Sir No → 9045852502
Kavinder@85@gmail.com .

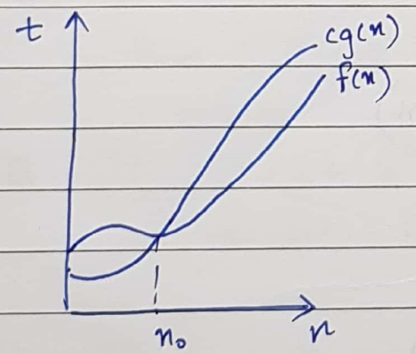
Algorithm Design And Analysis (ADA)

→ ① Complexity Analysis .
② Searching and Sorting (Divide and Conquer)
③ Dynamic Programming (0-1 Knapsack) .
Greedy Algos .
Computational Complexity .

→ Most efficient algo is one which takes less time and less space .

Time Complexity → Asymptotic Notation
① Big Oh Notation . (O) → Worst
② Big Omega Notation (Ω) → Best
③ Theta Notation (θ) → Average .

Concept of Algorithm Efficiency .
↳ Time Complexity
↳ Space Complexity .



Input
↑
$f(n) \le c\,g(n)$ → Output .

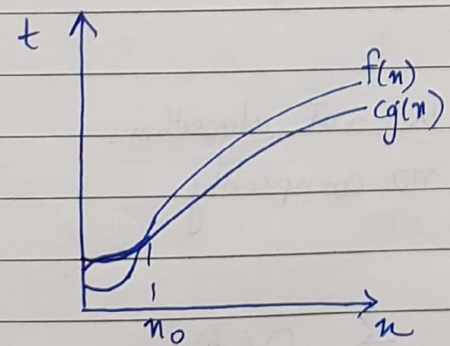
$n \geq n_0$

Eg ⇒ $f(n) = 3n+4$

$g(n) = n$.

$\cancel{3n+4 \le cn}$ $c=4$

$\Rightarrow \cancel{(c-3)n \geq 4}$ $n_0 = 4$



$f(n) \geq c\,g(n)$.

$n \geq n_0$

$f(n) = 3n+4$

$g(n) = n$.

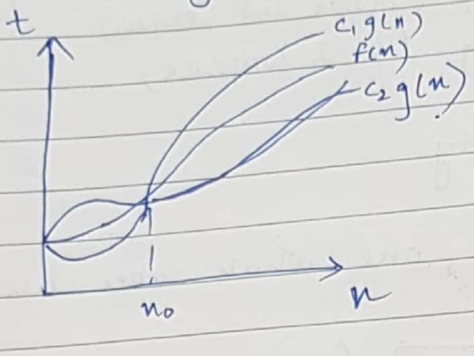
$c = 4$ $n_0 = 5$

$c = 3$ $n_0 = 1$

$O(n)$ $\Omega(n)$

- Theta Notation

→ $c_1 g(n) \le f(n) \le c_2 g(n)$



→ ALGORITHM

Iterative — Recursive.

Q.(1)
```
main()
{   while(n>1)
        n=n/2;
}
```
Time Complexity — $O(\log_2 n)$

Q.2)
```
while(n>1)
   n=n/3;
```
Time complexity = $O(\log_3 n)$.

Q.(3)
```
n=2.
while (n>1)      // Since it is not algorithm.
   n=n+1;           there is no complexity.
```

Q.4)
```
for(i=1; i<=n; i++)
   for(j=1; j<=i^2, j++)        → O(n^4).
      for(k=1; k<=n/2; k++)
         print "ABC";.
```

$n^{\log_4 3} = n^{0.8}$ $\frac{n(n+1)(2n+1)}{6}$ $n\log(n)$ $2^{2^3} = 2^{8}$ $n = 2$ $i = 1$ $j = 2$

⑦ for(i=1; i≤n; i++)
{ j=2;
while(j<=n)] k+1
j=j²;
}

2^{2^0} $2^{2^1} \to 2^{2^2} \to 2^{2^3} \to \cdots 2^{2^k}$

$n(k+1)$

$\cancel{\Theta(n\log n) = O(2^{2^k} \cdot 2^k) = O(2^{2^k+k})}$ $n = 2^{2^k} \Rightarrow k = \log_2 \log_2(n)$

$\boxed{O(n \log_2 \log_2(n))}$

→ Master's Theorem

- $1 < \log(\log n) < \sqrt{\log n} < \log n < \sqrt{n} < n < n\log n < n^2 < n^2\log n < n^3 < n^c < c^n < n^n < n!$

→ $n^{\log n} < 2^n < n^n$

→ $T(n) = a\,T(n/b) + f(n)$

Case I: $f(n) = O(n^{\log_b a - \epsilon})$ where $\epsilon > 0$
then $T(n) = \Theta(n^{\log_b a})$.

$T(n) = 8T\left(\frac{n}{2}\right) + n^3$

$n^{\log_2 8} = n^3$

$T(n) = \Theta(n^3 \log n)$

Case II: If $f(n) = \Theta(n^{\log_b a})$
then $T(n) = \Theta(n^{\log_b a} \lg n)$.

Case III: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ where $\epsilon > 0$.
then $T(n) = \Theta(f(n))$.

→ Eg $T(n) = 9T\left(\frac{n}{3}\right) + n$.
⇒
$a = 9$; $b = 3$; $f(n) = n$

$n^{\log_3 9} = n^2$.

$n^2 > n$ ⇒ Case I.

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

Eg $T(n) = T\left(\frac{2n}{3}\right) + 1$
⇒
$a = 1$; $b = 3/2$; $f(n) = 1$

$n^{\log_{3/2} 1} = 1$ ⇒ Case II.

So, $T(n) = \Theta(\log_2 n)$.

Eg $T(n) = 3T(n/4) + n\log(n)$
⇒ $a = 3$; $b = 4$; $f(n) = \log n$ $n^{\log_4 3} = n^{0.8}$. $n\log n > n^{0.8}$ Case (III
So, $T(n) = \Theta(n\log$

Page

Q. A(n)

```
A(n)
{
    if (n ≤ 1)
        return 1;
    else
        return [A(n/2) + A(n/2) + n];
}
```

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

$a = 2 \quad b = 2 \quad f(n) = n$.

$n^{\log_2 2} = n$

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

• ① Substitution Method.

→

Eg

$T(n) = T(n-1) + 1$ —①

$T(n-1) = T(n-2) + 1$.

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

Adding above eqn's.

$T(n) = T(n-3) + 3$

$T(n) = T(n-k) + k$

Boundary cond^n:

$T(n) = 1$ if $n = 1$.

• For $n-k = 1 \Rightarrow k = n-1$

$\Rightarrow T(n) = T(1) + n - 1$

$\Rightarrow \boxed{T(n) = O(n)}$.

Eg

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

$T(n-1) = T(n-2) + n - 1$

$T(n-2) = T(n-3) + n - 2$.

$T(n) =$ ~~[illegible]~~ $T(n-3) + 3(n-1)$

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

~~$T(n) = 2nT(n/2)$~~

n^2

For $n-k=1 \Rightarrow k=n-1$

$T(n) = T(1) + (n-1)^2$

$\Rightarrow T(n) = 1 + (n-1)^2$

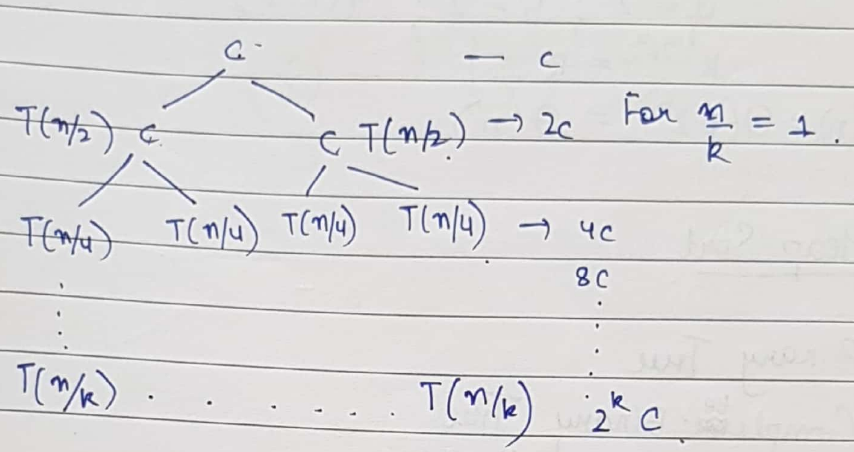
$\Rightarrow \boxed{T(n) = O(n^2)}$

② Recursive Tree Method

→

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

→

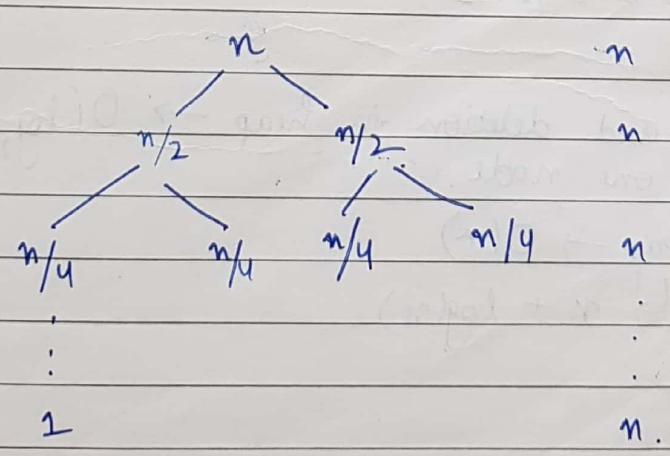


$T(n) = c(1 + 2 + 4 + \dots 2^k)$

$= c.(2^k - 1)$ $\qquad k = \log n$

$= c.(n-1) = O(n)$

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



$\boxed{T(n) = \theta(n \log n)}$

$= (k+1)n$

Scanned by CamScanner

$n^{\log_b a}$ $f(n)$. $O(f(n))$

① $T(n) = 2T(\sqrt{n}) + n$ $\longrightarrow$ $T(n) = O(\log n)$.

② $T(n) = 2T(\sqrt{n}) + C$.

① $T(n) = 2T(\sqrt{n}) + n$.

Let $n = 2^k$.

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

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

$S(k) = 2S(k/2) + 2^k$.

$a = 2$; $b = 2$; $f(k) = 2^k$.

$k^{\log_b a} = k$.

$T(n) = \Theta(2^k) = \Theta(n)$

- Heap Sort

Binary Tree

Complete Binary Tree

i level $\rightarrow$ $2^i - 1$.

① Min Heap

$\rightarrow$ Parent Node is always smaller than child node

② Max Heap

$\rightarrow$ Parent node is greater than child node.

$i = 1$ to n .

Left child $= 2 * i$

Right child $= 2 * i + 1$

Insertion and deletion in heap of one node. $\longrightarrow$ $O(\log_2 n)$

Build Heap $\rightarrow$ $O(n)$.

$\hookrightarrow$ $n + \log(n)$

$\lceil n/2 \rceil - 1$ 5 6 7 8 9 2 5

Q $T(n) = 2T(\sqrt{n}) + 1$

let $n = 2^k$

$T(2^k) = 2T(2^{k/2}) + 1$

let $T(2^k) = S(k)$

$S(k) = 2S\left(\frac{k}{2}\right) + 1$ → $k > 1$ $\Theta(k)$

$(2^k)^{\log_2 2} = 2^k = n$.

$k = \log_2 n$ → $\Theta(\log_2 n)$.

→ $T(n) = 7T\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right)^2 = 7T\left(\frac{n}{2}\right) + \frac{n^2}{4}$.

largest = i

l

r

$n/2 - 1$

→ **Heapify**

HeapSort(A)

1. Build max_heap(A)] $O(n)$
2. for i = A.length down to 2.
3. exchange A[1] with A[i].
4. A.heapsize = A.heapsize - 1;
5. Max_heapify(A, 1)] $O(\log 2)$.

$n^{2.81}$. Page

→ Strassen Matrix Multiplication

$$a_{ij} = \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}_{3\times3}}_{b} \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{c}$$

$$a_{ij} = \sum_{k=0}^{n} b_{ik}\, c_{kj}$$

Conditions :

(1) $n \times n$

(2) $n = 2^k$.

→ By divide and conquer.

$$T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

$$\Rightarrow \theta(n^3)$$

Cormen ⟶ Pg 80, 81

↳ $S_1 = B_{12} - B_{21}$

$S_2 = A_{11} + A_{12}$

$S_3 = A_{21} + A_{22}$.

$S_4 = B_{21} - B_{11}$

$S_5 = A_{11} + A_{22}$

→ $T(n) = 7T\left(\frac{n}{2}\right) + cn^2$.

$$T(n)$$

$$\underbrace{T\left(\frac{n}{2}\right) \quad T\left(\frac{n}{2}\right) \quad \cdots \quad T(n/2)}_{7 \text{ times}}$$

$$T(n) = n^2 + 7\left(T\left(\frac{n}{2}\right)\right)^2 + 7^2\left(T\left(\frac{n}{2}\right)\right)^2 + \ldots$$

→ Quick Sort

Worst Case – $O(n^2)$

↳ Array is sorted.

Randomized worst case – 1 1 1 1 1 (Same element)

→ Merge Sort:

$$T(n) = 2T\left(\frac{n}{2}\right) + n.$$

↳ (Also for best case quick sort).

→ Binary Search.

$$T(n) = T\left(\frac{n}{2}\right) + k.$$

•

```
main()
{
    i=25
    while(i<n)
    {
        i=i^20;
    }
}
```

$O(n) = \log_{20}\left(\log_{25} n\right)$.

$n^{1/2}, n^{1/4}, \ldots n^{1/2^k}$

$n^{1/2^k} = 1$

$\frac{1}{2^k} = \log_n 1 = \quad n^{\frac{1}{2^k}} = 1$

$\frac{1}{2^k} = 0$

$\Rightarrow 2^k = \frac{1}{\log_2 n}$

$\Rightarrow k = \log_2\left(\frac{1}{\log_2 n}\right)$

→ K(n)

```
{ for(i=0; i<n; i++);
      if(n>1)
          Return K(n/2) + K(n/2) + 1;
}
```

$T(n) - O(n \log n)$

Date
Page

$T(n) = T(\sqrt{n}) + n\log n$.

let $n = 2^k$.

$T(2^k) = T(2^{k/2}) + 2^k k$.

let $T(2^k) = S(k)$

$S(k) = S\left(\frac{k}{2}\right) + 2^k \cdot k$.

$n^{\log_2 1} = n^0 = 1$

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

Q $T(n) = T(n-1) + \log n$

$T(n-1) = T(n-2) + \log(n-1)$

$T(n-2) = T(n-3) + \log(n-3)$

$T(n) = T(n-3) + \log n(n-1)(n-2)$

$= T(n-k) + \log[n(n-1).(n-2)\ldots(n-k+1)]$

$n-k=1$.

$T(n) = T(1) + \log(n!) = \log(n^n) = \boxed{n\log n}$

Q $T(n) = \sqrt{n}\,T(\sqrt{n}) + n$.

Let $n = 2^k$.

$T(2^k) = 2^{k/2}\,T(2^{k/2}) + 2^k$.

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

$S(k) = 2^{k/2}\,S\left(\frac{k}{2}\right) + 2^k$.

$k^{\log_2 2^{k/2}} =$

$T(n) = \sqrt{n}\,T(\sqrt{n}) + n$

$\Rightarrow \frac{T(n)}{n} = \frac{T(\sqrt{n})\sqrt{n}}{\sqrt{n}.\sqrt{n}} + \frac{n}{n}$

$\Rightarrow \frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + 1$

let $n = 2^k$.

$\frac{T(2^k)}{2^k} = \frac{T(2^{k/2})}{2^{k/2}} + 1$

Let $S(k) = \frac{T(2^k)}{2^k}$.

$S(k) = S(k/2) + 1$

$k^{\log_2 1} = 1$

$\theta(k) = \log k = \log(\log n)$.

$\frac{T(n)}{n} = \theta(\log(\log n))$

$\Rightarrow T(n) = \theta(n\log(\log n))$

$\frac{n!}{n-k}$ $n \cdot n-1 \ldots n-k$

Greedy

→ Optimization Problems.

① MST ② Knapsack

→

③ Activity Selection ④ Huffman Coding.

→

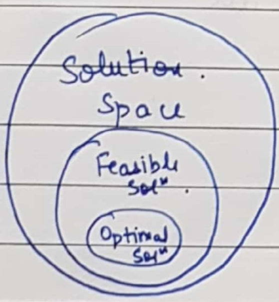
$T(n) = T(n-1) + \log n.$

$T(n-1) = T(n-2) + \log(n-1)$

- Greedy

Optimization Problem → Min cost.
→ Max profit.
→ Max reliability.
→ Min Risk.

→ Greedy Dynamic Programming.



Solution Space → Set of solutions

Feasible Space → Set of possible soln.

Optimal Space → Set of efficient soln.

→ Knapsack Problem

Greedy → Fractional.

Dynamic → 0/1

① Greedy abt weight → Select min weight

② Greedy abt profit → Select max profit

③ Greedy for per unit profit.
↳ Select max per unit profit first.

	01	02	03
P_i	25	24	15
W_i	18	15	10

Net profit $= 25 + \frac{24^{8}}{15_{5}} \times 2 = 25 + 3.2$

$= \boxed{28.2}$

30 . 15

$2^{2^k} = n$.

Divide and conquer algo.

① Divide
② Conquer
③ Combine

① Net profit = $15 + 10 \times \frac{24}{15}$ 8

$= \boxed{31}$

③ ~~Net profit~~

	01	02	03
$\frac{P_i}{W_i}$	1.4	1.6	1.5

$\frac{P_i}{W_i}$ should be max.

Net profit $= 24 + 5 \times 1.5$
$= 24 + 7.5$
$= \boxed{31.5}$

→ Algo:

① Calculate per unit profit for all
② Sort by descending.
③ for $i \leftarrow 1$ to n.
  $K \leftarrow K + P_i$
  $w = W - w_i$
  if $(w < W)$
   $K = K + w * \frac{P_i}{w_i}$

~~④ Return K~~

④ Return K.

→ Minimum Cost Spanning Tree

Huffman, MST, Knapsack, Activity — Shortest path ✗

Let a graph be have n vertices.

Max edges = $n(n-1)/2$

No. of simple graph = $2^{n(n-1)/2}$.

- Spanning Tree → A connected subgraph of a graph is spanning tree if

- n vertices in complete graph.

No. of spanning tree possible = $(n)^{n-2}$

① Kruskal Algo.

Step 1 : Sort weight of edges and create min heap.

Step 2 : Take min weight edge. If adding this edge does not form a cycle, then edge is included in MST. Else it is not added.

Step 3 : Repeat step 2 till no. of edges in MST become equa to $V-1$.

→ n_{max}

Q. When will edge with max weight included in MST?

Q. $T(n) = 4T(n/2) + n^2$.

$$T\left(\frac{n}{2}\right) = 4T(n/4) + \left(\frac{n}{2}\right)^2$$

$$T\left(\frac{n}{4}\right) = 4T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2$$

$$T(n) = 4\left[4T\left(\frac{n}{4}\right) + \frac{n^2}{4}\right] + n^2.$$

$$= 16\left[4T\left(\frac{n}{8}\right) + \frac{n^2}{16}\right] + 2n^2.$$

$$= 64T\left(\frac{n}{8}\right) + 3n^2$$

$$T(n) = 4^k T\left(\frac{n}{2^k}\right) + kn^2$$

$$\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n.$$

$$2^{2\log_2 n} = 2^{\log_2 n^2} = n^2$$

$$T(n) = 4^{\log_2 n} + n^2 \log n$$

$$T(n) = O(n^2 \log n)$$