CS202

HOMEWORK 1

Muhammad Arham Khan

21701848 – CS

Section: 3

Dated: 9 March, 2019

Question 1

(a)

We need "100n^3 + 8n^2 + 4n <= Cn^4" for n >= n_0
So (100/n) + (8/n2) + (4/n^2) <= c
Choose C = 200 (as 100/n <= 100, 8/n2 <= 8, 4/n3 <= 1 for n >= 1)
since n0 = 1,
(100/n) + (8/n^2) + (4/n^3) <= 100 + 8 + 4 = 112 <= 200 = C

(b)

Assume n is a power of 2, so

$T(n) = 8T(n/2) + n^3$
$= 8 (8T (n/4) + (n/2)^3) + n^3$
$= 8^2(T (n/2^2) + 8 . (n^3/2^3) + n^3$
$= 8^2(8T(n/2^3) + (n/3)^3) + 2n^3$
$= 8^3T(n/2^3) + 3n^3$
…
$= 8^kT(n/2^k) + kn^3$ where k >= 1

Taking $T(n/2^k) = T(1)$ after enough k's
$n/2^k = 1$ tends to $n = 2^k$ tend to $k = \log_2 n$
$= 8^kT(1) + kn^3$
$= 8\log_2 n + \log_2 n . n^3$
$= (n\log_2 n)^3 + n^3\log_2 n$
$= n^3 + n^3\log n$
$= T(n^3\log n)$

(c)

Outer loop operates log(n) times, the middle loop runs n times and inner most loop runs n/2 times. Hence, the run-time magnitude is $O(\log(n) . n . n/2) = O(n^2\log(n))$

(d)

Selection Sort: [16, 6, 39, 21, 10, 21, 13, 7, 28, 19]

[16, 6, 19, 21, 10, 21, 13, 7, 28, 39]

[16, 6, 19, 21, 10, 21, 13, 7, 28, 39]

[16, 6, 19, 7, 10, 21, 13, 21, 28, 39]

[16, 6, 19, 7, 10, 13, 21, 21, 28, 39]

[16, 6, 13, 7, 10, 19, 21, 21, 28, 39]

[10, 6, 13, 7, 16, 19, 21, 21, 28, 39]

[10, 6, 7, 13, 16, 19, 21, 21, 28, 39]

[7, 6, 10, 13, 16, 19, 21, 21, 28, 39]

[6, 7, 10, 13, 16, 19, 21, 21, 28, 39]

Insertion sort: [16, 6, 39, 21, 10, 21, 13, 7, 28, 19]

[6, 16, 39, 21, 10, 21, 13, 7, 28, 19]

[6, 16, 39, 21, 10, 21, 13, 7, 28, 19]

[6, 16, 39, 21, 10, 21, 13, 7, 28, 19]

[6, 16, 21, 39, 10, 21, 13, 7, 28, 19]

[6, 10, 16, 21, 39, 21, 13, 7, 28, 19]

[6, 10, 16, 21, 21, 39, 13, 7, 28, 19]

[6, 10, 13, 16, 21, 21, 39, 7, 28, 19]

[6, 7, 10, 13, 16, 21, 21, 39, 28, 19]

[6, 7, 10, 13, 16, 21, 21, 28, 39, 19]

[6, 7, 10, 13, 16, 19, 21, 21, 28, 39]

Question 2

```
MF:Downloads Mohammad$ ./arham
 [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,  ]
 [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,  ]
 [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,  ]
 [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,  ]
-----------------------------------------------------
Part c - Time analysis of Radix Sort
Array Size      TimeElapsed
2000            2.121
6000            6.235
10000           8.901
14000           13.857
18000           18.458
22000           22.447
26000           25.326
30000           28.933
-----------------------------------------------------
Part c - Time analysis of Bubble Sort
Array Size      TimeElapsed     compCount       moveCount
2000            8.747           1530004         2302809
6000            126.429         17993430                26873172
10000           366.761         49991760                75686793
14000           723.106         97992829                146828022
18000           1220.06         161981955               242433879
22000           1816.73         241974465               363517746
26000           2564.72         337964422               507462552
30000           3411.38         449963679               670906920
-----------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size      TimeElapsed     compCount       moveCount
2000            1.146           487979          36359
6000            0.817           84891           148694
10000           1.373           168061          271455
14000           1.872           247810          397387
18000           2.333           293768          464803
22000           2.925           368405          571226
26000           3.506           438801          715223
30000           4.378           539824          852862
-----------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size      TimeElapsed     compCount       moveCount
2000            0.289           19334           43904
6000            0.944           67819           151616
10000           1.611           120562          267232
14000           2.293           175262          387232
18000           3.083           231928          510464
22000           3.856           290067          638464
26000           4.776           348829          766464
30000           5.678           408627          894464
```

Question 3

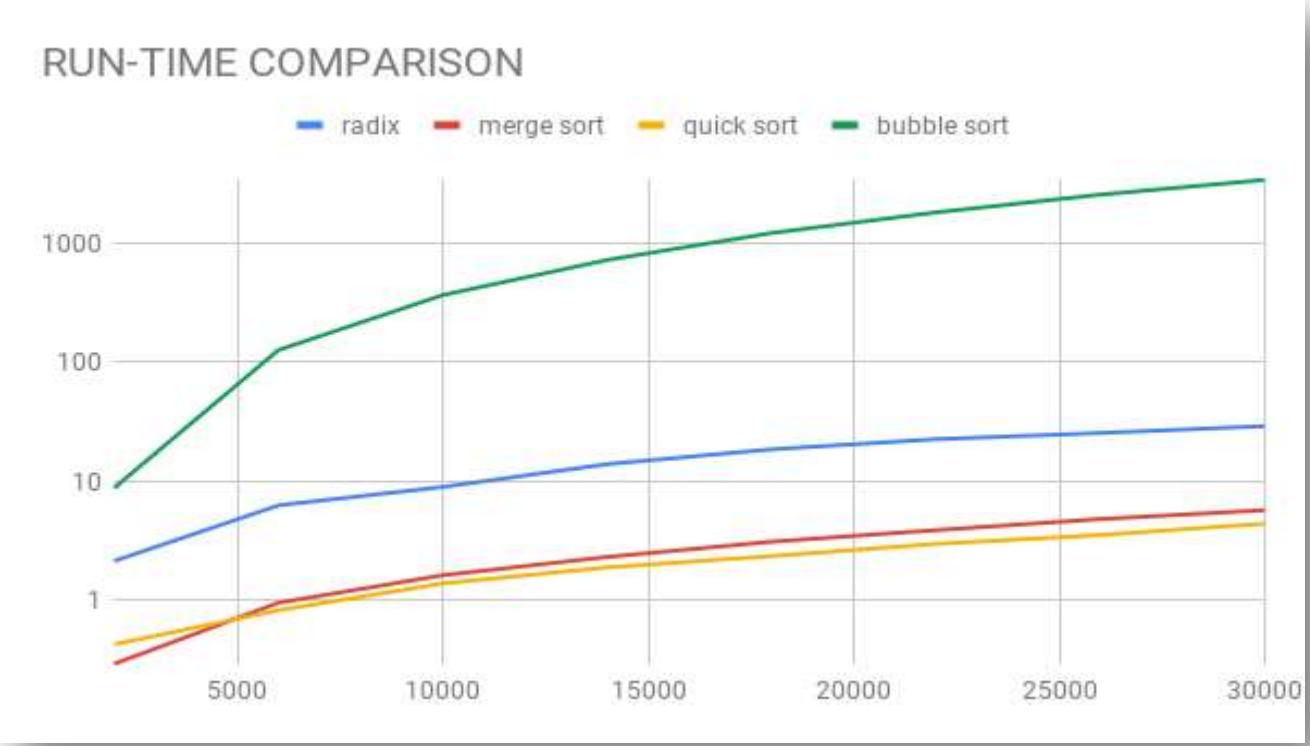

Figure 1: Plot of elapsed time (on the y-axis – logarithmic scales) vs. array size (on the x-axis)

RADIX SORT:

EMPIRICAL Vs. THEORETICAL:

Theoretically, this algorithm is supposed to progress with a time complexity of O(kn) where k is the length of the maximum integer and it should produce a linear graph. But, in this graph, since the y-axis is logarithmic, it appears to follow a downward bent trend testifying the its linear time complexity. Hence, the graph complies with the theoretical assumptions.

CHANGE IN COMPLEXITY WITH REVERSED INPUT:

Assuming that the input array was completely reversed, the time complexity would still be the same since the algorithm runs in linear time and it does not matter how misarranged the items are.

BUBBLE SORT:

EMPIRICAL Vs. THEORETICAL:

Theoretically, the time complexity of the bubble sort is $O(n^2)$ And, as is evident from the obtained results, the trend of the bubble sort takes longer than other sorting algorithm and increases on an exponential level on the logarithmic axis. Hence, the results comply with the theoretical values.

CHANGE IN COMPLEXITY WITH REVERSED INPUT:

Considering such a case, the complexity of the algorithm would still be the worst-case ie: $O(n^2)$ and infact, would take more longer than the current trend. Since there would be more swaps and comparisons in this case, it would definitely take longer and time complexity would increase

MERGE SORT:

EMPIRICAL Vs. THEORETICAL:

Theoretically, this has a time complexity of O(nlogn) and as is evident from the graph, the graph on the logarithmic axis appears to be following the O(nlogn) increase trend and appears to be nearly straight due to logarithmic axis and hence, it complies.

CHANGE IN COMPLEXITY WITH REVERSED INPUT:

If the array was placed in reverse order, the time complexity would not change. (worst case time complexity is O(nlogn)

QUICK SORT:

EMPIRICAL Vs. THEORETICAL:

Theoretically, the time complexity of quick sort is O(nlogn) and as is evident from the graph, the trend is followed as the line appears to be nearly straight due to logarithmic axis and hence, its time complexity is O(nlogn)

CHANGE IN COMPLEXITY WITH REVERSED INPUT:

Since O(nlogn) is the average case time complexity, the time complexity would increase to $O(n^2)$ if the whole array is reversed as that is the worst case scenario.