**CSE 203**

**Class Work on Time complexity and Big-Oh notation**

***Answer to Problem 1:***

From some given expression dominant term(s) and lowest big-Oh complexity is determined.

|  |  |  |
| --- | --- | --- |
| *Expression* | *Dominant Term(s)* | *Big-Oh* |
| 5 + .001n3 + 0.025n | 0.001n3 | O (n3) |
| 500n + 100n1.5 + 50n log10n | 100n1.5 | O (n1.5) |
| 0.3n + 5n1.5 + 2.5n1.75 | 2.5n1.75 | O (n1.75) |
| n2log2n + n(log2n)2 | n2log2n | O (n2log2n) |
| nlog3n + nlog2n | nlog2n | O (nlog2n) |
| 3log8n + log2log2log2n | 3log8n | O (log8n) |
| 100n + .01n2 | .01n2 | O (n2) |
| .01n + 100n2 | 100n2 | O (n2) |
| 2n + n0.5 + 0.5n1.25 | 0.5n1.25 | O (n1.25) |
| 0.01nlog2n + n(log2n)2 | n(log2n)2 | O(n(log2n)2) |
| 100nlog3n + n3 + 100n | n3 | O (n3) |
| 0.003log4n + log2log2n | 0.003log4n | O (log4n) |

***Answer to Problem 2:***

Some expressions are given. Truth value of the expressions are determined.

|  |  |  |
| --- | --- | --- |
| Expression | True or False | Correct Formula if False |
| Sum rule:  O (f + g) = O (f) + O (g) | False | O (f + g) = maximum (O (f), O (g)) |
| Product rule:  O (f \* g) = O (f) \* O (g) | True |  |
| Transitivity:  If g = O (f) and h = O (f)  then g = O (h) | False | If g = O (f) and f = O (h)  then g = O (h) |
| 5n + 8n2 +100n3 = O (n4) | True |  |
| 5n + 8n2 +100n3 = O (n2log n) | False | 5n + 8n2 +100n3 = O (n3)  Assuming lower complexity |

***Answer to Problem 3:***

Two algorithms A and B with spend time TA(n) = 0.1n2log10n and TB(n) = 2.5n2 are given.

So according to the rules, complexity for each algorithm in terms of big-Oh notation is

CA = O (n2log10n)

CB = O (n2)

Comparing this two algorithm’s complexity second one is better than the first one.

To prove this, let n = 100 and by computing it is clear that first one takes time which is 2 times greater than the second one in big-Oh sense.

So, if the size of n 🡪 109 then second one will be better in big-Oh sense.

But for smaller size of n, let n = 10 then both algorithms run in same time in sense of big-Oh.

So, if the size of n > n0 = 10 then the first one is much better than the other one.

But if the size of n 🡪 109 or > 109 then I will recommend second one to use according to big-Oh sense.

***Answer to Problem 4:***

According to the problem statement, randomValue takes constant number of computational steps ‘c’ and goodSort takes n log n. with the help of the data Big-Oh complexity is determined,

For (i = 0; i < n; i++) { complexity: O(n)

For (j = 0; j < n; j++) complexity: O(n)

a [j] = randomValue (i); complexity: O (1) taken total n\*c steps

goodSort(a); complexity: O (n log n) taken total n\* n log n steps

}

Total taken steps for the code segment:

n \* ( n \* ( c ) + n log n )

or, n2c + n2 log n steps

So, in big-Oh sense:

Complexity: O (n2 log n).

***Answer to Problem 5:***

For the given code segment, the most inner loops

for (int j = 1; j < n; j \*= 2) takes ‘log2(n-1) 〜 log2n’ steps for each time or O (log2n)

and the other one, for (int j = 1; j < n; j += 2) takes n/2 steps for each time or O (n)

So, inner loops will take,

X = log2n + n/2 steps for each time

Now, for the middle one, it will continue with total steps

1, 3, 5, ……., n steps or O(n)

as the outer one executes.

And for the outer one, for (int bound = 1; bound <= n; bound \*= 2) takes log2n steps or O (log2n)

So, complexity in sense of big-Oh will be,

O (log2n \* (n \* (log2n + n)))

or, O (n2log2n)