Winter : 2020

Q.1

(a) What is algorithm ? Why analysis of algorithm required ?

→ A process or a set of rules to be followed to achieve desired output, especially by a computer.

→ A step by step procedure to solve the different kinds of problem

→ An unambigeous sequence of computative step that transforms the input into the output.

→ An algorithm is any well defined computational procedure that takes some values or set of values as input and produce some value or set of value as a output.

* Why it is required ?

→ To understand the basic idea of problem.

→ To find an approach to solve problem.

→ To improve effciency of existing techniques.

→ To understand basic principle of designing the algorithm.

→ To compare the performance of algorithm with respect to other technique.

→ Algorithm gives a clear description of requirments and goals of the problem to designer.

→ A good design can produce good solution.

→ To understand the flow of the problem.

→ To measure behaviour of the method in all cases.

→ With the help of algorithm, we convert art into science.

(b) Explain about asymptotic notation.

-> Asymptotic notation are language that allows us to analyze an algorithm at running time. by identifying it's behaviour as input size of algorithm is increased. This is also known as an algorithm growth rate.

* Used

-> To characterize the complexity of an algorithm.

-> To compare the performance of two or more algorithm solving the same problem.

* Types :

-> (i) O - Notation (Big - O notation) [wrost case]
(ii) Ω - Notation (omega - notation) [best case]
(iii) Θ - Notation (Theta - notation) [average case]

(i) O - Notation (Big - oh notation)

-> Big - O notation represent the upper bound of the running time algorithm.

-> It gives worst - case complexity of algorithm.

-> It is measure of the longest ammount of time.

→ Big - O is the formal method of expressing the upper bound of algorithm.

→ The function $f(n) = O(g(n))$ as, "f of n is big-O of g of n" If and If only exist positive constant C and such that,

$$0 \leq f(n) \leq C\, g(n) \quad \forall \quad n \geq n_0$$

Ex. $3n^2 + 5n + 10$

→ Let, $g(n) = n^2$

$C = 18$

→ $n = 1$,

$$3(1)^2 + 5(1) + 10 \leq 18(1)^2$$

$18 \leq 18$, True, $\forall \quad n \geq 1$

It is true for all $n \geq 1$

$$f(n) = O(g(n))$$

→ $g(n)$ is an asymptotically upper bound of $f(n)$.

$f(n) = O(g(n))$ implies :

$$f(n) \leq C.g(n)$$

(ii) Ω - notation (Omega - notation)

-> Omega notation represent the lower bound of running algorithm.

-> It provides best case complexity of an algorithm.

-> Thus, function $f(n) = \Omega(g(n))$ as, "f of n is omega g of n" if and if only if there exists positive constant c and n_0 such that,

$$0 \leq c\,g(n) \leq f(n), \text{ for } \forall\ n \geq n_0$$

-> g(n) is asymptotically lower bound of f(n)

-> $f(n) = \Omega(g(n))$ implies :

$$f(n) \geq c.g(n)$$

Ex. $f(n) = 3n + 2$

-> $g(n) = n$

$n^c = 1$

-> for, $n = 1$,

$3n + 2 \geq c \cdot g(n)$	$1.(1) \leq 3 + 2$
$3 + 2 \geq 1$	$1 \leq 5$
$5 \geq 1$ $(f(n) \geq^c g(n))$	$(c.g(n) \leq f(n))$

-> It is true for all $n \geq 1$

(iii) Θ - notation (Theta notation)

→ It is represent upper and lower bound of running algorithm.

→ It is used to analyze the average case complexity of an algorithm.

→ The function $f(n) = \Theta(g(n))$ as, "f is the Θ of g of n." if and if only there exists positive constant c_1, c_2 and n_0 such that,

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0$$

→ Ex. $f(n) = 3n + 2$

→ $c_1 = 1$, $c_2 = 4$, $g(n) = n$

→ For , $n = 1$

$$1 \cdot n \leq 3 + 2 \leq 4$$
$$1 \leq 5 \leq 4$$

∴ False

→ For , $n = 2$

$$2n \leq 6 + 2 \leq 8$$
$$4 \leq 8 \leq 8 \quad \text{, True}$$

So, $f(n) = \Theta(g(n))$, $\forall\ n \geq 2$, $c_1 = 1$, $c_2 = 4$.

(c) Write an algorithm for insertion sort. Analyze insertion sort algorithm for all cases.

→ The insertion algorithm works by insert an element to it's appropriate position during each iteration.

→ The insertion sort comparing an element to all its previous elements untill an appropriate position is not found.

* Algorithm

```
for j = 2 to A.length
    key = A[j]
    i = j - 1
    While i > 0 and A[i] > key
        A[i+1] = A[i]
        i = i - 1
    A[i+1] = key
```

* Cases

→ Best case (Ω) : If the given number are sorted then the running time is : $O(n)$

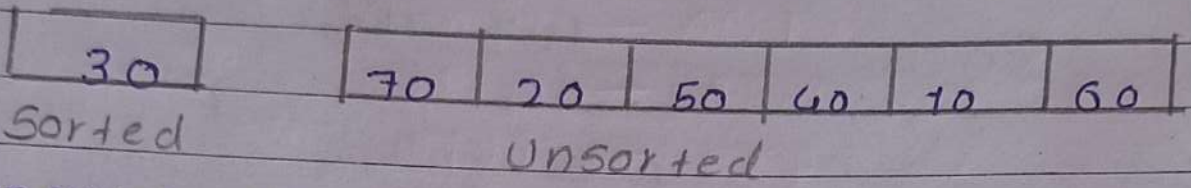
→ Average case (Θ) : If there is wide variation with running time : $\Theta(n^2)$

→ Worst case (o) : If number are sorted in reverse order : $O(n^2)$

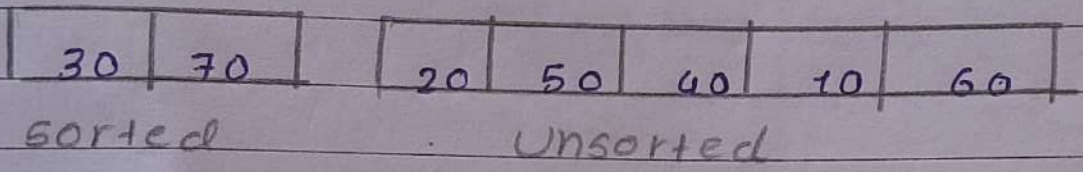
☆ Insertion Sort

-> In this method the elements are inserted at their appropriate place.

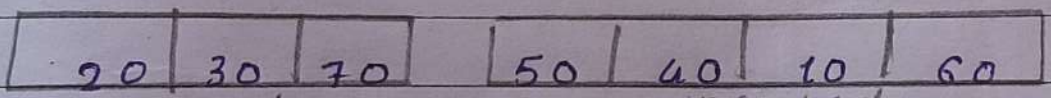
Ex. 30, 70, 20, 50, 40, 10, 60

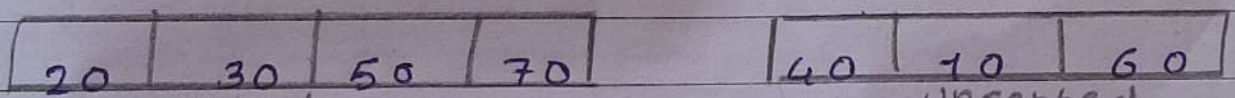


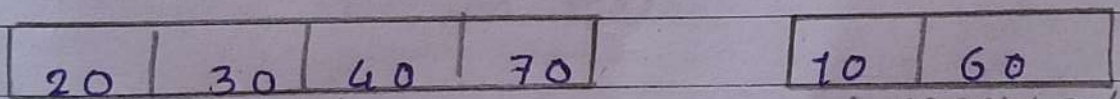
-> Compare 70 with sorted element and insert it an appropriate location.

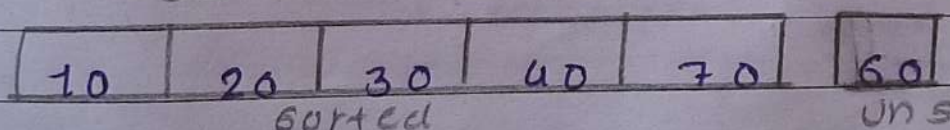


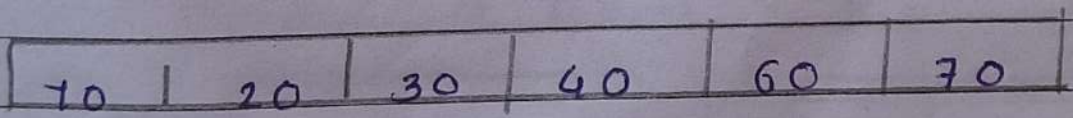
-> Compare 20 with sorted element and insert it an appropriate location.



-> 

-> 

-> 

-> 

Q. 2

(a) What is difference between selection sort and bubble sort.

->

Selection sort	Bubble sort
Selection sorting is a sorting algorithm where we select minimum element from the array and put that it's correct position.	Bubble sorting is a sorting algorithm where we check two elements and swap them at their correct position.
Time complexity is best case is $O(n^2)$	Time complexity in best case is $O(n)$
Time complexity is worst case is $O(n^2)$.	Time complexity in worst case is $O(n^2)$.
It uses selection method.	It uses exchanging method
It is efficient technique.	It is not effcient technique.
This method is faster.	This method is slower.

(b) Write iterative and recursive algorithm for finding the factorial N. Derive time complexity of both algorithms.

→ * Recursive

```
int Fact (int n)
  {
    if (n == 1)
      return 1;
    else
      return (n * Fact(n-1));
  }
```

Time complexity : O(n)

* Iterative

```
int fact (int n)
  {
    int res = 1, i;
    for (i = 2; i <= n; i++)
      res *= i;
    return res;
  }
```

Time complexity : O(n)

Solve following recurrence relation using itearative method. [$T(n) = 2T(n/2) + n$]

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

$T(n/2) = 2T(n/4) + n/2$

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

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

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

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

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

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

$\vdots$

$= 2^k T\left(\frac{n}{2^k}\right) + k.n$

Let, $n/2^k = 1$,

$n/2^k = 1 \quad \therefore n = 2^k \quad \therefore \log n = \log 2^k$

$\therefore \log_2 n = k \log_2 2$

$\therefore \log n = k$

$= n \cdot 1 + n \log n$

$= O(n \log n)$

Q.3

(a) Explain divide and conquer approach.

→ Many useful algorithm are recursive in structure : To solve given problem they call themselves recursively one or more time.

→ Thus algorithm follow a divide and conquer approach.

→ The divide and conquer approach includes three step at each level.

(i) Divide : Break the problem into serval sub problems that are similar to original problem but small in size.

(ii) Conquer : Solve the sub problem recursively if the sub problem sizes are small engough Solve sub problem in a straight forward manner.

(iii) Combine : Combine the solution of sub problem to create solution of original problem.

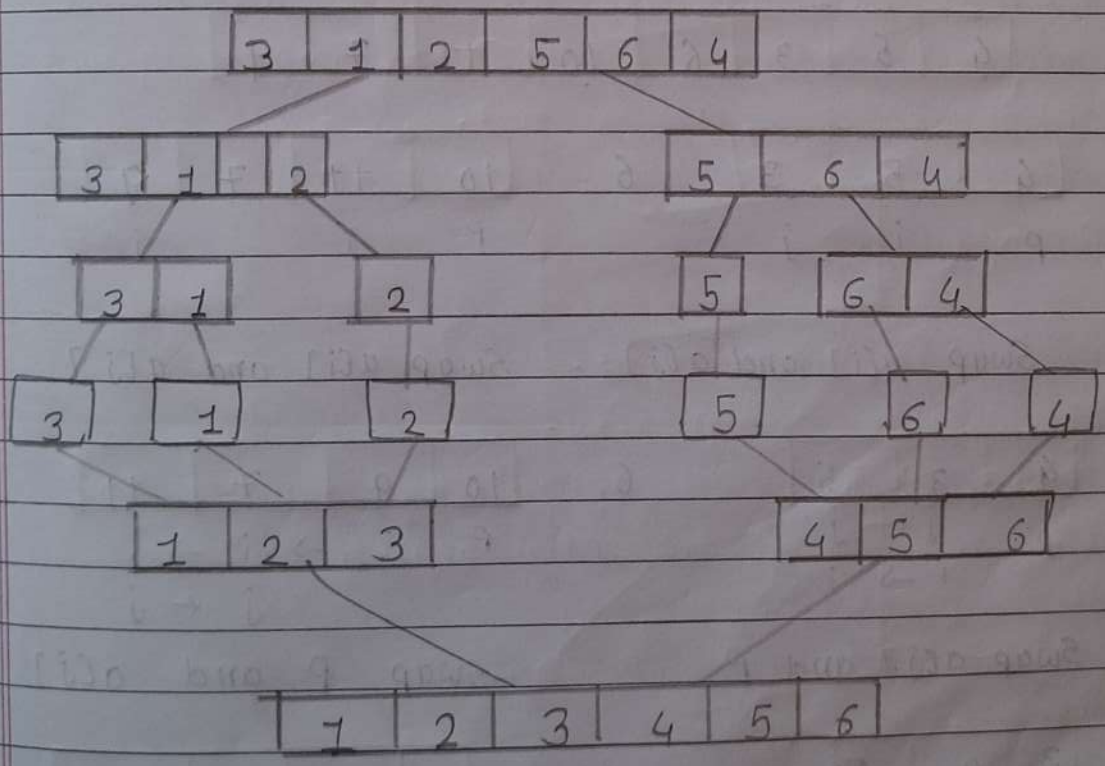
* Advantage

- Difficult problem can be solved easily.
- Efficiently use cache memory without occupying much space.
- Reduce time complexity of problem.

* Disadvantage

- It involves recursion which is sometime slow.
- Efficiency depends on implementation of logic.
- It may crash system if recursion perform rigorously.

Ex.



(b) Perform quick sort algorithm:

{6, 5, 3, 11, 10, 4, 7, 9}

→

6	5	3	11	10	4	7	9
p	i						j

Step 2:

6	5	3	11	10	4	7	9
p			i		j		

Swap a[i] and a[j]

Step 3:

6	5	3	4	10	11	7	9
p			i				
			j				

Swap p and a[j]

Step 4:

4	5	3	6	10	11	7	9

Step 5:

4	5	3		6		10	11	7	9
p	i	j				p	i		j

Swap a[i] and a[j] Swap a[i] and a[j]

Step 6:

4	3	5		6		10	9	7	11
p	j ← j					p	i → i	→ i	
	i → i							j ← j	

Swap a[j] and p Swap p and a[j]

Step 7:

3	4	5		6		7	9	10	11

Step 8:

3	4	5	6	7	9	10	11

☆ Quick Sort

→ Quick sort is a sorting algorithm that uses the divide and conquer strategy.

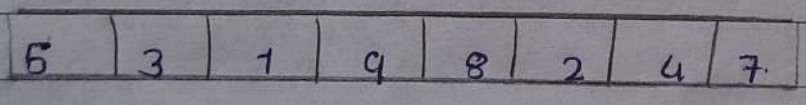
→ Divide : Split the array into two sub array and each element in left sub array is less than or equal to middle and right sub array is greater than or equal to middle element.

Conquer : Recursively sort two sub array.

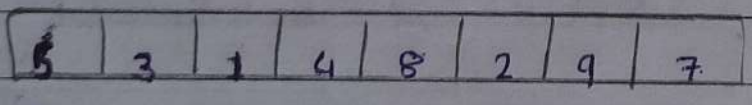
Combine : Combine all sorted element in a group to form a list of sorted element.

Ex. 5, 3, 1, 9, 8, 2, 4, 7

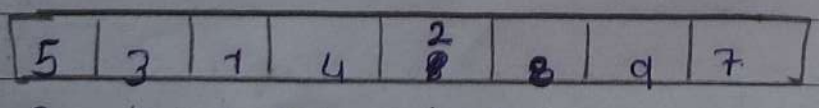
→ Step 1 : arrange element in the array



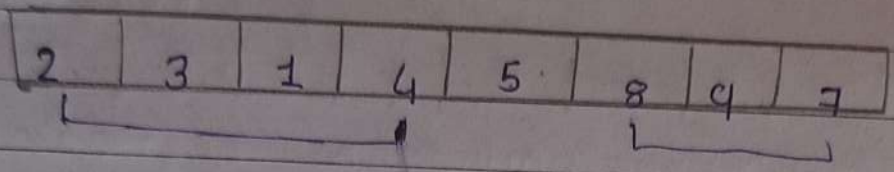
Exchange a[i] and a[j]



Exchange a[i] and a[j]



Exchange p and a[j]



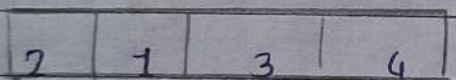
Left sub list Right sub list

→ Step 2 :

Left sub-list

p i j ← j

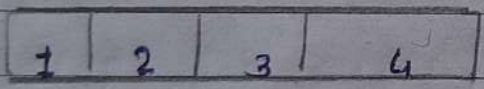
Swap a[i] and a[j]



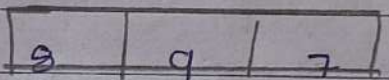
p i j ← j

j

Swap a[j] and p

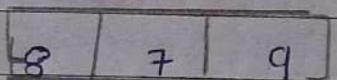


Right sub list



p i j

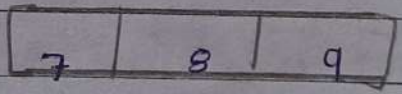
Swap a[j] and a[i]



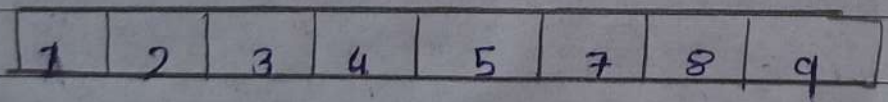
p i → i

j ← j

Swap a[j] and p



→ Combine sub list.



(1) Explain Master theorem and solve the recurrence relation using master method.
$T(n) = 9T(n/3) + n$

→ $T(n) = aT(n/b) + f(n)$

- n is the size of the problem.
- a is number of sub problem in recursion.
- n/b size of each sub problem.
- $f(n)$ is the sum of the work done outside the recursive call, which includes sum of dividing the problem and combine the solution of sub problem.
- It is not possible always bound the function according to the requirment,
- So we make three cases which tell us what kind of bound we can apply on the function

$T(n) = \Theta(n^d)$ if $a < b^d$
$T(n) = \Theta(n^d \log n)$ if $a = b^d$
$T(n) = \Theta(n^{\log_b a})$ if $a > b^d$

Ex. $T(n) = 9T(n/3) + n$

→ Here, $a = 9$, $b = 3$, $d = 1$
$b^d = 3$
So, $a > b^d$

$T(n) = \Theta(n^{\log_b a})$
$= \Theta(n^{\log_3 9})$
$= \Theta(n^2)$

Q.4

(a) Write characterstics of greedy algorithm.

-> First we select some solution from input domain.

-> Then we check whether solution is possible or not.

-> From the set of possible solution, particular solution that satisfies or nearly satisfy the objective of function. Such a solution is called optimal solution.

-> As greedy method work in stages. At each stage only one input is considered at each time. Based on this input we decide whether particular input gives optimal solution or not.

* Advantage

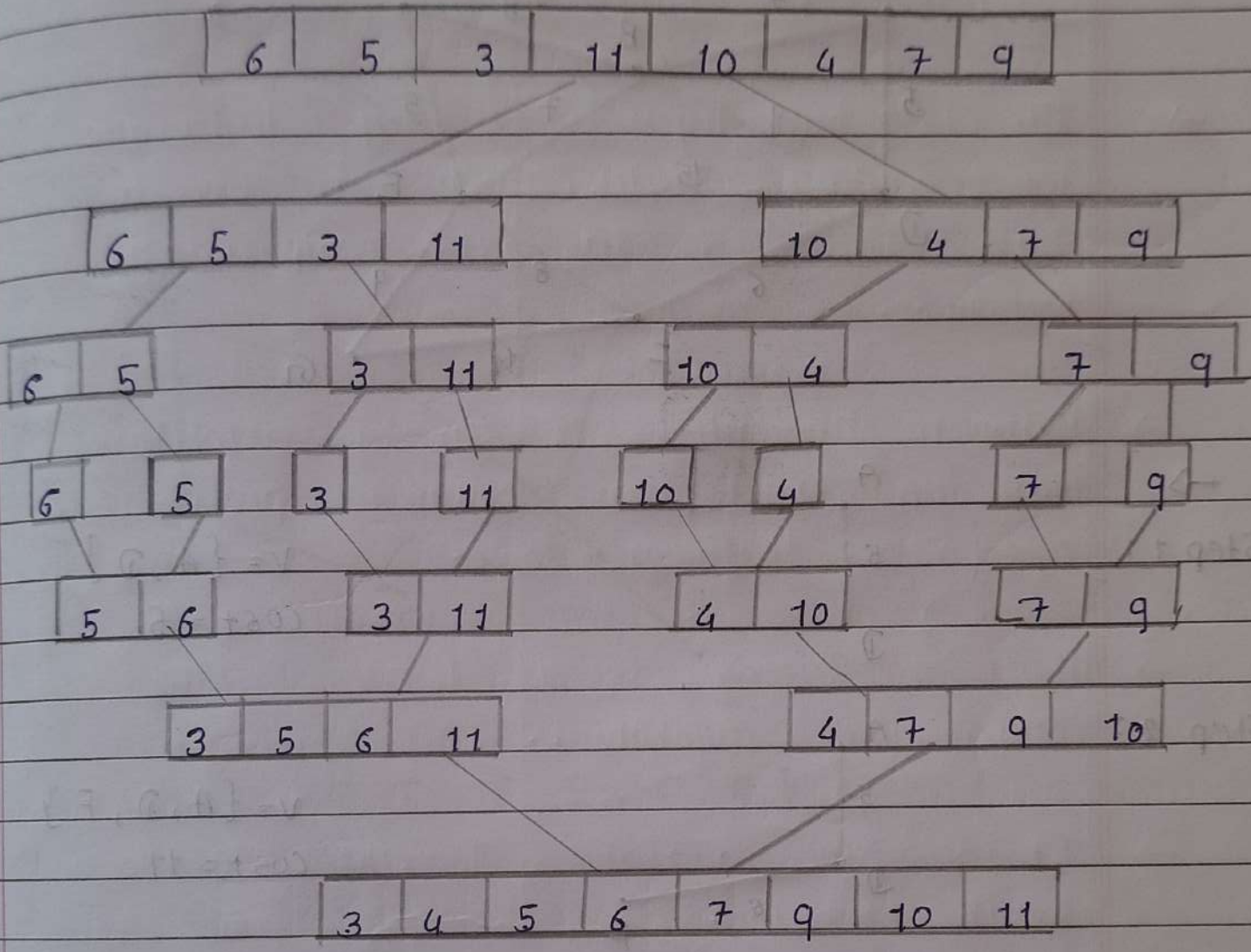
- Easy to implement.
- Less time complexity.
- It can be used optimizing purpose or finding close to optimization in case of hard problems.

* Disadvantage

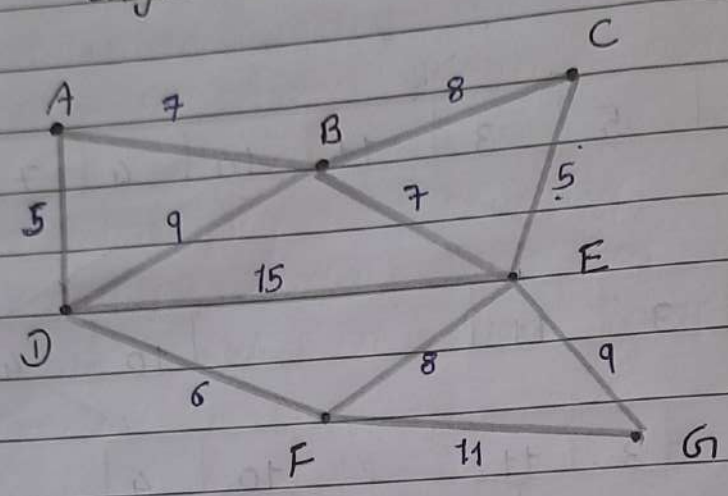
- Local optimal solution may not be globally optimal

(b) Trace a merge sort for data.
$A = \{6, 5, 3, 11, 10, 4, 7, 9\}$

→

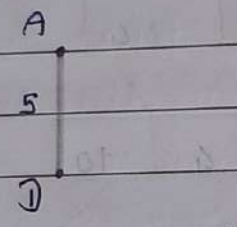


(c) Find minimum spanning tree using prims algorithm.



→

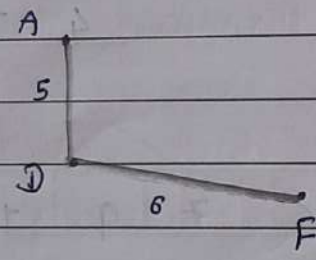
Step 1:



$V = \{A, D\}$

$cost = 5$

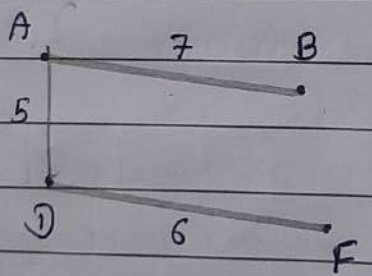
Step 2:



$V = \{A, D, F\}$

$cost = 11$

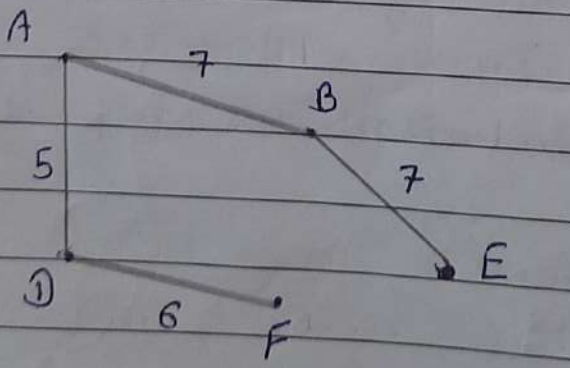
Step 3:



$V = \{A, D, F, B\}$

$cost = 18$

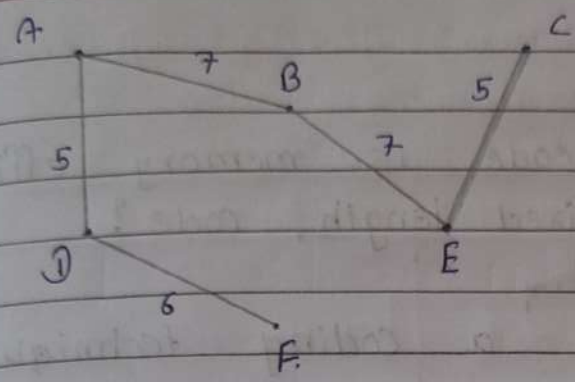
Step 4:



$V = \{A, D, F, B, E\}$

$cost = 26$

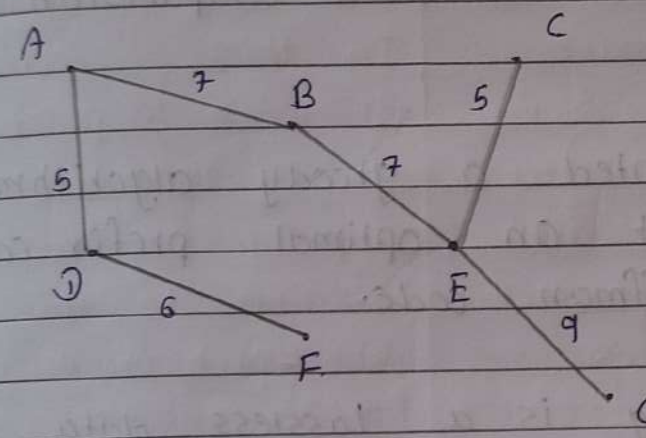
Step 5:



V = { A, D, F, B, E, C }

cost = 31

Step 6:



V = { A, D, F, B, E, C, G }

cost = 40

-> Extra

Prim's algorithm	Kruskal's algorithm
This algorithm is used for obtaining minimal spanning tree by selecting the adjacent vertex of already selected verticies.	This algorithm is used to obtaining minimal spanning tree but it is not necessary to choose adjacent verticies of already selected.

Q.5

(a) How Huffman code is memory efficient compare to fixed length code?

→ It is basically a coding technique for encoding data. Such an encode data is used in data comparision technique.

→ Huffman invented a greedy algorithm that construct an optimal prefix code called a Huffman code.

→ Huffman coding is a lossless data comparision algorithm.

→ It assign variable length codes to input character.

→ It assign ~~variable~~ length of codes which are based on the frequency of corresponding characters.

→ The ~~most~~ least frequent character gets the smallest code the large frequent character gets the largest code.

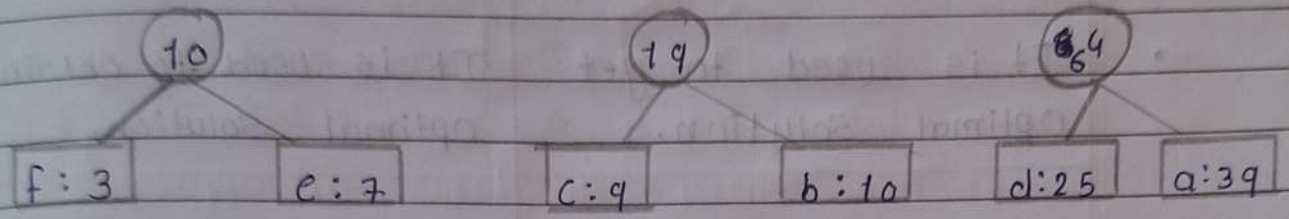
→ Variable length code assigned to input character are prefix code.

Ex. a : 39 , b : 10 , c : 9 , d : 25 , e : 7 , f : 3

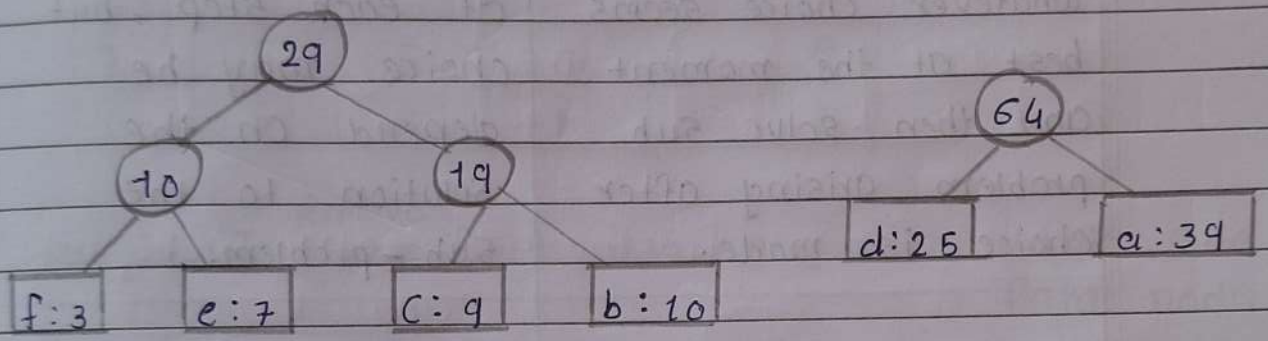
→ f : 3 , e : 7 , c : 9 , b : 10 , d : 25 , a : 39

Step 1:

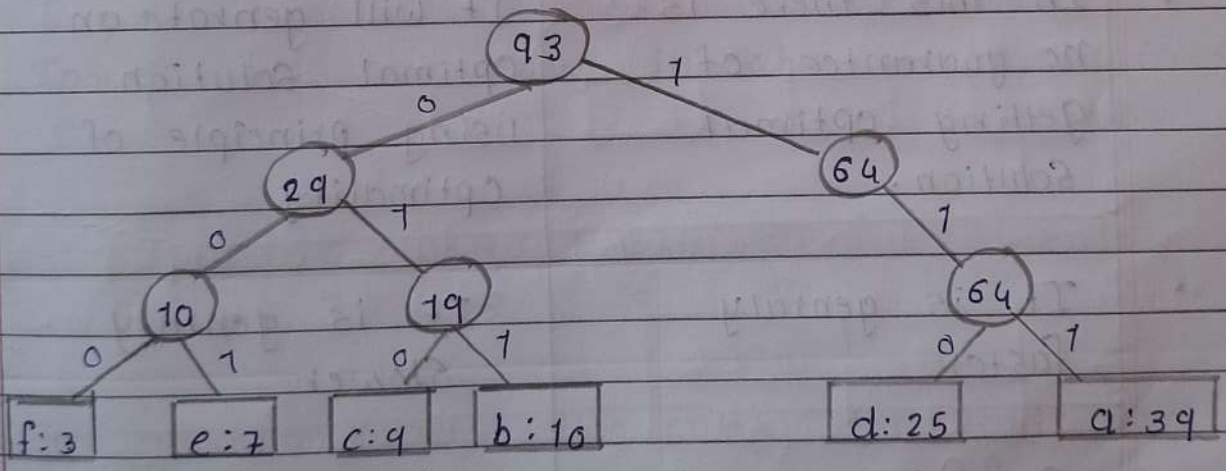
Step 2:



Step 3:



Step 4:



a	111
b	011
c	010
d	110
e	001
f	000

(b) Difference between greedy approach and dynmic programing

→	Greedy Method	Dynamic programming
•	It is used to get optimal solution.	It is used to obtain optimal solution.
•	In this, we make whatever choice seems best at the moment and then solve sub problem arising after choice is made.	In this, we choose at each step, but choice may be depend on the solution to the sub-problem.
•	It is more efficient.	less efficient
•	In this there is no guarantee of getting optimal solution.	It will genrate an optimal solution using principle of optimality
•	It is genraly faster	It is genraly slower.
•	Ex. Prim's algorithm	Ex. 0/1 knapsack

(c) Find the Huffman code for each symbol
A B C C D E B A B F F B A C B E B D F A A A A B C D E E D C C B F E B F C A E

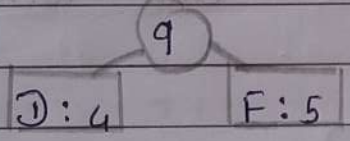
→ Step 1: Count frequency of each symbol

A	B	C	D	E	F
8	9	7	4	6	5

→ Step 2: Arrang in ascending order of frequncies

D	F	E	C	A	B
4	5	6	7	8	9

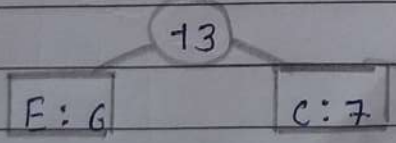
→ Step 3: Remove first two entry from table and form node



Insert this new entry in table at appropriate position

F	C	A	B	D F
6	7	8	9	9

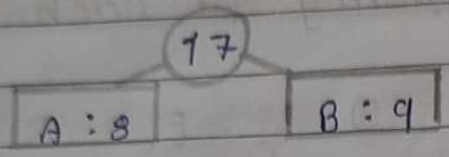
→ Step 4: Remove first two entry from table and form node.



Insert this new entry in table at appropriate position

A	B	D F	E C
8	9	9	13

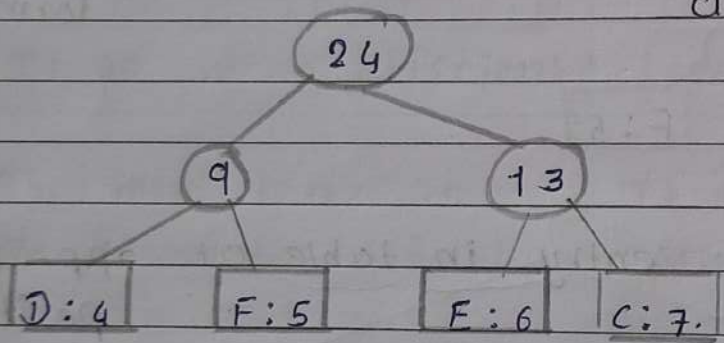
-> Step 5 : Remove first two entry from table and form node.



Insert new entry in table at appropriate location.

DF	EC	AB
9	13	17

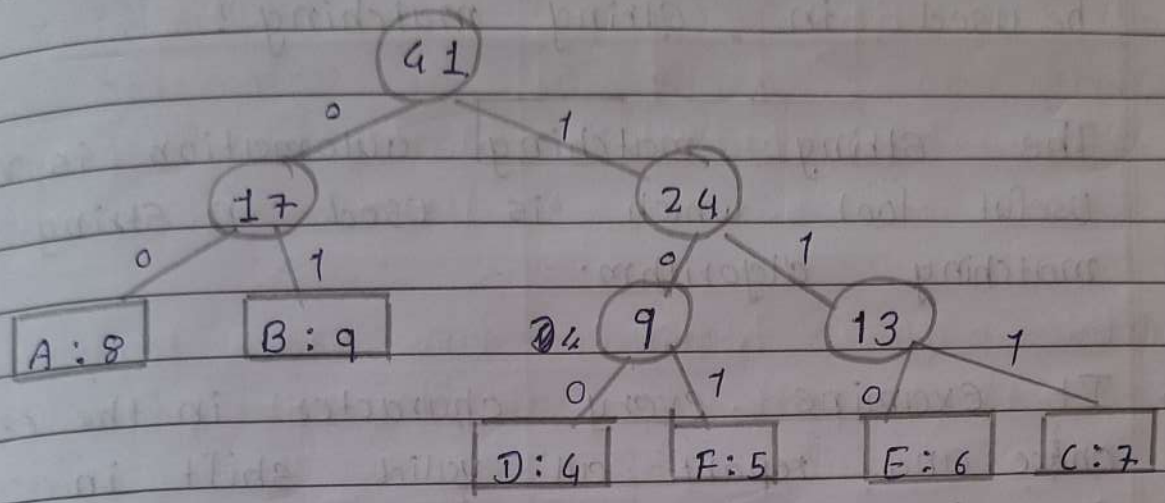
-> Step 6 : Remove first two entry from table and form node



Insert new entry in table at appropriate position.

AB	DFEC
17	24

Step 7: Remove first two entry from table and form node



Letter	code
A	00
B	01
C	111
D	100
E	110
F	101

Q.67

(a) What is finite automate? How it can be used in string matching?

→ The string matching automation is very useful tool which is used in string matching algorithm.

→ It examine every character in the exactly once and report all valid shift in O(n) time.

→ The goal of string matching is to find location specific text pattern within the larger body of text.

→ A finite automateion M is a 5-tuple (Q, q_0, A, E_δ), where,

- Q is a finite set of states.
- $q_0 \in Q$ is start state
- $A \subseteq Q$ is notable set of accepting state.
- Σ is finite input alphabate.
- δ is a function from $Q \times \Sigma$ int Q called transition function of M

Extra:

-> The finite automaton starts in state q_0 and reads the character of its input string one at a time.

-> If automaton is in state q and read input character a, it moves from q to state $\delta(q,a)$.

Whenever its current state q is a member of A, the machine M has accept the string read so far. An input that is not allowed is rejected.

-> The finite automaton M induces a function ϕ called final state function.

-> Function f defined as,

$$\phi(\epsilon) = q_0$$

$$\phi(wa) = \delta(\phi(w), a) \text{ for } w \in \Sigma^*, a \in \Sigma)$$

(b) Differentiate BFS and DFS

→

BFS	DFS
• Stands for bredth first search.	Stands for depth first search
• Uses queue to find shortest path.	Use stack to find shortest path.
• Better when target is closer to source.	Better is when target is far from source.
• Slower than DFS	Faster than BFS
• No backtracking is required.	Backtracking is implemented
• required more memory	Less memory required
• Traverse the tree level wise	Traverse the tree depth wise

(c) Explain Backtracking Method. What is N-Queens problem? Give solution of 4-queen problem using backtracking method.

→ Backtracking is a technique based on algorithm to solve problem. It uses recursive calling to find the solution by building a solution step by step increasing value with time.

→ It removes the solution that does not give rise to the solution of the problem based on the constraints given to solve the problem.

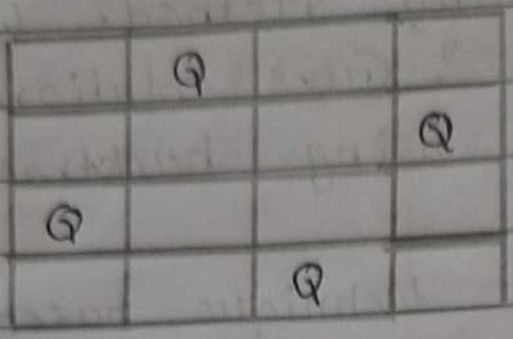
→ Backtracking algorithm is applied to some specific types of problem,

- Decision problem used to find possible solution.
- Optimisation problem used to find best solution that can be applied.
- Enumeration problem used to find set of all possible solution.

* N-Queen Problem

→ The N queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.

For example the following solution for the 4 queen problem.



→ The expected output is a binary matrix that has 1s for the blocks where queens are placed. For example, the following is the output matrix for the above 4-Queen solution.

{0, 1, 0, 0}
{0, 0, 0, 1}
{1, 0, 0, 0}
{0, 0, 1, 0}

Q.6

(a) Explain principle of optimality and describe it's use in dynamic programming.

-> The dynamic programming algorithm obtains the solution using principle of optimality.

-> The principle of optimality states that " in an optimal sequence of decision or choices, each sub sequence must also be optimal."

-> When it is not possible to apply principle of optimality it is almost impossible to obtain the solution using dynamic programming approach.

-> Ex. Finding of shortest path in a given graph uses the principle of optimality.

-> In dynamic programing method duplicate solution are totally avoided.

-> It is efficient than the divide and conquer algorithm.

-> In divide and conquer algorithm input is split at its middle point, However, in dynamic programming method splits at every possible split points. After trying each split points it determines which split point is optimal.

(b) Find out minimum number of multiplication required for multiplying.
A [7×6], B [5×4], C [4×3], D [3,2], E [2,1]

→

$$C[i,j] = \min_{i \le k < j} \{ C[i,k] + C[k+1,j] + d_{i-1} \times d_k \times d_j$$

A B C D E

$\underbrace{7 \times 5}_{d_0}$ $\underbrace{5 \times 4}_{d_1}$ $\underbrace{4 \times 3}_{d_2}$ $\underbrace{3 \times 2}_{d_3}$ $\underbrace{2 \times 1}_{d_4 \quad d_5}$

C	1	2	3	4	5
1	0	20	34	40	36
2		0	60	64	31
3			0	24	18
4				0	6
5					0

K	1	2	3	4	5
1		1	2	3	1
2			2	22	2
3				3	3
4					4
5					

→ $C[1,1] = 0$
$C[2,2] = 0$
$C[3,3] = 0$
$C[4,4] = 0$
$C[5,5] = 0$

→ $C[1,2] = \min_{1 \le k < 2} \{ C[1,1] + C[2,2] + d_0 \times d_1 \times d_2$

$= \min^{k=1}_{1 \le k < 2} \{ 0 + 0 + 1 \times 5 \times 4$

$= 20$

(b) Find out minimum number of multiplication required for multiplying.
A [1×6], B [5×4], C [4×3], D [3,2], E [2,1]

→

$$C[i,j] = \min_{i \le k < j} \{ C[i,k] + C[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

A B C D E

$\underbrace{1 \times 5}_{d_0} \quad \underbrace{5 \times 4}_{d_1} \quad \underbrace{4 \times 3}_{d_2} \quad \underbrace{3 \times 2}_{d_3} \quad \underbrace{2 \times 1}_{d_4} \quad \underbrace{}_{d_5}$

C	1	2	3	4	5
1	0	20	34	40	36
2		0	60	64	31
3			0	24	18
4				0	6
5					0

K	1	2	3	4	5
1		1	2	3	1
2			2	22	2
3				3	3
4					4
5					

→ $C[1,1] = 0$
$C[2,2] = 0$
$C[3,3] = 0$
$C[4,4] = 0$
$C[5,5] = 0$

→ $C[1,2] = \min_{1 \le k < 2} \{ C[1,1] + C[2,2] + d_0 \times d_1 \times d_2$
$= \min^{k=1}_{1 \le k < 2} \{ 0 + 0 + 1 \times 5 \times 4$
$= 20$

→ $C[2,3] = \min_{2 \le k < 3} \begin{cases} k=2 \{ C[2,2] + C[3,3] + d_1 \times d_2 \times d_3 \end{cases}$

$= \min_{2 \le k < 3} \{ k=2 \{ 0 + 0 + 5 \times 4 \times 3$

$= 60$

→ $C[3,4] = \min_{3 \le k < 4} \begin{cases} k=3 \{ C[3,3] + C[4,4] + d_2 \times d_3 \times d_4 \end{cases}$

$= \min_{3 \le k < 4} \{ k=3 \{ 0 + 0 + 4 \times 3 \times 2$

$= 24$

→ $C[4,5] = \min_{4 \le k < 5} \begin{cases} k=4 \{ C[4,4] + C[5,5] + d_3 \times d_4 \times d_5 \end{cases}$

$= \min_{4 \le k < 5} \{ k=4 \{ 0 + 0 + 3 \times 2 \times 1$

$= 6$

→ $C[1,3] = \min_{1 \le k < 3} \begin{cases} k=1 \{ C[1,1] + C[2,3] + d_0 \times d_1 \times d_3 \\ k=2 \{ C[1,2] + C[3,3] + d_0 \times d_2 \times d_3 \end{cases}$

$= \min \begin{cases} 0 + 60 + 1 \times 5 \times 3 \\ 260 + 0 + 1 \times 4 \times 3 \end{cases}$

$= \min \begin{cases} 75 \\ 34 \end{cases}$

$= 34$, when $k = 2$

→ $C[2,4] = \min_{2 \le k < 4} \begin{cases} k=2 \{ C[2,2] + C[3,4] + d_1 \times d_2 \times d_4 \\ k=3 \{ C[2,3] + C[4,4] + d_1 \times d_3 \times d_4 \end{cases}$

$= \min_{2 \le k < 4} \begin{cases} k=2 \{ 0 + 24 + 5 \times 4 \times 2 = 64 \\ k=3 \{ 60 + 0 + 5 \times 3 \times 2 = 90 \end{cases}$

$= 64$, when $k = 2$

→ $C[3,5] = \min_{3 \le k < 5} \begin{cases} k=3 & C[3,3] + C[4,5] + d_2 \times d_3 \times d_5 \\ k=4 & C[3,4] + C[5,5] + d_2 \times d_4 \times d_5 \end{cases}$

$= \min_{3 \le k < 5} \begin{cases} k=3 & 0 + 6 + 4 \times 3 \times 1 = 18 \\ k=4 & 24 + 0 + 4 \times 2 \times 1 = 32 \end{cases}$

$= 18$, when $k = 3$

→ $C[1,4] = \min_{1 \le k < 4} \begin{cases} k=1 & C[1,1] + C[2,4] + d_0 \times d_1 \times d_4 \\ k=2 & C[1,2] + C[3,4] + d_0 \times d_2 \times d_4 \\ k=3 & C[1,3] + C[4,4] + d_0 \times d_3 \times d_4 \end{cases}$

$= \min_{1 \le k < 4} \begin{cases} k=1 & 0 + 64 + 1 \times 5 \times 2 = 74 \\ k=2 & 20 + 24 + 1 \times 4 \times 2 = 52 \\ k=3 & 34 + 0 + 1 \times 3 \times 2 = 40 \end{cases}$

$= 40$, when $k = 3$

→ $C[2,5] = \min_{2 \le k < 5} \begin{cases} k=2 & C[2,2] + C[3,5] + d_1 \times d_2 \times d_5 \\ k=3 & C[2,3] + C[4,5] + d_1 \times d_3 \times d_5 \\ k=4 & C[2,4] + C[5,5] + d_1 \times d_4 \times d_5 \end{cases}$

$= \min_{2 \le k < 5} \begin{cases} k=2 & 0 + 18 + 5 \times 4 \times 1 = 38 \\ k=3 & 60 + 6 + 5 \times 3 \times 1 = 81 \\ k=4 & 64 + 0 + 5 \times 2 \times 1 = 74 \end{cases}$

$= 38$, when $k = 2$

$$C[1,5] = \min_{1 \le k < 5} \begin{cases} k=1 & C[1,1] + C[2,5] + d_0 \times d_1 \times d_5 \\ k=2 & C[1,2] + C[3,5] + d_0 \times d_2 \times d_5 \\ k=3 & C[1,3] + C[4,5] + d_0 \times d_3 \times d_5 \\ k=4 & C[1,4] + C[5,5] + d_0 \times d_4 \times d_5 \end{cases}$$

$$= \min_{1 \le k < 5} \begin{cases} k=1 & 0 + 31 + 1 \times 5 \times 1 = 36 \\ k=2 & 20 + 18 + 1 \times 4 \times 1 = 42 \\ k=3 & 34 + 6 + 1 \times 3 \times 1 = 43 \\ k=4 & 40 + 0 + 1 \times 2 \times 1 = 42 \end{cases}$$

$= 36$, when $k = 1$

-> C[1,5] determines optimal cost 36

-> ~~((A).(B).(C).(D).E)~~

-> ((A).(B).(C).(D).E)

(4) Solve following knapasck problem using dynamic programing algorithm with given capacity $w = 5$ and values.
(2, 12), (1, 10), (3, 20), (2, 15)

→ $w_i = \{2, 1, 3, 4\}$
$p_i = \{12, 10, 20, 15\}$

$w = 5$
$n = 4$

w_i	p_i	0	1	2	3	4	5
		0	0	0	0	0	0
12	2	1	0	0	12	12	12
10	1	2	0	10	12	22	22
20	3	3	0	10	12	20	32
15	2	4	0	10	15	25	37

p_i	w_i	0	1	2	3	4	5
		0	0	0	0	0	0
12	2	1	0	0	12	12	12
10	1	2	0	10	12	22	22
20	3	3	0	10	12	20	30	32
15	2	4	0	10	15	25	30	37

→ Max profit is : $[4, 5] = 37$

Solution = { item 1, item 2, item 4 }

Q.8

(a) Explain Min-Max principle.

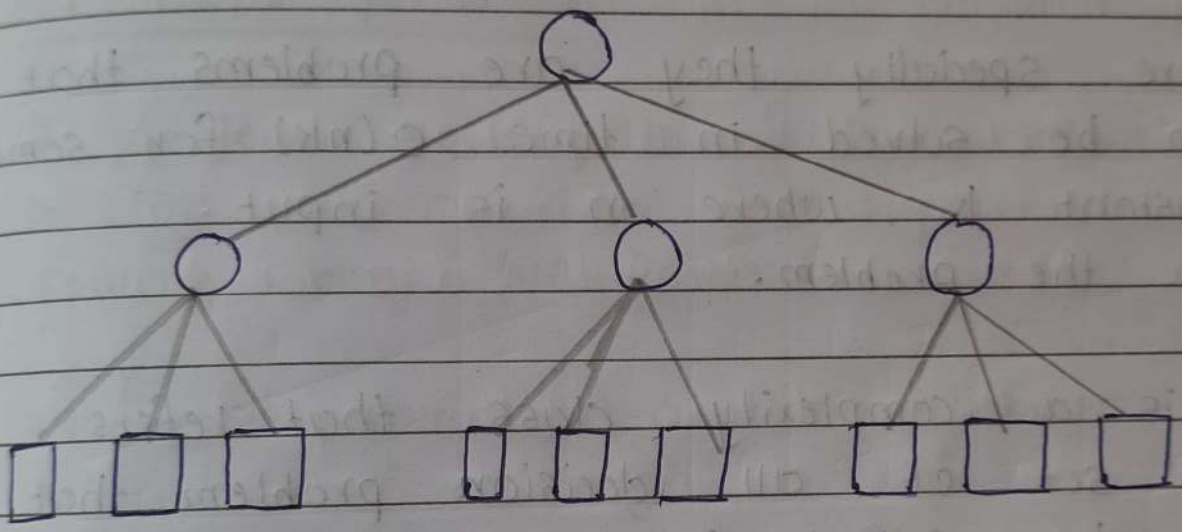
-> Minmax is a decision rule used in decision theory, game theory and philosophy for minimizing the possible loss and maximizing the profit.

-> Min max principle can be used for playing tic-tac-toe game.
In this game each player can win, lose or draw.

-> If there are two players A and B.
If player A wins by one move then that move is said to be best move.
If player B knows that one move will lead to the situation where player A can win in one move, while another move will lead to situation where player A can at best draw, then player B best move is the one leading to draw.

-> The Minmax algorithm helps find the best move, by working backwards from the end of the game.

-> At each steps it assume that player A is trying to maximize the chance of A winning, on the other hand, player B trying to minimize the chance of A winning so that B can win.
Thus min max principle is applied to making out decision.



(b) Define P, NP and NP hard problem.

* Class P

-> The class P consists of those problem that are solvable in polynomial time by deterministic algorithm.

-> More specially they are problems that can be solved in time O(nk) for some constant k, where n is input to the problem.

-> P is a complexity class that refers the set of all decision problem that can be solved in polynomial time. That is given an instance of the problem the answere yes and no can be decide in polynomial time.

* Class NP

-> NP is a non-deterministic polynomial time.

-> The class NP consists of those problem that are verifiable in polynomial time.

-> Easy to check correctness of a claimed answer with the help of little extra information.

-> We are not asking for a way to find a solution but only to verify that an alleged solution really is correct.

-> Class can be solved in exponential time using exhaustive search.

* NP Hard problems

-> A problem is NP-Hard if an algorithm for solving it can be translated into one for solving any NP-Problem.

-> NP-Hard means "at least hard as any NP problem" although it might in fact be harder.

-> NP hard problem can be solved if and only if there is a NP-Complete problem that can be reducible into x in polynomial time.

-> To solve this problem it do not have to be in NP.

(c) Explain Robin karp algorithm.

→ It is based on hashing technique.

→ This algorithm assume each character to be a digit in radix-d notation. It makes use of equivalence of two number modulo a third number.

→ Let, P[1...m] be a pattern then p denotes its corresponding decimal value. The d=10 for decimal, d=2 for binary.

→ The method follow following step:

(i) Compute P in O(m) time.
(ii) Compute all t_i values in total of O(m) time.
(iii) Find all valid shift s in total of O(n) time.

Ex.

Text: T :	3	1	4	1	5	9	2	6	5	3	5
Pattern: P :	2	6									

→ Let $P = P \bmod q$
$= 26 \bmod 11$
$= 4$

3	1	4	1	5	9	2	6	5	3	5

9	3	8	4	4	4	4	10	9	2

$ts_0 = 31 \mod 11 = 9$

$ts_1 = 14 \mod 11 = 3$

$ts_2 = 41 \mod 11 = 8$

$ts_3 = 15 \mod 11 = 4$

$ts_4 = 59 \mod 11 = 4$

$ts_5 = 92 \mod 11 = 4$

$ts_6 = 26 \mod 11 = 4$

$ts_7 = 65 \mod 11 = 10$

$ts_8 = 53 \mod 11 = 9$

$ts_9 = 35 \mod 11 = 2$

9 3 8 4 4 4 4 10 9 2

Supurious match

Valid match

Winter : 2021

Q.1

(a) Define algorithm and explain its characteristics.

-> Algorithm is a step by step procedures which defines a set of instruction to be executed in a certain ~~folder~~ order to get desired output.

* Characteristics

-> Unambiguous : Algorithm should be clear and unambiguous. Each of its step and input /output should be clear and it must lead only one meaning.

-> Input : An algorithm should have 0 or more well-defined inputs.

-> Output : An algorithm should have 1 or more well defined outputs.

-> Finiteness : Algorithm must terminate after a step.

-> Feasible : Should be feasible with avalaible resources.

-> Independent : Algorithm should have step by step direction, which should be independent of programing language.

(b) Explain Worst case, Average case and best case time complexity with example.

→

* Worst-Case

→ In this case, we calculate the upper bound on the running time of an algorithm.

→ We must know the case that cause a maximum number of operation to be executed.

→ Worst case resource usage is maximum.

→ For linear search, the worst case happnes when element to be searched (x) is not present in the array.

→ When x is not present, the search () function compares it with all elements of arr[] one by one.

→ Therefore the worst case time complexity of the linear search would be O(n).

* Average case

-> In this case, we take all possible inputs and calculate the computing time for all of the inputs.

-> Sum all the calculated value and divide by the total number of input.

-> It is rarerly used case.

-> Algorithm behaviour under random condition.

-> For linear search problem let us assume that all cases are uniformaly distributed.

* Best case

-> In this case, we calculate the lower bound of the running time of an algorithm.

-> We must know the case that causes a minimum number of operation to be executed.

-> Algorithm behaviour under optimal condition.

-> Minimum number of steps and operation.

-> In linear search problem the best case occurs when x is present at the first location.

(c) Write and analyze insertion sort algorithm to arrang n iteams into ascending order.

-> Insertion sort is a simple sorting algorithm that works similarly the way you sort playing cards in your hand.

-> The array is virtually split into a sorted and unsorted part.

-> Values from unsorted part are picked and placed at correct position in sorted path.

-> This is one of the simplest algorithm with simple implementation.

-> It is efficient for small data values.

-> It is appropriate for data set which is partially sorted.

* Sort array of size N in ascending order

-> Iterate from arr[1] to arr[N] over the array.

-> Compare the current element to its predcessor.

-> If the key element is smaller than its predecessor, compare it to elements before. Move greater element one position up to make space for swapped element.

* Algorithm

```
for i ← 2 to n do
    x ← T[i];
    j ← i-1;                                   ← O(n) (Best)
    while x < T[j] and j > 0 do
        T[j+1] ← T[j];                         ← O(n²) (worst + Avg)
        j ← j-1;
    T[j+1] ← x;
```

→ Best case time complexity : $O(n)$

→ Worst case time complexity : $O(n^2)$

Q.2

(a) Write an algorithm of selection sort.

→ Algorithm :

```
for i ← 1 to n-1 do
    min_j ← i
    min_x ← A[i]
    for j ← i+1 to n do
        if A[j] < min_x then
            min_j ← j
            min_x ← A[j]
    A[min_j] ← A[i]
    A[i] ← min_x
```

```
for i ← 1 to n-1 do
    min_x ← A[i]
    for j ← i+1 to n do
        if A[j] < min_x then
            temp ← A[i]
            A[i] ← A[j]
            A[j] ← Temp
```

→ Ex

40	20	30	10
min(x)	j		

pass 1:

20	40	30	40

pass 1 min[x] > A[j] → swap

20	40	30	10

Ex.

70	30	20	50	60	10	40

Pass 1: find smallest element, swap

10	30	20	50	60	70	40

min(x) j

Pass 2: min[x] > A[j] ⟷ swap

10	20	30	50	60	70	40

min(x) A[j]

Pass 3: min[x] > A[j] → swap

10	20	30	40	60	70	50

min(x) j

Pass 4: min[x] > A[j] → swap

10	20	30	40	50	70	60

min(x) j

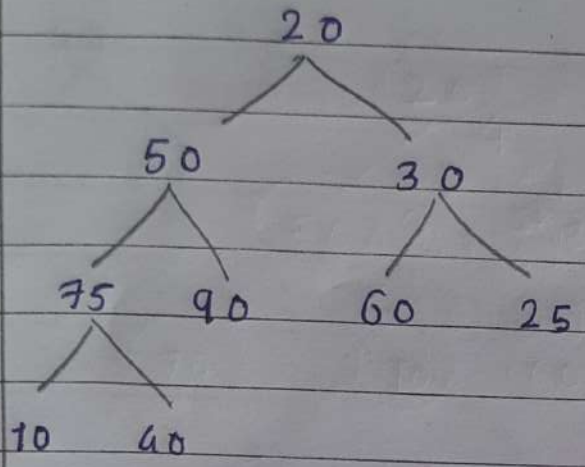
Pass 5: min[x] > A[j] → swap

10	20	30	40	50	60	70

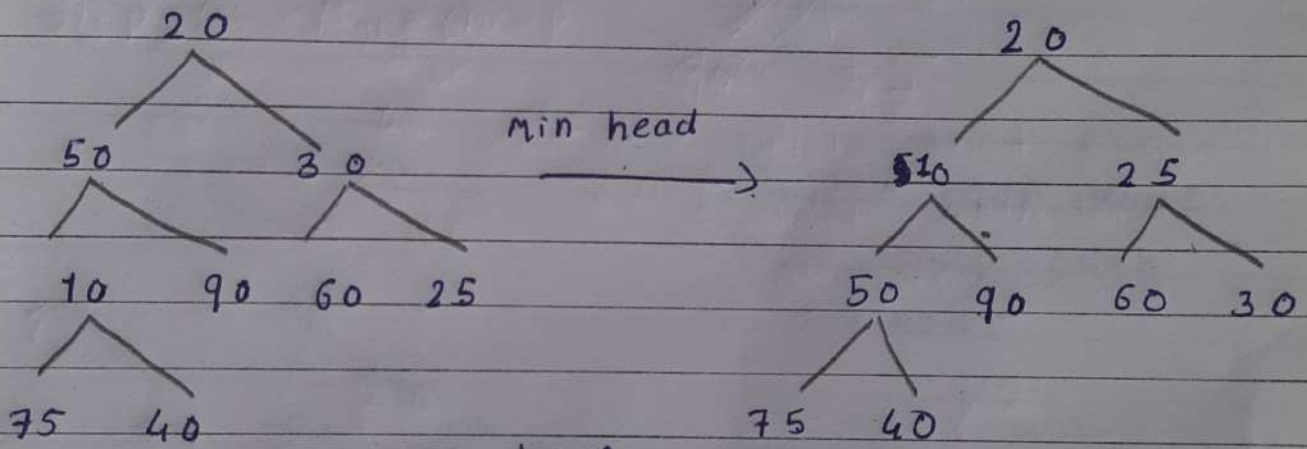
Heap Sort

Ex. 20, 50, 30, 75, 90, 60, 25, 10, 40

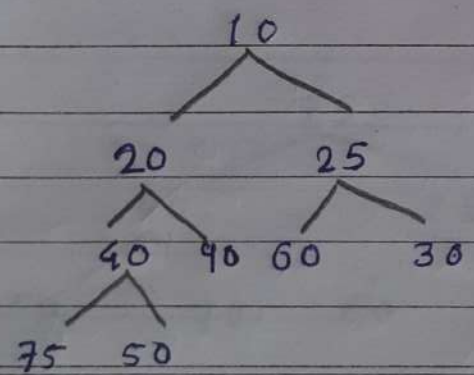
→



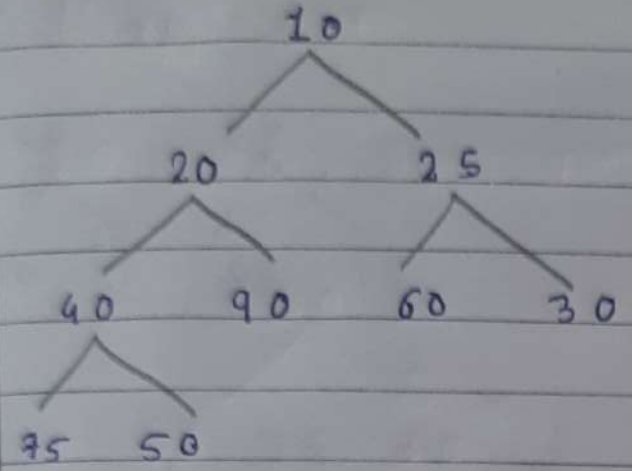
→ Step 1 : Min heap



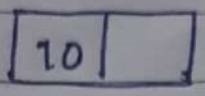
Min heap



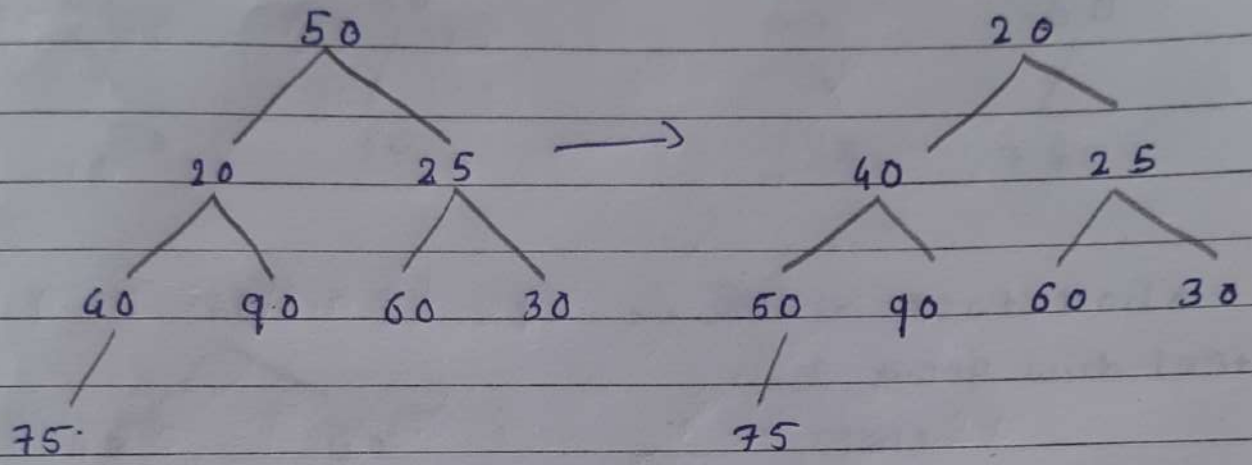
→



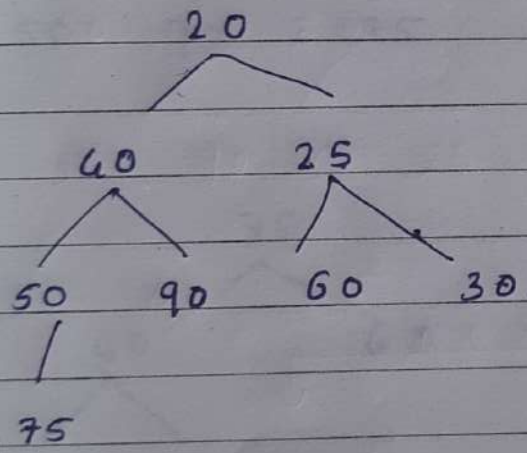
→ Delete first (root) node and swap with last element.



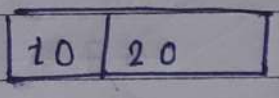
→



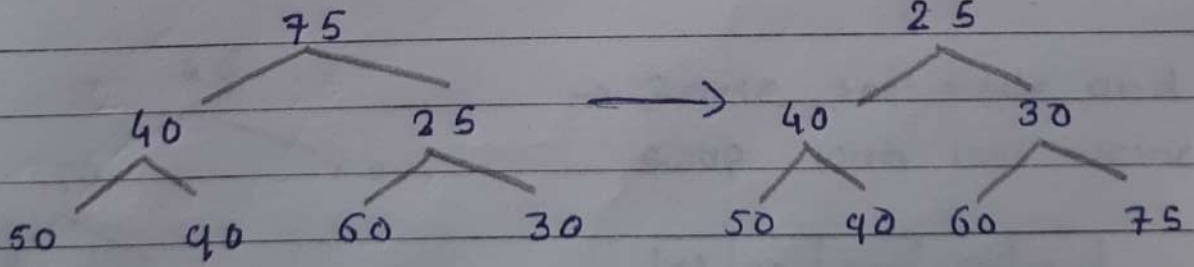
→

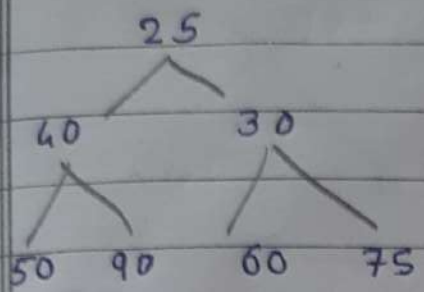


→ Delete root node and swap with last node



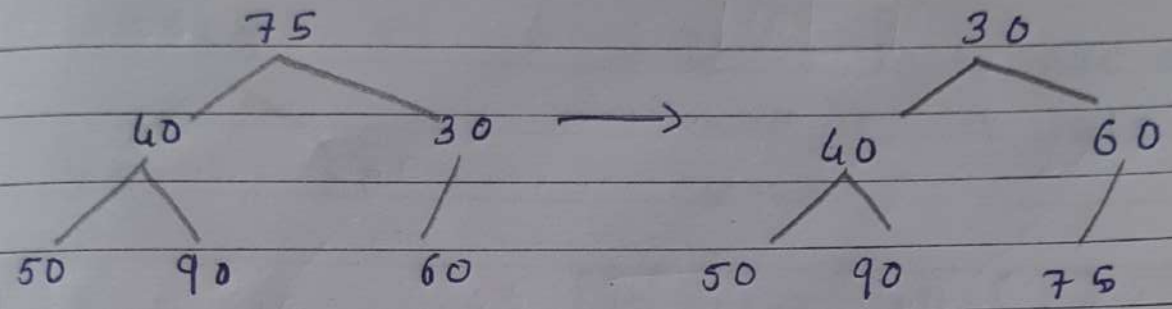
→

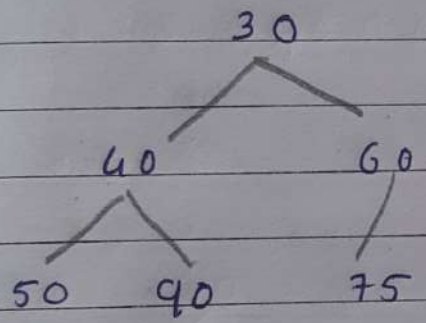




→ Delete root node and swap with last element.

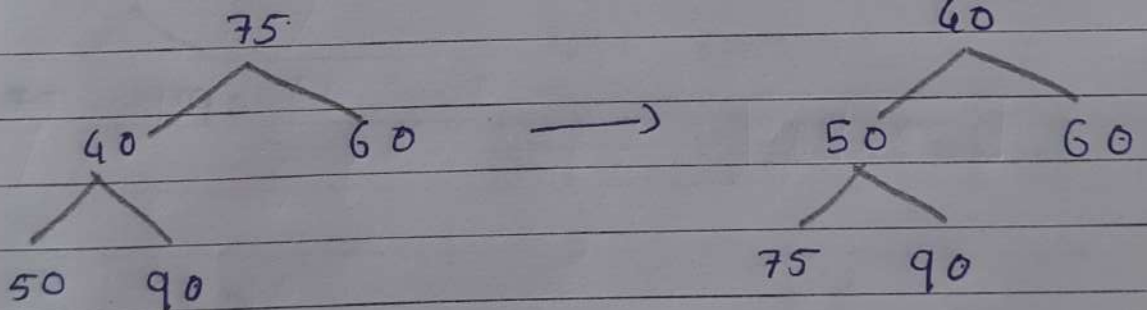
10	20	25

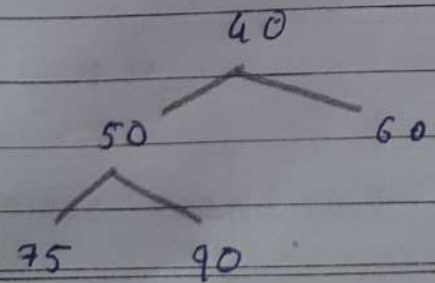




→ Delete root node and swap with last element.

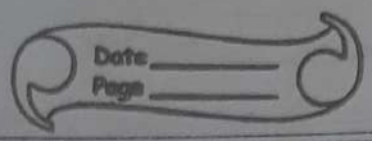
10	20	25	30



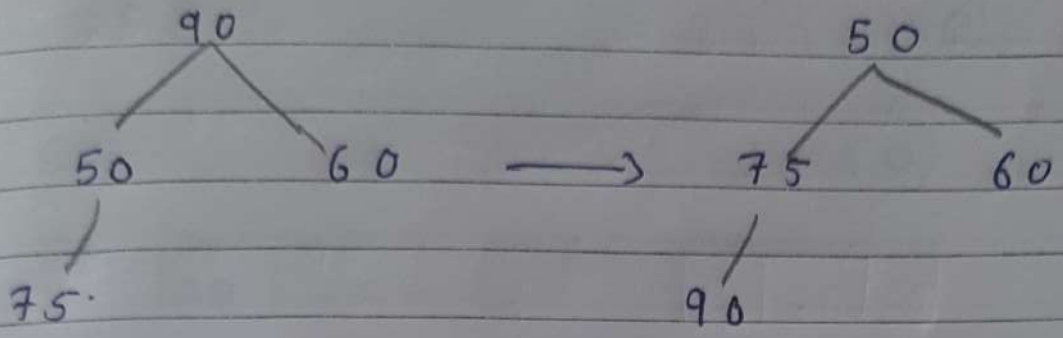


→ Delete root node and swap with last element

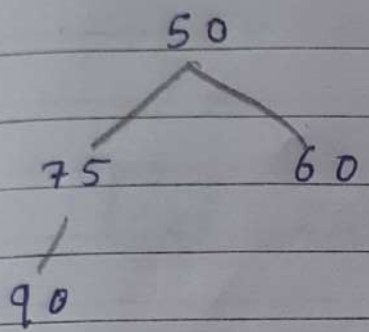
10	20	25	30	40



->



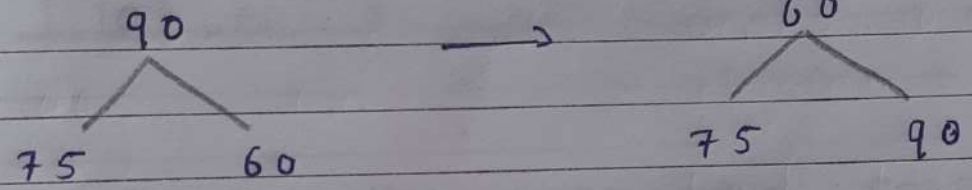
->



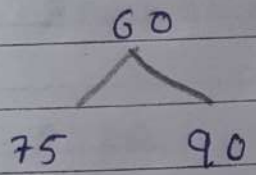
-> Delete root node and swap it with last element

10	20	25	30	40	50

->



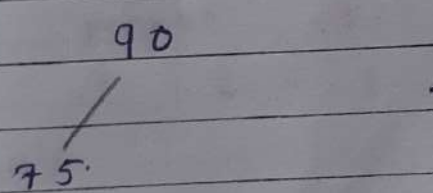
->



-> Delete root and swap with last element

10	20	25	30	40	50	60

->



-> Delete root node swap with last element

10	20	25	30	40	50	60	75	90

Q.3

(a) What is principle of optimality ? Explain its use in dynamic programing method.

→ W:2020 → Q.6.(a)

(b) Explain Bionomial coefficient algorithm using dynamic programming.

→ Computing binomial coefficient is typical example of applying dynamic programing.

→ In mathamatics, particulary in combinatorics, binomial coeffiencent is coefficient of any of the terms in expression of $(a+b)^n$. It is denoted by $C(n,k)$ or $\binom{n}{k}$ where $(0 \leq k \leq n)$

→ Formula :

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

and $C(n,0) = 1$

$C(n,n) = 1$

where, $n > k > 0$.

(c) Solve following 0/1 knapasck problem using dynamic programing

$w_i = \{1, 2, 5, 6, 7\}$

$P_i = \{1, 6, 18, 22, 28\}$

Weight capacity = 11

→ W:2020 → Q.6.(c)

Q.3

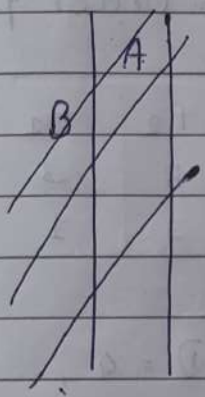
(a) Compare dynamic programing and Greedy algorithm

→ W:2020 : Q.5.(b)

(b) Give characterstics of greedy algorithm

→ W:2020 → Q.4.(a)

(c) obtain longest common sequence using dynamic programing A = acabaca , B = bacac



		B→	b	a	c	a	c
	A↓	0	0	0	0	0	0
✓	a	0	0	↖1	←1	↖2	←2
✓	c	0	0	↑1	↖2	↑1	↖3
	a	0	0	↖2	↑3	↖4	←4
	b	0	↖1	↑2	↑3	↑4	↑4
✓	a	0	0	↖3	↑3	↖5	←5
✓	c	0	0	↑3	↖4	↑5	↖6
	a	0	0	↖4	↑4	↖6	↑6

LCS = acac

Q.4

(a) Using greedy algorithm find an optimal schedule for following jobs with $n = 7$. profits: $(P_1, P_2, P_3, P_4, P_5, P_6, P_7) = (3, 5, 18, 20, 6, 1, 38)$ and deadline $(d_1, d_2, d_3, d_4, d_5, d_6, d_7) = (1, 3, 3, 4, 1, 2, 1)$

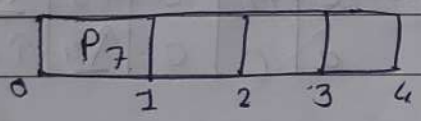
→

Job	P_1	P_2	P_3	P_4	P_5	P_6	P_7
Profit	3	5	18	20	6	1	38
deadline	1	3	3	4	1	2	1

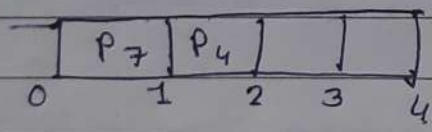
→ Sort Job in descending order profit value

Job	P_7	P_4	P_3	P_5	P_2	P_1	P_6
Profit	38	20	18	6	5	3	1
deadline	1	4	3	1	3	1	2

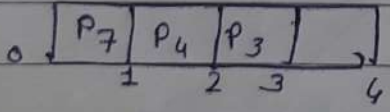
→ Select maximum deadline ① = 4,
Select Job $P_7 = 38$, ① = 1



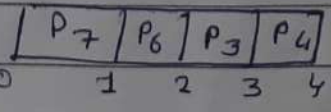
→ Select Job $P_4 = 20$, ① = 4



→ Select Job $P_3 = 18$, ① = 3



→ Select Job $P_6 = 1$, ① = 2



→ $P_7 + P_6 + P_3 + P_4 = 38 + 1 + 18 + 20$
$= 77$

(b) Find minimum spanning tree for given graph using prim's algorithm.

(c) Explain BFS and DFS technique.

→ * BFS following Rules

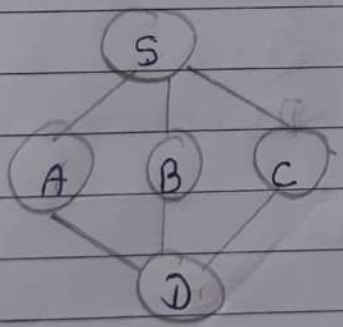
(i) Select an unvisited node V, visit it, have it be the root in a BFS treee being formed its level is called current level.

(ii) For each node x in the current level, in the order in which the level nodes were visited, visit all unvisited neighbour of x.

(iii) Repeat step 2 and for all unvisited node

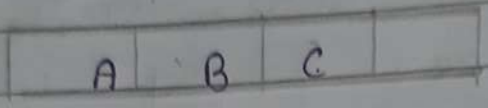
(iv) Repeat from step 1 until no more verticies are remaining.

→ Ex.

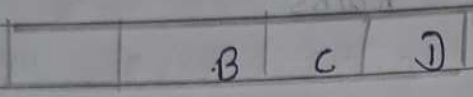


→ Step 1: Start with S and visit all unvisited node of S.

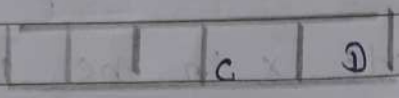
Step 2: Visit all unvisited node of S and pop S



Step 3: Visit all unvisited node of A and pop A



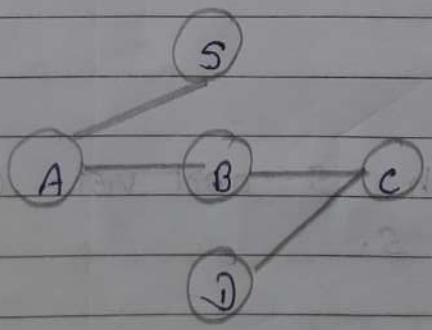
Step 4: Visit all unvisited node of B and pop B



Step 5: Visit all unvisited node of C and pop C.

Step 6: Visit all unvisited node of D and pop D.

Ans: S, A, B, C, D



-> * DFS following Rules

(i) Select an unvisited node v, visit it and treat as current node

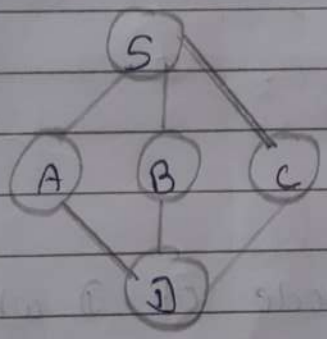
(ii) Find an unvisited node of current node, visit it and make it current node

(iii) If current node has no unvisited neighbour then pop it and make precedecer node to current node.

(iv) Repeat step 2 and 3 untill no more nodes can be visited.

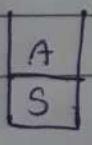
(v) Repeat from step 1 for remaining node.

-> Ex.

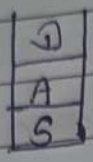


Step 1: Mark S as visited and put into stack.

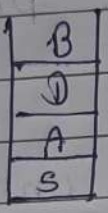
Step 2: Visit adjacent node of S which is A and put into stack



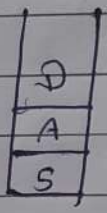
Step 3: Visit adjacent node of A which is D and put into stack



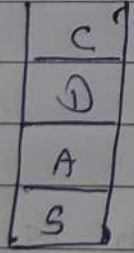
Step 4: Visit adjacent node of D which is B and put into stack.

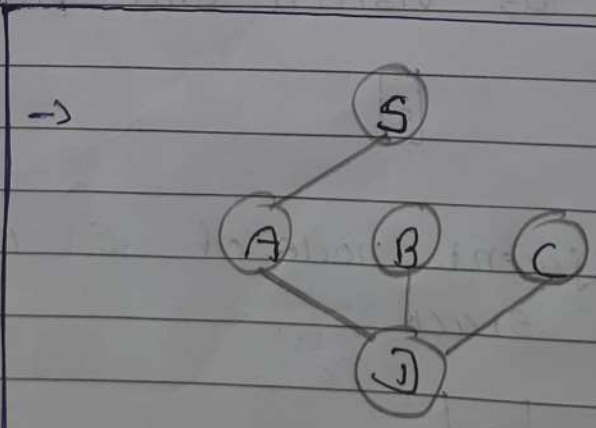


Step 5: Visit adjacent node of B.
There is no unvisited node from B so pop B from stack.



Step 6: visit adjacent node of D which is C and put into stack





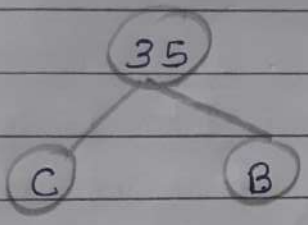
Q.4

(a) Find an optimal Huffman code for following frequency. A:50, B:20, C:16, D:30

-> Step 1: Arrange in ascending order

C	B	D	A
15	20	30	50

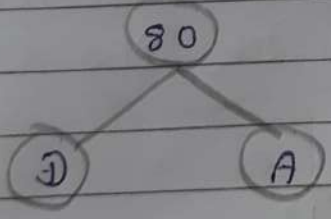
-> Step 2: Remove first two node from table and form node



Insert this new entry at appropriate location in table

D	A	CB
30	50	35

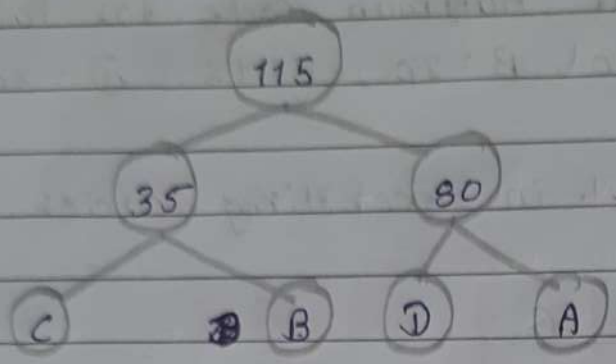
-> Step 3: Remove first two node from table and form node



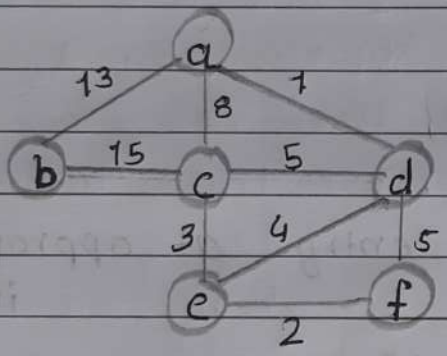
Insert this new entry at appropriate location in table.

CB	DA
35	80

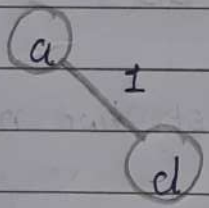
→ Step 4: Remove first two entry from table and form node.



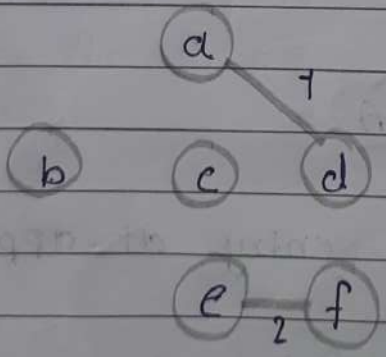
(b) Find minimum spanning tree using krushak's algorithm.



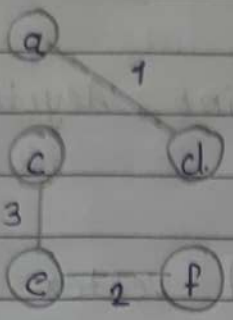
→ Step 1:



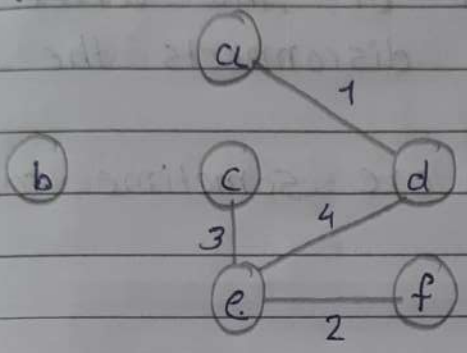
Step 2:



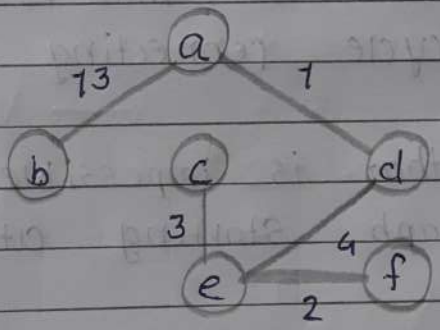
Step 3:



Step 4:



Step 5:



(c) Explain Backtracking method. What is N-Queen problem ? Give solution of 4-Queen problem using backtracking method.

Q.5

(a) Define Articulation point, Acyclic directed graph, Back edge.

-> Articulation point : A vertex is said to be articulation point in a graph if removal of the vertex and associated edge disconnects the graph.

- Articulation points are sometime called cut verticies.

-> Directed acyclic Graph : It is a graph that is directed and without cycle connecting the other edges.

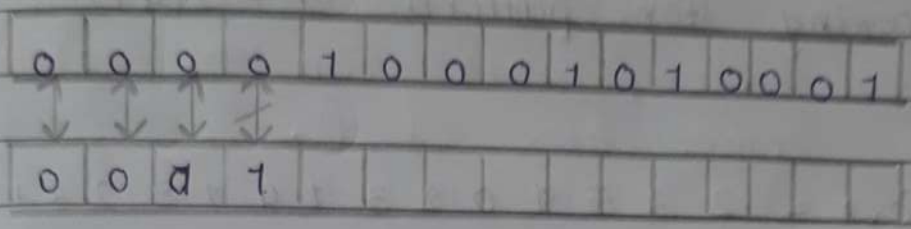
- This means that is impossible to traverse the entire graph starting at one edge.

-> Back edge : It is an edge (u, v) such that v is an ancestor of node u but not part of DFS traversal of the tree.

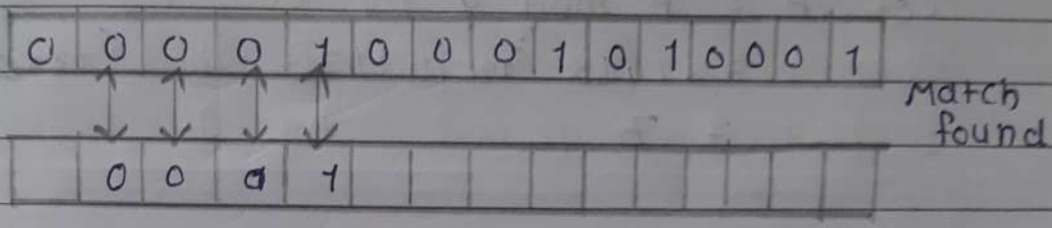
- The Back edge indicates cycle in directed graph.

(b) Show the comparision that naive using string matcher makes for the pattern P = 0011 in Text T = 0000100010101001

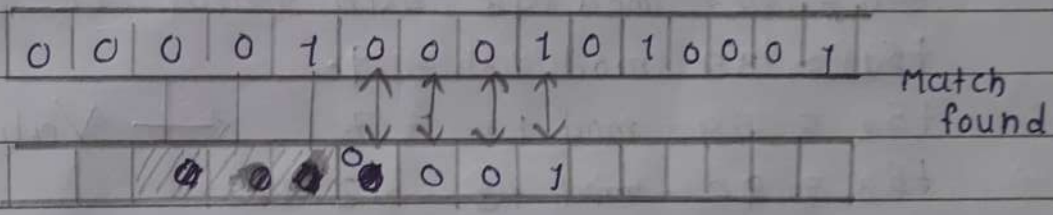
→



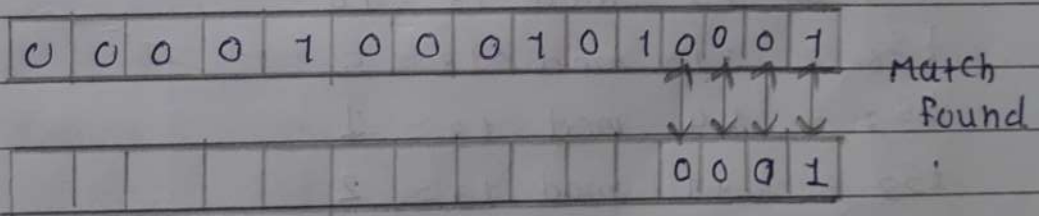
Shift to right by one position



Shift to right,



Shift to right;



→ Shift pattern right by one position, each time and check pattern against text.

(c) Solve Rabin-karp string matching algorithm when modulo $q = 13$, Text $T = 2359023141526739921$ when looking for pattern $P = 31415$

-> Text = 2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1
pattern (P) = 31415

Let, $P = P \bmod q$
$= 31415 \bmod 13$ $(13 \times 2416) = 31408$
$= 7$

-> 2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1

$ts_0 = 23 \bmod 13 = 10$
$ts_1 = 35 \bmod 13 = 9$
$ts_2 = 59 \bmod 13 = 7$ ← Valid match
$ts_3 = 90 \bmod 13 = 12$
$ts_4 = 02 \bmod 13 = 2$
$ts_5 = 23 \bmod 13 = 10$
$ts_6 = 31 \bmod 13 = 5$
$ts_7 = 14 \bmod 13 = 1$
$ts_8 = 41 \bmod 13 = 2$
$ts_9 = 15 \bmod 13 = 2$
$ts_{10} = 526 \bmod 13 = 0$
$ts_{11} = 26 \bmod 13 = 0$
$ts_{12} = 67 \bmod 13 = 2$
$ts_{13} = 73 \bmod 13 = 9$
$ts_{14} = 39 \bmod 13 = 0$
$ts_{15} = 99 \bmod 13 = 8$
$ts_{16} = 92 \bmod 13 = 1$
$ts_{17} = 21 \bmod 13 = 8$

Q.5

(a) Explain polynomial reduction.

→ To prove whether particular problem is NP complete or not we use polynomial time reducibility. That means if,

$A \xrightarrow{Poly} B$ and $B \xrightarrow{Poly} C$ then $A \xrightarrow{Poly} C$

→ The reduction is an important task in NP completness proof.

→ Let us consider a decision problem, A which we would like to solve in polynomial time.

→ Now suppose that we already know how to solve different decision problem B in polynomical time.

→ These are important problem and people have spent significant effort on trying to find better algorithm for them.

→ Each one of these is a search problem whereby we search for solution that is good in some easy to define sense.

→ Local replacment : In this reduction A→B by dividing input to A in the form of components and then these components can be converted to component of B.

→ Component design : In this reduction A→B by building special component for input B that enforce properties required by A.

(c) Explain P, NP, NP complete, NP-hard problems.

→ W: 2020 → Q. 8. (b)

(b) Differentiate between backtracking and branch and bound.

Backtracking	Branch and Bound
→ It is used to find all possible solution available to a problem.	It is used to solve optimisation problem.
→ Traverse the state space tree by DFS.	Traverse the tree in any manner.
→ Involves feasibility functions.	Involves bounding function.
→ It is used for solving decision problem.	It is used for solving optimisation problem.
→ It is more efficient.	It is less efficient.
→ Useful in solving N-Queen problem.	Useful in solving Knapsack problem.
→ It can solve almost any problem.	It can not solve almost any problem.

Summer : 2022

Q.1

(a) Define algorithm. Time and space complexity.

→ Algorithm is a step by step procedure which defines set of instruction to be executed in certain order to get desire output.

→ Time complexity : It is a time taken by algorithm to execute each set of instructions.
It is always better to select most efficient algorithm when a simple problem can be solved with different method.

→ Space complexity : It is usually reffered to as a amount of memory consumed by the algorithm.
It is composed of two different spaces Auxilary space and Input space.

(b) Explain worst, best and average case complexity.

→ Winter : 2021 → Q.1 (b)

(c) Sort the following list using quick sort algorithm.
< 5, 3, 8, 7, 4, 6, 2, 7 >

→ Winter : 2020 → Q.3 (b)

Extra

Q.2.(b) Binary Search algorithm.

→

```
low ← 0
high ← n-1
While ( low < high) do
    {
        m ← (low + high) / 2
            if (key = A[m]) then
                return m.
            else if (key < A[m]) then
                high ← m-1
            else
                low ← m+1
    }
return -1.
```

Q.2

(a) Write and algorithm for selection sort.

→ Winter : 2021 → Q.2 (a)

(b) Demonstrate binary search method to search key = 14, from the array
A = < 2, 4, 7, 8, 10, 13, 14, 60 >

→

2	4	7	8	10	13	14	60
0	1	2	3	4	5	6	7

$m = (0+7)/2$

$= 3.5$

≈ 4

→

2	4	7	8	10	13	14	60
0	1	2	3	4	5	6	7

↑ Mid

key > mid, we will search right sublist.

→

13	14	60
5	6	7

$m = 5/7$

$= 12/2 = 6$

key = mid

The element 14 is present at A[6]

(c) Solve using master theorem.

(i) $T(n) = T(n/2) + 1$

(ii) $T(n) = 2T(n/2) + n \log n$

-> (i) $T(n) = O(n^a)$ if $a < b^d$

$T(n) = O(n^d \log n)$ if $a = b^d$

$T(n) = O(n^{\log_b a})$ if $a > b^d$

(i) $T(n) = T(n/2) + 1$

-> $a = 1, b = 2, d = 1$

$b^d = 2$

$a < b^d$

$T(n) = O(n)$

(ii) $T(n) = 2T(n/2) + n \log n$

-> $a = 2, b = 2, d = 1$

$b^d = 2$

$a = b^d$

$T(n) = O(n \log n) + n \log n$

$= O(n (\log)^2 n)$

(1) Solve using recurrence relation.

$T(n) = T(n-1) + 1$ where $T(0) = 0$

→ $T(n) = T(n-1) + 1$

$T(n) = [T(n-2) + 1] + 1$

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

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

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

⋮

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

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

$= T(n-2) + 1$

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

Assume, $n - k = 0$

$\therefore n = k$

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

$T(n) = T(0) + n$

$= 0 + n$

$= n$

Q.3

(a) What is principle of optimality ? Explain it's use in dynamic programing method.

→ Winter : 2020 → Q.6.(a)

(b) Find out LCS of A = { K, A, N, D, L, A, P }

and B = { A, N, D, L }

(↖) ⇒ When character match

(↑) ⇒ When cross value same.

	B→	A	N	D	L
A↓	0	0	0	0	0
K	0	↑ 0	↑ 0	↑ 0	↑ 0
A	0	↖ 1	← 1	← 1	← 1
N	0	↑ 1	↖ 2	← 2	← 2
D	0	↑ 1	↑ 2	↖ 3	← 3
L	0	↑ 1	↑ 2	↑ 3	↖ 4
A	0	↖ 1	↑ 2	↑ 3	↑ 4
P	0	↑ 1	↑ 2	↑ 3	↑ 4

→ LCS = A N D L

(c) Discuss Assembly line scheduling problem using dynamic programing with example.

→

Q.4

(a) Explain Articulation point, Graph, tree.

→ Articulation point : A vertex is said to be articulation point in a graph if removal of vertex and associated edge disconnects the graph.

Articulation points are some time called cut vertices.

→ Graph : A graph is a set of vertices and edges.

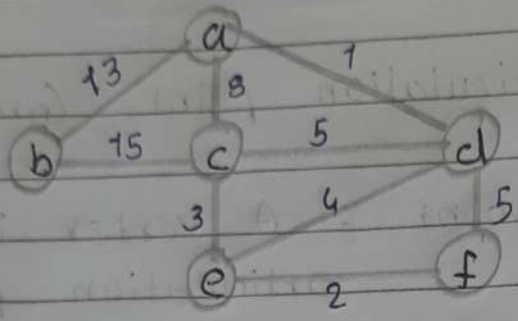
→ Tree : A tree is a set of nodes and edge.

→ Graph : A graph is a collection of two sets V and E where V is a finite non-empty set of vertices and E is a finite non empty set of edge.

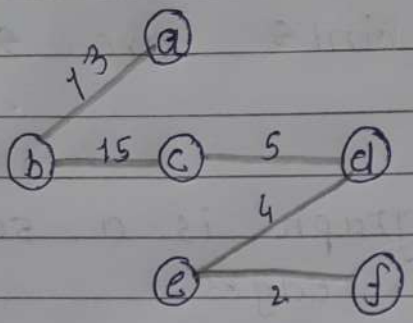
(b) Explain Bredth First Search

→ Winter : 2020 → Q.7 - (b)

(b) Find MST using prim's algorithm.



→ Winter : 2021



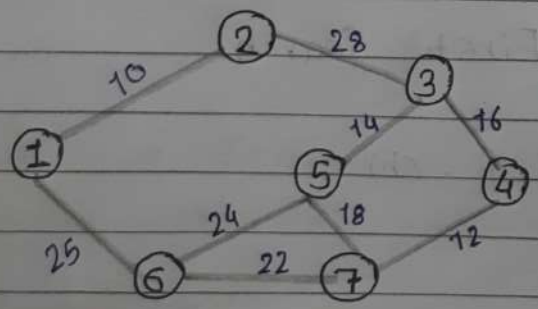
MST = 39

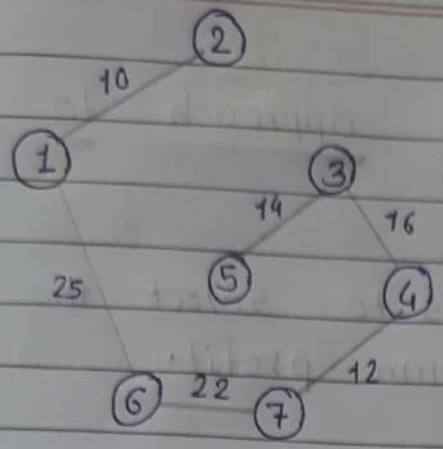
Q.4

(a) Explain Huffman code with example.

→ Winter : 2020 → Q.5.(a)

(b) Find MST Using Krushalk's algorithm.





(c) Explain fractional knapasack problem with example.

-> The fraction knapasack problem is also one of the techniques which are used to solve the knapsack problem.
In fractional knapasack the items are broken in order to maximize the profit.
The problem in which we break the item is known as a fractional knapsack problem.

* Problem can be solved using two techniques:

-> Brute-Force : This approach tries all the possible solution with all the different fractions but it is time consuming approach.

-> Greedy : In this approach we calculate the ratio of profit, and accordingly we will select the item. The item with the highest ratio would be select first.

Q.5

(a) What is String matching problem? Define valid and Invalid shift.

→ String matching algorithm is also called String searching algorithm.
This is vital case of string algorithm is declared as " this is the method to find a place where one is serval string are found within larger string.

. Algorithms : Naive string matching,
Rabin-Karp,
Finite Automate

→ Shift : The position before complete pattern occurs in text.

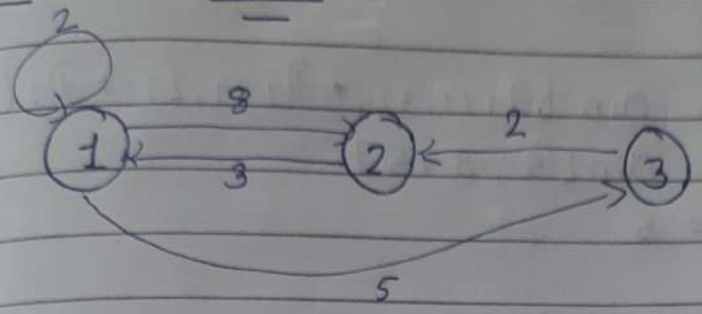
→ Invalid shift : The position after which partial matching occurs.

→ Valid shift : The position after which complete matching occurs.

(b) Define P, NP, NP-Hard and complete problem.
→ Winter : 2020 → Q.8 (b.)

(c) Explain Backtracking method? What is N-queen problem? Give solution of 4-queen problem.
→ Winter : 2021 → Q.4 (c)

Floyd's Algorithm



$$D_0 = \begin{bmatrix} 2 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{bmatrix}$$

→ D_1 from D_0

$D_1(2,3) = \text{Min}[D_0(2,3),\ D_0(2,1) + D_0(1,3)]$
$= \text{Min}[\infty,\ 3+5]$
$= 8$

$D_1(3,2) = \text{Min}[D_0(3,2),\ D_0(3,1) + D_0(1,2)]$
$= \text{Min}[2,\ \infty+8]$
$= 2$

$$D_1 = \begin{bmatrix} 2 & 8 & 5 \\ 3 & 0 & 8 \\ \infty & 2 & 0 \end{bmatrix}$$

→ D_2 from D_1

$D_2(1,3) = \text{Min}[D_1(1,3), D_1(1,2) + D_1(2,3)]$
$= \text{Min}[5, 8+5]$
$= 5$

$D_2(3,1) = \text{Min}[D_1(3,1), D_1(3,2) + D_1(2,1)]$
$= \text{Min}[\infty, 2+3]$
$= 5$

$$D_2 = \begin{bmatrix} 2 & 8 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix}$$

→ D_3 from D_2

$D_3(1,2) = \text{Min}[D_2(1,2), D_2(1,3) + D_2(3,2)]$
$= \text{Min}[8, 5+2]$
$= 7.$

$D_3(2,1) = \text{Min}[D_2(2,1), D_2(2,3) + D_2(3,1)]$
$= \text{Min}[3, 8+5]$
$= 3$

$$D_3 = \begin{bmatrix} 2 & 7 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix}$$

Knapsack problem Using Greedy method

Q. W = 15
$P_i = (10, 15, 12, 8)$
$w_i = (4, 5, 6, 3)$

object	1	2	3	4
P_i	10	15	12	8
w_i	4	5	6	3
P_i/w_i	2.5	3	2	2.6

A. → Method 1 : Select maximum profit.

Object	Profit	weight	Remaining w_i
-	-	-	15
2	15	5	15 - 5 = 10
3	12	6	10 - 6 = 4
1	10	4	4 - 4 = 0

Total Profit = 15 + 12 + 10 = 37.

→ Method 2 : Select minimum weight.

Object	Profit	weight	Remaining w_i
-	-	-	15
4	8	3	15 - 3 = 12
1	10	4	12 - 4 = 8
2	15	5	8 - 5 = 3
3	12	6	3 - 3 = 0

Total Profit = 8 + 10 + 15 + 12
= 45

→ Method 3 : Select maximum Pi/wi

object	Profit	Weight	Remaining wi
–	–	–	15
2	15	5	15 - 5 = 10
4	8	3	10 - 3 = 7
1	10	4	7 - 4 = 3
3	3 × 2 = 6	3	3 - 3 = 0

Total Profit = 15 + 8 + 10 + 6

= 39