# California State University, Fresno

# DEPARTMENT OF COMPUTER SCIENCE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **Algorithms & Data Structures** | | | Semester: | **Fall 2021** |
|  | | | | | |
| Points |  | Document author: | **Ojas Deshmukh** | | |
|  | Author’s email: | **ojas@mail.fresnostate.edu email** | | |
| Laboratory number: | **Lab 5** | | |
|  | | | | | |

**1. Statement of Objectives**

For this lab we will look at two different linear order sorting algorithms. Count sort and Radix sort.

We will check their reason for linear order time complexity.

**2. Experimental Procedure**

Let us check how count sort works-

We will create an array that stores the count for each number in the given array. Count is nothing but the number of times the number repeats itself. We create an array of range in which we can store number of occurrences for each number. Then we add the previous count of the number to the count of next. Then the values in the count array become the locations of element where we will be placing them.

For example, we have an array 1, 3, 2, 4, 2, 7 🡪 we create count arr = (0=0,1=1, 2=2, 3=1, 4=1, …, 7=1,…) 🡪 we make the count add in the next count, count arr = (0=0, 1=1, 2=3, 3=4, 4=5, 5=5, 6=5, 7=6, 8=6, 9=6) now we have locations, so we rearrange accordingly, 1 at location 1, 1 at location 2, 2 at location 3, and so on---

We get count or locations 🡪 1 2 3 4 5 6 7 8 9

Numbers from array🡪 1 1 2 3 4 7 🡪 this is our sorted array. …. we discard other number copying the same location value.

Radix sort – We have implemented radix sort using the count sort, so let us see the working.

First, we check for the largest number or max in the list and then we check the number of digits in that number. Let us say that there are 4 digits in the largest element, then we will have to run loops for four times. Now, for every digit starting from unit’s place or least significant place we sort number based on count sort then we go to next digit place or ten’s place and so on. The time complexity can be seen similar to count sort because of use of count sort in the algorithm.

**3. Analysis**

From this, we have witnessed that for problems where merge sort and selection sort were taking long time to execute code from last lab, count sort and radix sort are extremely powerful in the sense of low run time needed to sort the array with different arrangements. There was not significant difference when we consider the difference in array types. Time complexity order for count sort is O(N+K) here n is the number of elements in the array and k is the range of the elements and for radix sort is O(D(N+B)) where D is the number of digits in the given list, N is the number of elements in the list, and B is the base or bucket size used, which is normally base 10 for decimal representation.

For arrays with small inputs it is extremely fast and for array with 200000 where other sorting algorithms were taking too long or not even able to complete the task sometimes, these two algorithms do it in linear time.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**4. Encountered Problems**

The count sort algorithm had some issues in the beginning, since my logic applied in theory and that I coded did not match, but once I found the bug, it all worked.

**5. Conclusions**

Count sort and Radix sort are algorithms of linear order and extremely convenient to use with simple logic and no comparison conditions. We can always use these for cases of big data set that needs sorting in linear order of run time complexity.