Report

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Assignment No. 2**

Average time of execution of different sorting Algorithms

for a sorted and unsorted array of numbers.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**BY**

Muhammad Tanzeel Saleem

BSCS-III

04071913026

**Introduction:**

In this report, I will explain how I have performed in my assignment 2 and it is working. I have two arrays of numbers having a size of 100k. One array of numbers is sorted while the other array of numbers is not sorted. So, by using our CPP program code which includes different sorting algorithms I have to sort both the arrays of numbers. And I have to calculate the**time (in microseconds)** of execution of those different sorting algorithms.

**Explanation:**

I have five different sorting algorithms that I have defined in our CPP program. These sorting algorithms are:

* Wrong Selection Sort
* Right Selection Sort
* Bubble Sort Without Flag
* Bubble Sort Without Flag
* Insertion Sort

Here the right or wrong sort doesn’t mean that these algorithms are performing right or wrong sort respectively. It means that the particular algorithm which is taking a smaller number of steps to execute will be the right otherwise it will be wrong but both the algorithms will perform sorting in the correct order. Because an excellent algorithm does not depend upon the machine time. It depends upon the total number of steps in which the algorithm is executing.

I have defined these sorting algorithms in our CPP program. So, by using our code I have calculated the average time of execution of these different sorting algorithms by running them ten times and store it in a CSV file that was created by our CPP program. Based on the average time of execution I will know that which algorithm is performing good (means taking a smaller number of steps to execute) or bad (means taking a larger number of steps to execute).

**Different Sorting Algorithms:**

Here I will discuss the time complexity and working of different sorting algorithms.

**Wrong Selection Sort:**

**What wrong selection sort is?**

It is a type of sorting in which I assume the element on the first index is minimum. After then I compare it with all the elements of an array and if swap needs at any index then it swaps the elements immediately. So, it takes a larger number of steps and greater time to perform sorting. That’s why we called it a wrong selection sort.

**The time complexity of wrong selection sort is: O(N2)**

* **An unsorted array of numbers:**

I performed sorting using the wrong selection sort for an unsorted array of numbers. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

* **A sorted array of numbers:**

I performed sorting using the wrong selection sort for a sorted array of numbers. If we compare the execution of time for a sorted array with an unsorted array. Then we come to know that this algorithm has taken greater time to execute. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

**Conclusion:**

**Average time of execution for an unsorted array = 22534111 µs**

**Average time of execution for a sorted array = 8411318 µs**

**How this algorithm is?**

So, from the above-average, I came to know that the execution time of the wrong selection sort function for an unsorted array is greater than a sorted array. It is the **worst algorithm** for sorting because it took lots of execution time and lots of steps as it swaps elements immediately when there is a need for swapping. In this way, the number of steps increased to perform sorting. That’s why it is the worst algorithm.

**Correct Selection sort:**

**What correct selection sort is?**

In this type of sorting, I assume that the element on the first index of the array is the minimum number. Then I compare the first index with all the index of the array. This algorithm doesn’t perform swapping until it doesn’t find the most minimum number in an array via index.

**The time complexity of correct selection sort is: O(N2)**

* **An unsorted array of numbers:**

I performed sorting using correct selection sort for an unsorted array of numbers. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

* **A sorted array of numbers:**

I performed sorting using correct selection sort for a sorted array of numbers. If we compare the execution of time for a sorted array with an unsorted array. Then we came to know that this algorithm has taken almost steps to execute. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

**Conclusion:**

**Average time of execution for an unsorted array = 9716665** **µs**

**Average time of execution for a sorted array = 9664147 µs**

**How this algorithm is?**

So, from the above-average, I came to know that the execution time of the correct selection sort function for an unsorted array is almost the same as a sorted array. It is the **best algorithm** for selection sorting as compare to wrong selection sorting. Because it took almost the same execution time and a smaller number of steps than the wrong selection sort. That’s why this algorithm is best for selection sorting.

**Bubble Sort Without Flag:**

**What bubble sort without a flag is?**

This is another type of sorting in which elements are sorted like bubbles. Such as if there are small elements at the middle or end of an array then this algorithm performs sorting in such a way that sorted element will come at the top indexes of an array just like bubbles came up in soda water. Here bubble sorting without a flag means that the algorithm will perform sorting if the array is already sorted.

**The time complexity of bubble sort without a flag is: O(N2)**

* **An unsorted array of numbers:**

I performed sorting using bubble sort without a flag for an unsorted array of numbers. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

* **A sorted array of numbers:**

I performed sorting using bubble sort without a flag for a sorted array of numbers. If we compare the execution of time of a sorted array with an unsorted array. Then we came to know that this algorithm has taken large execution time for unsorted array and small execution of time for a sorted array. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

**Conclusion:**

**Average time of execution for an unsorted array = 26867574** **µs**

**Average time of execution for a sorted array = 9887899 µs**

**How this Algorithm is?**

So, from the above-average execution time, I came to know that the execution time of this algorithm for a sorted array is half of the execution time of an unsorted array. Its execution time is also greater than the execution time of selection sort. So, it is the **worst algorithm** for bubble sorting. This algorithm is performing an equal number of steps for the sorting of a sorted and unsorted array.

**Bubble Sort with Flag:**

**What bubble sort without a flag is?**

This is another type of sorting in which elements are sorted like bubbles. Such as if there are small elements at the middle or end of an array then this algorithm performs sorting in such a way that sorted element will come at the top indexes of an array just like bubbles came up in soda water. Here bubble sorting with a flag means that the algorithm will not perform sorting if the array is already sorted.

**The time complexity of bubble sort without a flag is: O(N)**

* **An unsorted array of numbers:**

I performed sorting using bubble sort with a flag for an unsorted array of numbers. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

* **A sorted array of numbers:**

I performed sorting using bubble sort with a flag for a sorted array of numbers. If we compare the execution of time of a sorted array with an unsorted array. Then we came to know that this algorithm has taken smaller execution times for the unsorted array as compared to the algorithm of bubble sort with flag. And 194 microsecond that is almost 0 sec of execution of time for a sorted array. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

**Conclusion:**

**Average time of execution for an unsorted array = 67757095** **µs**

**Average time of execution for a sorted array = 194** **µs**

**How this Algorithm is?**

So, from the above-average execution time, I came to know that the execution time of this algorithm for an unsorted array is larger. While the execution of time for a sorted array is almost 0 sec. Because this algorithm does not perform sorting if the array is already sorted. So, it is the **best algorithm** for bubble sorting.

**Insertion Sort:**

**What insertion sort is?**

This type of sorting algorithm works similarly to the way we sort playing cards in our hands. The array is virtually split into a sorted and unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

**The time complexity of bubble sort without a flag is: O(N)**

* **An unsorted array of numbers:**

I performed sorting using insertion sort for an unsorted array of numbers. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

* **A sorted array of numbers:**

I performed sorting using insertion sort for a sorted array of numbers. If we compare the execution of time of a sorted array with an unsorted array. Then we came to know that this algorithm has taken smaller execution time for the unsorted array as compared to any other sorting algorithms. And 319 microsecond that is almost 0 sec of execution of time for a sorted array. This sorting algorithm was executed for ten-times. The detail of execution time (in microseconds) of this algorithm is given in the chart.

**Conclusion:**

**Average time of execution for an unsorted array = 6054860** **µs**

**Average time of execution for a sorted array = 319** **µs**

**How this Algorithm is?**

So, from the above-average execution time I came to know that the execution time of this algorithm for the unsorted array is smaller than the other algorithms. While the execution of time for a sorted array is almost 0 sec. So, it is the most **efficient and best algorithm** for sorting. The array is virtually split into a sorted and unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

**Summary:**

Finally, if I conclude from the above detail of algorithms. **The most efficient and best algorithm is an insertion sort**. Because it takes a small number of steps to execute. It also takes small execution of time. And its good thing is that it takes almost no time (0 sec) to execute if the array is already sorted.

In this sorting, the array is virtually split into a sorted and unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_