Algorithms and Data Structures ITCS 6114

Project Team Members:

Manasi Prabhune

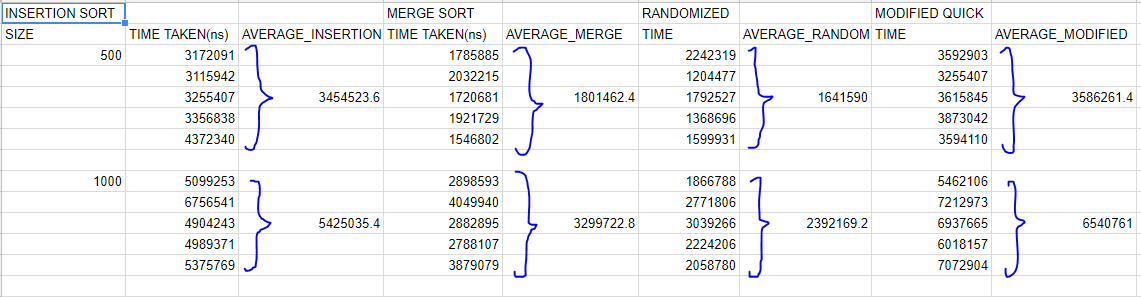
Nirja Sanjay Ayare

We have implemented 4 different sorting techniques to understand the time complexity of these algorithms with different inputs.

**Variation in Inputs:**

When we take Insertion Sort’s time noted for 500 integers(nanosecs), 1831270 ns. This is the first reading.An average value of the time noted is 1897346 ns. Recorded time may vary as there are other processes running at the same time such as comparison etc. There are other factors such as the computer used for running algorithm for variation in the time. The sample of variations can be seen in the below image. Same goes for all the time values. Hence, we have averaged the time taken by the algorithms for same input size of array in 4 executions.

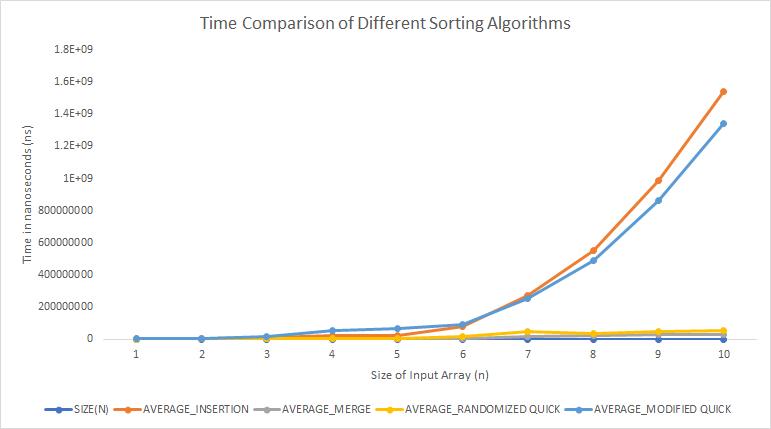
The input array size varies from 500, 1000 .. , 20000,50000.



**Time Complexities of different sorting techniques:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sorting Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| **Insertion Sort** | O(n) | O(n2) | O(n2) |
| **Merge Sort** | O(nlogn) | O(nlogn) | O(nlogn) |
| **Randomized Quicksort** | O(nlogn) | O(nlogn) | O(n2) |
| **Modified Quicksort** | O(nlogn) | O(nlogn) | O(n2) |

**Graphical Representation :**

****

**Observation:**

The above graph represents Time Comparison of Different Sorting Techniques

X- axis represents number of elements used for sorting.

Y- axis represents the time(ns) taken by the algorithm.

As seen from the above graph:

Insertion Sort - This sorting algorithm works better for small array size. It is in place comparison sort, i.e. it takes one item at a time and inserts in the already sorted sub array. As the size of the array increases, the time taken for sorting the array also increases.

Merge Sort - This sorting algorithm uses “ Divide and conquer” method for sorting the elements. It divides the array into 2 subarrays(n/2) and goes on sorting and finally merge to give final sorted array. As it is not in place sorting, memory space needs to be allocated. From the graph, it is seen that as the size of the array increases it takes the least time to sort the array compared to other algorithms.

Randomized Quicksort - This algorithm can be implemented for huge input sets. The randomized quicksort uses most of the time in random partition. It selects the pivot element randomly. From the graph, it is seen that randomized quicksort takes the same time as merge sort as both use divide and conquer technique.

Modified Quicksort - It is better than merge sort although it has same time complexity as it consists inner loop, making it 1 time faster. It is more optimal sorting.

**Justification and Conclusion :**

X- axis represents number of elements used for sorting.

Y- axis represents the time(ns) taken by the algorithm.

The graph clearly shows that merge sort and randomized sort are straight line with same time complexity

Insertion sort is good for array with small sizes like 500, 1000. But when the size increases to 20000, 40000, 50000,... it is noticed that a small amount of delay is caused while generating the results.

Merge sort: It uses the divide and conquer technique. It works well for large input sizes greater than 10000. The time taken is less for greater input sizes.

Randomized quicksort: It behaves similar to merge sort. For input sizes greater than 10000, it works well too.

Modified quicksort: For small input sizes it works well as it implements the insertion sort. As the input size increases it will increase the time consumed as the complexity and loop executions increase.

Performances when:

1. Input Array is already sorted: This is also known as the best case analysis for the algorithms.

The below output is obtained for best case analysis:

Time taken for randomized quick sorting of is 3652673.0 nanoseconds or 0.0036s  
 Time taken for insertion sorting of is 52526.0 nanoseconds or 0.000005s  
 Time taken for merge sorting of is 1621667.0 nanoseconds or 0.0016s  
 Time taken for modified quicksort sorting of is 69431.0 nanoseconds or 0.000006s

The input size taken is 500.

From the above readings it can be seen that the least time to sort an array is done by insertion sort and the most time taken is by randomized quick sorting.

This is because when a call to insertion sort is made, nine of the elements slide over if the key inserted is greater than or equal to the element to its left. So, if every element is greater than or equal to every element to its left, the runtime is 𝚯(n).

2) Input array is inversely sorted: This is the worst case as the array elements are sorted in descending order. The below output is obtained when the array is inversely sorted:

Time taken for quick sorting of is 1875845.0 nanoseconds

Time taken for insertion sorting of is 3903229.0 nanoseconds

Time taken for merge sorting of is 722686.0 nanoseconds

Time taken for modified quicksort sorting of is 473942.0 nanoseconds

The input size taken is 500.

From the above readings it can be seen that time taken by randomized quick sorting is the least, and merge sort takes the most time.

In case of insertion sort, the first loop executed n-1 times irrespective of the input order of elements so the differentiation factor is second loop execution. If the input array is reversely sorted, then the second loop will execute maximum times.