Analysis of Algorithm
(May 2019)

Q.P. Code - 55801

Q 1 Solve any 4

1. **Derive the complexity of quick sort for best case and worst case.** **5M**

Best Case:

In the best case, every time we partition the array, we divide the list into two nearly equal pieces. Partitions are balanced and it is identical to merge sort. Hence complexity of best case will be O(n $\log_2 n$), Let us prove it by solving recurrence equation.

T(1) = 0

T(n) = 2T(n/2) + θ (n)

2T(n/2): is the time required to solve the problems of size (n/2).

O(n): is the time required to fix the position of the pivot.

Using Master method: $T(n) = 2\,T(\frac{n}{2}) + n$

Comparing with : $T(n) = a\,T(\frac{n}{b}) + f(n)$

a = 2, b = 2 & f(n) = n with k = 1

where k is degree pf polynomial function f(n)

$a = b^K$, $2 = 2^1$.

So from the case I variant 2

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

T(n) = O(n log n)

"Hence Proved"

Time Complexity - T(n) = O(n $\log_2$ n)

Worst Case:

Worst case for quick sort occurs when the list is already sorted. To divide the list, algorithm scans the array and determines the correct position of the pivot, so division cost of Quick Sort is linear, i.e θ (n). In worst case, one of the sub list has size 0. And other has size (n-1).

Thus,

T (0) = 1

T (n) = T(conquer)+T(divide)+ T (combine)

T (n) = T(n-1) + θ (n) + θ (1)

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

Using iterative approach

Substitute n by (n-1) in equation (1),

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

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

Substitute n by (n-1) in equation (2)

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

From equation (3),

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

After k iterations,

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

Let k = n,

$T(n) = T(0) + 1 + 2 + 3 + \ldots + (n-2) + (n-1) + (n)$

$= 1 + 2 + 3 + \ldots + n = \sum n$

$= n(n+1)/2 = (n^2/2) + (n/2)$

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

2. What is asymptotic analysis? Define Big O, Omega and Theta notations. 5M

Asymptotic Notation:

Asymptotic analysis are a mathematical tool to find time or space complexity of an algorithm without implementing it in a programming language. It is a way of describing a major component of the cost of the entire algorithm.

- Asymptotic notation does the analysis of algorithm independent of all such parameters

Big O:

Let f(n) and g(n) are two nonnegative functions indicating running time of two algorithms. We say, g(n) is upper bound of f (n) if there exist some positive constants c and n_0 such that $0 \le f(n) \le c \cdot g(n)$ for all $n \ge n_0$. It is denoted as $f(n) = O(g(n))$.

- The Big O notation, where O stands for 'order of', is concerned with what happens for very large values of n.
- The Big O notation defines upper bound for the algorithm, it means the running time of algorithm cannot be more than its asymptotic upper bound for any random sequence of data

Big omega:

Let f(n) and g(n) are two nonnegative functions indicating running time of two algorithms. We say, g(n) is lower bound of function f (n) if there exist some positive constants c and n_0 such that $0 \le c \cdot g(n) \le f(n)$ for all $n \ge n_0$. It is denoted as $f(n) = \Omega(g(n))$.

- This notation is denoted by 'Ω', and it is pronounced as "Big Omega".
- Big Omega notation defines lower bound for the algorithm.

Big Theta:

Let f(n) and g(n) are two nonnegative functions indicating running time of two algorithms. We say, g(n) is tight bound of function f (n) if there exist some positive constants c1, c2 and n_0 such that 0 ≤ c1. g(n) ≤ f(n) ≤c2. g(n) for all n ≥ n_0. It is denoted as f (n) = θ(g(n)).

- This notation is denoted by 'θ' and it is pronounced as Big theta.
- Big theta defines tight bound for the algorithm.

3. Write an algorithm to find all pairs shortest path using dynamic programming. 5M

Algorithm for all pair shortest path:

```
for all vertices u
 for all vertices v
 if u = v
dist [u, v] ← 0
 else
dist [u, v] ←∞
 for l← 1 to V - 1
 for all vertices u
 for all edges x → v
 if dist [u, v] > dist [u, x] + w(xv)
dist [u, v] ← dist [u, x] + w(xv)
```

4. Write a note on " Optimal Storage on Tapes". 5M

Problem:

Given n programs P1, P2, …, Pn of length L1, L2, …, Ln respectively, store them on tap of length L such that Mean Retrieval Time (MRT) be minimum.

Retrieval time of the j^{th} program is a summation of the length of first j programs on tap. Let T_j be the time to retrieve program P_j. Retrieval time of P_j is computed as,

$$T_j = \sum_{k=1}^{j} L_k$$

Mean retrieval time of n programs. It is required to store programs in an order such that their Mean Retrieval Time is minimum.

Optimal Storage on tape is minimization problem which,

$$\text{Minimize } \sum_{i=1}^{n} \sum_{k=1}^{i} L_k$$

$$\text{Subjected to } \sum_{i=1}^{n} L_i \le L$$

- In case we have to find the permutation of the program order which minimizes the MRT after storing all programs on single tape only.
- Greedy algorithm stores the programs on tape in nondecreasing order of their length, which ensures the minimum MRT.

- A magnetic tape provides only sequential access of data, In an audio tape, cassette, unlike a CD, a fifth song from the tape cannot be just directly played. The length of the first four songs must be traversed to play the fifth song. So in order to access certain data, head of the tape should be positioned accordingly.

5. Define master theorem. Solve the following using master method T(n) = 8T(n/2) + n² 5M

Definition: Divide and conquer strategy uses recursion. The time complexity of the recursive program is described using recurrence. The master theorem is often suitable to find the time complexity of recursive problems.

- Master method is used to quickly solve the recurrence of the form $T(n) = a \cdot T(n/b) + f(n)$. Master method finds the solutions without substituting the values of T(n/b). In the above equation,
  n = Size of the problem
  a = Number of sub problems created in recursive solution.
  n/b = Size of each sub problem
  f(n) = work done outside recursive call.

This includes cost of division of problem and merging of the solution.

Example:

Given: $T(n) = 8T(n/2) + n^2$

Compare this equation with $T(n) = a\,T(n/b) + f(n)$

Here, a = 8, b = 2 and $f(n) = n^2$

Checking the relation between a and b^2

As $a > b^2$

i.e $8 > 2^2$, **Solution for this equation is given as,**

$$T(n) = \left(n^{log_b^a}\right) = \theta\left(n^{log_2^8}\right) = \theta\left(n^{log_2^{2^3}}\right) = \theta\left(n3^{log_2^2}\right) = \theta\left(n^3\right)$$

Q 2

A) write an algorithm for finding minimum and maximum using divide and conquer. Also derive its complexity. 10M

Algorithm: **Algorithm DC_MAXMIN (A, Low, high)**

```
If n = = 1 then
```

```
return (A[1], A[1])
else if n = = 2 then
if A[1] < A[ 2] then
return (A[1], A[2])
else
return (A[2], A[1])
else
mid ← (low + high)/2
[LMin, LMax]= DC_MAXMIN(A, low,mid)
[RMin, RMax]= DC_MAXMIN(A, mid + 1, high)
If LMax > RMax then
Max ← LMax
Else
Max ← RMax
End
If LMin < RMin then
min← LMin
else
min←RMin
end
return (min, max)
end
```

Complexity of algorithm:

- DC_MAXMIN does two comparisons two determine minimum and maximum element and creates two problems of size n/2, so the recurrence can be calculated as,

$$T(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ 2T(n/2) + T(n/2) + 2 & n>2 \end{cases}$$

When n is a power of two, $n = 2^k$
for some positive integer k, then

$$\begin{aligned} T(n) &= 2T(n/2) + 2 \\ &= 2(2T(n/4) + 2) + 2 \\ &= 4T(n/4) + 4 + 2 \\ &\;. \\ &\;. \\ &\;. \\ &= 2^{k-1}\, T(2) + \textstyle\sum(1 \le i \le k-1)\, 2^k \\ &= 2^{k-1} + 2^k - 2 \\ &= 3n/2 - 2 = O(n) \end{aligned}$$

3n/2 – 2 is the best, average, worst case number of comparison when n is a power of two.

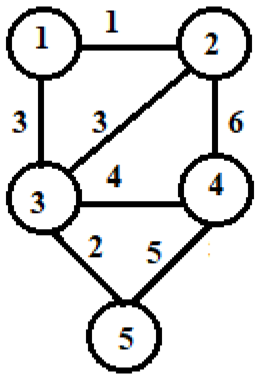
B) write Kruskal's algorithm and show it's working by taking suitable example of graph with 5 vertices.

10M

```
Algorithm KRUSKAL_MST(G)
A = Ø
Foreach  v ∈ G . V:
MAKE-SET (v)
Foreach ( u, v) in G . E ordered by weight (u, v), increasing:
if FIND-SET (u) ≠ FIND-SET(V):
A = A ∪ {(u, v)}
UNION(FIND-SET(u), FIND-SET(v))
return A
```

Example:

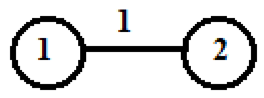
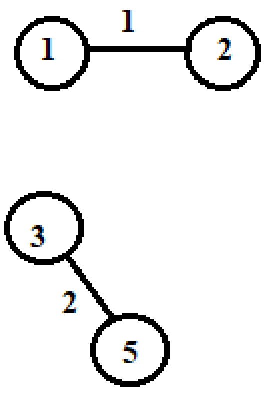
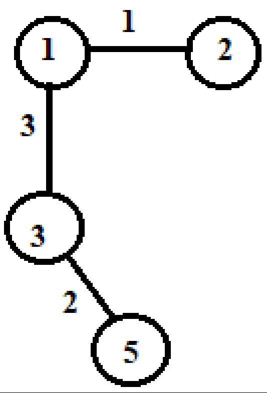
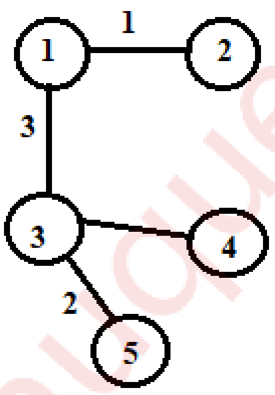
Find the cost of minimal spanning tree of the given graph by using Kruskal's Algorithm.



Solution:

Edge	<1,2>	<3,5>	<1,3>	<2,3>	<3,4>	<4,5>	<2,4>
Cost	1	2	3	3	4	5	6

- Kruskal's algorithm sorts the edges according to decreasing order of their weight. Sorted edges are listed in following table.
- One by one edge is added to partial solution if it is not forming the cycle.
- Initially, set of unvisited edges U_E = { <1,2>, <3,5>, <1,3>, <2,3>, <3,4>, <4,5>, <2,4>}

Partial solution	**Updated U_E**
	U_E = {<3,5>, <1,3>, <2,3>, <3,4>, <4,5>, <2,4>}
	U_E = {<1,3>, <2,3>, <3,4>, <4,5>, <2,4>}
	U_E = {<2,3>, <3,4>, <4,5>, <2,4>}
	U_E = {<4,5>, <2,4>}
Minimum cost edge <2,4> creates cycle so remove it from U_E	**Cost of solution:** **w(1,2)+w(1,3)+W(3,4)+w(3,5)** **=1+3+4+2 = 10**

Q 3

A) Solve fractional knapsack problem for the following.

n = 6, p= (18, 5, 9, 10, 12, 7) w = (7, 2, 3, 5, 3, 2), Max sack capacity M = 13. **10M**

Solution:

Item	Weight	Value	Value / Weight
P1	7	18	2.571
P2	2	5	2.5
P3	3	9	3
P4	5	10	2
P5	3	12	4
P6	2	7	3.5

Arrange the Item in increasing order of value / Weight Ratio

P5, P6, P3, P1, P2, P4

Capacity of Bag is 13.

Capacity of Bag	Items	value
13	-----	0
10	P5	12
8	P5, P6	7
5	P5, P6, P3	9

Now, P1 arrives but capacity of bag is only 5 and weight is 7 so we are going to take fraction

Maximum Profit = $28 + \frac{5}{7} * 18$ **= 40.852**

B) Write an algorithm for Knuth Morris Pratt (KMP) pattern matching. **10M**

i) This is first linear time algorithm for string matching. It utilizes the concept of naïve approach in some different way. This approach keeps track of matched part of pattern.
ii) Main idea of this algorithm is to avoid computation of transition function ∂ and reducing useless shifts performed in naive approach.
iii) This algorithm builds a prefix array. This array is also called as ∏ array.
iv) Prefix array is build using prefix and suffix information of pattern.
v) This algorithm achieves the efficiency of O(m+n) which is optimal in worst case.

<u>Algorithm</u> – **KNUTH_MORRIS_PRATT (T, P)**

```
n = T.length
m = P.length
∏ = Compute prefix
q ← 0
for i = 1 to n
while q > 0 and P[q+1] ≠ T[i]
    q = ∏ [q]
if P[q+1] = = T[i]
```

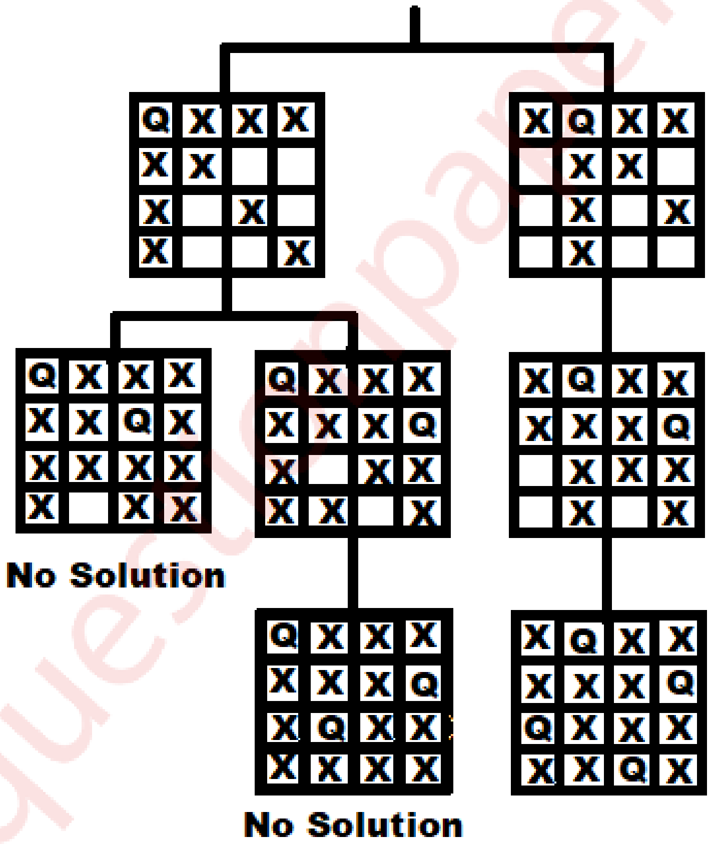
```
        q = q+1
      if q = = m
    Print “pattern found”
       q = ∏ [q]

   COMPUTE_PREFIX (P)
   M = P.length
   Let ∏ [1……m] be a new array
     ∏ [1] = 0
     K = 0
   for k = 0 to m
       while k > 0 and P[k+1] ≠ T[q]
         k = ∏ [k]
        if P[k+1] = = T[q]
         k = k + 1
       ∏ [q] = k
        return ∏
```

Q 4

A) Write an algorithm to solve N Queens problem. Show its working for N = 4. **10M**

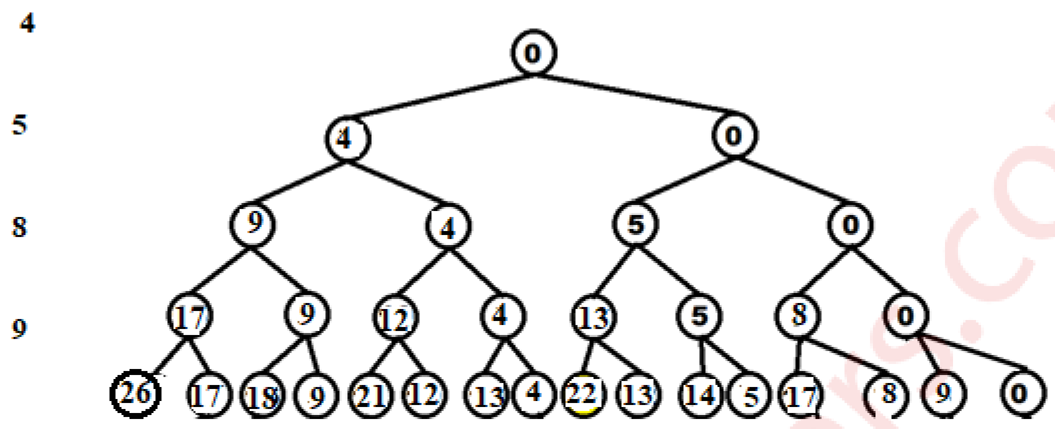
```
Algorithm Queen (n)
         for column ← 1 to n do
          {
          if (Place(row, column)) then
           {
           Board [row]=column;
           if (row = = n) then
              Print_board (n)
           else
             Queen(row+1, n)
           }
        }

       Place(row, column)
        {
         for i ← row -1 do
          {
           if (board [i] = column) then
             return 0;
           else if (abs(board [i] = column)) = abs(i-row) then
             return 0;
          }
        return 1;
      }
```

i) n-queens problem is a problem in which n-queens are placed in n*n chess board, such that no 2 queen should attack each other.
ii) 2 queens are attacking each other if they are in same row, column or diagonal.
iii) For simplicity, State space tree is shown below. Queen 1 is placed in a 1st column in the 1st row. All the position is closed in which queen 1 is attacking. In next level, queen 2 is placed in 3rd Column in row 2 and all cell are crossed which are attacked by already placed 1 and 2. As can be seen below. No place is left to place neat queen in row 3, so queen 2 backtracks to next possible and process continue.
iv) In a similar way, if (1, 1) position is not feasible for queen 1, then algorithm backtracks and put the first queen cell (1, 2), and repeats the procedure. For simplicity, only a few nodes are shown below in state space tree.



v) Complete state space tree for the 4 – queen problem is shown above. 2 – queen problem is not feasible.
vi) This is how backtracking is used to solve n-queens problems.

B) Write an algorithm to solve sum of subset problem and solve the following problem. n =4, w = {4, 5, 8, 9}, required sum = 9. 10M



Solutions : Subset **1) {4, 5}**

<u>**Algorithm**</u>**:**

Let W be a set of elements & M be the expected sum of subset then

Step1: Start with empty set.

Step2: Add to the subset, the next element from the list.

Step3: If the subset is having sum equal to M then Stop with that subset as solution,

Step4: If the subset is not matching with the M or if we have reached to the end of the set Then backtrack through that subset until we find suitable value.

Step5: If the subset is less then M then repeat Step2.

Step6: If we have visited all the elements without finding suitable & No backtrack is possible Then stop without solution.

Time Complexity – O(2^n)

Q 5

A) Prove that Vertex Cover problem is NP Complete. 10M

Given that the Independent Set (IS) decision problem is N P-complete, prove that Vertex Cover (VC) is N P-complete.

Solution: 1. Prove Vertex Cover is in N P. I Given VC ,

vertex cover of G = (V, E), |VC | = k I We can check in O(|E| + |V|) that VC is a vertex cover for G.

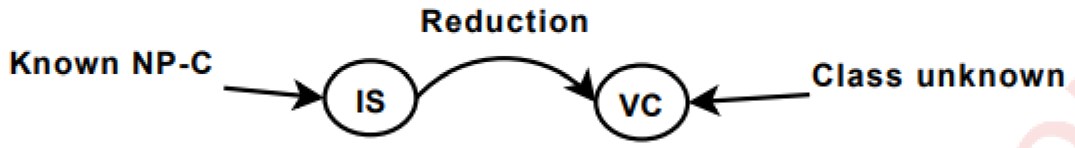
For each vertex ∈ VC , remove all incident edges.

Check if all edges were removed from G.

Thus, Vertex Cover ∈ N P

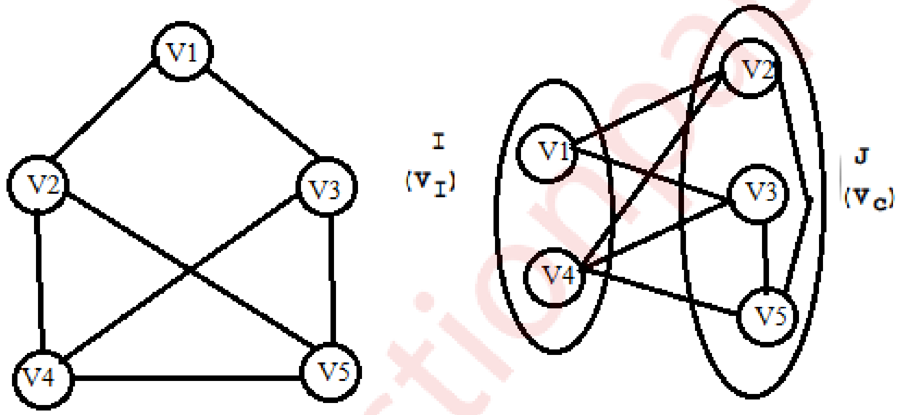
Select a known N P-complete problem.

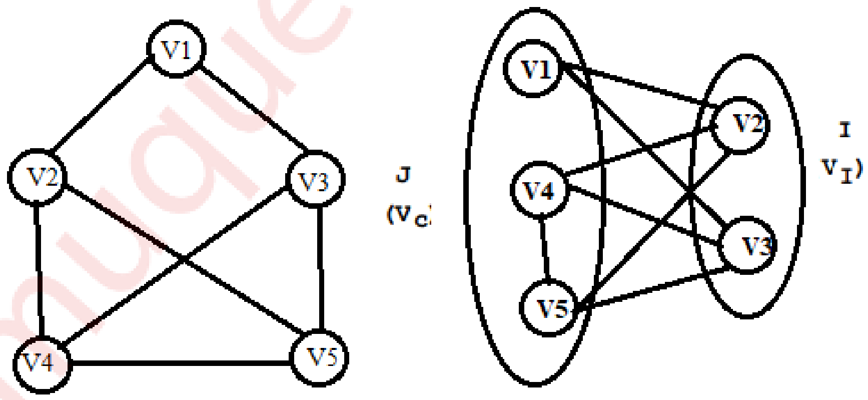
- Independent Set (IS) is a known N P-complete problem.
- Use IS to prove that VC is N P-complete.



3. Define a polynomial-time reduction from IS to VC:

- Given a general instance of IS: G 0=(V 0 , E 0), k 0
- Construct a specific instance of VC: G=(V, E), k I V =V 0 I E=E 0 I (G=G 0) I k=|V 0 | − k 0
- This transformation is polynomial:
- Constant time to construct G=(V, E)
- O(|V|) time to count the number of vertices
- Prove that there is a VI (|VI | = k 0) for G 0 iff there is an VC (|VC | = k) for G.





B) Find the longest common subsequence for the following two strings.

X = ABACABBY = BABCAB **10M**

i) The longest common sequence is the problem of finding maximum length common subsequence from given two string A and B.
ii) Let A and B be the two string. Then B is a subsequence of A. a string of length m has 2^m subsequence.
iii) This is also one type of string-matching technique. It works on brute force approach.
iv) Time complexity = O(m*n)

Algorithm **LONGEST_COMMON_SUBSEQUENCE (X, Y)**

```
// X is string of length n and Y is string of length m
for i ← 1 to m do
   LCS [i, 0] ← 0
   end
for j ← 0 to n do
    LCS [0, j] ← 0
   end
for i ← 1 to m do
   for j ← 1 to n do
     if X_i = = Yj then
        LCS [i, j] = LCS [i-1, j-1] +
    else if LCS [i-1, j] ≥ LCS [i, j-1]
        LCS [i, j] = LCS [i-1, j]
    else
        LCS [i, j] = LCS [i, j-1]
   end
  end
 end
end
return LCS
```

Let us consider P=(A, B, A, C, A, B, B, Y) and Q=(B, A, B, C, A, B)

Q $\longrightarrow$

P $\downarrow$

			1	2	3	4	5	6
			B	A	B	C	A	B
		0	0	0	0	0	0	0
1	A	0	0	1	1	1	2	2
2	B	0	1	1	2	2	2	3
3	A	0	1	2	2	2	3	3
4	C	0	1	2	2	3	3	3
5	A	0	1	3	3	3	4	4
6	B	0	2	3	4	4	4	5
7	B	0	3	3	5	5	5	6
8	Y	0	3	3	5	5	5	6

LCS : (A,B, C,Y)

Q 6

A) Assembly Line Scheduling **10M**

- Assembly line scheduling is a manufacturing problem. In automobile industries assembly lines are used to transfer parts from one station to another station.

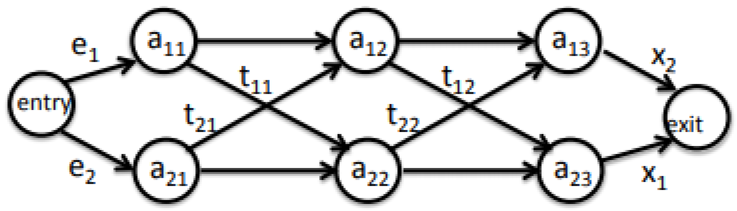
- Manufacturing of large items like car, trucks etc. generally undergoes through multiple stations, where each station is responsible for assembling particular part only. Entire product be ready after it goes through predefined n stations in sequence.

- Manufacturing of car may be done through several stages like engine fitting, coloring, light fitting , fixing of controlling system, gates, seats and many other things.

- The particular task is carried out at the station dedicated to that task only. Based on the requirement there may be more than one assembly line.

- In case of two assembly lines if the load at station j at assembly 1 is very high, then components are transfer to station j of assembly line 2 the converse is also true. This technique helps to speed ups the manufacturing process.

- The time to transfer partial product from one station to next station on the same assembly line is negligible. During rush factory may transfer partially completed auto from one assembly line to another, complete the manufacturing as quickly as possible.



B) Job Sequencing with deadlines **10M**

i) Job sequencing is a problem in which n jobs are arranged in such a way that we get maximum Profit. The job should complete their task within deadlines.
ii) It is also called as Job scheduling with deadlines.
iii) Time complexity = $O(n^2)$

Algorithm

Step1: Sort the jobs J_i into non-increasing order.
Step2: Assign Empty slot as Maximum deadline value i.e. Empty slot = Dmax.
Step3: Take i index value compute value of k
K = min (Dmax, deadline(i))
Step4: If value of K is greater and equal to one check empty slot. Empty slot = k
If empty slot is full. check previous slot if it is free assign job to that slot.
If slot is full ignore that job.
Step5: increment value of i till all the jobs are not finished. Repeat Step3.
Step6: Stop.

Example

Let n=4, (J1, J2, J3, J4)=(100, 10, 15, 27), (D1, D2, D3, D4)=(2, 1, 2, 1) find feasible solution. With job scheduling with deadlines.

Index	1	2	3	4
Job	J1	J2	J3	J4
Value	100	10	15	27
Deadlines	2	1	2	1

Arrange the jobs in non-increasing value.

Index	1	2	3	4
Job	J1	J4	J3	J2
Value	100	27	15	10
Deadlines	2	1	2	1

Dmax = 2

Time slot	1	2
Status	Empty	Empty

i=1
K= min (Dmax, Deadline(i))
K= min (2, 2)
K=2 (k ≥ 1)
Time slot = k = 2 (empty)

Time slot	1	2
Status	Empty	J1

i=2
K= min (Dmax, Deadline(i))
K= min (2, 1)
K=1 (k ≥ 1)
Time slot = k = 1 (empty)

Time slot	1	2
Status	J4	J1

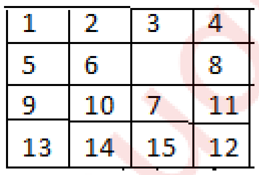
Maximum Profit = J1 + J4
= 27 + 100
= **127**

C) 15 Puzzle Problem **10M**

- In this problem there are 15 tiles, which are numbered from 0 – 15.

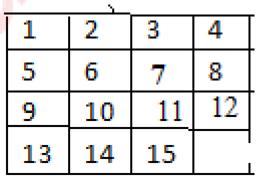
- The objective of this problem is to transform the arrangement of tiles from initial arrangement to a goal arrangement.

- initial and goal arrangement



1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

a) Initial state



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

b) Goal state

- There is always an empty slot in the initial state.
- In the initial state moves are the moves in which the tiles adjacent to ES are moved to either left, right, up or down.
  - Each move left, right, up and down creates a **new** arrangement in a tile.

- These arrangements are called different states of the puzzle.
- The initial arrangement is called as initial state and goal arrangement is called as goal state.
- The state space tree for 15 puzzle is very large because there can be 16! Different arrangements.
- In state space tree, the nodes are numbered as per the level.
- Each next move is generated based on empty slot positions.
- Edges are label according to the direction in which the empty space moves.
- The root node becomes the E – node.
- The child node 2, 3, 4 and 5 of this E – node get generated.
- Out of which node 4 becomes an E – node. For this node the live nodes 10, 11, 12 gets generated.
- Then the node 10 becomes the E – node for which the child nodes 22 and 23 gets generated.
- We can decide which node become an E – node based on estimation formula.
- Cost estimation formula
  C (x) = f (x) + G (x)

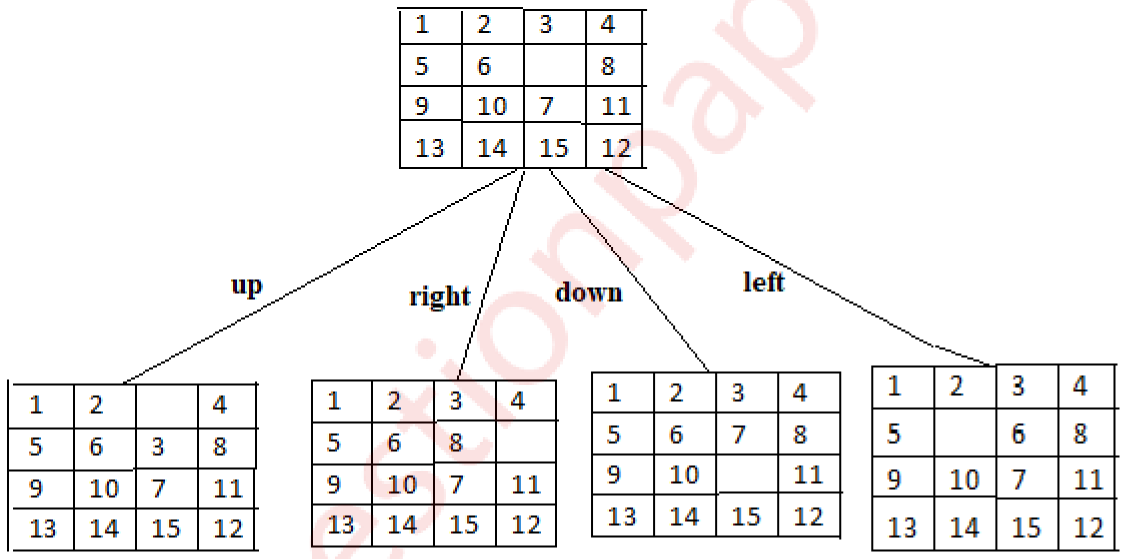


Fig. Cost f(x) + h(x) after first move

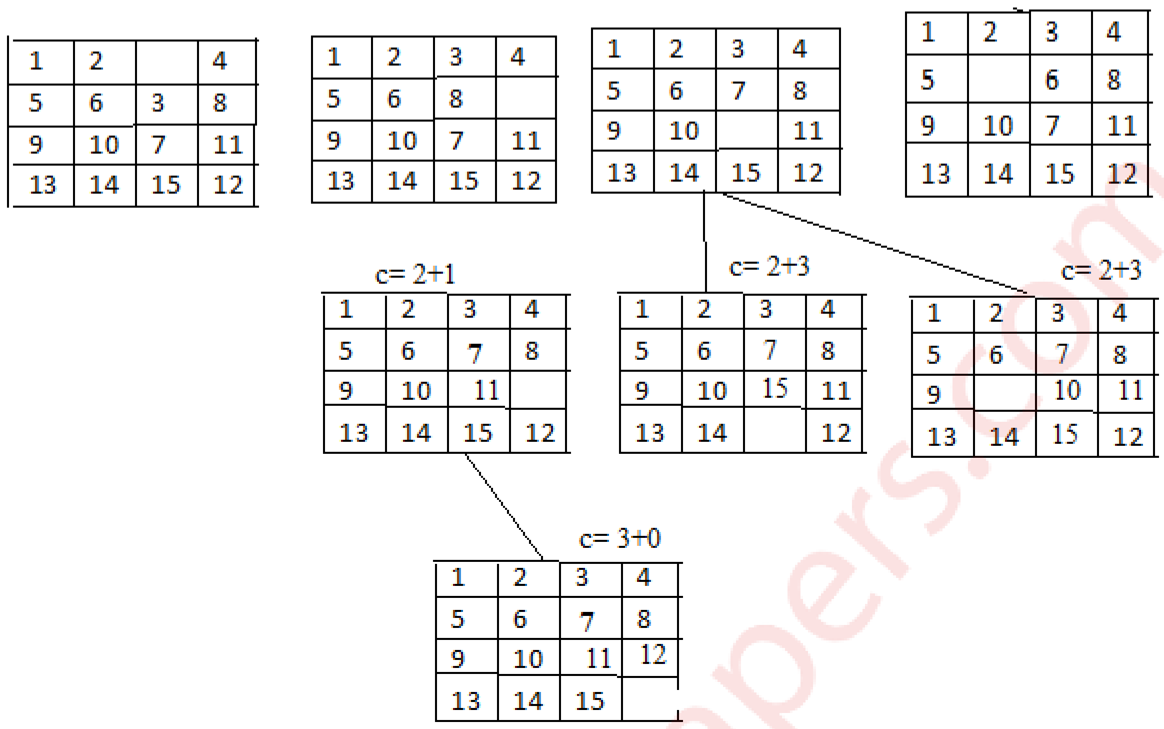


Fig. Tile positions after subsequent moves

D) P, NP, and NPC Classes **10M**

P:

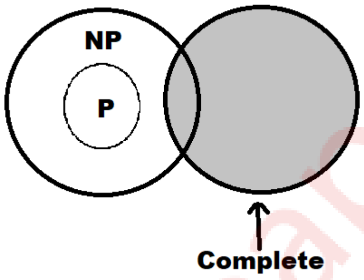
- Polynomial time solving .
- Problems which can be solved in polynomial time, which take time like O(n), O(n2), O(n3).
- Example: finding maximum element in an array or to check whether a string is palindrome or not.
- so there are many problems which we can solved in polynomial time.
- P problems are set of problems that can be solved in polynomial time by deterministic algorithm.
- P is also known as PTIME and DTIME complexity class.
- Example of P problem: Insertion sort, Merge sort, Linear search, Matrix multiplication.
-

NP:

- Non deterministic Polynomial time solving. Problem which can't be solved in polynomial time like TSP(travelling salesman problem) or An easy example of this is subset sum: given a set of numbers, does there exist a subset whose sum is zero?
- But NP problems are checkable in polynomial time means that given a solution of a problem ,

NP-Complete:

- The group of problems which are both in NP and NP-hard are known as NP-Complete problem.
- Now suppose we have a NP-Complete problem R and it is reducible to Q then Q is at least as hard as R and since R is an NP-hard problem. therefore Q will also be at least NP-hard , it may be NP-complete also.
- NP complete is the combination of both NP and NP hard problem.
- Decision problem C is called NP complete if it has following two properties.
- C is in NP, and
- Every problem X in NP is reducible to C in polynomial time, i.e. For every X ε NP, X $\leq_p$ C
  This two factor prove that NP-complete problems are the harder problems in class NP.
  They often referred as NPC.



NP Complete

****