CS/CPE590: Algorithms
Spring 2023

Assignment2

Name: Chao Zheng
CWID: 20017348

Lab Report

03/15/2023

Abstract:

The purpose of this experiment is to observe the runtime difference between the radix sort implemented by insertion sort and the radix sort implemented by counting sort. From the theoretical comparison average case one is O(n^2) and another is O(n), and the result of the experiment would show corresponding behavior between those radix sorts.

Observation:

	m =25		m = 35		m = 45	
n	Random Vector	Sorted Vector	Random Vector	Sorted Vector	Random Vector	Sorted Vector
5000	2.0	763.0	2.8	909.2	2.2	1277.0
6000	2.0	860.0	2.8	1726.6	3.0	1817.6
7000	4.0	1480.0	2.8	2106.0	3.4	2591.6
8000	2.4	1551.4	4.0	2454.2	5.0	4550.2

Table 1: Radix sort implement by insertion sort, with different size of arrays n and the length of string m to runtime of random vector generated and sorted vectors generated (Average summary).

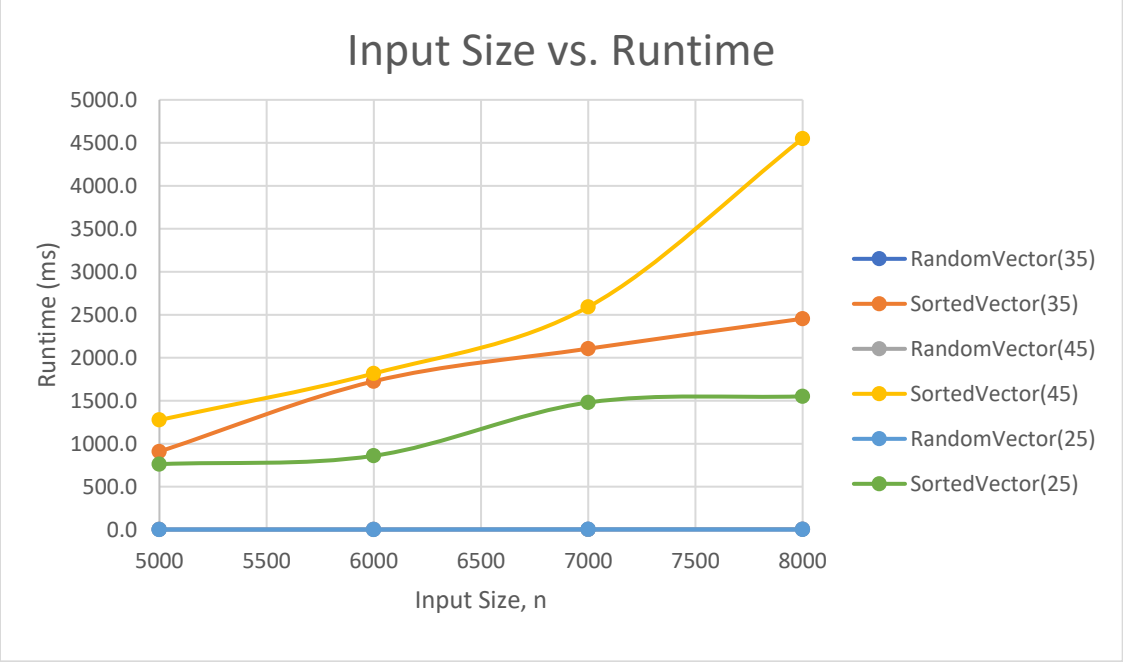

Figure 1: The graph shows the performance of radix sorts on 5000 - 8000size arrays and 25 - 45 length strings using insertion sort based, Sorted and random vectors have significantly different sorting times (Average summary).

	m =25		m = 35		m = 45	
n	Random Vector	Sorted Vector	Random Vector	Sorted Vector	Random Vector	Sorted Vector
5000	1.8	3.6	1.2	1.5	1.4	2.0
6000	2.2	3.2	3.0	2.4	1.6	2.6
7000	4.2	4.4	2.0	2.4	3.4	2.4
8000	3.0	4.0	3.0	3.8	2.0	4.0

Table 2: Radix sort implement by counting sort, with different sizes of arrays n and the length of string m to runtime of random vector generated and sorted vectors generated.

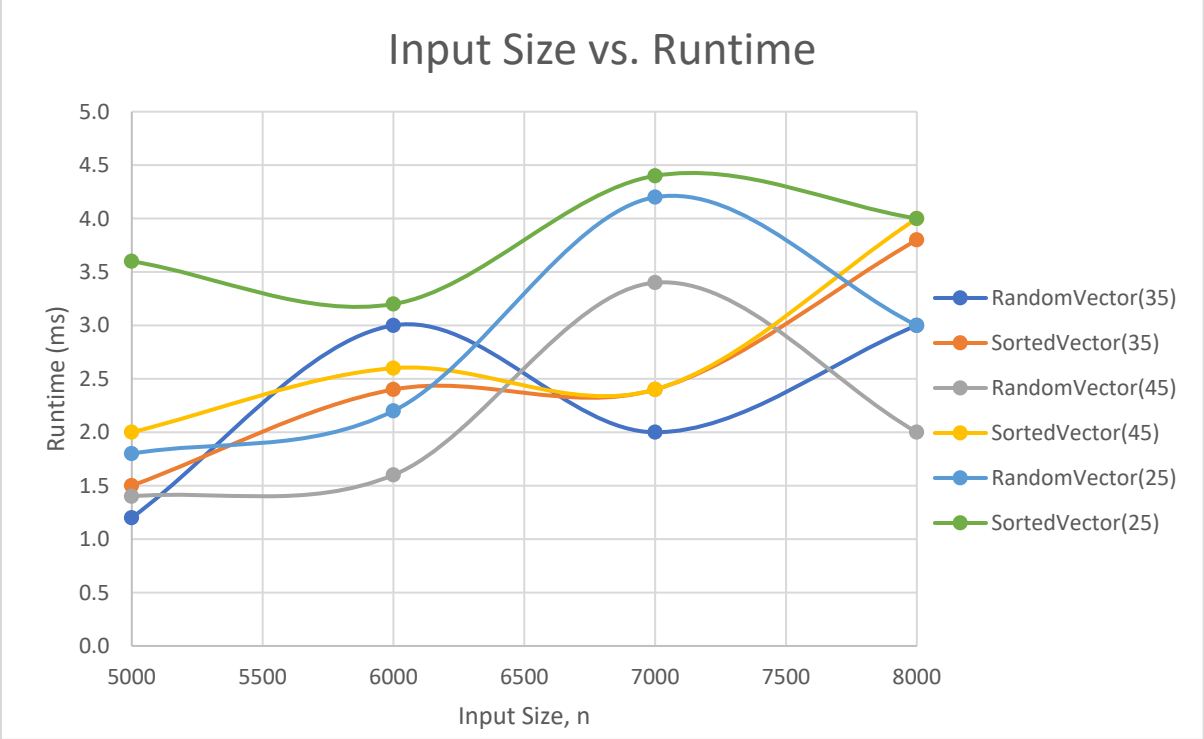

Figure 2: The graph shows the performance of radix sort on 5000 – 8000 size arrays and 25 - 45 length strings using counting sort based, and sorting times are similarly.

Discussion:

The experiment compared the runtime difference between the radix sort implemented by insertion sort and the radix sort implemented by counting sort. From the tables, it is evident that counting sort implementation is much faster than the insertion sort implementation. The tables show that the average runtime for the counting sort implementation is O(n), while the insertion sort implementation has an average runtime of O(n^2).

The runtime of both implementations is dependent on the size of the array (n) and the length of the string (m). The runtime increases with an increase in the size of the array and the length of the string. This increase is expected because the number of comparisons and swaps required to sort the array increases with an increase in the array size and string length. From the

tables, it is evident that the runtime of the insertion sort implementation increases significantly with an increase in the size of the array and string length. On the other hand, the counting sort implementation has a slower runtime increase with an increase in the array size and string length.

The theoretical analysis suggests that the counting sort implementation is faster than the insertion sort implementation because counting sort has a complexity of O(n) in the average case, while insertion sort has a complexity of O(n^2). Counting sort is more efficient because it is a non-comparison-based sorting algorithm, and its runtime is dependent on the range of values in the input array. On the other hand, insertion sort is a comparison-based sorting algorithm that requires swapping elements in the array, which makes it slower.

The experiment's results are consistent with the theoretical analysis because the counting sort implementation has a significantly faster runtime than the insertion sort implementation. The tables also show that the runtime difference between the two implementations increases with an increase in the size of the array and string length. The insertion sort implementation becomes increasingly inefficient as the array size and string length increase, while the counting sort implementation remains efficient.

Conclusion:

In conclusion, the experiment has demonstrated that the radix sort implemented by counting sort is significantly faster than the radix sort implemented by insertion sort. The runtime of both implementations is dependent on the size of the array and the length of the string, with an increase in the size of the array and string length resulting in a slower runtime. The theoretical analysis and experimental results are consistent, showing that the counting sort implementation is more efficient because it is a non-comparison-based sorting algorithm.