02/03/22.

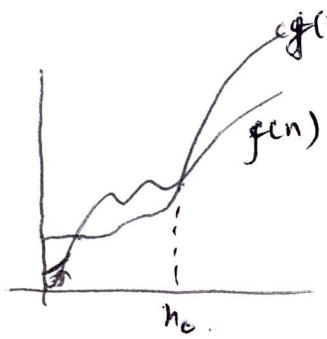
1st Sessional

Design & Analysis of Algorithms.

YASH VINAYVANSHI
19BCS081.

Q1. (i) O Notation.

→ asymptotic upper bound ⇝ binds a function from above.

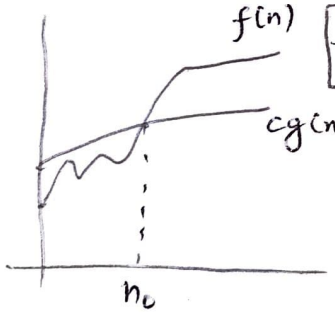


$$O(g(n)) = \{ f(n) : \exists\ c, n_0 > 0 \text{ s.t } 0 \le f(n) \le cg(n)\ \forall\ n \ge n_0 \}$$

→ Since O Notation describes the upper bound, It is used to bound the worst case running time of an algorithm.

(ii) Ω Notation

→ Asymptotic lower bound ⇝ bind a function from below.



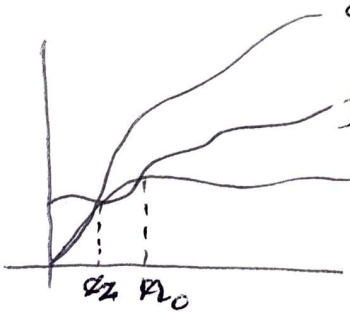
$$\Omega(g(n)) = \{ f(n) : \exists\ c, n_0 > 0 \text{ s.t } 0 \le cg(n) \le f(n)\ \forall\ n \ge n_0 \}$$

→ It is used to bound the best case running time of an algorithm.

(iii) Θ Notation

→ Asymptotically binds a function from above & below : Sandwiching.

$$\Theta(g(n)) = \{ f(n) : \exists\ c_1, c_2, n_0 > 0 \text{ s.t } 0 \le c_1 g(n) \le f(n) \le c_2 g(n)\ \forall\ n \ge n_0 \}$$



→ It is used to bound the average case running time of an algorithm.

Q1 i) $f(n) = 3 \cdot 2^n + 7n^2 + 8n + 9$

2^n is exponential term & thus dominates all polynomial terms of type n^k $k \geq 1$

$\therefore \boxed{f(n) \in O(2^n)}$

ie $f(n) \leq c\, 2^n$ for $c \geq 3$, $n_0 > 1$.

Since recurrence relation doesn't satisfy

$\therefore \boxed{f(n) \in \Omega(2^n)}$.

& since $f(n) = O(2^n)$ & $\Omega(2^n)$.

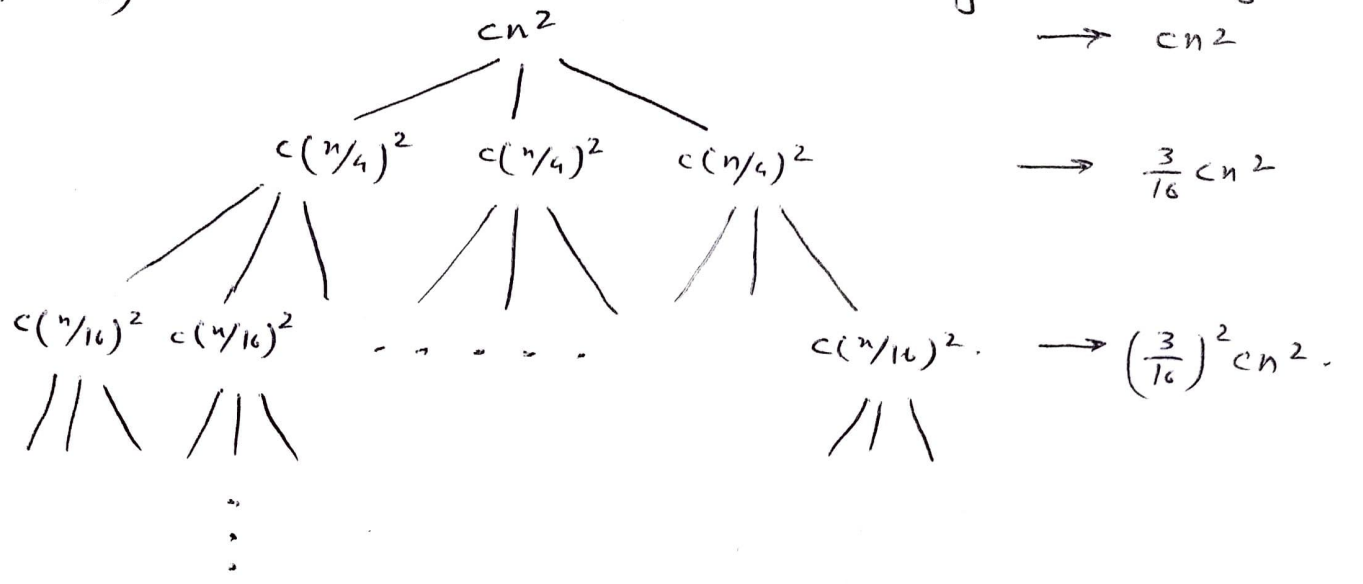
$\therefore \boxed{f(n) = \Theta(2^n)}$

(ii) $f(n) = 3n^2 + \log n$.

$$\boxed{\begin{aligned} f(n) &= O(n^2) \\ f(n) &= \Omega(n^2) \\ f(n) &= \Theta(n^2) \end{aligned}}$$

Due to same reasons as in (i). Polynomial terms asymptotically dominate logarithmic terms.

Q2 b) Assuming n is power of 4



height of tree : $\frac{n}{4^h} = 1 \rightarrow h = \log_4 n$

width of tree : $3^h = 3^{\log_4 n} = n^{\log_4 3}$

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \dots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \qquad \left[\because \sum_{i=0}^{\infty} a^i = \frac{1}{1-a}\right]$$

∞ GP summation.

$$= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{16-3} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$

verifying result by substitution.

1. Induction hypothesis : $T(n) \leq cn^2$
2. Assume $T(m) \leq cm^2$ holds $\forall\ m < n$.

① $m = n/4 \rightsquigarrow T(n/4) \leq c(n/4)^2$

③ proving it holds for n

$$T(n) = 3T(n/4) + cn^2$$
$$\leq 3d(n/4)^2 + cn^2$$
$$= \frac{3}{16}dn^2 + cn^2$$
$$\leq dn^2 \quad \text{if } d \geq \frac{16}{13}c.$$

$\therefore T(n) = O(n^2)$. — ①

Also, since the tree is complete,

$T(n) = \Omega(n^2)$ — ②

using ① & ②,

$$\boxed{T(n) = \Theta(n^2)}$$

Q2 c) $T(n) = T(n-1) + 1$. Assuming $T(1) = 1$.

using iteration method,

$$T(n) = T(n-1) + 1.$$
$$= (T(n-2) + 1) + 1$$
$$= T(n-2) + 1 + 1$$
$$= (T(n-3) + 1) + 1 + 1$$
$$= T(n-3) + 1 + 1 + 1$$
$$\vdots$$
$$= T(n-k) + k.$$

for $k = n-1$

$$T(n) = T(n-(n-1)) + n-1$$
$$= T(1) + n - 1$$
$$= 1 + n - 1$$
$$= n$$
$$= O(n)$$

Also, since the recursion tree is linear,

$T(n) = \Omega(n)$

$$\therefore \boxed{T(n) = \Theta(n)}$$

Q2 a) $T(n) = 2T(\sqrt{n}) + n$.

using change of variables.

$m = \log n \rightsquigarrow n = 2^m$.

$T(2^m) = 2T(2^{m/2}) + 2^m$

Let $T(2^m) = S^m$

$S(m) = 2T(2m/2) + m$.

using Master's ~~Theorem~~ Methods

$T(n) = aT(n/b) + f(n)$.

$a = 2, \; b = 2, \; f(m) = m$.

$n^{\log_b a} = n^{\log_2 2} = n^1$

$\therefore f(n) = \Theta(n^{\log_b a - \varepsilon}) \rightsquigarrow n = \Theta(n^{1-\varepsilon})$

$\rightsquigarrow \boxed{\varepsilon = 0}$

$\therefore$ <u>case 2</u>

if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

$\therefore S(m) = O(m \log m)$

putting back $m = \log n$

$= \boxed{O(\log n \log \log n)}$

Q3. (i) Merge Sort.

```
MERGE-SORT (A, P, r) {
    if P < r
        q = ⌊(P+r)/2⌋
        MERGE-SORT (A, P, q)
        MERGE-SORT (A, q+1, r)
        MERGE (A, P, q, r)
}
```

Divide : O(1)

Divide : To compute middle of subarray. Takes constant time as it takes constant primitive operations.

$$D(n) = c$$

Conquer : Recursively solve two subproblems each of size $n/2$ → contributes $T(n/2)$ time each.

$$2T(n/2)$$

Combine : MERGE procedure on element subarray takes $\theta(n)$ time.

$$C(n) = \theta(n).$$

Divide & conquer recurrence relation.

$$T(n) = \begin{cases} \theta(1) & \text{if } n \le c. \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

↓

Merge sort recurrence relation

$$\boxed{T(n) = \begin{cases} \theta(1) & \text{if } n = 1 \\ 2T(n/2) + \theta(n) & \text{if } n > 1 \end{cases}}$$

(ii) Solving Recurrence relⁿ using masters method.

$$T(n) = 2T(n/2) + cn$$

$$T(n) = aT(n/b) + f(n).$$

$a = 2, \; b = 2, \; f(n) = n.$

$$n^{\log_b a} = n^{\log_2 2} = n^1$$

$$f(n) = \theta(n^{\log_b a - \varepsilon}) \rightarrow n = \theta(n^{1-\varepsilon}) \rightarrow \varepsilon = 0$$

∴ <u>case 2</u>

if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

$$\therefore T(n) = \Theta(n \log n)$$

Q4. Binary search Recurrence equation.

```
BinSea (A, l, r, key) {
    if (l > r) return;
    mid = l+r/2
    if A[mid] == key
        return mid;
    if (A[mid] < key)
        return BinSea (A, l, mid, key);
    return BinSea (A, mid+1, r, key);
}
```

→ we match the key with middle element of sorted array A, if it matches → we found the key → return mid index

→ If key < A[mid] → the key shall exist in first half of array, otherwise in second half of array

→ with each recursive call, we reduce the search space by half.

∴ Divide : To compare A[mid] with key → $O(1)$
Conquer : In each iteration, search space halves → $T(n/2)$
Combine : No combination → 0 cost.

∴ recurrence relation for binary search is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(n/2) + \Theta(1) & \text{otherwise} \end{cases}$$

→ $\boxed{T(n) = \Theta(\log n)}$