Term Project

Tuyet Mai Pham - 991545166

Sheridan College

PROG20799 – Data Structures & Algorithm Development – C

Dr. Ben Kam

April 12, 2023

Table of Contents

I. Introduction

Sorting algorithms play a vital role in computer science and programming. They allow putting a group of items in a particular order using a comparison operator. These algorithms work well for various data types such as numbers, strings, and more. In computer programs, sorting algorithms are an essential part as they make data operations such as searching and analysis more manageable by rearranging the data.

When it comes to sorting challenges, selecting the appropriate algorithm is of utmost importance and is contingent on the specific requirements and constraints of the task at hand. A plethora of sorting algorithms exists, each with its unique approach and benefits. The aim of this project is to compare Radix Sort and Bubble Sort algorithms to evaluate their efficiency and performance. This comparison intends to provide insights into the advantages and disadvantages of different algorithms.

II. Radix Sort Algorithm

Radix sort is a sorting algorithm that works by grouping the digit from right to left at the same place of all the elements given. It starts with the most right which is the least significant digit till the most significant digit. The total number of runs is determined by the number of digits in the biggest number in the given list.

- Firstly, find the biggest element in the list then figure out the number of runs. The number of runs is the number of digits of the biggest element.
- Then go through each element in the list, and group them based on each digit on each run from the least significant to the most significant digit.
- After grouping at the most significant digit, the list is now in order.

III. Bubble Sort Algorithm

Bubble sort is a sorting algorithm that works by checking if two adjacent elements do not match a given sort order and swapping them. Multiple iterations of this procedure are performed until the full list is sorted.

- In the first Run:
    - The process of Bubble Sort begins at the first index of a list and compares the first and second elements.
    - If the first element is greater than the second, they are swapped.
    - The algorithm then compares the second and third elements and swapped if they are not in order.
    - This process continues until the last element of the list is reached.
    - At the end of the run, the largest element is placed last on the list.
- The process above is repeated until the list is in proper order. After each run, the largest among unsorted elements is placed at the end.

IV. Radix Sort Big O Notation

Code	Complexity
void radixSort(int* array, int size) {	
int runs = getNumberOfRuns(array, size);	
Queue queues[10]; //10 digits from 0-9	
for (int I = 0; I < 10; i++) {	
Queue newQueue;	

`    initQueue(&newQueue);`	
`    queues[i] = newQueue;`	
`  }`	
`  int modulous = 10;`	
`  int divisor = 1;`	
`  for (int I = 0; I < runs; i++) {`	k
`    for (int j = 0; j < size; j++) {`	n
`      int digit = (array[j]/divisor) % 10;`	
`      enqueue(&queues[digit], array[j]);`	
`    }`	
`    int index = 0;`	
`    for (int j = 0; j < 10; j++) {`	10
`      while(queues[j].nodeCount > 0) {`	
`        array[index] = dequeue(&queues[j]);`	
`        index++;`	
`      }`	
`    }`	
`    modulous *= 10;`	
`    divisor *= 10;`	
`  }`	
`}`	

O(k(n+10))

V. Bubble Sort Big O Notation

Code	Complexity
`void bubbleSort(int array[], int size) {`	
`    Queue queue;`	
`    initQueue(&queue);`	
`    for (int i = 0; i < size; i++) {`	n
`        for (int j = 0; j < size - 1 - i; j++) {`	n
`            if (array[j] > array[j + 1]) {`	
`                enqueue(&queue, array[j]);`	
`                array[j] = array[j + 1];`	
`                array[j + 1] = dequeue(&queue);`	
`            }`	
`        }`	
`    }`	
`}`	

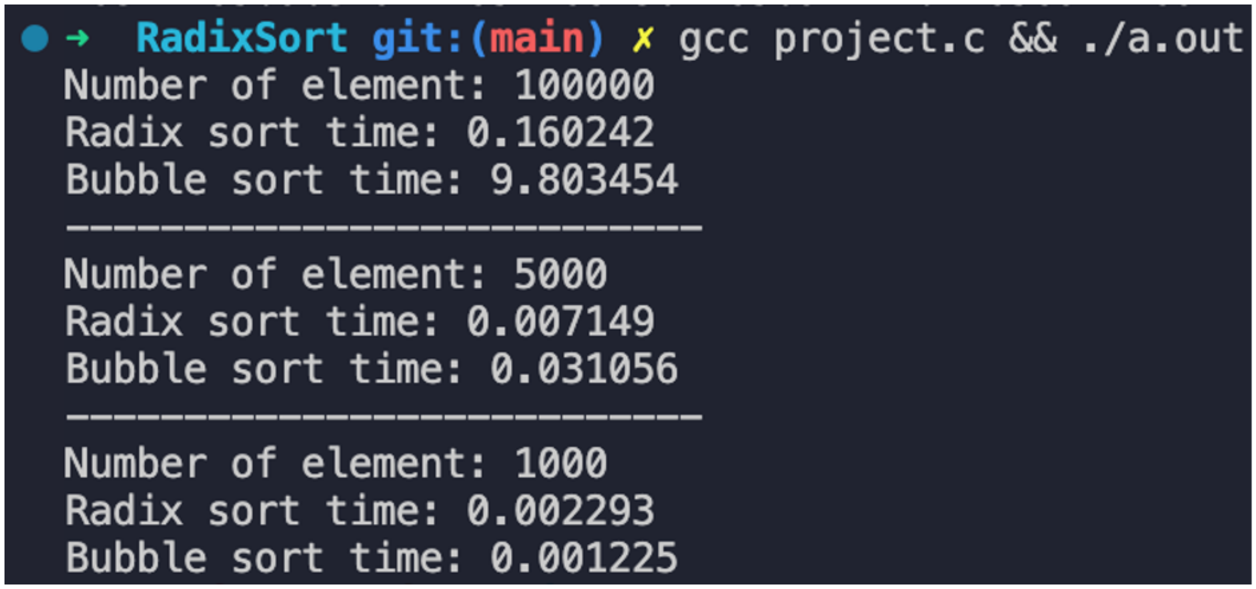
$O(n^2)$

VI. Setting Up Controlled Experiments

In order to accurately measure the time it takes for the two sorting algorithms to complete, the program utilizes the clock() function provided by the C standard library. To ensure reliable results, the program is run three times and records the varying results. Additionally, the program tests the algorithms' performance with small, medium, and large datasets.

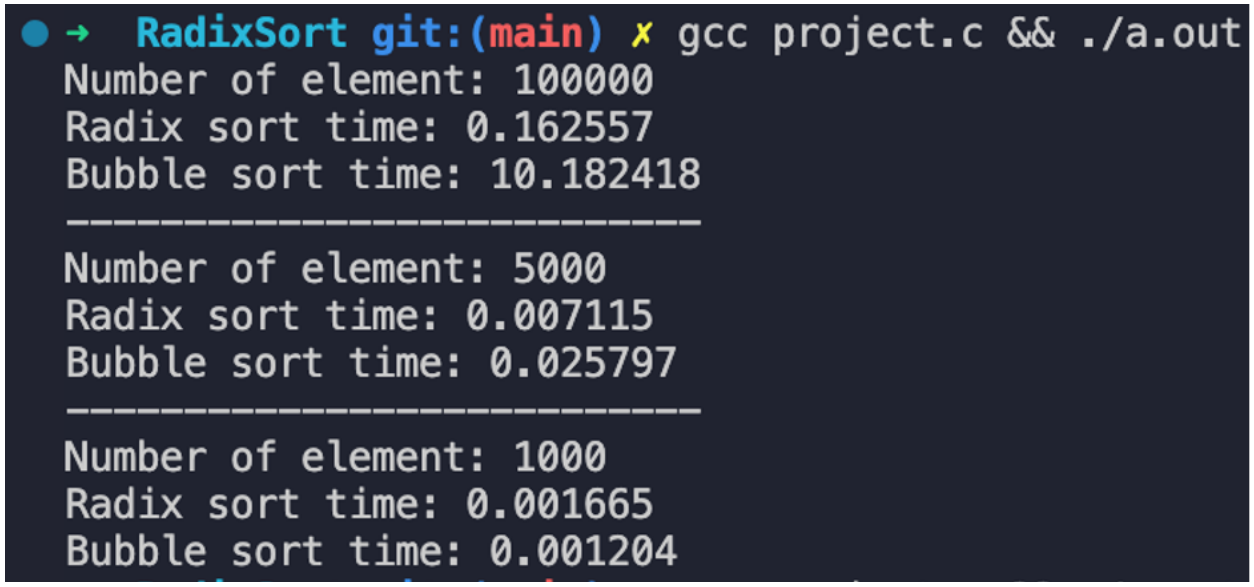
VII. Experiment results

First Run:



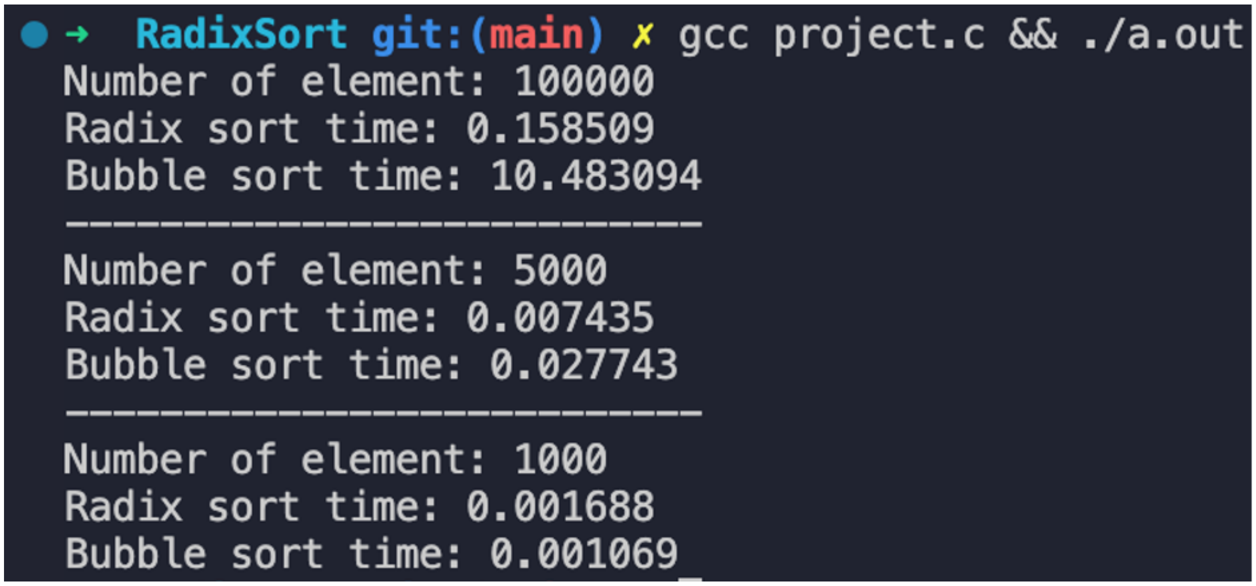
```
● ➜  RadixSort git:(main) ✗ gcc project.c && ./a.out
  Number of element: 100000
  Radix sort time: 0.160242
  Bubble sort time: 9.803454
  ---------------------------
  Number of element: 5000
  Radix sort time: 0.007149
  Bubble sort time: 0.031056
  ---------------------------
  Number of element: 1000
  Radix sort time: 0.002293
  Bubble sort time: 0.001225
```

Second Run:



```
● ➜  RadixSort git:(main) ✗ gcc project.c && ./a.out
  Number of element: 100000
  Radix sort time: 0.162557
  Bubble sort time: 10.182418
  ---------------------------
  Number of element: 5000
  Radix sort time: 0.007115
  Bubble sort time: 0.025797
  ---------------------------
  Number of element: 1000
  Radix sort time: 0.001665
  Bubble sort time: 0.001204
```

Third Run:



```
● ➜  RadixSort git:(main) ✗ gcc project.c && ./a.out
  Number of element: 100000
  Radix sort time: 0.158509
  Bubble sort time: 10.483094
  --------------------------
  Number of element: 5000
  Radix sort time: 0.007435
  Bubble sort time: 0.027743
  --------------------------
  Number of element: 1000
  Radix sort time: 0.001688
  Bubble sort time: 0.001069
```

VIII. Data Analysis

Based on the results obtained from three separate program executions, it is evident that the difference in implementation times between the two sorting algorithms is significantly noticeable with a larger data set of 100,000 elements. In this scenario, the Radix sort algorithm executes much faster, taking only about 0.15 seconds, while the Bubble Sort algorithm requires more time (9 – 10 seconds). The latter takes a considerable amount of time to execute the program.

In the case of the medium data set of 5,000 elements, the difference in implementation time between the two algorithms still exists, but it is not as significant as with the larger data set. Bubble Sort takes longer to execute than Radix Sort.

For a smaller data set of 1,000 elements, the results show that the implementation of the Bubble Sort algorithm is now faster than that of Radix Sort. Bubble Sort executes in about 2/3 or 1/2 of the time taken by Radix Sort.

IX. Conclusion

Based on the Big-O notation, we can see that Radix Sort has better time complexity. The complexity time of Radix Sort is O(k(n+10)), while Bubble Sort has a time complexity of $O(n^2)$.

Based on the results of the program, it can be concluded that Radix Sort outperforms Bubble Sort in terms of efficiency for sorting large datasets. The execution time of Radix Sort is significantly faster than Bubble Sort, especially for bigger data sets. However, for smaller data sets, Bubble Sort can be a more practical option as it takes less time to execute than Radix Sort. Therefore, the choice of which algorithm to use depends on the size of the data set and the specific needs of the program.