Ans-(2)

=> For multiplication of large integers, we can use 2 approaches:-

(1) Naive Approach
(2) Divide and Conquer

=> Now, in Naive approach, we take individual bits of second number and multiply it with all numbers of first.

=> Finally, we add all the multiplications.

=> Here, time complexity becomes $O(n^2)$

=> By using the Divide and conquer approach we can reduce the time complexity.

=> Firstly, we divide two numbers in two halves.

=> let the given numbers be X & Y.

=> we assume that n = even

=> So, we divide both numbers such that it satisfies the following:-

$X = X1 * 10^{n/2} + X2$

$Y = Y1 * 10^{n/2} + Y2$

⇒ Here, X1 and Y1 contains leftmost n/2 bits of X and Y

⇒ Also, X2 and Y2 contains rightmost n/2 bits of X & Y

⇒ So, product of X and Y can be written as:-

$$XY = (X1 * 2^{n/2} + X2)(Y1 * 2^{n/2} + Y2)$$

$$= 10^{n} (X1 * Y1) + 10^{n/2} (X1*Y2 + X2*Y1) + X2*Y2$$

⇒ Here, there are 4 multiplications involved.

⇒ Here, each multiplication is of size n/2

⇒ To reduce the 4 multiplications into 3 multiplications, we do the following for middle two terms:→

$$X1*Y2 + X2*Y1 = (X1 + X2) * (Y1 + Y2) - (X1 * Y1) - (X2 * Y2)$$

⇒ So, final equation becomes;

⇒ $$XY = 10^{n} (X1 * Y1) + 10^{n/2} * [(X1 + X2) * (Y1 + Y2) - (X1 * Y1) - (X2 * Y2)] + X2 * Y2$$

Page:
Date:

=> Now; the Recurrance becomes

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

=> So; Time complexity becomes $O(n^{1.59})$

⇒ $a = 26$; $b = 13$

⇒ $a = a_1 a_0 \Rightarrow a_1 = 2$; $a_0 = 6$

⇒ $b = b_1 b_0 \Rightarrow b_1 = 1$; $b_0 = 3$

⇒ $c = c_2 10^n + c_1 10^{n/2} + c_0$ — (1)

⇒ n = total digits = 2

⇒ $c_2 = a_1 * b_1 = 2 \times 1$

$\boxed{c_2 = 2}$ — (2)

⇒ $c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$

$= (2+6) * (1+3) - (2+18)$
$= 8 * 4 - 20$
$= 32 - 20$

$\boxed{c_1 = 12}$ — (3)

⇒ $c_0 = a_0 * b_0 = 6 \times 3 = \boxed{18}$ — (4)

⇒ By (1), (2), (3), (4)

⇒ $26 \times 13 = (2 \times 10^2) + (12 \times 10^{2/2}) + 18$

$= (2 \times 100) + (12 \times 10) + 18$
$= 200 + 120 + 18$

$\boxed{26 \times 13 = 338}$

=>

Ans - (3) The longest common subsequence problem is about finding the longest sequence which exists in both the strings.

=> We normally use Dynamic Programming for solving this problem

=> Here; S1 = ABCBDAB ; length = 7
S2 = BDCA BA ; length = 6

=> Solution :->

j	0	1	2	3	4	5	6
i	yj	B	D	C	A	B	A
0	xi 0	0	0	0	0	0	0
1	A 0	0↑	0↑	0↑	1↖	1←	1↖
2	B 0	1↖	1←	1←	1↑	2↖	2←
3	C 0	1↑	1↑	2↖	2←	2↑	2↑
4	B 0	1←	1↑	2↑	2↑	3↖	3←
5	D 0	1↑	2↖	2↑	2↑	3↑	3↑
6	A 0	1↑	2↑	2↑	3↖	3↑	4↖
7	B 0	1←	2↑	2↑	3↑	4↖	4↑

Start

=> It is clear that length of

LCS = 4

=> Considering only ↖ ↑ ←

The longest common subsequences are:-> <BCBA> ; <BCAB> ; <BDAB>

Ans-(4)

* NP - Completeness *

=) The class of problem "NP-complete" stands for the sub-class of decision making problems in NP that are hardest.

=) The class NP complete is abbreviated as NPC and comes from

- Non deterministic Polynomial

- Complete - "Solve one, solve them all"

=) A decision problem L is said to be NP-complete if :=>

(i) L is in NP that means any given solution to this problem can be verified quickly in polynomial time

(ii) Every problem is NP reducible to L in polynomial time

=> Informally, it is believed that if a NPC problem has a solution in polynomial time then all other NPC problems can be solved in polynomial time.

=> Some well-known problems that are NP-complete :->

(1) N- Puzzle Problem
(2) Knapsack Problem
(3) Traveling Salesmen Problem
(4) Graph coloring problem

=> Following techniques are applied to solve NPC problems :->

(1) Randomization
(2) Heuristics
(3) Approximation Approach
(4) Restriction
(5) Parameterization

* NP-Hard *

=> It stands for non-deterministic polynomial time hard.

=> It can be defined as:->

(i) NPH is a class problem that are "Atleast as hard as the hardest problems in NP"

(ii) A problem H is NP-Hard if there is a NPC problem L that is polynomial time reducible to H.

(iii) A problem H is said to be NPH if satisfiability reduces to H

(iv) If NPC problem L can be solved in polynomial time by machine with oracle for H.

=> These problems are generally type-decisions, search problems, optimization problems.

(1). 19ORS098

Page:
Date:

Ans-(1) * Binomial coefficient *

=> Firstly, computing binomial coefficient is non optimization problem but can be solved using dynamic programming

=> $C(n,k) \Rightarrow \binom{n}{k}$ are represented as :-

$$(a+b)^n = C(n,0)a^n + \cdots + C(n,k)a^{n-k}b^k + \cdots + C(n,n)b^n$$

=> The recursive relation is defined by prior power :->

$$C(n,k) = C(n-1,k-1) + C(n-1,k)$$

for $n > k > 0$

=> Dynamic algo. constructs a $n \times n$ table.

=> Main logic :->

function $C(n,k)$

if $k=0$ or $k=n$
then return 1

else
return $C(n-1,k-1) + C(n-1,k)$

* Finding Binomial Coefficient of C(9,4)

	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1
5	1	5	10	10	5
6	1	6	15	20	15
7	1	7	21	35	35
8	1	8	28	56	70
9	1	9	36	84	126

=> The value of C(9,4) = 126

=> Pascal's Triangle is used to find the coefficient.

⇒ In N-Queen Problem, we have to place the Queens in such a way that no Queen can kill the other Queen.

⇒ We use Back tracking to solve the problem.

⇒ Total Given Queens = 4

∴ $\boxed{N = 4}$

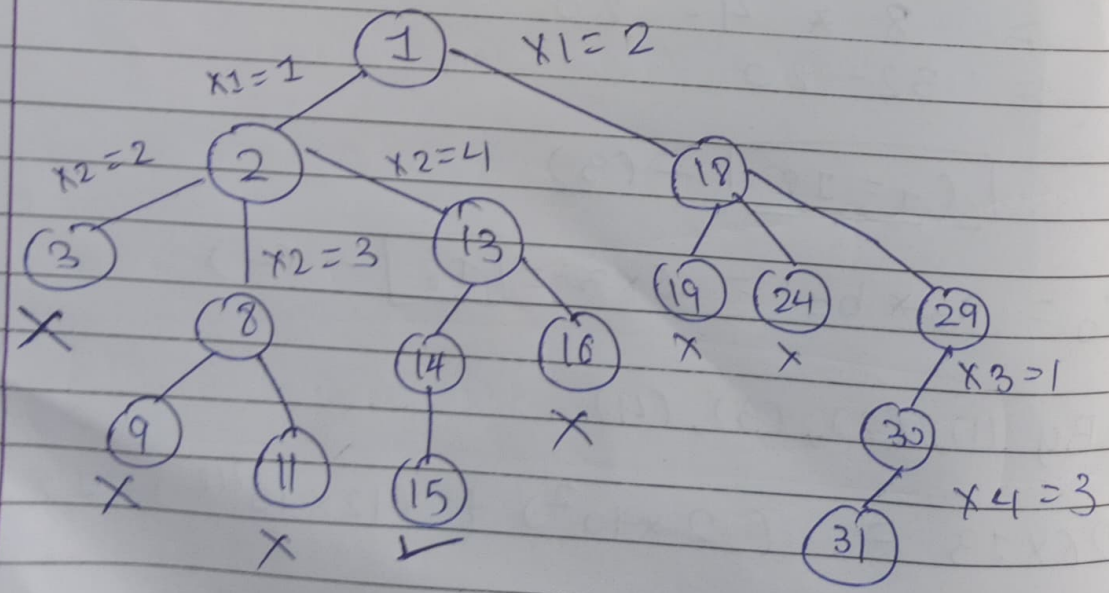
⇒ x ⇒ | 1 | 2 | 3 | 4 |

↳ Indicates the column number

⇒ The chess board will be of

4 × 4

⇒



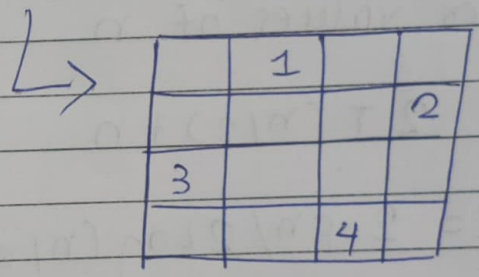
=> Here, in the tree :-

x => Dead Node.

=> There are 2 Total solutions for 4 Queen

(1) (2, 4, 1, 3)

(2) (3, 1, 4, 2)



Ans -(7)

=> Given the values :->

a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

=> From the above table;

max. deadline = 6.

=> Now, we will require to sort the given table in descending order with respect to w_i

=> Here, the given table is already sorted in descending order.

=> The optimal Job scheduling is as below.

1	2	3	4	5
J4	J2	J3	J1	J7

=> It is clear from above that Jobs J5 and J6 are not scheduled.

=> So, we will need to pay penalty for that.

=> Penalties = $w_i(J5) + w_i(J6)$

= 30 + 20

= 50

=> The optimal sequence of Jobs :->

< J4, J2, J3, J1, J7 >

=> The total penalty to pay = 50

Page:
Date:

Ans-(8)
=

=> Basically, Greedy algorithms are used for optimization problems.

=> Activity Selection Problem is an optimization problem which deals with selection of non-conflicting activities that needs to be executed in a given time frame.

=> Each Activity is marked by start time and finish time.

i	1	2	3	4	5	6	7	8	9	10	11
Si	1	3	0	5	3	5	6	8	8	2	12
fi	4	5	6	7	8	9	10	11	12	13	14

=> Greedy algorithm will always choose the local best solution for getting best or optimal global solution.

Activity 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

1
4
8
11

Page:
Date:

=> For selecting max. activities, we need to sort the array with respect to finish time

=> After sorting; we can choose activities: 1, 4, 8, 11.

=> Hence, maximum 4 activities can be choosen.

=> By following greedy approach; we selected the activity, that has least finish time among the remaining activities.

Ans-
(9)

* Recurrance Relation *

=> A recurrance Relation is an equation that defines a sequence based on a rule that gives the next term as a function of the previous term

=> We can also that a recurrance is a equation that describes a ~~term~~ function in terms of its values on smaller input.

=> Here, a recursive relation, T(n) is a recursive function of integer n.

=> Recurrance Relations are very useful in calculating the running time complexities of recursive algorithms.

=> There are 4 methods for solving Recurrance Relations:

(1) Master method

(2) Substitution method

(3) Recursion Tree method

(4) Iteration Method

* Master Method *

=> By using this method, we can get immediate solution

=> This method only works on the recurrance relations that can be transformed into following:-

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

Here, $a >= 1$ & $b >= 1$

=> There are 3 cases:-

(1) If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then follows $T(n) = O$

(2) If it is true for some constant $K \geq 0$ such that:
$F(n) = O$, then it follows : $T(n) = O$

(3) If it is true $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$ and also true that
: $af(\frac{n}{b}) \leq cf(n)$ for $c < 1$ for large value of n.

then ; $T(n) = \Theta((f(n))$

* Substitution method *

=> It consists of two main method:-
(1) Guess the solution
(2) Use the mathematical induction to find the boundary condition and proves that the guess is correct.

* Iteration method *

=> It basically means to expand the recurrance and express it as a summation of terms of n and initial condition.

* Recursion Tree Method *

=> It is a pictorial representation of iteration method in the form of a tree where at each level nodes are expanded.

=> It is generally used when Divide and Conquer technique is applied.

=> A Recursion Tree is best used to generate a good guess which can be verified by Substitution method.

Ans -(10) In merge Sort, we divide the array into nearly two equal halves and then, solve them recursively using merge sort only.

So, we have;

$$T\left(\frac{n_L}{2}\right) + T\left(\frac{n_R}{2}\right) = 2\,T\left(\frac{n}{2}\right)$$

=) Here; n_L = left Half ; n_R = Right half.

=) $n_L \cong n_R$

=) Finally, we merge these two sub-array using merge produce which takes $\Theta(n)$ times

=) If $T(n)$ is the time required by merge sort for array of size n.

=) Then, recurrence Relation :->

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

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

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

=> Solving using substitution method.

=> We need to prove that

$$T(n) <= c n \log n.$$

=> Assuming that it is true for smaller values of n.

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

$$<= 2\, c\, n/2 \log(n/2) + n$$

$$= cn\log n - cn\log 2 + n$$

$$= cn\log n - cn + n$$

$$= cn\log n + n(1-c)$$

$$<= cn\log n$$

=> Time complexity: => $n\log n$

$\Rightarrow T(n) = 2T(\frac{n}{2}) + n \quad -(1)$

$\Rightarrow T(n/2) = 2T(n/4) + n/2 \quad -(2)$

$\Rightarrow T(n/4) = 2T(n/8) + n/4 \quad -(3)$

$\Rightarrow 2[2T(n/4) + n/2] + n$

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

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

$= 2^2[2T(n/8) + n/4] + 2n$

$= 2^3[T(n/2^3) + 3n$

$= 2^4$

$\Rightarrow$ After k times;

$$\boxed{2^k T(n/2^k) + kn}$$

$\Rightarrow \frac{n}{2^k} = 1 \; ; \; T(1) = 1$

$\Rightarrow n = 2^k \Rightarrow \log n = \log_2 2^k$

$\Rightarrow \boxed{k = \log n}$

⇒ On substituting in - (1) it becomes

⇒ $n(1) + n\log n$

⇒ $\boxed{O(n\log n)}$