**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 >= n0

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) + n3

= 8 (8T (n/4) + (n/2)^3) + n3

= 82(T (n/22) + 8 . (n3/23) + n3

= 82(8T(n/23) + (n/3)3) + 2n3

= 83T(n/23) + 3n3

…

= 8kT(n/2k) + kn3 where k >= 1

Taking T(n/2k) = T(1) after enough k’s

n/2k = 1 tends to n = 2k tend to k = log2n

= 8kT(1) + kn3

= 8log2n + log2n . n3

= (nlog2n)3 + n3log2n

= n3 + n3logn

= T(n3logn)

**(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( n2log(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]

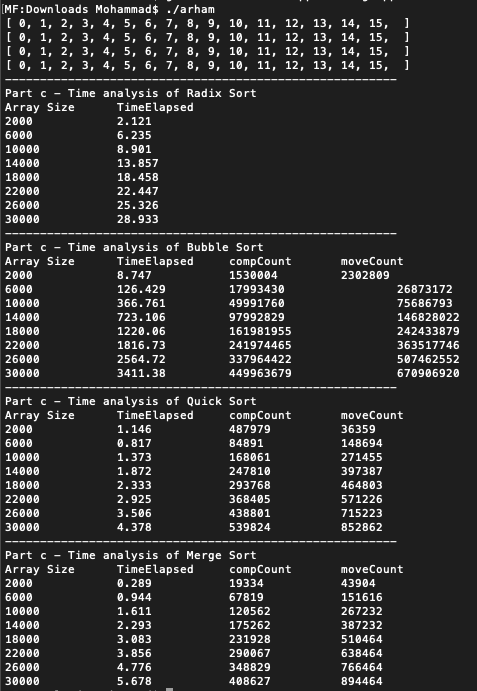
[ 6, 10, 13, 16, 21, 21, 39, 7, 28, 19]

[ 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**

****

**Question 3**

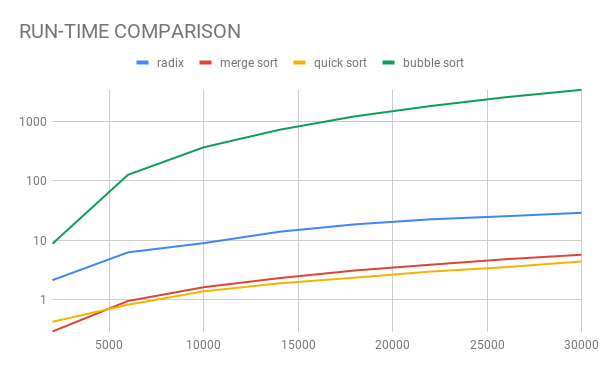


Figure : Plot of elapsed time (on the y-axis – logarithmic scales) vs. array size (on the x-axis)

**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(n2) 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(n2) 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(n2) if the whole array is reversed as that is the worst case scenario.