# California State University, Fresno

# DEPARTMENT OF COMPUTER SCIENCE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **Algorithms & Data Structures** | | | Semester: | **Fall 2021** |
|  | | | | | |
| Points |  | Document author: | **Marlon Branham** | | |
| **`** | Author’s email: | **zekebranham@mail.fresnostate.edu** | | |
| Laboratory number: | **Laboratory 4** | | |
|  | | | | | |

**1. Statement of Objectives**

In this lab, the goal was to gain hands-on experience with and understanding of Counting Sort and Radix Sort, and then comparing the two theoretical time complexities with their actual runtimes. The significance of this lab is experimenting with the design of the two sorting algorithms and ultimately increasing our toolbox as far as sorting goes, these were all accomplished. This report will discuss my experimental procedure, analysis, encountered problems, and conclusion with my findings.

**2. Experimental Procedure**

The procedure used in this lab was straightforward and concise. The procedures included implementing two sorting algorithms, one being Counting Sort, and the other being Radix Sort. We were to create an array of reasonable size to sort, and then calculate the runtime of the algorithms. Finally, once we have found the runtime of the algorithms, compare the runtimes to the theoretical time complexity of each, which should be linear.

I approached this by creating an unordered array of 500,000 randomly generated elements to use for my first algorithm, and then I copied the first array to a second array that I need for the second algorithm. This is when I started sorting the first array using Counting Sort and found the runtime using Chrono. After this, I sorted the second unsorted array using Radix Sort and found the runtime using Chrono. I created my Radix Sort algorithm by using Counting Sort to sort each decimal value of the array.

**3. Analysis**

The goal of the experiment was to implement the two sorting algorithms, find their runtimes, and then compare them with their theoretical values. Both Counting Sort and Radix Sort are linear sorting algorithms that are non-comparative. Counting Sort is T(n)=θ(n) while Radix Sort is T(n)=(d(n+k)). The calculated average runtime of the two algorithms on my systema are: Counting Sort: 0.0118 seconds, Radix Sort: 0.0324 seconds. This shows that while both algorithms are linear, Counting Sort is still around 3 times faster than of Radix Sort, mainly due to the extra processing it takes to use Radix Sort.

A screen shot of a computer

Description automatically generated

**4. Encountered Problems**

The lab was generally straightforward and did not spark any issues in the understanding of the assignment. My only issue was regarding the creation of the arrays for Counting Sort. Since the counting array and frequency array are based on the size of the maximum value of the primary array, the created arrays need to be formed using dynamic memory allocation, which presents some issues in C++. So, to find my way around this, I allocated the arrays using “new” and “delete” operators and had no issues after.

A black background with white text

Description automatically generatedA black background with white text

Description automatically generated

A computer screen with white text

Description automatically generated

**5. Conclusions**

In this lab, I can conclude that both algorithms, Counting Sort, and Radix Sort, are linear non-comparison sorting methods that are much faster and more efficient than other comparison methods. With this being so, the Counting Sort method was significantly faster than Radix Sort as the amount of the elements grew. Overall, successful implementation of the two algorithms, we can see that the theoretical time complexity of both closely resembles the actual runtime on my system.

**6. References**

<https://www.guru99.com/cpp-dynamic-array.html>

<https://www.programiz.com/dsa/counting-sort#google_vignette>