

CALIFORNIA STATE UNIVERSITY, FRESNO
DEPARTMENT OF COMPUTER SCIENCE

Class:	Algorithms & Data Structures			Semester:	Fall 2023
Points		Document author:	Pratham Aggarwal		
		Author's email:	Pratham_aggarwal@mail.fresnostate.edu		
		Laboratory number:	Lab 5		

1. Statement of Objectives

This lab's goal is to compare two sorting algorithms—Counting Sort and Radix Sort and evaluate how well they perform. The implementation and analysis of these sorting algorithms are covered in this report, along with measurements of their average execution times over a number of iterations and comparisons between the experimental findings and the theoretical time complexities.

2. Experimental Procedure

Implementation of the algorithm: The code implements the Counting Sort and Radix Sort sorting algorithms.

Data: For both sorting algorithms, the following initial array of numbers is used: 170, 45, 75, 90, 802, 24, 2, 66.

The main goal of this experiment is to evaluate and compare the execution times of the Counting Sort and Radix Sort algorithms. I used the clock function from the C++ standard library to accomplish this. I can measure the average execution time of each sorting algorithm by recording the system time before and after each algorithm's execution using the clock function for several iterations.

Average Execution Time: I estimated the average execution time after doing 1000 iterations of each sorting algorithm. This average execution time offers a more accurate estimation of how long the algorithms actually need to sort the array.

Results display: For both Counting Sort and Radix Sort, I displayed the sorted results from the first iteration in order to better understand how the algorithms work. Then, later on I display the average execution time.

3. Analysis

Counting Sort and Radix Sort (First Iteration) Results: {2, 24, 45, 66, 75, 90, 170, 802} is the results of the first iteration of Counting Sort and Radix Sort. This indicates that the array was successfully sorted using both algorithms in ascending order.

Average Execution Time for Counting Sort: Over 1000 iterations, the average execution time for counting sort is estimated to be around 0.022 milliseconds.

Average Execution Time for Radix Sort: Calculated over 1000 iterations, the average Radix Sort execution time is roughly 0.002 milliseconds. This indicates, that Radix Sort performs even better than Counting Sort.

Theoretical Time Complexity: The time complexity of counting sort is $O(n + r)$, where n is the number of elements and r is the range of input value. Radix Sort has an $O(d(n + k))$ time complexity, where d is the maximum number of digits, n is the number of elements, and k is the range of input values.

Screenshot attached in the end of the report.

4. Encountered Problems

Pseudo Code: I had to look for the sorting algorithms' pseudo code.

Predefined Algorithms: Predefined functions like `findMax` and `clock`, I took help from instructors and internet resources to understand how they were used.

Timing issues: At first, the sorting algorithms' timing displayed 0 seconds. I had to run the algorithms 1000 iteration to solve this problem, and consider the average execution time of the 1000 iterations.

5. Conclusions

Counting Sort and Radix Sort are both efficient sorting algorithms with linear time complexity.

Both algorithms correctly sorted the input array, according to the first iteration results.

Over 1000 iterations, Counting Sort showed an average execution time of about 0.022 milliseconds.

With an average execution time of roughly 0.002 milliseconds over 1000 iterations, Radix Sort displayed a better result.

Theoretical Time Complexity:

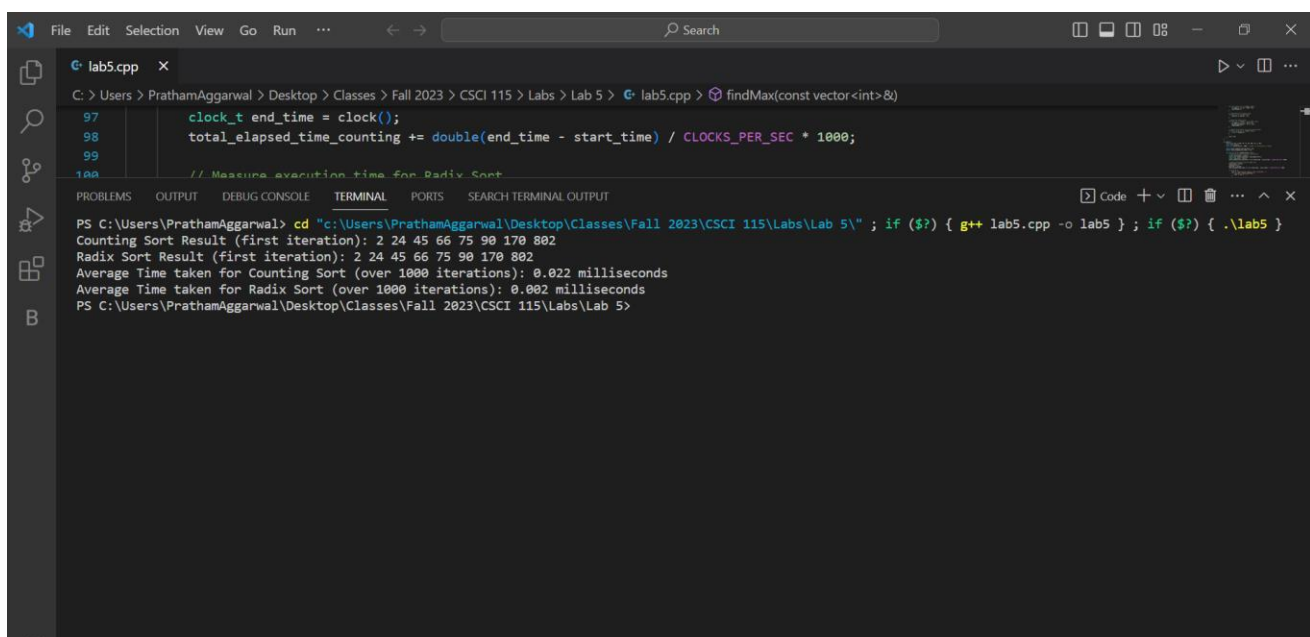
Counting Sort: $O(n + r)$

Radix Sort: $O(d(n + k))$

Comparing the experimental and theoretical time complexities, both algorithms performed close to their theoretical expectations. Radix Sort performed particularly well, demonstrating how effective it is in sorting integers without comparisons.

6. References

Lecture notes, mainly for theoretical time complexity.



The screenshot shows a Visual Studio Code editor with a C++ file named `lab5.cpp` open. The file contains code for finding the maximum element in a vector and measuring the execution time of Counting Sort and Radix Sort. The terminal output shows the results of running the program, including the sorted array and the average execution times for both sorting algorithms over 1000 iterations.

```
lab5.cpp
C: > Users > PrathamAggarwal > Desktop > Classes > Fall 2023 > CSCI 115 > Labs > Lab 5 > lab5.cpp > findMax(const vector<int>>8)
97      clock_t end_time = clock();
98      total_elapsed_time_counting += double(end_time - start_time) / CLOCKS_PER_SEC * 1000;
99
100      // Measure execution time for Radix Sort

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH TERMINAL OUTPUT
PS C:\Users\PrathamAggarwal> cd "C:\Users\PrathamAggarwal\Desktop\Classes\Fall 2023\CSCI 115\Labs\Lab 5\"; if ($?) { g++ lab5.cpp -o lab5 }; if ($?) { .\lab5 }
Counting Sort Result (first iteration): 2 24 45 66 75 90 170 802
Radix Sort Result (first iteration): 2 24 45 66 75 90 170 802
Average Time taken for Counting Sort (over 1000 iterations): 0.022 milliseconds
Average Time taken for Radix Sort (over 1000 iterations): 0.002 milliseconds
PS C:\Users\PrathamAggarwal\Desktop\Classes\Fall 2023\CSCI 115\Labs\Lab 5>
```