Q.1

Ans:

=> we have 4-pegs & have to move n disks from peg-1 to peg-4 with the standard restriction of this problem.

=> Basically, we can somehow compare & relate the problem with original T.O.H problem which has 3 disk & transfer to the pegs.

=> so, we have moves in total = $2^n - 1$ to perform / calculate this problem.

=> It would be no. of moves which can not be more than $2^n - 1$.

$\boxed{= d}$

Q.2

Ans

"I don't know"

Q. 3

Ans: Let $OPT[i, j]$ = minimum possible weight (smallest) considering items $\{1 \dots i\}$ such that the total values is $\geq j$

$\Rightarrow 1 \leq i \leq n$ and $0 \leq j \leq \sum_{k} v[k]$.

Base Case: $OPT(i, j) = 0$ if $i = 0$ or $j = 0$.

$OPT(i, j) = \min_{k \leq i} \{w[i]\}$ if $j = 0$.

otherwise:

$OPT(i, j) = \min \{ OPT(i-1, j), OPT(i-1, j - v[i]) + w[i] \}$

not picking an element

picking an element.

Time-Complexity

$= O(n \cdot v)$, $v = \sum_{i} v[i]$.

$\Rightarrow$ The Memo is of size $n \times \sum_{i} v[i]$. Now if $\sum_{i} v[i]$ is $O(n^k)$

But if $\sum_{i} v[i]$ is arbitrarily large then it is no more polynomial.

Q.4

Ans:

⇒ Main aim to determine the graph has a negative-weight cycle. So avoid this cycle (circular dependency) we need an additional parameter which decreases at each cycle of Recursion.

APSPdist (u, v, k)

$$= \begin{cases} 0 & \text{if } u = v \\ \infty & \text{if } k = 0 \ \&\& \ u \neq v \\ \min\limits_{x} \left(\cancel{\text{APSP}}\ \text{APSP}_{dist}(u, x, k-1) + w(x \to v) \right) & \end{cases}$$

⇒ APSP dist (u, v, k) is the shortest path length from u to v with at most k edges.
$(u \rightsquigarrow v)$

∴ APSP (dist (u, v, k)) $\equiv$ APSP(dist $(u, v, k-1)$)

⇒ we're really trying to compute APSPdist $(u, v, k-1)$ to determine the length. (negative-weight)

Q.5

Ans:

$$w^i[j,k] = \begin{cases} k & \text{if } j = i \quad (\text{Base case}). \\ w^i[j-1,k] & \text{if } j > i \ \& \ E(j,k) = E(j-1,k) + b. \quad // \text{deletion.} \\ w^i[j,k-1] & \text{if } j > i \ \& \ E(j,k) = E[j,k-1] + a \quad // \text{insertion.} \\ w^i[j-1,k-1] & \text{if } j > i \ \begin{cases} E(j,k) = E(j-1,k-1) \\ E(j,k) = E(j-1,k-1) + c \end{cases} \end{cases}$$

Q. 6.

Ans:

$$
w^i(j,k) = \begin{cases} \infty & \text{if } j < i \\ k & \text{if } j = i \\ w^i[j-1,k] & \text{if } j > i \ \& \ E(j,k) = E(j-1,k). \\ w^i[j,k-1] & \text{if } j > i \ \& \ E(j,k) = E(j,k-1] \\ w^i[j-1,k-1] & \text{if } j > i \ \left\{ \begin{array}{l} E(j,k) = E(j-1,k-1) \\ E(j,k) = E(j-1,k-1) \end{array} \right. \end{cases}
$$

=) Here, we set if $j < i$ then there is no meaning of the given solution so at that point we set to '∞'.

Q.7

Ans:

1.

As we have $\lceil \frac{n}{3} \rceil$ blocks of 3 element each & we need to find MoM.

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

2. we need space to store the result.

$T(n) = O(n)$. to store.

3. $\lceil \frac{n}{11} \rceil$

$$T(n) \leq O(n) + T\left(\frac{n}{11}\right) + T\left(\frac{2n}{11}\right).$$

Q.8

Ans: ~~(i) ~~ ~~statement~~

~~(ii)~~

(a) statements are as follows:

(i) IsSubseqRecursive (i, j+1)

(ii) IsSubseqRecursive (i+1, j+1).

(b) Time Complexity of above equation is

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

=) let it be the time Complexity of the IsSubsequence (i, j) whether case they both the string is subsequence of the string or not.

~~Q(8)~~ Q. 9

~~Ans:~~

Ans:

(a). ReverseSort (A, s)

if A[length] = s : return A.

else if
minimum = select (s, A)
// it will gives the s^{th} smallest element.
x' = Rank (minimum, A)

if A[s] == minimum and s = x'
return ReverseSort (A, s+1).
else.
tempo = A[s]
A[s] = minimum
A[x] = tempo.
return ReverseSort (A, s+1)

(b). T(k-2) + 2. (∵ T(k) = call to Reverse ( )
∴ T(2) = 1

(c) T(k) = # calls to Reverse made by RS (A, n-k, +1)

~~RS~~. Reverse (A, n-k + 1)

$\downarrow$
k-sized
Subarray.

(d). $T(K-1)+2$. $T(2) \leq 1$.

=) tightest upper bound.
Reverse A[s...n].

(e) space complexity as we need K-sized subarray.

$T(K) = \underline{O(K)}$.

Q. 10

Ans: ⇒ In DP we need design an algorithm for the problem which is given here.

⇒ we have to earn max profit.

Profit (~~Vect~~ price).
size = length of the values of prices.

// base case.
if (size == 0) : return 0
else,
for <- i to size (length of prices).
if
prices [0] > prices [i],
prices [0] = prices [i]. // prices [0] = U.

else if
prices [0] = prices [i] // price [0] = B.

Result [] = max (Result [i-1], U - B].

return result [size -1].

Time Complexity = O(n).
Space - " = O(n) // Result [] array