Design and Analysis of Algorithms

IGV19CS406
Tharun Kumar R

(1) a. Basic asymptotic notation

Ans :- the asymptotic efficiency analysis framework concentrates on the order of growth of an algorithm basic operation count as the principal indicator of an algorithm efficiency

to compare and rank such orders of growth computer scientists use 3 notations O (big-oh), Ω (big-omega) and Θ (big theta)

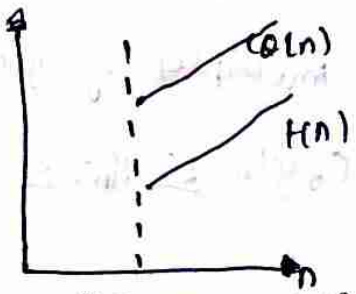
t(n) and g(n) can be any be any non-negative function defined on the set of natural numbers

t(n) → algorithm's running time
C(n) → basic operation count
g(n) → some simple function to compare with count

* O - definition (Big Ohm)

A function t(n) is said to be O(g(n)) denoted $t(n) \in O(g(n))$ if t(n) is bounded above by some constant multiple of g(n) for all large n, ie. If there exist some positive ~~concept~~ constant c and some non-negative integer n_0 such that

$t(n) \leq c\, g(n)$ for all $n \geq n_0$



Big-Oh notation
$t(n) \in O(g(n))$

Example :-

$100+5 \leq 100n + 5n \quad \forall \quad n \geq 1$

$100+5 \leq 105n$

$\downarrow \quad \downarrow \quad \downarrow$

$f(n) \quad c \quad g(n)$

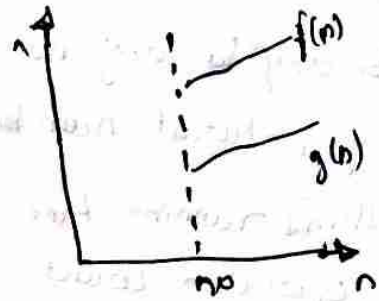
~~f(n) should be the~~

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

$\therefore 100n+5 \in O(n)$

* Ω - notation definition :

A function $f(n)$ is said to be $\Omega(g(n))$ denoted as $f(n) \in \Omega g(n)$ if $f(n)$



Big - Omega Notation

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

$\geq n^2 \quad \forall \quad n \geq 0$

$\downarrow$

$n) \quad \Omega(n) \quad c=1$

$f(n) \geq c \cdot g(n)$

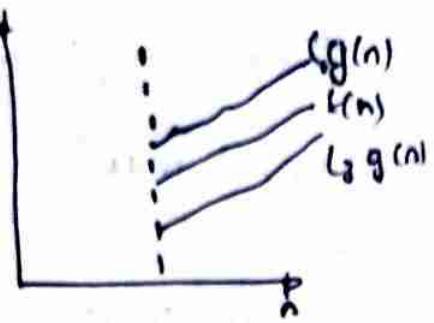
$\therefore n^3 \in \Omega(n^2)$

Θ notation definition

A function $f(n)$ is said to be in $\Theta(g(n))$, denoted $f(n) \in \Theta(g(n))$ if $f(n)$ is bounded above and between by some positive constant multiples of $g(n)$ for all large n

$$c_2 g(n) \leq f(n) \leq c_1 g(n)$$

(3)



$n^3 \geq n^2 + n \geq 0$

$\downarrow \quad \downarrow$

$f(n) \quad g(n) \quad c = 1$

$f(n) \geq c_1 g(n)$

$\boxed{n^3 \in \Omega(n^2)}$

(4)

the Analytic axiom for the Ω and Θ notations as well

the proof Extented to orders of growth the following simple facts about arbitrary real numbers a_1, b_1, a_2 and b_2

If $a_i \leq b_i$ and $a_2 \leq b_2$ then $a_1 + a_2 \leq 2 \max [b_1, b_2]$

Since $t_1(n) \in O(g_1(n))$

then there Exist some positive Constant c_1 and some non-negative Integers n_1 such that

$$t_1(n) \leq c_1 g_1(n) \quad \forall \quad n \geq n_1$$

Similarly

$$t_2(n) \leq c_2 g_2(n)$$

$$t_2(n) \leq c_2 \cdot g_2(n) \quad \forall \quad n \geq n_2$$

let us denote $c_3 = \max \{c_1, c_2\}$ and consider $n \geq \max \{n_1, n_2\}$ adders two Integers above yields the following

$$t_1(n) + t_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$$

$$\leq c_3 g_1(n) + c_3 g_2(n)$$

$$\leq c_3 [g_1(n) + g_2(n)]$$

$$\leq c_3 \cdot 2 \max [g_1(n), g_2(n)]$$

(4)

$$t_1(n) + t_2(n) \leq 2.(3.\max[g_1(n), g_2(n)]$$

$$\therefore \boxed{t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})}$$

(1c) algorithm :-

Sequencial Search :-

Sequencial Search for a given value in a given array by Sequential Sear

```
i ← 0
while i < n and A[i] ≠ K do
    i ← i+1
If (i < n return i)
Eleye return -1;
```

Worst case Analysis :-

~~Algori~~ where case analysis (usually done) : In the worst case we calculate upper ~~case~~ bound on running time of an algorithm we must know the the case that causes maximum number option to be executed for linear search the worst case happens when the element to be searched is not present In the array when k is not present in the sequencial search function compare it with all the element of A[] are by one therefore the worst case time complexity of linear search would be $O(n)$

Average case Analysis :- In average case analysis we have to take all possible Input and calculate computing time for all the Input Sum all the calculated values and Divide the Sum by total Number of Input let us assume that all cases are uniformly distributed (Including the case for x not being played in array) so we Sum

case and Divide the Sum by (n+1) following the rule of average case time complexity

$$\text{Average case time} = \frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)}$$

$$= \frac{\theta((n+1) * (n+2)/2)}{(n+1)}$$

$$= \theta(n)$$

Best case Analysis :- In the best case analysis we calculated lower bound on running time of be Execute In the linear search problem the best case occurs when x is present at the first location the number of operat In the best case is constant most of the time we do worst case analysis to analyse algorithm In the worst analysis we guarantee an upper bound on the running time of an algorithm which is good Informal and the average case analysis Is not Easy to do In ~~the~~ mathematical distribution of all possible Inputs.

Part = B

(3.b) Recursive algorithm for binary Search

If n=1 return 1

Else return BinRec(n/2) + 1

$A(n) = A(n/2) + 1$ for $n > 1$

(6)

Since the recursive calls End when n is Equal to 1 and there are no additions made the Initial condition $A(1) = 0$

$A(2^k) = A(2^k - 1) + 1$ for $k > 0$

$A(2^0) = 0$

$A(2^k) = A(2^k - 1) + 1$ Substituted $A(2^k - 1) = A(2^k - 2) + 1$

$= [A(2^k - 2) + 1] + 1 = A(2^k - 2) + 2$ Substituted $A(2^k - 2)$

$= A(2^k - 3) + 1$

$= [A(2^k - 3) + 1] + 2 = A(2^k - 3) + 3$

$= A(2^k - i) + i$

$= A(2^k - k) + k$

where.

$A(2^k) = A(1) + k = k$ or after returning the orginal variable $n = 2^k$ and hence $k = \log_2 n$.

$A(n) = \log^2 n \in$ theta $(\log n)$

Worst case:- In algorithm efficient Input of size n which is the Input of size n for which algorithm runs the longest among all possible Input of that size.

Case value $C_{worst}(n)$ for given example

$C(n) = 1$

Best case :- In algorithm efficient Input of size n for an operation the algorithm runs the fastest among all possible Input of the size.

$C(n) = O(n)$

(4)

Average case :- the algorithm input and output Response either Best case or worst case And Hence the algorithm may Response either faster nor Slower in Average case Efficiency

a) the probability of a successful search is equal to $p (0 \le p \le 1)$

b) the probability of first occurs in the position of the list is the same you very

$$C_{avg}(n) = \left[1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + n \cdot \frac{p}{n} \right] + n(1-p)$$

$$= \frac{p}{n} [1 + 2 + 3 + \cdots + n] + n(1-p)$$

$$= \frac{p}{n} \cdot \frac{n(n+1)}{2} + n(1-p) = \frac{p(n+1)}{2} + n(1-p)$$

Part C

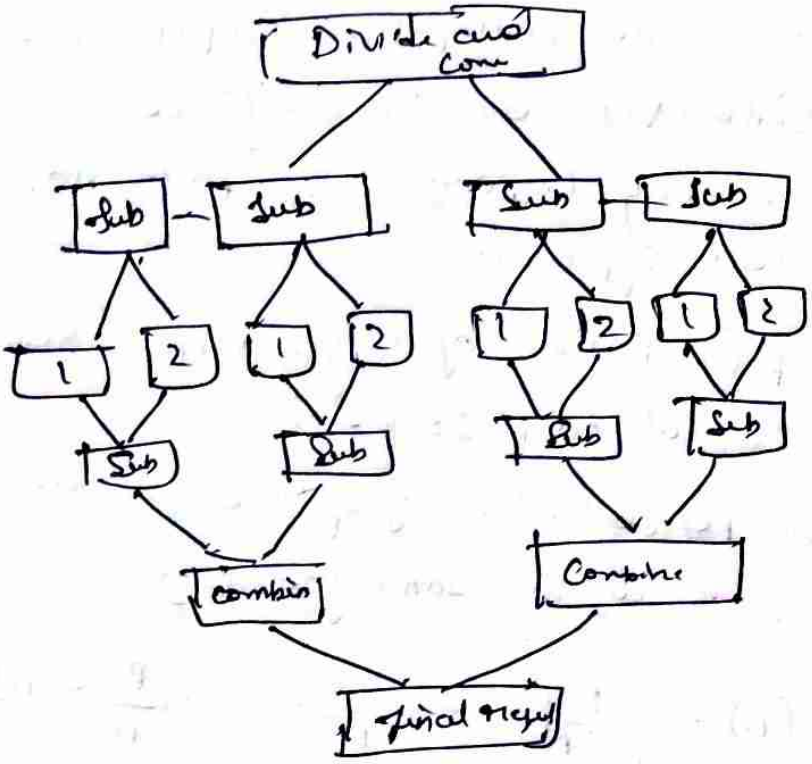
(5) Divide and conquer method

*) A problem is divided into several sub problems

*) Each subproblem are solved (typically recursively)

*) If necessary solution to the subproblems are combined together and solution to the orginal problem

(8)




```
algorithm maximum (int arr[], int i, int len)
{ If (i >= len - 2)
    {
      if (arr[i] > arr[i+1]
        return arr[i]
            Else
              return arr[i+1]
    }
max = maximum (arr, i+1, len)
{ If (arr[i] > max) return arr[i]
    Else
        return arr[i+1]
}
    max = maximum(arr, i+1, len);
```

(4)

```
If (i. >= len - 2)
{
   If (arr[i] < arr[i+1])
      return arr[i]
   Else
      return arr[i+1];
   min = minimum(arr, i+1, len)
If (arr[i] < min)
  return arr[i];
  Else
     return min;
}
```