problem 8.2 – Radix Sort

a) Hollerith's Radix Sort is implemented in the file "*HollerithRadixSort.cpp*" , and the usage is in "*main.cpp*".

b) **Asymptotic time complexity:** Hollerith's Radix sort, also known as Most Significant Digit Radix sort, uses bucket sort's classification method to group the elements based on the left most digit (most significant digit) and recursively sort the buckets. Each of the next recursions will be classified by the next most significant digit than the recursion before it until it reaches the least significant digit. Therefore, the depth of the recursion will be the number of digits (d) of the largest element. Since, Bucket sort classification takes $\Theta(n)$ time and the recursion depth is d, the Asymptotic time complexity of Hollerith's Radix sort **$\Theta(d*n)$**.

Asymptotic space complexity: Hollerith's Radix sort will require additional space for the creation of buckets. The algorithm will n buckets to be created, and since the recursion depth is d. The Asymptotic Space complexity would be $\Theta(d*n)$.