Md. Sahil
001710501029

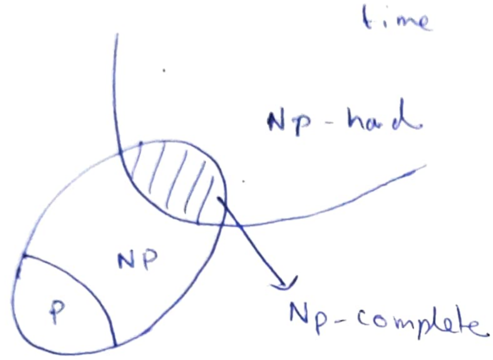
1.

a) NP- Completeness

A given problem π is NP complete if

i) $\pi \in NP$

ii) π is NP hard (i.e. if every problem in NP is polynomial time reducible to π)



i) Showing a problem is NP

This can be achieved simply using a certificate verification strategy, where a certificate is an answer to our problem & verifier is a polynomial algorithm that can tell us whether the answer to the problem is "yes" or "No".

ii) showing a problem is NP-hard

The theory of reduction is used to prove this. let H be an NP-hard problem and X be our target problem. We have to reduce H to X. This means that given a polynomial time subroutine for X, we can use it to solve H in polynomial time.

SATISFIABILITY (SAT) problem was the first problem to be proved as NP-hard by Stephen-Cook. All proofs ultimately follow reduction from the NP-completeness of CIRCUIT-SAT

Example:- SAT problem or satisfiability problem. Md. Sahid, 0017105010(?) (2)

i) SAT ∈ NP

→ A proof (proposed solution) is an assignment to the variables.

→ The proposed solution can be verified in polynomial time.

ii) SAT is NP-hard

Proof If there exists a polynomial time algorithm for SAT, then there is one for all problems in NP.

b) In order to prove that ~~a~~ an NP-hard problem is not necessarily in NP, we take an example of such a problem:-

given some ~~time~~ n, find all cliques in all graphs with n vertices. Clearly this problem is harder than the general clique problem (i.e. find the largest clique given a graph of n verticues & m edges).

Since, we can solve this ~~prob~~ problem, we can solve the previous clique problem. The answer to this problem is actually all subsets of n vertices which form cliques.

But also note that to verify that we have the ~~correct~~ correct answer, we have to check that we have all sub sets which form cliques.

So this problem is NP-hard, but not NP.

1. c) Answer (II) R is NP complete.

Because ~~an~~ NP-complete problem, S, is polynomial time reducible to R.
Thus R is NP-hard.

~~B~~. Option (I) is incorrect because R is not NP.
An NP complete problem has to be both NP-hard & NP.

Option (III) is incorrect because Q is not NP. (Same reason as in (I))

Option (IV) is incorrect. as there is not NP complete problem that is polynomial time reducible to Q.

1. d) P denotes polynomial time solvable problems, i.e. a ~~cl~~ problem such that there is a solution that solves it in $O(n^k)$ time when k is a constant & n is ~~xx~~ input size.

A problem is said to be NP if.
→ the solution can be verified in polynomial time.
→ solution always have length. polynomial in ~~xx~~ input size.

To show $P \subset NP$:-

Let ~~x~~ Problem X be in P.

We need to show X is also in NP.

P.T.O

~~For~~ For a given instance i of x_i, and a candidate ~~s~~ solution S_i, the given steps will decide if S_i is correct solution or not.

1. As X is ~~as~~ in P, we have a polynomial solvable problem. ~~atop~~ algorithm that can determine the solution of x_i.

2. let the answer of X be S, then it will ~~b.~~ have length polynomial in input size.

3. Now, S can be matched with S_i, ~~then~~ the candidate solution, and tell that if solution is correct or not, Hence verification of solution S is ~~sol~~ done in polynomial time.

So the problem can be said to be in NP too as both the criteria are fullfilled.

Hence $X \subset P \Rightarrow X \subset NP$

$\therefore P \subset NP$.

2 a) Input :- sequence of numbers $a_1, a_2, a_3 \ldots a_n$

To find the longest increasing subsequence.

Using recursion :-

Let $D[i]$ be the length of the longest Increasing subsequence for the Sub array $(a_1 \ldots a_i)$. Such that the sub sequence ends with $a[i]$.

~~LIS~~ Our Longest increasing subsequence will be $=$ $Max(D(i))$ for $0 < i \leq n$.

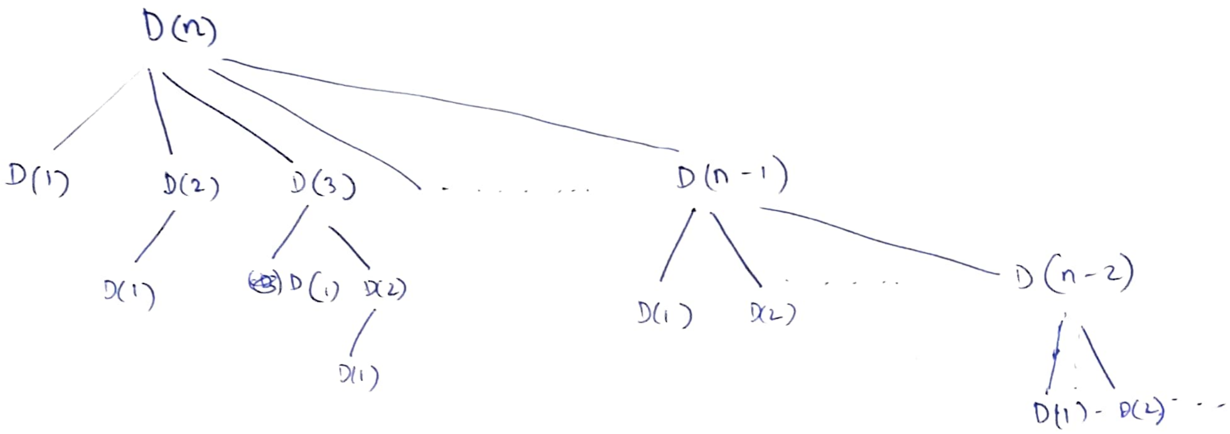
$\therefore$ $D[i]$ can be recursively evaluated as follows :-

$D[i] = max(D[j]) + 1$; where $0 < j < i$ and $a[i] > a[j]$

if no such j exists.

then $D[i] = 1$.

Thus the recursion tree will look like this :-



Calculating time complexity :-

In order to calculate $D(n)$, $D(1) \ldots D(n-1)$ needs to be calculated.

$\therefore$ $T(n) = T(n-1) + T(n-2) + \ldots T(1) + k$. —(I)

$\Rightarrow$ $T(n-1) = T(n-2) + T(n-3) \ldots T(1) + k$ —(II)

(I)-(II) $T(n) - T(n-1) = T(n-1)$

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

$= 2 \times [2T(n-2)]$

$= 2 \times 2 \times 2 \times T(n-3)$

$= 2^{n-1} T(1)$ $\therefore$ The time complexity is of order $O(2^n)$

Using Dynamic Programming :-

Here, instead of recalculating the values of $D[j]$ (of $0<j<i$) at each call, we use the concept of memoization so that each $D(i)$ is calculated only once.

Let dp be an array such that $dp[i] = D(i)$

$$\therefore \quad dp[i] = \begin{cases} 1 + \max(dp[j]) & \text{where } 0<j<i \ \& \ a[j] < a[i] \\ 1 & \text{if no such } j \text{ exists.} \end{cases}$$

$$i = 1, \ldots n.$$

$$LIS = \max(dp[i]) \quad \text{where } 0 < i \leq n$$

We calculate $dp[i]$ starting from $i=1$ with increasing value of i for each iteration.

$$\therefore \quad T(1) = k$$

$$T(2) = T(1) + k$$

$$T(3) = T(2) + 2k$$

$$\vdots$$

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

$$\therefore \quad T(n) = T(n-1) + (n-1)k$$

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

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

$$\vdots$$

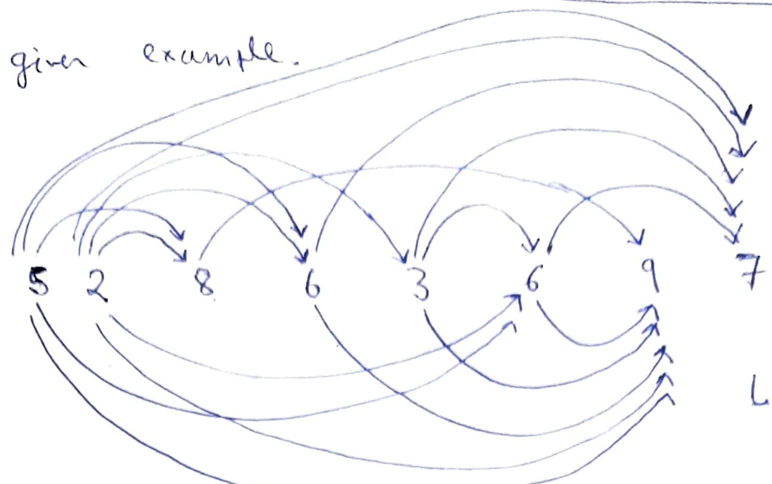
$$= k + 2k + \ldots (n-1)k$$

$$T(n) = \frac{(n-1)(n)}{2} k.$$

$\therefore$ Time complexity will be of order $O(n^2)$.

DAG for the given example.

a = 5 2 8 6 3 6 9 7



LIS :→ 2→3→6→7

or

2→3→6→9

c) Table of computation.

X	X	A	L	T	R	U	I	S	T	I	C
X	0	1	2	3	4	5	6	7	8	9	10
A	1	0	1	2	3	4	5	6	7	8	9
L	2	1	0	1	2	3	4	5	6	7	8
G	3	2	1	1	2	3	4	5	6	7	8
O	4	3	~~3~~ 2	2	2	3	4	5	6	7	8
R	5	4	3	3	2	3	4	5	6	7	8
I	6	5	4	4	3	3	3	4	5	6	7
T	7	6	5	4	4	4	4	4	4	5	6
H	8	7	6	5	5	5	5	5	5	5	6
M	9	8	7	6	6	6	6	6	6	6	6

∴ Minimum edit distance = $E(9, 10) = 6$.

The path to reach goal is marked in the table.

Other possible paths are marked with dotted paths.

Md. Sahil, 001710501029

~~Alignment~~

Optimal solution Alignment:-

A	L	G	O	R	_	I	_	T	H	M
A	L	_	T	R	U	I	S	T	I	C
		1	2		3		4		5	6

Other ~~solution~~ possible optimal solution:-

A	L	G	O	R	_	I	_	T	H	M
A	L	T	_	R	U	I	S	T	I	C
		1	2		3		4		5	6

A	L	G	O	R	I	_	T	H	M
A	L	T	R	U	I	S	T	I	C
		1	2	3		4		5	6