CS6033 - ASSIGNMENT 1

Name : Seetharaman Krishnamoorthy

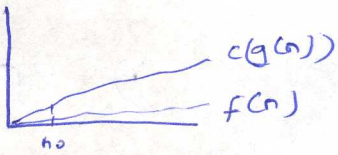
Id : N13274420

1) Conditions : ① $f(n) \in o(g(n))$ and

② $f(n) \notin \Theta(g(n))$

i) $f(n) \in o(g(n))$

$\Rightarrow 0 \leq f(n) < c\,g(n)$



ii) $f(n) \notin \Theta(g(n))$

$\Rightarrow 0 \leq c_1(g(n)) \nleq f(n) \nleq c_2\,g(n)$

∴ This Implies $f(n)$ should be small enough so that there will be no $c_1(g(n))$ which will hold true.

Example :

$f(n) = \log n \qquad g(n) = 5n^2$

when $n = 1$

$f(n) = 0 \qquad g(n) = 5$

$0 < 5$

Condition ① : So $f(n)$ is $o(g(n))$

Condition ② : $0 \leq c_1 5 \leq 0 \leq c_2 5$

Here we cannot find any positive constant c_1 which will result in $c_1 5 \leq 0$

$\therefore$ $f(n) = \log n$

$g(n) = 5n^2$

These functions satisfies both the conditions.

2) Conditions : ① $f(n) \in \Omega(g(n))$

② $f(n) \notin O(g(n))$

i) $f(n) \in \Omega(g(n))$

$0 \leq c(g(n)) \leq f(n)$

ii) $f(n) \notin O(g(n))$

$0 \nleq f(n) \nleq c(g(n))$

- Here as it depend on constant, there can always be a large constant to make this true.

Ans:

- No such functions exist.

3)

Given: $f(n)$ → microseconds

1 month → 31 days

	sec	min	hour	day	month	year	Century
$\lg n$	2^{10^6}	$2^{6\times10^7}$	$2^{3.6\times10^9}$	$2^{8.6\times10^{10}}$	$2^{267\times10^{10}}$	$2^{3.2\times10^{13}}$	$2^{3.2\times10^{15}}$
$\sqrt{n}$	10^{12}	36×10^{14}	$(3.6\times10^9)^2$	$(8.6\times10^{10})^2$	$(267.84\times10^{10})^2$	$(3.2\times10^{13})^2$	$(3.2\times10^{15})^2$
n	10^6	6×10^7	3.6×10^9	8.6×10^{10}	267.84×10^{10}	3.2×10^{13}	3.2×10^{15}
n^2	10^3	$(6\times10^7)^{1/2}$	$(3.6\times10^9)^{1/2}$	$(8.64\times10^{10})^{1/2}$	$(267.84\times10^{10})^{1/2}$	$(3.2\times10^{13})^{1/2}$	$(3.2\times10^{15})^{1/2}$
n^3	10^2	$(6\times10^7)^{1/3}$	$(3.6\times10^9)^{1/3}$	$(8.64\times10^{10})^{1/3}$	$(267.84\times10^{10})^{1/3}$	$(3.2\times10^{13})^{1/3}$	$(3.2\times10^{15})^{1/3}$
2^n	19	25.83	31.74	36.33	41.28	44.86	51.51

i) $\lg n$

$\log_2 n = 1000000$ (ms to sec)

$n = 2^{1000000}$

$n = (2)^{10^6}$

$\log_2 n = 60\times10^6$

$= 6\times10^7$

$n = (2)^{6\times10^7}$ (min to ms)

$\log_2 n = 3.6\times10^9$

$n = (2)^{3.6\times10^9}$ (Hr to ms)

$\log_2 n = 8.64\times10^{10}$

$n = (2)^{8.64\times10^{10}}$ (Day to ms)

$\log_2 n = 267.84 \times 10^{10}$

$n = (2)^{267.84 \times 10^{10}}$ (month to µs)

$\log_2 n = 3.214 \times 10^{13}$

$n = (2)^{3.214 \times 10^{13}}$ (year to µs)

$\log_2 n = 3.214 \times 10^{15}$

$n = (2)^{3.214 \times 10^{15}}$ (century to µs)

ii) $\sqrt{n}$

$\sqrt{n} = 10^6$

$n = 10^{12}$ (sec to µs)

$\sqrt{n} = 6 \times 10^7$

$n = 36 \times 10^{14}$ (min to µs)

$\sqrt{n} = 3.6 \times 10^9$

$n = (3.6 \times 10^9)^2$ (Hour to µs)

$\sqrt{n} = 8.64 \times 10^{10}$

$n = (8.64 \times 10^{10})^2$ (Day to µs)

$\sqrt{n} = 267.84 \times 10^{10}$

$n = (267.84 \times 10^{10})^2$ (month to µs)

$\sqrt{n} = 3.214 \times 10^{13}$

$n = (3.214 \times 10^{13})^2$ (year to µs)

$\sqrt{n} = 3.214 \times 10^{15}$

$n = (3.214 \times 10^{15})^2$ (century to µs)

iii) n

$n = 10^6$

$n = 6 \times 10^7$

$n = 3.6 \times 10^9$

$n = 8.64 \times 10^{10}$

$n = 267.84 \times 10^{10}$

$n = 3.214 \times 10^{13}$

$n = 3.214 \times 10^{15}$

iv) n^2

$n^2 = 10^6$

$n = 10^3$

$n^2 = 6 \times 10^7$

$n = (6 \times 10^7)^{1/2}$

$n^2 = 3.6 \times 10^9$

$n = (3.6 \times 10^9)^{1/2}$

$n^2 = 8.64 \times 10^{10}$

$n = (8.64 \times 10^{10})^{1/2}$

$n^2 = 267.84 \times 10^{10}$

$n = (267.84 \times 10^{10})^{1/2}$

$n^2 = 3.214 \times 10^{13}$

$n = (3.214 \times 10^{13})^{1/2}$

$n^2 = 3.214 \times 10^{15}$

$n = (3.214 \times 10^{15})^{1/2}$

v) n^3

$n^3 = 10^6$

$n = 10^2$

$n^3 = 10^7 \times 6$

$n = (6 \times 10^7)^{1/3}$

$n^3 = 3.6 \times 10^9$

$n = (3.6 \times 10^9)^{1/3}$

$n^3 = 8.64 \times 10^{10}$

$n = (8.64 \times 10^{10})^{1/3}$

$n^3 = 267.84 \times 10^{10}$

$n = (267.84 \times 10^{10})^{1/3}$

$n^3 = 3.214 \times 10^{13}$

$n = (3.214 \times 10^{13})^{1/3}$

$n^3 = 3.214 \times 10^{15}$

$n = (3.214 \times 10^{15})^{1/3}$

vi)

$2^n = 10^6$

$n = 6 \lg 10 = 19$

$2^n = 6 \times 10^7$

$n = \log_2 (6 \times 10^7) = 25.83$

$2^n = 3.6 \times 10^9$

$n = \log_2 (3.6 \times 10^9) = 31.74$

$2^n = 8.64 \times 10^{10}$

$n = \log_2 (8.64 \times 10^{10}) = 36.33$

$2^n = 267.84 \times 10^{10}$

$n = \log_2 (267.84 \times 10^{10}) = 41.28$

$2^n = 3.214 \times 10^{13}$

$n = \log_2(3.214 \times 10^{13}) = 44.86$

$2^n = 3.214 \times 10^{15}$

$n = \log_2(3.214 \times 10^{15}) = 51.51$

4)

```
PRINT_DUPLICATES(A)
for i=1 to A.length
    For j=i to A.length
        if A[i] == A[j]
            PRINT(A[i])
```

→ This algorithm does not correctly print the duplicates.

→ Consider the example,

A = | 5 | 1 | 3 | 1 |

Here for first iteration i=1 & j=1 so obviously A[i] will be equal to A[j] and it prints '5' which is not a duplicate element.

5) PRINT_DUPLICATES(A)

```
for i=1 to A.length
    for j=i+1 to A.length
        if A[i] == A[j]
            PRINT(A[i]
```

→ This algorithm runs in $O(n^2)$ time because the inner loop runs for 'n' times and outer loop runs for 'n' times in worst case scenario.

Running times:

big-oh → $O(n^2)$

Theta → $\theta(n^2)$

omega → $\Omega(n^2)$

Even if the array is sorted, the inner for loop will run 'n' times & so does the outer loop.

6) - The algorithm to sum the items in the array is correct.

- The loop invariant would be that the "variable s contains sum of items in the array A[1..i]".
- This condition holds good during initialization, at the start of each iteration and also during termination.

7) a) GAUSSES_SUMMATION_SERIES_1

big-oh → $O(n^2)$

little-oh → $o(n^2)$

Theta → $\theta(n^2)$

omega → $\Omega(n^2)$

b) GAUSSES_SUMMATION_SERIES_2

big-oh → $O(n)$

little-oh → $o(n)$

Theta → $\Theta(n)$

omega → $\Omega(n)$

c) PRINT_REPEATED_ITEMS(A)

big-oh → $O(n^2)$

little-oh → $o(n^2)$

Theta → $\Theta(n^2)$

omega → $\Omega(n^2)$

8) - In insertion sort, the while loop lines searches backword in the sorted array and swap the element at each iteration of the loop if the condition is passed.

- If we use binary search instead of ~~log~~ linear search, the worst case time can be reduced to $O(\log n)$.

- But the binary search only finds the position in which the element should be placed. It should be followed by swapping elements next to it. For this we use a loop and it takes 'n' time.

- So we cannot use a binary search to improve worst-case running time of insertion sort to $O(n \log n)$.

9) Function is called with MAXIMUM(A, 1, 5)

output :

2, 1

3, 2

5, 4

5, 3

10) Define a problem where sorting the data plays a crucial role. write an algorithm to solve the problem and analyze the running time with sorted data and without sorting. Provide the running time of algorithm in big-oh, Theta and omega notation.