Q. [Recurrence]

Q: Solve the Recurrence Relation $T(n) = 3T\left(\frac{n}{4}\right) + n$ using iteration (Substitution) method?

⇒ Given $T(n) = 3T\left(\frac{n}{4}\right) + n$ —(i)

$T\left(\frac{n}{4}\right) = 3 \cdot T\left(\frac{n}{4^2}\right) + \frac{n}{4}$ substitute back to (i)

$T(n) = 3 \cdot \left[3T\left(\frac{n}{4^2}\right) + \frac{n}{4}\right] + n$

$T(n) \Rightarrow 3^2 \cdot T\left(\frac{n}{4^2}\right) + n + 3\cdot\frac{n}{4}$ —(ii)

$T\left(\frac{n}{4^2}\right) = 3^2 \cdot T\left(\frac{n}{4^3}\right) + \frac{n}{4^2}$ — Put in eqn-(ii)

$T(n) = 3^2 \left[3T\left(\frac{n}{4^3}\right) + \frac{n}{4^2}\right] + n + \frac{n}{4}$

$T(n) \Rightarrow 3^3 \cdot T\left(\frac{n}{4^3}\right) + n + 3\frac{n}{4} + 3^2\frac{n}{4^2}$

⋮

Generalizing term for $n = K$ say

$T(n) = 3^K \cdot T\left(\frac{n}{4^K}\right) + n + 3\frac{n}{4} + 3^2\frac{n}{4^2} + \cdots + 3^{K-1}\frac{n}{4^{K-1}}$

$$\boxed{T(n) \Rightarrow 3^K \cdot T\left(\frac{n}{4^K}\right) + \sum_{i=0}^{K-1} n\left(\frac{3}{4}\right)^i} \quad \text{—(iii)}$$

Base condition—

$\frac{n}{4^K} = 1$ i.e. $n = 4^K$ or

$$\Rightarrow \boxed{\log_4 n = K}$$

⇒ Putting this value to eqn-③

$$T(n) = 3^{\log_4 n} T\left(\frac{n}{4^{\log_4 n}}\right) + \sum_{i=0}^{\log_4 n - 1} n\left(\frac{3}{4}\right)^i$$

$\frac{n}{4^{\log_4 n}} \Rightarrow n$

$$\Rightarrow 3^{\log_4 n} \underbrace{4T(1)}_{\Rightarrow 1} + \sum_{i=0}^{\log_4 n - 1} n\left(\frac{3}{4}\right)^i \qquad \left[\because 3^{\log_4 n} = n^{\log_4 3}\right]$$

$$\Rightarrow n^{\log_4 3} + \sum_{i=0}^{\log_4 n - 1} n\left(\frac{3}{4}\right)^i$$

$$T(n) = \Phi n^{\log_4 3} + n \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{4}\right)^i$$

decreasing GP

$S_n \Rightarrow \dfrac{1-r^n}{1-r}$

$$\Rightarrow n^{\log_4 3} + n \cdot \frac{1-\left(\frac{3}{4}\right)^{\log_4 n}}{1-\frac{3}{4}}$$

$$T(n) \Rightarrow n^{\log_4 3} + n \times 4 \left(1-\left(\frac{3}{4}\right)^{\log_4 n}\right)$$

for large value of n. 3ars dominate to $n^{\log_4 3}$

ie $T(n) = \Theta(n^{\log_4 3})$ $\Theta(n)$ Ans

Verify using master theorem

$a = 3$
$b = 4$
$k \geq 1$
$p \geq 0$

Case-3

$a > b^k$

$T(n) \Rightarrow n^k \log^p n$

$\Rightarrow O(n)$

Q-2 @ $T(n) = T(n-1) + n$ — (1) using iterative method.

$T(n-1) = T(n-1-1) + n-1$ Put back to Eq (1)

② $T(n) = T(n-2) + n-1 + n$ — (ii)

from (i) $T(n-2) = T(n-3) + n-2$ Put

③ $T(n) = T(n-3) + n-2 + n-1 + n$

⋮

for K term

$$T(n) = T(n-K) + n + n-1 + n-2 + \cdots + n-K-1$$

Base case $n-K=1$, $n = 1+K$ or $K = 1+n$

$$T(n) = T(1) + \sum_{i=0}^{K-1} (n-i)$$

$$T(n) = c + \sum_{i=0}^{n} (n-i) \quad [n + (n-1) + (n-2) + \cdots 2+1+0]$$

nothing but

$$\frac{n(n+1)}{2} \Rightarrow O(n^2)$$ Ans

③ $T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$ —(i)

$T\left(\frac{n}{2}\right) = T\left(\left\lceil \frac{n}{2^2} \right\rceil\right) + 1$

$T(n) = T\left(\left\lceil \frac{n}{2^2} \right\rceil\right) + 1 + 1$

generalizing for K term

$T(n) = T\left(\left\lceil \frac{n}{2^k} \right\rceil\right) + k$ —(ii)

$\frac{n}{2^k} = 1$ base case

$n = 2^k$

$k = \log_2 n$ Put in eq-(ii)

$T(n) = T(1) + \log_2 n$

$T(n) = O(\log_2 n)$ Ans

④ $T(n) = 2T\left(\frac{n}{2}\right) + n$ —(i)

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right) + \frac{n}{2}$ Put in eq-(i)

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

$T(n) = 2^2 \cdot T\left(\frac{n}{2^2}\right) + 2n$ —(ii)

from eq-(i)

$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}$ Put in eq-(ii)

for

$T(n) \Rightarrow 2^2\left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right] + 2n$

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

for K term

$T(n) \Rightarrow 2^k T\left(\frac{n}{2^k}\right) + kn$ —(iii)

Base case -

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

$n = 2^k$

$k = \log_2 n$

Eq-3 becom

$T(n) = 2^{\log_2 n} T(1) + \log_2 n \times n$

$T(n) \Rightarrow \underbrace{n^{\log_2 2}}_{(1)} + n \log n$

Domenating factor

$T(n) \Rightarrow O(n \log n)$ Ans

Que $T(n) = T(\sqrt{n}) + \log n$ $n \geq 2$ —(i)

let $n = 2^m$ then $\sqrt{n} = 2^{\frac{m}{2}}$

eq-(i) beam.

$T(2^m) = T(2^{m/2}) + \log_2 2^m$

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

$S(m) = S\left(\frac{m}{2}\right) + m$ —(iii)

$S\left(\frac{m}{2}\right) = S\left(\frac{m}{2^2}\right) + \frac{m}{2}$ Putting

$S(m) = S\left(\frac{m}{2^2}\right) + \frac{m}{2} + m$ —(iv)

from eq-(iii)

$S\left(\frac{m}{2^2}\right) = S\left(\frac{m}{2^3}\right) + \frac{m}{2^2}$ Put in eq-(iv)

$S(m) = S\left(\frac{m}{2^3}\right) + \frac{m}{2^2} + \frac{m}{2} + m$

generalizing for K-term

$S(m) S(m) = S\left(\frac{m}{2^k}\right) + m + \frac{m}{2} + \frac{m}{2^{k-1}}$

$S(m) = S\left(\frac{m}{2^k}\right) + m\left[\frac{1}{2^0} + \frac{1}{2} + \frac{1}{2^2} + \cdots \frac{1}{2^{k-1}}\right] \Rightarrow \frac{1}{1-\frac{1}{2}} \quad \frac{1}{\frac{1}{2}}$ ——(2)

Decreasing GP with $r = \frac{1}{2}$

Base case- $\frac{m}{2^k} = 1$

ie $m = 2^k$, $k = \log_2 m$ $\quad \therefore n = 2^m$

$\log_2 n = m$

$S(m) = S(1) + 2 \cdot m$

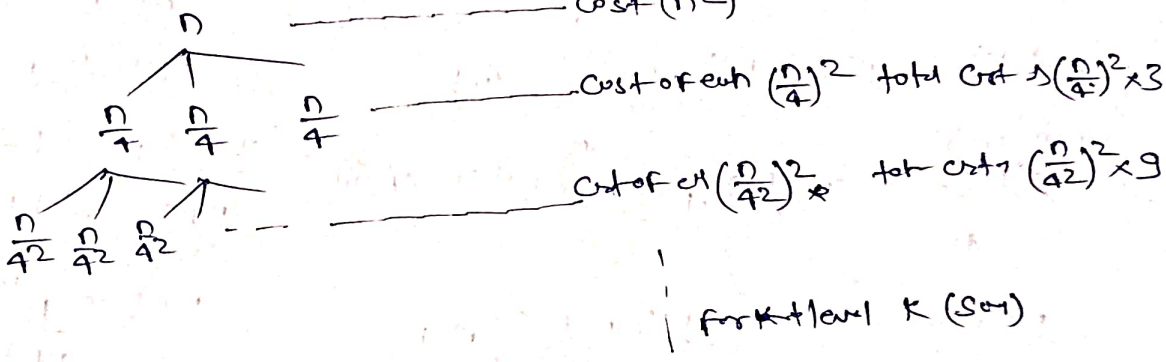
Now $S(m) = \theta(m) = T(n) = 2 \cdot \log_2 n$

ie $T(n) = \log_2 n$ Ans

Tree-method

↳ $T(n) = 3T\left(\frac{n}{4}\right) + cn^2$ Solve $n \to n^2$

Level-0



Cost →

$T(n) = n^2 + 3 \cdot \frac{n^2}{16} + 3^2 \cdot \frac{n^2}{16^2} + \cdots$

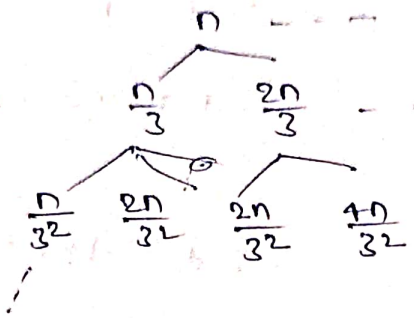
$T(n) = n^2 \sum_{K=0}^{\infty} \left(\frac{3}{16}\right)^K$ ——(i)

$\sum_{K=0}^{\infty} \left(\frac{3}{16}\right)^K$ Decreasing GP sum → $\frac{1}{1-\frac{3}{16}} \to \frac{16}{13}$

now $T(n) \to \theta(n^2) \frac{16}{13}$

ie $\theta(n^2)$ Ans

Qu: $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$



Cost → n

Cost → $\frac{n}{3} + \frac{2n}{3}$ → n

Cost → $\frac{n+2n+2n+4n}{3^2}$ → $\frac{9n}{9}$ → n

for K term

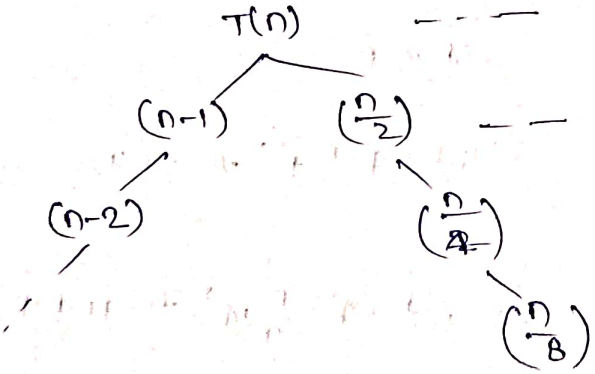
Cost → n (only)

Depth of recursion tree depend how many time we can divide n until we reach to base case -

$\frac{n}{3^K} = 1$ ie $n = 3^K$ OR $\boxed{K = \log_3 n}$

∴ Since we have $\log_3 n$ level and each level has cost n then time complexity becomes $O(n \cdot \log_3 n)$ Ans

Qu: $T(n) = T(n-1) + T\left(\frac{n}{2}\right) + n$



root has cost n.

Cost $n-1+\frac{n}{2}$

Height of tree blc of left subtree → $n + (n-1) + (n-2) + \cdots$

→ $\frac{n(n+1)}{2}$ → $O(n^2)$

Right subtree give $n + \frac{n}{2} + \frac{n}{4} + \cdots$ &

$n\left[1 + \frac{1}{2} + \frac{1}{4} + \cdots\right]$

decreasing GP

$O(n)$ × $\left(\frac{1}{1-\frac{1}{2}}\right)$ constant

total complexity

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

Ans

Qu (1.1.7) $T(n) = 4T\left(\frac{n}{2}\right) + cn$

$T(n)$ — — — Cost cn

$4T\left(\frac{n}{2}\right)$ — — — Cost $4 \times \frac{cn}{2} \Rightarrow 2cn$

$4 \times 4 T\left(\frac{n}{4}\right)$ — — Cost $\Rightarrow 4 \times 4 \times \frac{cn}{4} \Rightarrow 4cn$

for K term

$4^K \times \frac{cn}{2^K}$

$\Rightarrow cn\left(\frac{4}{2}\right)^K$

$\Rightarrow cn \cdot 2^K$

$\Rightarrow$ Problem end when $\frac{n}{2^K} = 1$ ie $T(1)$

ie $\boxed{n = 2^K}$

total time $c \cdot n \cdot n$

$\Rightarrow O(n^2)$

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

$T(n)$ — — — Cost $\Rightarrow n$

$4T\left(\frac{n}{2}+2\right)$ — — — Cost $\Rightarrow 4\left(\frac{n}{2}+2\right) \Rightarrow 2n+8$

$4^2 T\left(\frac{n}{4}+4\right)$ — — — Cost $4^2\left(\frac{n}{4}+4\right) \Rightarrow 4n+64$

$4^3 T\left(\frac{n}{8}+6\right)$ — — Cost $4^3\left(\frac{n}{8}+6\right) \Rightarrow 8n+512$

for K term

Height become 1 when $\frac{n}{2^K}+2K = 1$

$\Rightarrow$ ignoring $+2K$

$\frac{n}{2^K} = 1$

$n = 2^K \longrightarrow$

$4^K\left(\frac{n}{2^K}+2K\right) \Rightarrow n \times 2^K + 4^K \times 2K$

$n \times (1+2+4+\cdots) +$ minor

$O(n^2)$ Ans

[Divide and conquer]

Q Given a set of n integer, and a value K, find the Kth smallest element? (DC-1)

⇒ 1. Find the Pseudomedian

① Divide the input array (say X) into group of 5.

② Find median of each grp

③ Use the select function recursively to find median of median, ie Pseudo median m.

Select (X, K) [Partitioned array based on m]

$U = \{x \in X \mid x < m\}$

$V = \{x \in X \mid x = m\}$

$W = \{x \in X \mid x > m\}$

2. Now check the size of U and Recurr-

if $|U| = K-1$, then m is Kth smallest element

if $|U| > K$, then desired element is in U, do Rec on U

otherwise $|U| < K$ the desired element will be in W. Rec on W.

Recurrence Rel^n ⇒

$$\boxed{T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + O(n)}$$ [Base case $T(6) = O(1)$]

where $T\left(\frac{n}{5}\right)$ is time taken to find median of median

$T\left(\frac{3n}{4}\right)$ represent the time to recursively find the Kth element in one of the Partition.

$O(n)$ represent time to Partition array.

Solve the Recurrence Rel^n using tree method you will get $\Theta(T(n) = O(n))$

Q: Consider the Previous Problem, Can we divide the set into subset of size 7 instead of 5, as done. Prove or disprove $|U| \leq \frac{3n}{7}$ in this case also?

⇒ - when we divide X into n of size 7 there will be $\frac{n}{7}$ groups.

- for each grp we need to find median which will be 4th element when grp is sorted.

- find Pseudomedian m

⇒ Partition of original array by using pseudo median into three part

U: Element less than m

V: element Equal to m

W: Element greater than m

Analysis

∵ Each grp of 7 has 3 element that are lesser than median 3 are greater than median where 4th will be median itself.

∴ out of $\frac{n}{7}$ grp $\frac{n}{7} \cdot 3 = \frac{3n}{7}$ element will be lesser than Pseudo median. Also $\frac{3n}{7}$ will greater than Pseudo-median

The key is that Pseudo median divide the remaining element fairly evenly.

we know - $|U| + |V| + |W| = n$

or $\boxed{|U| \leq n - |W| - |V|}$ —(i)

Assume worst-case

$$|U| + |W| \geq \frac{3n}{7} + \frac{3n}{7}$$

$$|U| + |W| \Rightarrow \frac{6n}{7}$$

thus Eqn.(1) becom

$$|U| \leq n - \frac{6n}{7}$$

$$\boxed{|U| \leq \frac{n}{7}}$$

⇒ this show that size of largest Partition $|U|$ can be ~~exceed~~ less than $\frac{3n}{7}$ while Divide in grp of 7

⇒ $\boxed{|U| \text{ is Bounded}}$

$\frac{1}{7} \Rightarrow 0.14$ | $\frac{3}{7} \Rightarrow 0.75$

Qw2 Compute $x = y \cdot z$, each of n bit,

Karatsuba Algorithm (कराटसुबा)

1. Divide the each n-bit number into two halve

for example-

$$y = y_1 \cdot 2^{n/2} + y_0$$
$$z = z_1 \cdot 2^{n/2} + z_0$$

where y_1 and y_0 are the upper and lower halve of y, similer to z also.

2. Conqure to compute $y \cdot z$ we can expend it as

$$y \cdot z = (y_1 \cdot 2^{n/2} + y_0)(z_1 \cdot 2^{n/2} + z_0)$$

$$\Rightarrow y_1 z_1 \cdot 2^n + (y_1 z_0 + y_0 z_1) \cdot 2^{n/2} + y_0 z_0$$

Insted of computing product seprately, Karatsuba compute three product

$$\left.\begin{aligned} A &= y_1 \cdot z_1 \\ B &= y_0 \cdot z_0 \\ C &= (y_1 + y_0) \cdot (z_1 + z_0) \end{aligned}\right] \text{3 time}$$

then compute: $D = C - A - B$

3. Combin

$$y \cdot z = A \cdot 2^n + D \cdot 2^{n/2} + B = x$$

Time complexity

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

Here, $T\left(\frac{n}{2}\right)$ represent the time to multiply the half-size number

By master theorem

$O(n)$ represent time for addn and substn

$$\Rightarrow T(n) = O(n^{\log_2 3})$$
$$\Rightarrow \boxed{O(n^{1.585})}$$

Example of Karatsuba Algorithm

Let x = 1234

z = 5678

1. split the number-

x = 1234

$x_1 10^{n/2} + x_0$ where

$x_1 = 12$, $x_0 = 34$

$12 \times 10^2 + 34$

z = 5678

$z_1 10^2 + z_0$

$z_1 \Rightarrow 56$

$z_0 \Rightarrow 78$

∵ Here we have used $10^{n/2}$ - base instead of $2^{n/2}$ b/c of number of base -10

Calculating A, B, C

A = $x_1 z_1$

$\Rightarrow 12 \times 56 \Rightarrow 672$

B = $x_0 z_0$

34×78

$\Rightarrow 2652$

C $\Rightarrow (x_0 + x_1)(z_0 + z_1)$

$46 \cdot 134$

$\Rightarrow$ ~~180~~ 6164

Compute D $\Rightarrow$ C - A - B

~~180~~ 2840

How combine- x·z $\Rightarrow$ $A \cdot 10^n + (C - A - B) 10^{n/2} + B$

$672 \times 10^4 + 2840 \times 10^2 + 2652$

$6720000 + 284000 + 2652$

7002652 Ans

Matrix multiplication

↳ For two matrix A and B, both of nxn, the Product matrix C = AxB is computed using formula

$$C[i][j] = \sum_{k=1}^{n} A[i][k] \times B[k][j]$$

this requir O(n3) operation bic each element C[i][j] requir O(n) multiplication, and there are n^2 element in resultant matrix C.

Divide and Conqure Approch (Strassen's Algo) (स्ट्रेसेन)

1. Divide the matrix

Given two matrix nxn, A, B divide each matrix into 4, $\frac{n}{2} \times \frac{n}{2}$ Submatrix.

for example-

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Here $A_{11}, A_{12}, A_{21}, A_{22}$ and $B_{11}, B_{12}, B_{21}, B_{22}$ are each $\frac{n}{2} \times \frac{n}{2}$ matr

2. Compute intermediate matrix

using the submatrix, compute 7 intermediate matrix m_1 to m_7 by following way-

$m_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$m_2 = (A_{21} + A_{22}) B_{11}$

$m_3 = A_{11} (B_{12} - B_{22})$

$m_4 = A_{22} (B_{21} - B_{11})$

$m_5 = (A_{11} + A_{21}) B_{22}$

$m_6 = (A_{21} - A_{11})(B_{11} + B_{12})$

$m_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

this require only 7 multiplication of $\frac{n}{2} \times \frac{n}{2}$ matrix, rather than B.

3. Combin result-

$C_{11} = m_1 + m_4 - m_5 + m_7$

$C_{12} = m_3 + m_5$

$C_{21} = m_2 + m_4$

$C_{22} = m_1 - m_2 + m_3 + m_6$

matrix C ⇒ $\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

time complexity:

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

solving using master theorem

$$T(n) = O(n^{\log_2 7}) \approx \underline{O(n^{2.81})} \text{ Ans}$$

Example:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}_{4\times4} \qquad B = \begin{bmatrix} 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 \end{bmatrix}$$

Step-1 we divide 4×4 matrix into 2×2

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

where $A_{11} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $B_{12} = \begin{bmatrix} 19 & 20 \\ 29 & 24 \end{bmatrix}$ and so on

Step-2 Compute the 7 Strases product:—

$m_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$

$m_2 = (A_{21} + A_{22}) \times B_{11}$

$m_3 = A_{11} \times (B_{12} - B_{22})$

$m_4 \Rightarrow A_{22} \times (B_{21} - B_{11})$

$m_5 \Rightarrow (A_{11} + A_{12}) \times B_{22}$

$m_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$

$m_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$

Step-3 combin

$C_{11} = m_1 + m_4 - m_5 + m_7$

$C_{12} = m_3 + m_5$

$C_{21} = m_2 + m_4$

$C_{22} = m_1 - m_2 + m_3 + m_6$

[Greedy Algorithem]

- 0/1 Knapsack [Scheduling Problem]

Job Scheduling Problem with deadline

- Assumption:
  - Let n be the number of job
  - each job i has profit P_i and deadline d_i
  - All job take unit time and we have single Processor to execute
  - ~~Greedy Algorithem:~~
    You have to Prof greedy Preform better than any other Algo.

Greedy: Sort the job by profit and assign to them least avilible time slot.

Optimal soln: Some solution S that we assuming give higher profit than Greedy, we aim to prove that this assumption lead to contradiction.

[Prove based on induction]

- To Prove that greedy algorithem is optimal we prof by contradiction we assume that them is a Schedul S that produce higher profit than Greedy G. we than show that this lead to a contradiction, providing that greedy algorithem is indeed optimal.

Lemma 1: Job execution and replacement

Suppose a Job J does not apper in G, if we can show that from every time interval from [1 to d_j] there is a job with profit higher than P_j scheduled in G, then J can not be included in a better scheduled S without decreasing the total profit of S.

Proof of Lemma 1

- Suppose Job J does not appear in the greedy scheduled G_r.

→ By definition of greedy algorithem, every job J' scheduled in G_r upto the deadline d_j must have the Profit greater than or Equal to P_j.

→ Now to include J in an alternative scheduled S, you have to displace one or more of jobs that are currently in G_r.
But bic of each job has profit grater than or Equal to P_j, replacing them with J will result in a reduction of total Profit.
therefor any Schedule S that include J while excluding jobs with higher Profit can't exceed the Profit of greedy Schedule G_r.

Lemma 2: Suppose the total Profit of an alternative scheduled S is greater than that of greedy schedule G_r and there exist a job J with same profit as some job in G_r. If J is scheduled at a time slot that could have been used by the job with same profit in G_r, then S can't have higher Profit than G_r unless J replace a job with lower Profit.

Proof:
Suppose J and J' have same profit
Case-1: If J is place in the slot where J' was scheduled, and both Job has same Profit then replacing J with J' does not affect the Profit.

Remaining Jobs in S must have been scheduled in slots that could potentially be occopied by jobs in G_r.
for S to achive a higher Profit, J must have replace a job with a lower Profit in G_r. other wise the total Profit can't be exceed G_r.

Theorum: Any greedy scheduled strategy will finish the job in time of most $(\lceil \frac{n}{p} \rceil + h)_{max}$ where n → num of job, p → num of Processor

h → height of the longest dependency chain (Path) in job dependency graph.

Proof: Proof involves understanding two main factor -

1. Parallelization (Load Balancing) ie n/p
2. Dependency chain (Height of graph)

Here we need to show that time required by any greedy strategy is bounded to $\lceil \frac{n}{p} \rceil + h$.

Case-1 All Processor are busy -

In this case number of time required to finish all jobs ignoring dependency is roughly n/p not execute fraction so it will $\lceil n/p \rceil$

But we have dependency constrain, job are not independent, they have dependency represented by height of h of tree or graph

Here h (longest chain) decide minimum time to complete all job this give addition h unit of time.

thus when all Process are busy total time bound is

$$T_{greedy} \leq \lceil \frac{n}{p} \rceil + h$$

Case-2 Some Processor are ideal

Since fewer than P jobs are being Processed in some step, the time required might still be dominated by dependency chain ie - h.

ie $\lceil \frac{n}{p} \rceil + h$ since the bottleneck caused by dependency prevents further Parallelization.

Conclusion Combining both case -

we can conclude that greedy algo will finish all jobs in at most $T_{greedy} \leq \lceil \frac{n}{p} \rceil + h$

[Dynamic Programming]

① ↳ Counting Combination
② ↳ Knapsack Problem
③ ↳ Matrix chain multiplication
④ ↳ All pair shortest Problem

Problem: Calculating Combination:

- we are trying to calculate the binomial coefficient $\binom{n}{r}$, with respect to the number of ways to choose r item from set of n item. This can be expressed as-

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

base case:

$\binom{n}{0} = 1$ choose 0 item from n is 1.

$\binom{n}{n} = 1$ chosing all n item.

Approach 1: Divide and Conquer Approch

To choose r thing out of n,

(i) either; choose the first item, then we must choose remaining r-1 item from n-1 item. ie $\binom{n-1}{r-1}$

(ii) OR; Don't choose the first item, then we must choose the r item from remaining n-1 item. ie $\binom{n-1}{r}$

By using Divide and Conquere approch this problem can be solved as-

```
Choose(r,n){
    if(r=0 or n=r then) return(1)
    else return(choose(n-1, r-1) + choose(n-1, r))
}
```

$T(n) = 2T(n-1)+d$ on solving it leads to $\Theta(2^n)$

↳ Too much time (exponential)
we can reduce this by DP.

Note: If we can store the value of $\binom{n}{r}$ in any table then we can reduce the operation of calculating again and again same value when need next time.

USE a table $[0 \dots n, 0 \dots r]$ $T[i,j]$ hold $\binom{i}{j}$

Optimize solution :-(DP)

```
Choose (n,r){
(r=0) for i=0 to n-r do T[i,0]=1
(n=r) for i=0 to r do T[i,j]=1

      For j=1 to r do
          for i= j+1 to n-r+j do }  O(n^2)
```

Generation of table- $T[i,j] = T[i-1,j-1] + T[i-1,j]$

```
      return (T[0,r])
}
```

→ Now time complexity reduced to- $O(n^2)$

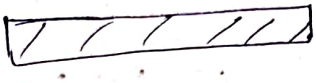
Example:

↳ 0/1 Knapsack problem, where the goal is to determine weather a subset of item can fill the knapsack to certain exact capacity.

(A) Divide and Conquer Approch

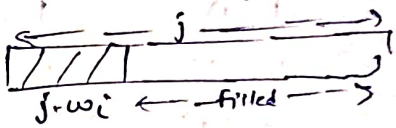
Here we have two possibilitic for each item
- Exclude the item from knapsack
- Include the item if it's weight is less than or equal to current knapsack capacity.

ie (1) if we exclude the item, the problem reduce to finding the soln for first i-1 item with the same capacity j ie K(i-1, j)



(2) if we include the item then problem reduced to finding the soln of first i-1 item with weight reduced to $j-w_i$ ie

$K(i-1, j-w_i)$



-Algo-

↳ if (i=0) then return (j=0)

↳ else if knapsack(i-1, j) then return (true)

↳ elseif $w_i \leq j$ then

return knapsack(i-1, j-w_i)

{K(i,j): ⇒ represent weather Possible to fill knapsack with first ith item and capacity of ~~this~~ is j
w_i ⇒ weight of ith item.}

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

↳ $\underline{\theta(2^n)}$

(B) Using DP Approch

```
knapsack(n,w):
    dp = [ ]                                   # Created a dp table of (n+1)x(w+1)
    for i in range(n+1):
        dp.append([False]*(w+1))
    for i in range(n+1):                       # Initilize the base case: for j=0, dp[i][w] = true.
        dp[i][0] = True
    for i in range(1, n+1):
        for j in range(1, w+1):
            if weight[i-1] > j:
                dp[i][j] = dp[i-1][j]
            else:
                dp[i][j] = dp[i-1][j] or
                           dp[i-1][j-weight[i-1]]
    return dp[n][w]
```

time complexity ⇒ $O(n \times w)$ where n is no. of item
w is knapsack capacity

in general it would be $O(n^2)$ space complexity ⇒ $O(n \times w)$

Example Given

item	weight
1	1
2	3
3	4
4	5

The knapsack has capacity 7. we want to find out if it is possible to select a subset of these item such that total weight is exactly 7.?

⇒

item/capacity	0	1	2	3	4	5	6	7
0-item	T	F	F	F	F	F	F	F
1 (1)	T	T	F	F	F	F	F	F
2 (3)	T	T	F	T	T	F	F	F
3 (4)	T	T	F	T	T	T	T	T
4 (5)	T	T	F	T	T	T	T	T

true

matrix chain multiplication

Question is in which order should I multiply to get minimum # of multiplication?

Approch-1 Using Divide and conquer

- Let cost(i,j) be the minimum cost of computing $m_i \cdot m_{i+1} \cdot m_{i+2} \cdots m_j$
- what is cost of breaking product m_k?

ie $(m_i \cdot m_{i+1} \cdots m_k)(m_{k+1} \cdot m_{k+2} \cdots m_j)$

we will get $\boxed{Cost(i,j) = cost(i,k) + Cost(k+1,j) + r_{i-1} * r_k * r_j}$

Algo-

```
Cost(i,j){
    if(i=j) then return(0)
    else
        return min[ cost(i,k) + cost(k+1,j) + r_{i-1} x r_k x r_j ]
                 i≤k<j
```

$T(n) = 3T(n-1)+d$ on solving will get

$O(3^n)$

Here we can do better by using DP Approch

Approch-2 DP

```
Stor cost(i,j) into m[i,j]
for i=1 to n do m[i,j]=0
for d=1 to n-1 do
    for i=1 to n-d do
        j= i+d
        m[i,j] = min min[m[i,k] + m[k+1,j] + r_{i-1} x r_k x r_j ]
                 i≤k≤j
```

time complexity reduced to 3^n to $O(n^3)$

Space complexity :) $O(n^2)$ For n×n table needed to store.

(1) Qu: Given a quick-sort Algorithem you task is to find worst-case time complexity for $n \geq 2$?

(n-1)

⇒ In the worst case the Pivot is selected either the smallest or the largest element, it is highely unballensed partation where one partation contain (n-1) element other 0(Empty).

Recurrence relation-

$T(n) = T(n-1) + O(n)$, where $O(n)$ is time taken to Partation array around Pivot. $n \geq 2$

$T(1) = O(1)$

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

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

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

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

Then $T(n) \Rightarrow T(n-3) + O(n-2) + O(n-1) + O(n)$

$\vdots$

$T(n-K) + n + (n-1) + (n-2) + \cdots (n-K+1)$

$n - K = 1$

$[n = 1+K]$

$1 + \underline{n + (n-1) + (n-2) + \cdots + 2}$

$\frac{n(n+1)}{2} \approx \underline{O(n^2)}$

time $\Rightarrow O(n^2)$ Ans

Qu: Suppose you are given a array $A[1 \cdots n]$ of sorted integer that hey been circuly shifted K-position to the right we can obsriouselY find largest element in $O(n)$ but your task is to find in $(\log n)$.

⇒
- Before the shift all the number are sorted in increasing order.
- After shift, there is point in the array (called pivot) wher order changes ie largest number followed by smallest number.

Eg: original: [5 15 27 29 35 42] [k=2 Shift] [35 42 5 15 27 29]

- we want to find Pivot point where this shift happen.

→ Approch: Calculate the middle index Set low=1, high = n

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

- Now compare the value of mid with the value of its neighbour

C-1 if A[mid] > A[mid+1] then A[mid] is largest and (high > mid)

if A[mid-1] > A[mid], then A[mid-1] is largest element ,(mid > low)

C-2 IF A[mid] > A[low], this mean left side is sorted largest will be on right side of mid

Set – low = mid+1

otherwise: Set high = mid-1

Eg → A = {27, 29, 35, 42, 5, 15} low = 0, high = 5

Step-1 mid → ⌊0+5/2⌋ → 2

step-2 A[mid] = A[2] → 35 Here 35 is not greater than its neighbour, go for subarray

Here 35

since A[low] ≤ A[mid] ie left Part is sorted go for right

Set – low = mid+1 → 2+1 → 3

Now Stt-3 mid → ⌊3+5/2⌋ → 4

A[4] → 5 since A[mid-1] > A[mid]
42 5
then largest is A[mid-1]
Ans

Time Complexity

↳ $T(n) = T\left(\frac{n}{2}\right) + O(1)$

↳ Recurrence call for Half Subarray

↳ constant time to comparing element at each step.

Time to find largest in array of size n.

T(n) → $\Theta(\log n)$ by master theorem

Qus: Suppose you are given a unsorted array A of all integer range 0 to n except for one integer, find missing number. Assume $n \geq 2^k - 1$
Design O(n) divide and conqure algo to find missing number?

⇒ Example: Let A = [3, 7, 1, 6, 0, 4, 2] where n = 7 and 5 is missing Here

Step-1 Divide based on MSB of each number

3 = 011, 7 = 111, 1 = 001, 6 = 110, 0 = 000, 4, 100, 2 ⇒ 010

(MSB-0) Group-0 [3, 1, 0, 2]

(MSB-1) Grp-1 = [7, 6, 4]

Step-2 Recursively search in group with fewer element

Here group 0 is 4 number then 1 must be 4 number

7 ⇒ 111 6 ⇒ 110 4 ⇒ 100

Divide based on second bit:

Grp-0 [4] Grp-1 [7, 6]

Step-3 Here grp-2 must contain 2-element but 1 is missing Here
Divide based on 3rd bit

4 ⇒ 100 101 → it will be next

Hence Ans 5

Algo: If there is only one number the return it.
Base

Input → An array A to $n = 2^k - 1$ element ie from 0 to n

Divide → Seprete the numbers in two grp O(n) Divide into two grp based on MSB

Grp 0: Number whose msb 0

Grp 1: ~~~~ msb 1

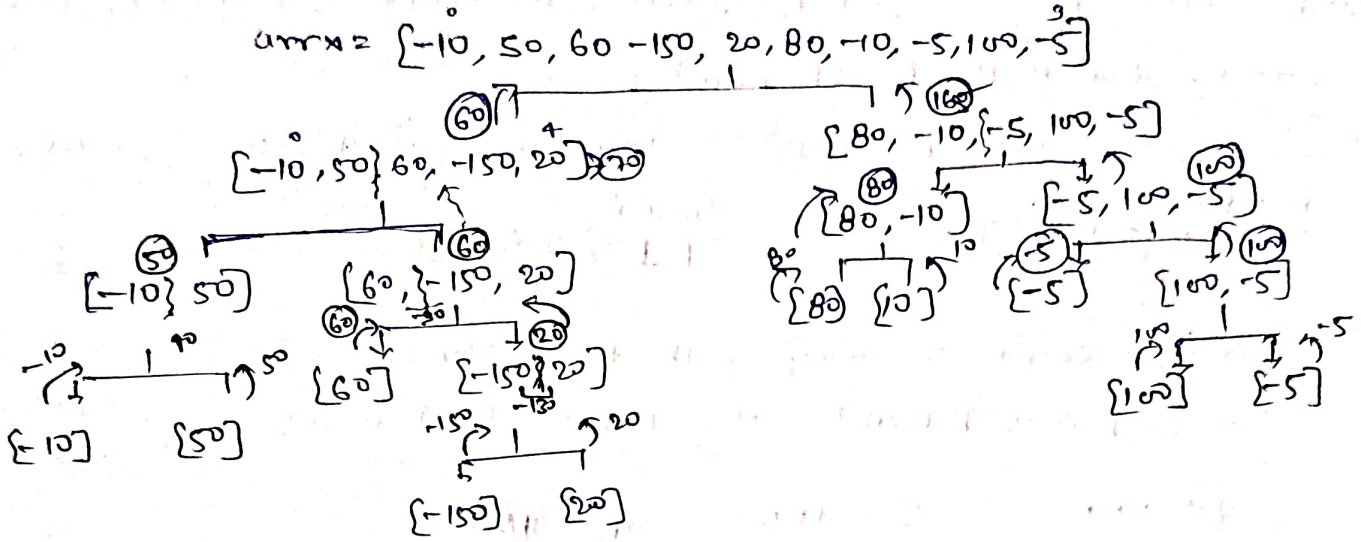
Conqures Count the number of element in each $T(\frac{n}{2})$ only one recursive need to call again with fewer number element grp recursively call again-

Combine: Return missing number once recursion reaches base case

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

By master theorem time O(n) Ans

Qu: You have given a array contain postive and -ve integers your task is to find a continuous subarray of max size in O(nlogn) time using divide and conqure method.



Proof:- ~~Divide~~ Approch divide the array into two halve and compute largest sum in three diferent places-

1. In left Half
2. Crossing the middle of array
3. right half.

Recurrence relation for doing this-

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

$2T\left(\frac{n}{2}\right)$ → it is for the two recursive call for left and right halve,

$O(n)$ → time taken to find maximum subarray sum that can be done in ~~constant~~ $O(n)$ time.

on solving using master method

$$n^{\log_2^2} \log n \Rightarrow \underline{O(n \log n)} \text{ Ans}$$

Qu! Explain Different type of randomize algorithem with an examples. (Ext 1)

Explain the randomized quick sort algorithem and derive its running time?

*) Randomized algorithem are versatiel and can be applied in varies Example of greedy algo, divide and conquar, dynamic programming.

By introducing we can reduce the chance of worst case time complexity.

few example-

(A) Randomized DP for approximate the number of solution by solving a subset of Problem.

(B) Randomized graph Algorithem such as DFS Randomized, Randomized MST ie kruskal with randomized edge selection, such that no cycle is formed.

(C) Randomized Quick sort

```
def randomized-partation(arr, low, high):
    Pivot-index = random.randint(low, high) // taking random pivot
    arr[pivot-index], arr[high] = arr[high], arr[pivot-index]
    return Partation(arr, low, high)


def randomized-quickSort(arr, low, high):
    if low<high:
        Pi = randomized-partation(arr, low, high) // Partation around random
                                                     Pivot.
        // recursively sorting before and after partation
        randomized-quickSort(arr, low, Pi-1)
        randomized-quickSort(arr, Pi+1, high)

def partation(arr, low, high):
    Pivot = arr[high]
    i = low-1
    for j in range(low, high):
        ~~i = i+1~~
        if arr[j]<Pivot:
            i = i+1
            arr[i], arr[j] = arr[j], arr[i]
    return i+1
```

Best / Avg Case:

$$T(n) = 2T\left(\frac{n}{2}\right) + \underbrace{O(n)}_{\text{time spent on partitioning.}}$$

using master theorem $T(n) = O(n \log n)$

Worst Case: $T(n) = T(n-1) + O(n)$

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