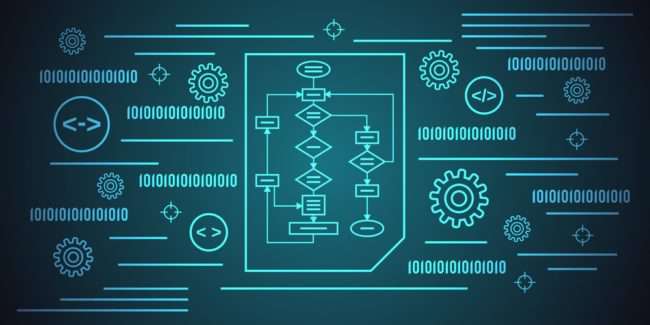


**CSE204: Data Structures and Algorithms I**

Sessional Assignment 7 on Sorting Algorithms



Submitted by:

Md. Tamimul Ehsan

Roll: 1805022

Dept. of CSE, BUET

Section: A1

***Machine Configuration:***

**Processor:** AMD Ryzen 5 3600 6-Core Processor 3.60GHz

**RAM:** 16GB

**Operating System:** Windows 10 64-bit Operating System, x64-based processor

***Complexity Analysis (Theoretical):***

**Merge Sort:**

In merge sort, whether an array is sorted or not doesn’t affect the running time,

Let the sorting time be T(n)

And for n = 1 , T(n) = O(1)

Otherwise,

T(n) = 2\*T(n/2) + k\*n ; because every time the array is divided into two part and then they are inserted in the main array. Here k is a constant.

And base case is T(n) = b;

Now, the worst case occurs when n = 2^p

* P = log n

So, T(n) = k\*n+k\*n\* log n

Thus T(n) = n log n

So, time complexity for merge sort in all case is O(n log n)

And during sorting we only create two temporary arrays to store the sorted results. So, the space complexity is O(n)

**Quick Sort:**

**Ascending and Descending:**

For quick sort the worst case occurs when the array is sorted in ascending or descending order.

Let the sorting time be T(n)

And for n = 1 , T(n) = O(1)

Otherwise,

T(n) = T(n-1) + k\*n ; because every time the pivot is the extreme element and the recursion is called for the left or right elements.

So, the time complexity is O(n^2)

Quicksort is an in place sort, so the space complexity is O(n)

**Random:**

And for random numbers, we can assume that the pivot is set to the middle element at average,

So, T(n) = 2\*T(n/2) + k\*n ; because every time the array is divided into two part and then they are inserted in the main array. Here k is a constant.

And base case is T(n) = b;

Now, the worst case occurs when n = 2^p

* P = log n

So, T(n) = kn + k\*n\* log n

Thus T(n) = n log n

Quicksort is an in-place sort, so the space complexity is O(n)

**Summary:**

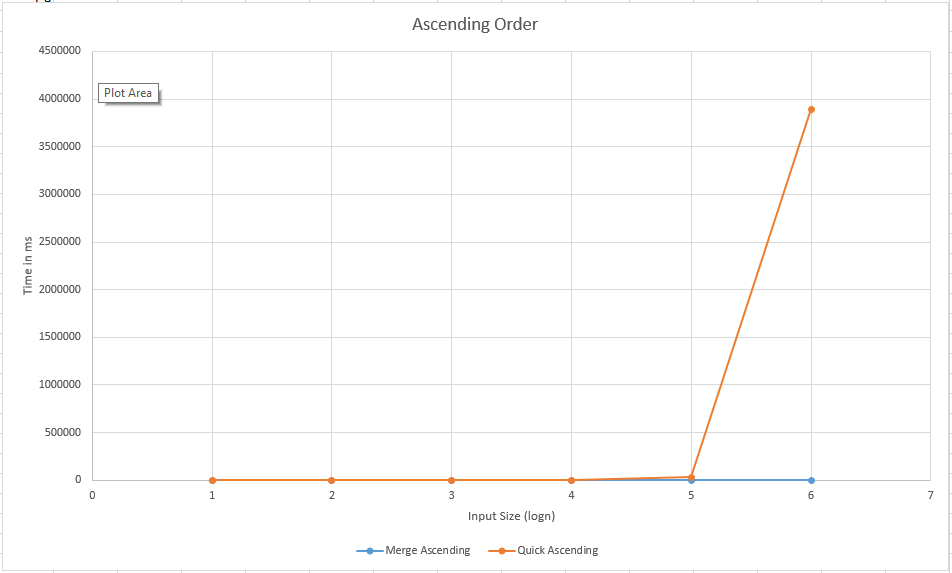
|  |  |  |  |
| --- | --- | --- | --- |
| Input Order | Sorting Algorithm | Time Complexity | Space Complexity |
| Ascending | Merge | O(nlogn) | O(n) |
| Quick | O(n^2) | O(n) |
| Descending | Merge | O(nlogn) | O(n) |
| Quick | O(n^2) | O(n) |
| Random | Merge | O(nlogn) | O(n) |
| Quick | O(nlogn) | O(n) |

***Time taken in milliseconds:***

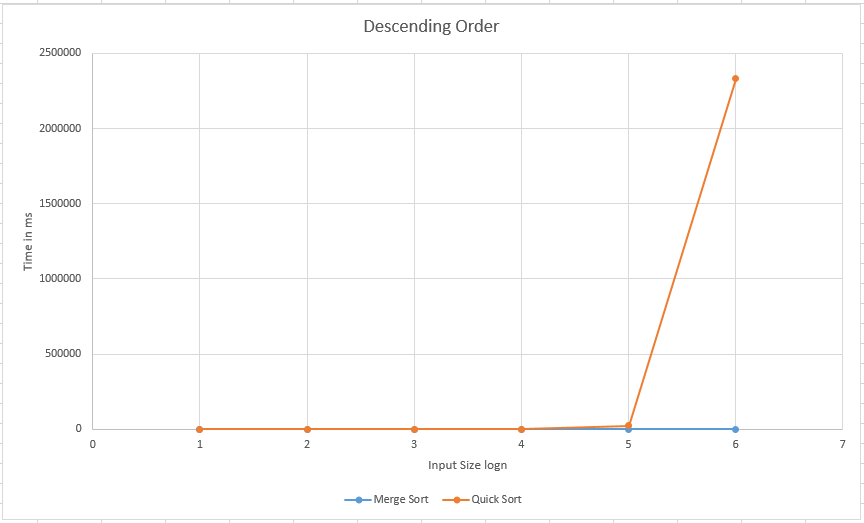
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Input Order | N = | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| Sorting Algorithm |
| Ascending | Merge | 0.000000 | 0.000000 | 0.020000 | 0.400000 | 5.800000 | 72.000000 |
| Quick | 0.000000 | 0.020000 | 2.740000 | 343.420000 | 34197.800000 | 3894732.000 |
| Descending | Merge | 0.000000 | 0.000000 | 0.000000 | 0.300000 | 5.900000 | 74.500000 |
| Quick | 0.000000 | 0.100000 | 1.900000 | 194.200000 | 22160.300000 | 2331852.000 |
| Random | Merge | 0.000000 | 0.000000 | 0.100000 | 1.200000 | 12.300000 | 142.800000 |
| Quick | 0.000000 | 0.050000 | 0.100000 | 1.000000 | 13.100000 | 163.300000 |

***Graphs:***

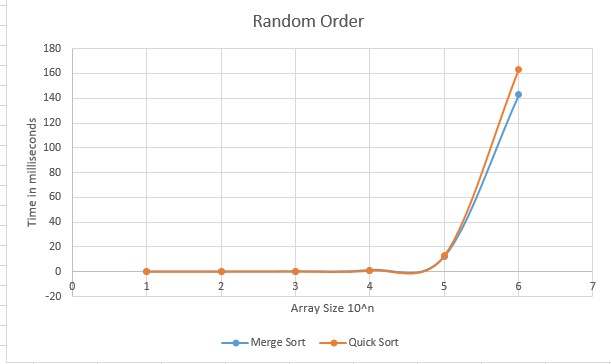
**Ascending Order:**



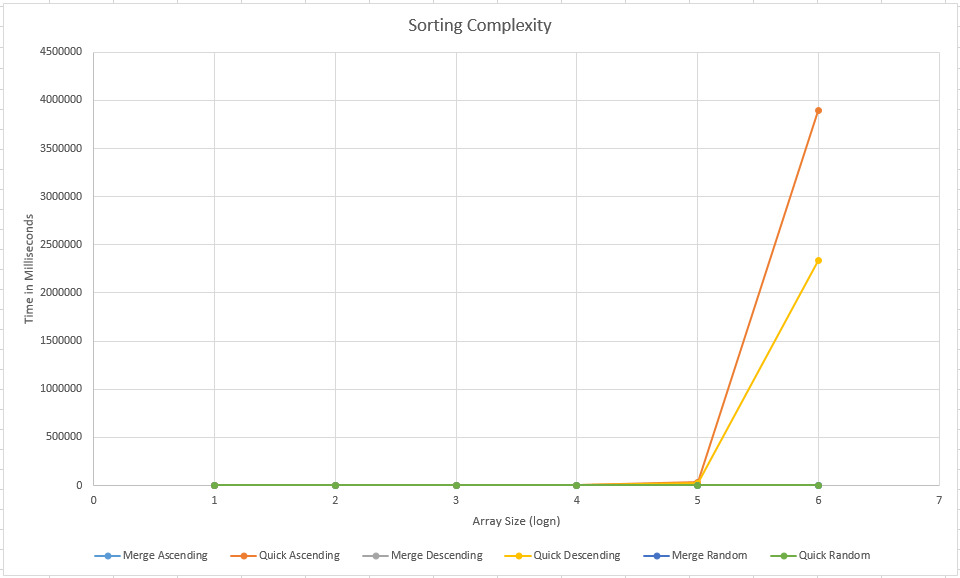
**Descending Order:**

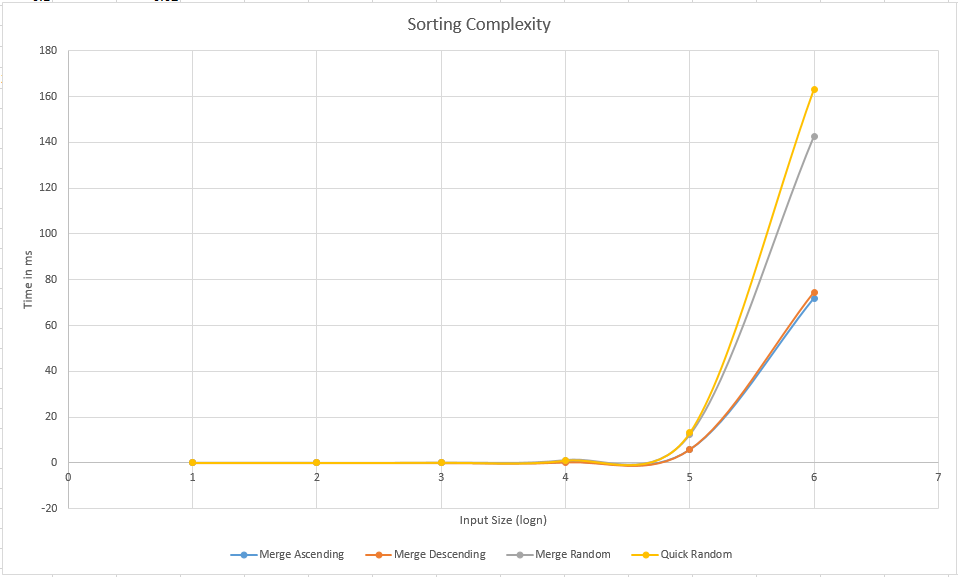


**Random Order:**



**All in One:**

****



***Complexity Analysis (Data based):***

**Merge Sort:**

We can see merge sort is providing almost consistent result in all the orders.

**Ascending:**

T(10^5)/T(10^4) = 20, which is almost equal to (10^4\*log10^4)/( 10^3\*log10^3) =13.3

T(10^5)/T(10^4) = 14.5, which is almost equal to (10^5\*log10^5)/( 10^4\*log10^4) =12.5

T(10^6)/T(10^5) = 12.41, which is almost equal to (10^6\*log10^6)/( 10^5\*log10^5) = 12

So, we can say that the time complexity is n logn

**Descending:**

T(10^5)/T(10^4) = 19.66, which is almost equal to (10^5\*log10^5)/( 10^4\*log10^4) =12.5

T(10^6)/T(10^5) = 12.62, which is almost equal to (10^6\*log10^6)/( 10^5\*log10^5) = 12

So, we can say that the time complexity is n logn

**Random:**

T(10^4)/T(10^3) = 12.00, which is almost equal to (10^4\*log10^4)/( 10^3\*log10^3) =13.3

T(10^5)/T(10^4) = 10.25, which is almost equal to (10^5\*log10^5)/( 10^4\*log10^4) =12.5

T(10^6)/T(10^5) = 11.61 , which is almost equal to (10^6\*log10^6)/( 10^5\*log10^5) = 12

So, we can say that the time complexity is n logn

**Quick Sort:**

**Ascending:**

T(10^4)/T(10^3) = 125.340, which is almost equal to ((10^4)^2)/(( 10^3)^2) =100

T(10^5)/T(10^4) = 99.580, which is almost equal to ((10^5)^2)/(( 10^4)^2) =100

T(10^6)/T(10^5) = 113.80, which is almost equal to ((10^6)^2)/(( 10^5)^2) = 100

So, we can say that the time complexity is n^2

**Descending:**

T(10^4)/T(10^3) = 102.21, which is almost equal to ((10^4)^2)/(( 10^3)^2) =100

T(10^5)/T(10^4) = 114.11, which is almost equal to (10000^2)/( 1000^2) =100

T(10^6)/T(10^5) = 105.23, which is almost equal to ((10^6)^2)/(( 10^5)^2) = 100

So, we can say that the time complexity is n^2

**Random:**

T(10^4)/T(10^3) = 10.00, which is almost equal to (10^4\*log10^4)/( 10^3\*log10^3) =13.3

T(10^5)/T(10^4) = 13.10, which is almost equal to (10^5\*log10^5)/( 10^4\*log10^4) =12.5

T(10^6)/T(10^5) = 12.47, which is almost equal to (10^6\*log10^6)/( 10^5\*log10^5) = 12

So, we can say that the time complexity is n logn

***Observation:***

It is seen that merge sort is always providing consistent result, while quicksort provides some inconsistency. This is mainly because of the choice of pivot point. This time consumption can be omitted if the pivot s taken at the middle. But even so, an anti-case may be generated based on fixed pivot. This can be omitted by using randomized pivot.

And again, for fixed pivot at last position, ascending order takes the maximum time. This is because the number of swap it needs is the maximum. But if we observe we will see that the swap is always occurring with itself. So, a precheck or a better performance swap may increase the efficiency by many folds.