**OFFLINE-7**

Course no. : CSE204

Course name : Data structure and algorithm

Name: Md. Farhan mahtab

ID : 1805096

Sec : B2

Date:  11.06.2021

Machine configuration:

| | |
|---|---|
| Device name | LAPTOP-BMC7C01U |
| Processor | Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz |
| Installed RAM | 8.00 GB (7.70 GB usable) |
| Device ID | 23452738-7156-4474-AEBA-94C5533135B9 |
| Product ID | 00327-35891-80748-AAOEM |
| System type | 64-bit operating system, x64-based processor |

Table:

| Input Order | n= / Sorting Algorithm | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
|---|---|---|---|---|---|---|---|
| Ascending | Merge | 0 | 0 | 1.6 | 3.6 | 62 | 360 |
| | Quick | 0 | 0 | 13.3 | 863.6 | 54827 | 6780000 |
| Descending | Merge | 0 | 0 | 1.05 | 2.5 | 70 | 338 |
| | Quick | 0 | 0 | 5.55 | 433.9 | 42107 | 7397000 |
| Random | Merge | 0 | 0 | 1.1 | 6.25 | 43.2 | 463.45 |
| | Quick | 0 | 0 | .75 | 1.1 | 20.9 | 254.45 |

Complexity Analysis:

**Merge Sort :**

Merge sort is divided into three parts: divide, conquer, combine.
For all three arrays the divide part takes the same time for the same n.
So, $D(n) = \Theta(1)$
For the conquer part we solve the two divided parts recursively.
So, $C(n) = 2*T(n/2)$
Finally merging or combining takes $\Theta(n)$ for n input.

$T(n) = 2*T(n/2) + \Theta(n) + \Theta(1)$
$= 2*T(n/2) + \Theta(n)$
By using master theorem we can say that

$T(n) = \Theta(nlgn)$
Which is the worst time complexity for n>1.

**Quick sort :**

For partitioning, function costs $\Theta(n)$ for n input.
As we partitioned the array into two parts and recursively sort the two part.
So, It will costs, C(n) = T(l) + T(n-l)

$T(n) = T(l) + T(n-l) + \Theta(n)$

For an ascending input array, as the array is already sorted and pivot is the last element, all elements from the left of pivot will remain on the left side. Which will partition the array of n-1 and 0 size.

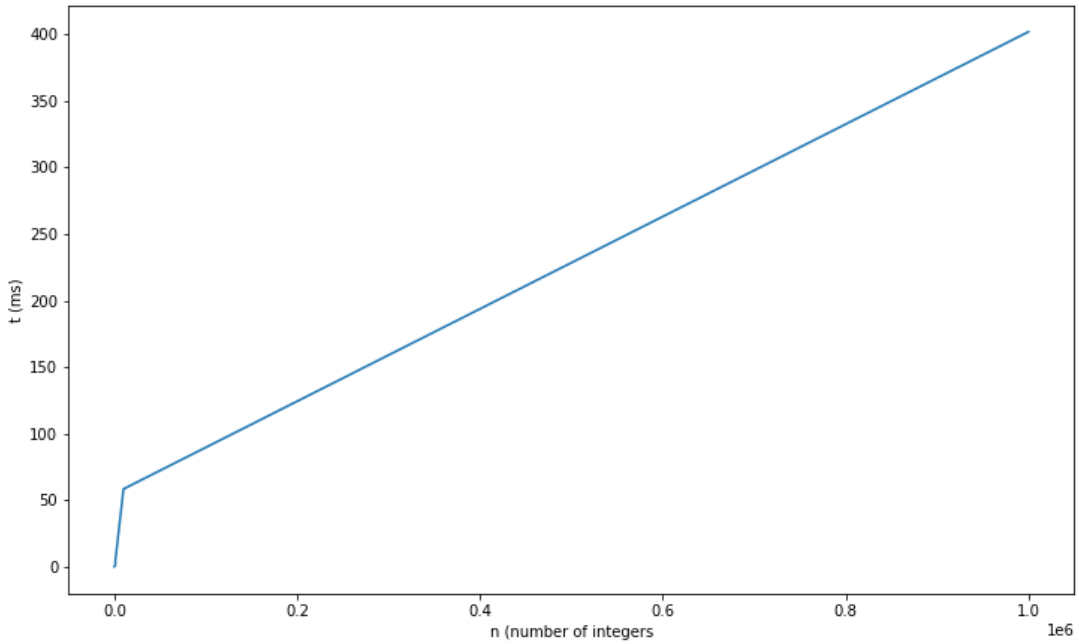
So, $T(n) = T(0) + T(n-1) + \Theta(n)$
$= T(n-1) + \Theta(n)$
$= T(n-1) + cn$ [ For $\Theta(n) = cn$]

$T(n-1) = T(n-2) + \Theta(n-1)$
$= T(n-2) + c(n-1)$
$T(1) = 0 + c.1$

So, $T(n) = cn + c(n-1) + \ldots\ldots\ldots\ldots + c.1$
$= c.(n + n-1 + \ldots\ldots\ldots\ldots + 1)$
$= c.n(n+1)/2$
$= cn^2$
$= \Theta(n^2)$

Which is the worst case scenario for Quicksort which can be also seen at the data table where n is a greatly large integer time taken to sort the algorithm is also lengthy .
For descending input arrays, the pivot will be largest . so, all elements from the left will move to the right and partition the array into two arrays of n-1 and 0.
In this case the cost will be same as Ascending array which is $\Theta(n^2)$.

In the best case scenario the partition will be two arrays of the same size of n/2.
In this case, $T(n) = 2*T(n/2) + \Theta(n)$
Which is the same as merge sort.
So, the best case time complexity of quick sort is $\Theta(nlgn)$.
Merge Sort plot:




Quick sort plot: