CSE-204: *Data Structures and Algorithms I Sessional*

Offline-7:
Sorting Algorithms

Submitted by:
Tanjeem Azwad Zaman
Roll: 1805006
Dept. of CSE, BUET
Section: A1 Batch:18

Date of Submission:
11.06.2021

Table:

Input Order	n = / Sorting Algorithm	10	100	1000	10000	100000	1000000
Ascending	Merge	2.82 ns	12.754 ns	67.33 ns	533.654 ns	5.446µs	60.227 µs
	Quick	2.02 ns	5.08 ns	201.975ns	60.83 µs	6115.55 µs	627.398 ms
Descending	Merge	3.05 ns	29.186 ns	92.943 ns	487.657 ns	4.645 µs	49.905 µs
	Quick	4.03 ns	8.245 ns	377.096 ns	39.375 µs	3875.89 µs	413.876 ms
Random	Merge	4.13 ns	11.17 ns	122.644ns	1.182 µs	12.885 µs	148.796 µs
	Quick	2.07 ns	5.85 ns	54.089 ns	609.543 ns	7.645 µs	91.055 µs

Complexity Analysis:

- In general, the readings for n = 10, 100 (and sometimes 1000) do not follow any pattern. This is possibly due to:
  - System.nanoTime() returns nanosecond precision, but **not nanosecond accuracy.** From the last updated documentation in JAVA7: *"no guarantees are made except that the resolution is at least as good as that of currentTimeMillis()"*
  - The time required for operations may trump the number of operations / order of the algorithm (very small in number)

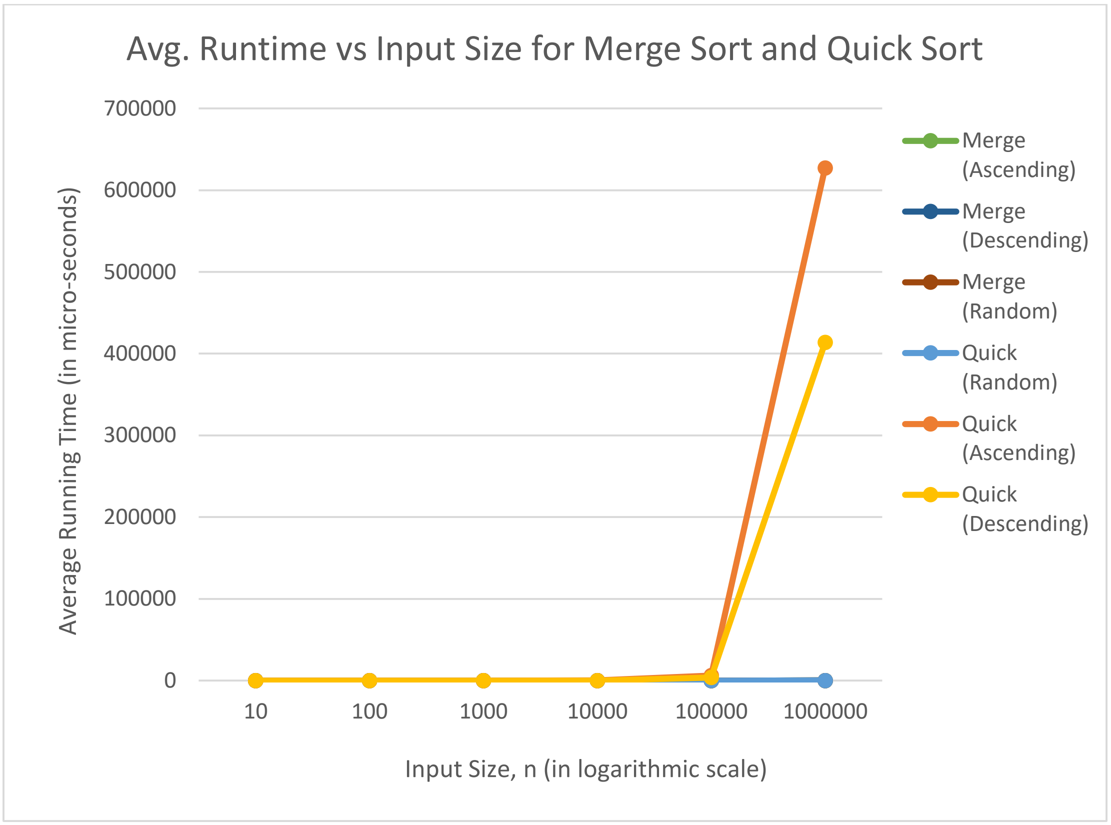
- For Merge Sort:
  - Theoretically, the time complexity is O(nlogn). Which means, ideally, an increase of 'n' by 10 times, would increase runtime by a little more than 10 times, for all orders.
  - Comparing the runtimes with change in ***input size,*** (n = 10^6 : 10^5: 10^4):
    - ***Ascending order-*** 60.227 : 5.445 : 0.533654 ≈ 100:10:1 (A bit more than 10x in each step, to be precise)
    - ***Descending order-*** 49.905 : 4.645 : 0.487675 ≈ 100:10:1 (A bit more than 10x in each step, to be precise)
    - ***Random order-*** 148.796 : 12.885 : 0.122644 ≈ 100:10:1 (A bit more than 10x in each step, to be precise)
    - This supports the theoretical O(nlogn) Time Complexity
  - Comparing the runtimes with change in ***input size,*** (Ascending : Descending : Random),
    - ***n=10^6-*** 60.227 : 49.905 : 148.796 ≈ 6:5:15
    - ***n=10^5-*** 5.446 : 4.645 : 12.885 ≈ 6:5:15
    - ***n=10^4-*** 0.533 : 0.487 : 1.182 ≈ 11:10:25
    - We see that, on average, Ascending takes less than Descending. And Random order takes more than 2.5 times than the rest.

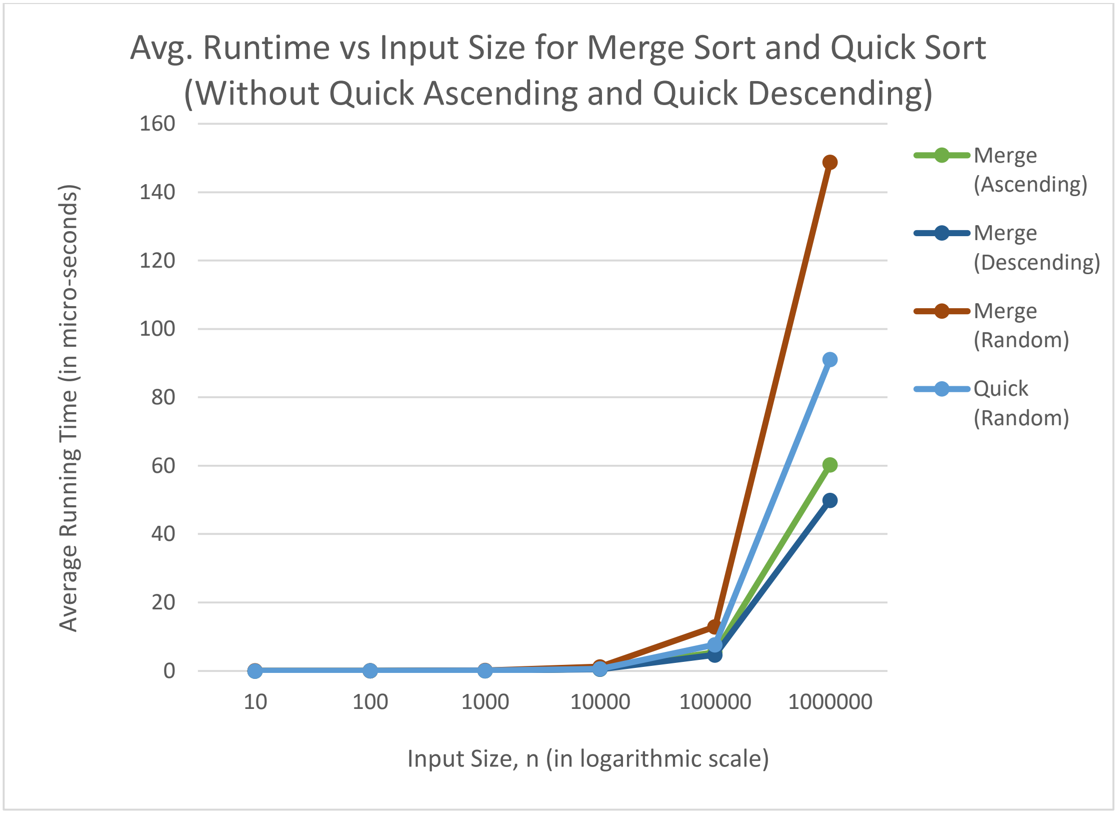
- For Quick Sort:
    - Theoretically, the average case time complexity is O(nlogn) for Random ordering. Which means, ideally, an increase of ‘n’ by 10 times, would increase runtime by just above 10 times.
    - For Ascending or Descending ordering, we face the worst case time complexity of O(n^2), which means the runtime should increase by a factor of 100, for 10 times increase in n. Comparing the runtimes with change in ***input size,*** (n = 10^6 : 10^5: 10^4):
        - ***Ascending order-*** 627398 : 6115.55: 60.83 ≈ 10000:100:1
        - ***Descending order-*** 413876 : 3875.89 : 39.375 ≈ 10000:100:1
        - These supports the theoretical O(n^2) Worst Case Time Complexity
        - ***Random order-*** 91.055 : 7.645 : 0.609543 ≈ 100:10:1 (A bit more than 10x in each step, to be precise)
        - This supports the theoretical O(nlogn) Average Case Time Complexity
    - Comparing the runtimes with change in ***input size,*** (Ascending : Descending),
        - ***n=10^6-*** 627.398: : 413.876 ≈ 6:4
        - ***n=10^5-*** 6115.55: 3875.89 ≈ 6:4
        - ***n=10^4-*** 60.83: 39.375 ≈ 6:4
        - We see that, on average, Ascending takes more time than Descending. This can be explained by the fact that when sorting descending arrays, there is no swapping in the “Partition” function other than the single swap of the pivot with the first element at the end. But in sorting an ascending array, a swap is done at every step of the “Partition” function.
- Merge Sort vs. Quick Sort:
    - Theoretically, although both algorithms have an average case time complexity of O(nlogn), the constant associated with Merge Sort > that of Quick sort.
    - This can be observed in sorting “***Random Order”*** arrays. Where, time for Merge : Quick is always Greater than 1:
        - 148.796: 91.055 ≈ 12.885: 7.645 ≈ 1182 : 609.543 ≈ 122.644: 54.089 ≈ 2 > 1
    - As for sorting in ascending and descending orders, the merge sort is faster by a huge margin, since the complexity is O(nlogn) vs. the O(n^2) worst case complexity of the Quick Sort

Machine Configuration:

- RAM: 16GB
- Processor: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
- System type: 64-bit operating system, x64-based processor

Plot:


Avg. Runtime vs Input Size for Merge Sort and Quick Sort
Average Running Time (in micro-seconds)
700000
600000
500000
400000
300000
200000
100000
0
10
100
1000
10000
100000
1000000
Input Size, n (in logarithmic scale)
Merge (Ascending)
Merge (Descending)
Merge (Random)
Quick (Random)
Quick (Ascending)
Quick (Descending)


Avg. Runtime vs Input Size for Merge Sort and Quick Sort (Without Quick Ascending and Quick Descending)
Average Running Time (in micro-seconds)
160
140
120
100
80
60
40
20
0
10
100
1000
10000
100000
1000000
Input Size, n (in logarithmic scale)
Merge (Ascending)
Merge (Descending)
Merge (Random)
Quick (Random)