Huzaifa Hashim
30001928

CH08-320201 Algorithms and Data Structures
Professor Kinga Lipskoch
Homework 6

Problem 6.1 Linear Time Sorting

a. **(3 points) Implement the algorithm for Counting Sort and then use it to sort the sequence < 9, 1, 6, 7, 6, 2, 1 >.**
b. **(3 points) Implement the algorithm for Bucket Sort and then use it to sort the sequence < 0.9, 0.1, 0.6, 0.7, 0.6, 0.3, 0.1 >**

check files marked 6.1a.cpp and 6.1b.cpp respectively

c. **(4 points) Given n integers in the range 0 to k, design and write down an algorithm (only pseudocode) with pre-processing time Θ(n+k) for building auxiliary storage, which counts in O(1) how many of the integers fall into the interval [a, b].**
Initial Observations = auxiliary storage required in Counting Sort that uses a third array during sorting has the complexity *O(n + k).*

Preprocessing would follow as would happen in Counting Sort, such that *A[i]* contains the numbers of elements that are less than or equal to *i* in the array.

Hence, the following pseudocode is basically an implementation of counting sort that processes its input by counting the number of objects with distinct key values, more or less similar to hashing. This preprocessed input takes up *O(n + k)* auxiliary storage, where *n* is the number of elements and *k* is the range of the input array.

To count the number of integers that fall into an arbitrary range of *[a … b]*, the computation of the following takes up *O(1)* time and yields the desired output: ***A[b] – A[a – 1].***

```
function countingSort(array, min, max):
    array of (max - min + 1) elements

    initialize count with 0

    for each number in array do
count[number - min] = count[number - min] + 1

    z = 0
for i from min to max do
 while ( count[i - min] > 0 ) do
    array[z] = i
    z++

    count[i - min] := count[i - min] - 1

    count[b] – count[a – 1]
```

```
    rangecount;

endfunc
```

d. (6 points) Given a sequence of n English words of length k, implement an algorithm that sorts them alphabetically in *Θ(n)*. Let k and n be flexible, i.e., automatically determined when reading the input sequence. You can consider k to behave like a constant in comparison with n. Example sequence of words to sort:
< ”word”, ”category”, ”cat”, ”new”, ”news”, ”world”, ”bear”, ”at”, ”work”, ”time” >.

**Check 6.1d.cpp*

e. (2 points) Given any input sequence of length n, determine the worst-case time complexity for Bucket Sort. Give an example of a worst-case scenario and then prove corresponding the complexity.

The worst case time complexity for Bucket Sort would involve just 1 bucket used to sort the input array. If, for instance, Bucket Sort uses insertion sort as a sorting algorithm, analysis suggests that Insertion Sort operates in constant time for n = 1, however, for n > 1, it would recursively sort the n elements in the input by T(n) = T(n – 1) + n, which translates into a time complexity of $O(n^2)$.
If Bucket Sort decides to place all elements in one bucket, then insertion sort would perform in its non-ideal state. Thus, the worst time complexity for Bucket Sort can be deduced as $\Theta(n^2)$.

However, if a stable algorithm that performs better in its worst case, say O(n * log(n)) replaces insertion sort as the sorting algorithm within bucket sort, the worst case time complexity can be improved for Bucket Sort.

The recurrence for insertion sort can be used to mathematically prove the worst case time complexity for Bucket Sort that uses Insertion Sort:

$$\sum_{i=0}^{n-1} i$$

$$= \frac{2n(n-1)}{2}$$

$$= (n\,(\,n-1))$$

$$T = O(n^2)$$

Example of worst case input = 0.65, 0.61, 0.63, 0.69, 0.67

f. (4 points) Given n 2D points that are uniformly randomly distributed within the unit circle, design and write down an algorithm (only pseudocode) that sorts the points by increasing

Euclidean distance to the circle's origin. Write also a pseudocode function for the computation of the Euclidean distance between two 2D points.

EuclideanDist(x_1, x_2, y_1, y_2)
double x, y;
$x = x_1 - x_2$
$y = y_1 - y_2$

lib <cmath>

distance = pow(x, 2) + pow(y, 2)
distance = sqrt(distance)
return distance

//end func

//To sort with respect to increasing Euclidean Distance with origin
//Store coordinates in std::pair
//one would always be the origin (0, 0) since question asks for distance with respect to center

input x;
input y;

call EuclideanDist with pair;

pushback all distances in a vector and call STL sort using iterator for vector

endfunc

Problem 6.2 Radix Sort

Consider Hollerith's original version of the Radix Sort, i.e., a Radix Sort that starts with the most significant bit and propagates iteratively to the least significant bit (instead of vice versa).

a. **(8 points) Implement Hollerith's original version of the Radix Sort.**
   Check 6.2a.c

b. **(4 points) Determine and prove the asymptotic time complexity and the asymptotic storage space required for your implementation.** [1]

   For Time Complexity,
   $N = Number\ of\ items\ to\ sort$
   $R = the\ radix$

[1] http://vlm1.uta.edu/~athitsos/courses/cse2320_summer2014/lectures/13_radix_sort.pdf

$w = number\ of\ digits\ in\ radix\ R\ representation$
*for inputs we need R^{w-1} recursive calls to sort each digit, which takes O(R) time
*we also need $O(N)$ time to sort through the array

Thus the time complexity is at least
$\boldsymbol{T(n) = O(N + R) * w}$

For Storage space,
*O(N) space for input arrays
*O(R) space for counter and indices
Thus Space Complexity is,
$\boldsymbol{O(N + R)}$

c. (2 points) Write down the pseudocode for an algorithm which sorts n integers in the range 0 to $n^3 - 1$ in *O(n)* time.

To sort in *O(n)* time, first convert integer base and sort using radix sort. Since range is till n^3, numbers will have at most $\log_n(n^3) = 3$ digits hence the function will be called thrice.
For each iteration of this pass, there are n possible values to be taken on and we can use Count Sort to sort each digit in ***O(n)*** time.

```
//Algorithm to sort n integers in range [0 ... n^3 – 1]
countSort(int arr[], int x, int exp)
count[n]
for i < n
count[i] = 0

// Store count of occurrences in count[]
for i < n
count [(arr[i] / exp) % n ] ++

// Change count[i] so that count[i] now contains actual
for i < n
count[i] += count[i - 1]


// Build the output array
for (i = n - 1; i >= 0; i--)
output[count[ (arr[i]/exp)%n] - 1] = arr[i]
count[(arr[i] / exp) % n]--

arr[i] = output[i]

//call countsort thrice function with exp = 1, n, n*n
//print array
```

Huzaifa Hashim
30001928

//main function that declares array static or dynamic
endfunc

****I seem to have misread the question and also edited my bucket sort code for 6.1 to adapt to this question. Find it as 6.2c.cpp**