# California State University, Fresno

# DEPARTMENT OF COMPUTER SCIENCE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **Algorithms & Data Structures** | | | Semester: | **Fall 2020** |
|  | | | | | |
| Points |  | Document author: | **Yet Chun Fong** | | |
|  | Author’s email: | [**Fongyc7177@mail.fresnostate.edu**](mailto:Fongyc7177@mail.fresnostate.edu) | | |
| Laboratory number: | **5** | | |
|  | | | | | |

**1. Statement of Objectives**

The main objective of this lab experiment is to get familiar with the concepts two linear sorting algorithm: Radix Sort and Counting Sort. Then, the experimental time complexities for these two linear sorting algorithms during this lab session are used to compare with the theoretical time complexities.

Counting Sort is an algorithm that sort numbers according to the keys that are small integers. Then, an array of length (max+1) will be initialized with the value of 0 for all elements. The purpose of this array is to store the count of the elements in the given array accordingly. After finding the index of each element from the count array, cumulative count will be performed. Then, the output array will be filled accordingly by the index of each elements. After placing each element at the correction position in the output array, the count of the element in the count array will be decrement by one. The theoretical time complexity for counting sort is O(n+k).

For Radix Sort, it uses a different concept to sort the list of numbers. It is a sorting algorithm that group the individual digits of the integers. Then, sort the elements according to their significancy, go through one by one. Radix Sort uses Counting Sort to perform the sorting because Counting Sort is a stable sorting algorithm that keeps the order of integers. The theoretical time complexity for radix sort is O(d\*(n+k)), where d is the number of digits of the maximum input number, n is the length of the input array, and k is the range of digits.

**2. Experimental Procedure**

Firstly, a getMax() function is initialized to return the largest element in the array. Both Counting Sort and Radix Sort need this function for important reasons. For Counting Sort, after retrieving the maximum element, a count array will be the size of (max+1) to count the elements from the original array. For Radix Sort, the purpose of getMax() function is to find the significant place of integer.

Counting Sort function is created that take 2 parameters. Then, two new arrays will be initialized, output array and count array. First, the max value will be retrieved, then the count array will be filled with the cumulative count of the array. Then, the next For loop is used to find the index of each element of the original array and place the elements in the output array. After each placement, the index of that element in count array will be deduct by one. Last For loop is for copy the sorted elements and place it into the original array, so that the array is sorted.

Moving to Radix Sort, it uses Counting Sort as a subroutine to sort the array because the Counting Sort is a stable sorting algorithm. The reason why a second Counting Sort function has to be created specifically for Radix Sort because this Counting Sort takes 3 parameters which are array, size, the basis of significant places of the elements in array as Radix Sort function sort the digits at each significant places. Then, a For loop will be used to calculate significant places of digits represented by k and insert the value of k into the parameter of array.

Last function is a print function to display the elements in array. In addition, Chrono timer has been used in this lab experiment to record the running time of both sorting algorithms.

**3. Analysis**

Text

Description automatically generated

The time taken for Counting Sort is slightly longer than Radix Sort. The reason why Radix Sort is faster in this case because the digit only has one significant place. Therefore, the For Loop in Radix Sort only loop once to get the array sorted. The theoretical running time of Counting Sort is O(n+k), if k = O(n), then for Radix Sort the theoretical running time is O(d\*(n+b)), d indicates the number cycle. In this case, the number of cycles is only one, so it is faster than Counting Sort. Based on the theoretical running time of Counting Sort, the Counting Sort is not efficient for large sized array. For Radix Sort, it can sort a large sized array much better than Counting Sort because it only requires the amount of ‘d’ passes through the list to get the array sorted. But it must uses a stable sorting algorithm as subroutine to sort the array accordingly.

**4. Encountered Problems**

There is only one major problem during this lab experiment, related to the Radix Sort. At the first place, the first Counting Sort is used but it is not sorting the array correctly, then the problem has been found. The first Counting Sort does not take k (Significant place) of digits while sorting the algorithm which might cause error when sorting values with more significant places up to hundreds. Therefore, the second Counting Sort is created specifically for Radix Sort that will take 3 parameters, the array, the size of array, and the significant places of digits inside the array.

Another minor problem is that the symbol for arrays like count[arr[i]] is supposed to be square bracket are sometimes mistakenly for other symbols like brackets, e.g.: count[arr(i)], which caused the Exception thrown at 0x000000 (ntdll.dll) to pop up. But after several sounds of checking, the problem is solved, symbols used are correct. The functions are running completely fine after all.

**5. Conclusions**

All in all, the performance of Counting Sort and Radix Sort is largely depending on the size of inputs. The downside of these both sorting algorithms is that they require extra memory storage for the auxiliary array during the sorting process, these two sorting algorithms are not in-place sorting method. Counting Sort uses the index of elements and cumulative count to sort the input array which needs two extra arrays to store the cumulative counts and outputs. For Radix Sort, it sort the least significant digit first and moving to the next significant digit until the array is sorted.

**6. References**

Coding Guidelines from Lab TA

Lecture slide L13- Counting and Radix Sort