Module - 3

Greedy Technique

This technique is used for designing optimization problems. Greedy Strategy always tries to find the best Soln for Each Sub problem with the hope that this will results in best Soln for a whole problem.

General plan :- Greedy Approach Suggests to construct a Soln for a problem through a Sequence of Steps, Each Step Expanding a partially obtained Soln so far, until a Complete Soln is reached.

→ It is Important to note that we've to make a choice in Each Step Such that choice Must be

* Feasible → It has to Satisfy the Constraints of the problem.
* locally optimal → It has to be the best local choice among all feasible Solns available at that Step.
* Irrevocable → Once a choice is made, it can't be changed on Subsequent Steps of the Algorithm.

As in Every Step we look at optimal Soln, greedy technique yields best Soln for Many of the problems.

→ The Drawback of greedy technique is without aiming at final Soln, we will think about the Step where we are at. this is just like grabing whatever available right now without thinking about the Future.
Thus for Some problems, Greedy Approach is Not Applicable.

* Coin change problem :-

In this problem, the goal is to find the Min no of coins which add up to a given amount of Money.

Minimum Coin change problem is often Solved by Greedy Algo.

→ Assuming there is an Infinite Supply of Currency, an Amount 'A' has to be paid with Min no of coins / notes, provided there are 'n' kinds of coins with denomination/value from V_1 to V_n

So the Greedy Strategy is to Visualize the best option i.e highest valued coin at Each Step for best results.

→ Consider the foll Ex

let an Amount of Rs. 27/- is to be paid & coins / notes of Values 1, 2, 5, 10 & 50 is available in Ample Amount.

⇒ Given are Coins of Values 1, 2, 5, 10 & 50 & Amount to be paid is Rs. 27

* First of all, Sort the Coin Value in decreasing order if not Sorted

coin_value[] =

50	10	5	2	1

(50 marked ×: greater than Amount)

Amount	Coin opted	Final Amount
27	10	27 − 10 = 17
17	10	17 − 10 = 7
7	5	7 − 5 = 2

Amount	Coin opted	Final Amount
2	2	2-2 = 0

∴ Solution is '4' coins/notes = {10, 10, 5, 2} //

* Algorithm

I/p :- Coin_value[] & Amount to be paid

O/p :- n is total kind of coins

```
min_coins (coin_value[], n, amount)
{
    for i = 1 to n
        while Amount >= coins[i]
        {
            amount = amount - coin_value[i]
            print coin_value[i]
        }
}
```

* let an Amount of Rs. 43/- is to be paid by coins/notes of Values 1, 5, 10 & 25 is available in Ample Amount.

⇒ Given are coins of Values 1, 5, 10 & 25 & Amount to be paid is Rs. 43/-

* First of all, Sort the Coins Value in decreasing order if not Sorted

Coin_value[] =

25	10	5	1

Amount	Coin opted	Final Amount
43	25	43-25 = 18
18	10	18-10 = 8
8	5	8-5 = 3
3	1	3-1 = 2
2	1	2-1 = 1
1	1	1-1 = 0

∴ Soln is 6 coin/Note = {25, 10, 5, 1, 1, 1} //

* Minimum Spanning Tree :- or Min Weight Spanning tree for a Weighted, Connected & undirected graph, a Spanning tree of that graph is a Subgraph that is a tree & Connects all the Vertices together.

→ A Single graph Can have Many diff Spanning trees.

→ There are two diff Algorithms used to obtain Min cost Spanning tree on given graph. using Greedy Approach

* prim's Algorithm
* Kruskal's Algorithm.

* prim's Algorithm :-

Constructs Min Spanning tree through a Sequence of Expanding Subtrees. The Initial Subtree is taken as any Arbitary Vertex.

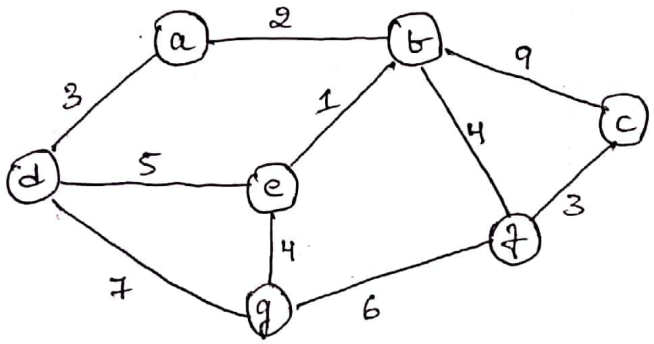
Then Among the other Vertices of a graph, the nearest Vertex to the Existing Vertex of tree is found & Attached to tree.

→ The Above procedure is Continued till all the vertices are Included into the tree.

Thus if V is the Set of all vertices of a graph & V_T is that of Min Spanning tree then we have to do the foll operations.

* Move u^* (presently chosen vertex) from the Set of $V - V_T$ to V_T
* For Each remaining Vertex 'u' in $V - V_T$ that is Connected to u^* by a Shorter distance, update its label by u^* & weight the Edge b/w them.

→ Consider the foll Example Connected weighted graph.

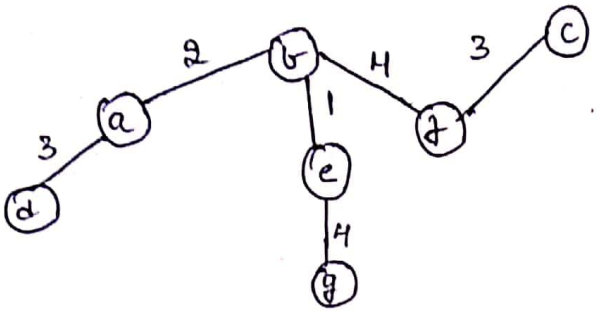


⇒

Tree vertices	Remaining vertices
a (-, -)	b (a, 2), d (a, 3), c (-, ∞), e (-, ∞) f (-, ∞), g (-, ∞)
b (a, 2)	d (a, 3), e (b, 1), c (b, 9), f (b, 4) g (-, ∞)
e (b, 1)	d (a, 3), d (e, 5), g (e, 4), c (b, 9), f (b, 4)
d (a, 3)	g (e, 4), g (d, 7), c (b, 9), f (b, 4)
g (e, 4)	g (g, 6), f (b, 4), c (b, 9)

f(b,4)	c(b,9), c(f,3)
c(f,3)	——— All vertices visited ———

∴ Tree vertices are a(-,-), b(a,2), e(b,1), d(a,3) g(e,4), f(b,4), c(f,3)



(Min cost Spanning tree)

* Algorithm prim's (G)

I/p:- A connected weighted graph G(V,E)

O/p:- E_T, the set of Edge composing Min Spanning tree of G.

$V_T \leftarrow \{v_0\}$

$E_T \leftarrow \phi$

for $i \leftarrow 1$ to $|V|-1$ do

find a Min-weight Edge $e^* = (v^*, u^*)$ among the all Edges (v,u) such that $v \in V_T$ & $u \in V - V_T$

$V_T \leftarrow V_T \cup \{u^*\}$

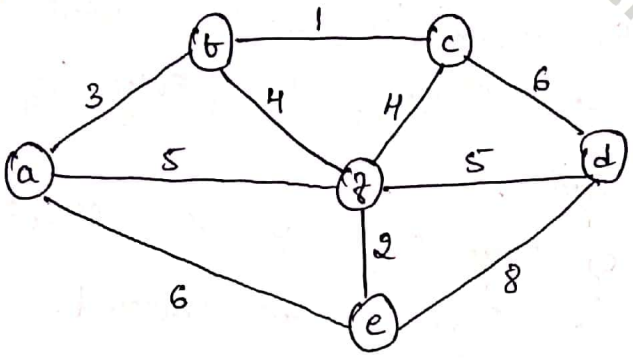
$E_T \leftarrow E_T \cup \{e^*\}$

return E_T

→ The time Complexity of prim's Algo is

$$O(|E| \log(|V|))$$

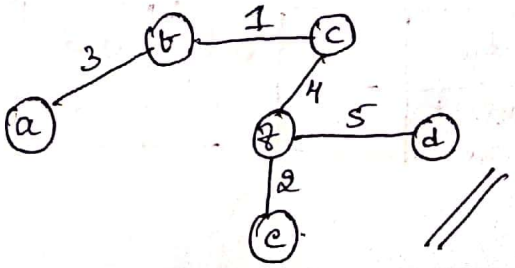
* Apply prim's Algo to find Min cost Spanning tree for given Weighted graph.



Tree vertices	Remaining vertices
a (-, -)	b(a,3), f(a,5), e(a,6), f(-,∞), c(-,∞) d(-,∞)
b(a,3)	f(a,5), e(a,6), f(b,4), c(b,1), d(-,∞)
c(b,1)	f(b,4), e(a,6), d(c,6)
f(b,4)	e(a,6), d(c,6), e(f,2), d(f,5)
e(f,2)	d(f,5), d(e,8)
d(f,5)	— All vertices are visited

∴ Tree vertices are

a(-,-) b(a,3) c(b,1)
f(b,4) e(f,2) d(f,5)



* Kruskal's Algorithm :-

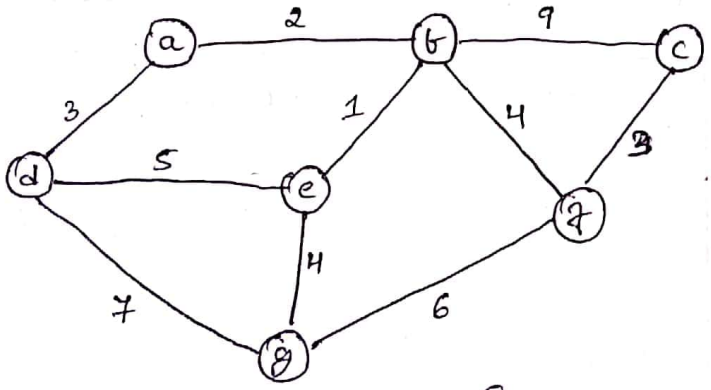
Used for finding Min Spanning tree of a Connected Weighted graph. Here, we will first Sort the Edges of the graph based on their weights.

→ The Edge with Min weight is added to the Empty tree then, Among the remaining list, next Smallest Edge is Consider -ed & is added to the Existing tree only "if the addition will not Create a Cycle".

If the Addition of any Edge to the Existing tree results in a cycle, Such an Edge is just ignored.

→ The procedure is Continued till the tree Contains all the Vertices of a graph.

→ Consider the foll Example



Find the Minimum Spanning Tree

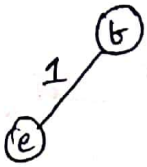
⇒ Firstly Sort the Edges of given graph based on weights (Non-decreasing order)

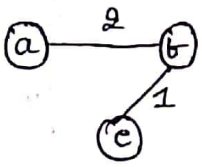
Edge	be	ab	ad	cf	bf	eg	de	gf	dg	bc
weight	1	2	3	3	4	4	5	6	7	9

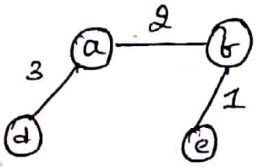
Tree Edge

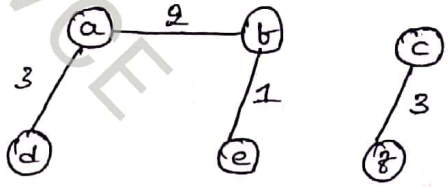
* be
* ab
* ad
* cf
* bf
* eg
* gf
* de
* dg
* bc

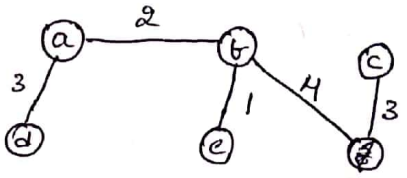
Constructed Tree

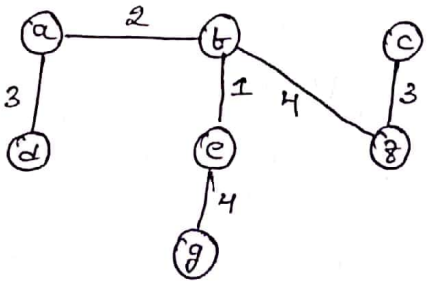












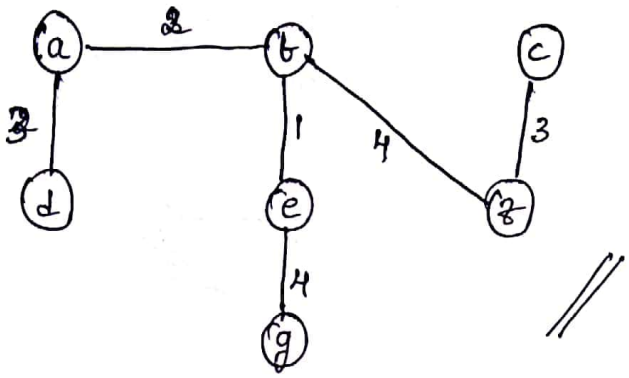
"gf" leads to cycle

"de" leads to cycle

"dg" leads to cycle

"bc" leads to cycle

∴ the Minimum Spanning tree is



* Algorithm Kruskal (G)

Ip:- Weighted Connected Graph G(V, E)

Op:- E_T, the Set of Edges, which is MST

Sort 'E' in non-decreasing order of Edge weights $w(e_1, \ldots e_n)$

$E_T \leftarrow \phi$

EdgeCount $\leftarrow 0$

$k \leftarrow 0$

while EdgeCount $< |V| - 1$ do

$k \leftarrow k + 1$

if $E_T \cup \{e_i\}$ is acyclic

$E_T \leftarrow E_T \cup \{e_i\}$

EdgeCount $\leftarrow$ EdgeCount $+ 1$

return E_T

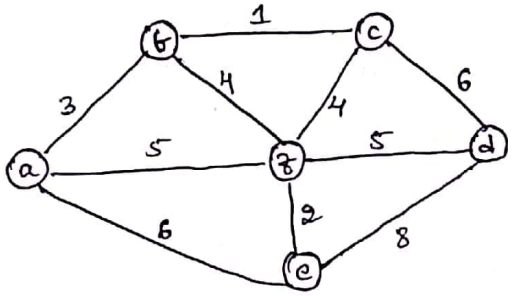
→ Even though Kruskal's Algorithm seems to be Simple than prim's Algo it is not fast than prim's, because, before adding Every Edge to the MST, we've to check whether it results in a cycle or not.

→ Unlike in prim's Algo, the Intermediate Steps of Kruskal's Algo May lead to disconnected Edges.

→ The Efficiency of Kruskal's Algorithm is

$$O(|E| \log |E|)$$

* Apply Kruskal's Algo to find Min Spanning tree for the following graph.



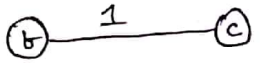
⇒ Sorted list of Edges

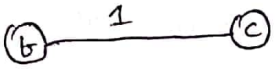
Edge	bc	ef	ab	bf	cf	af	df	ae	cd	de
weight	1	2	3	4	4	5	5	6	6	8

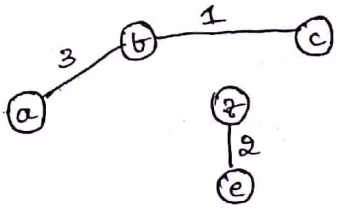
Tree Edges

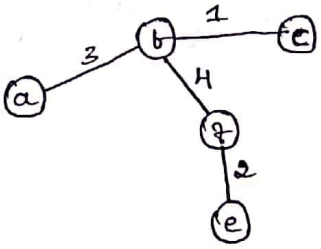
* bc
* ef
* ab
* bf
* cf

Constructed tree



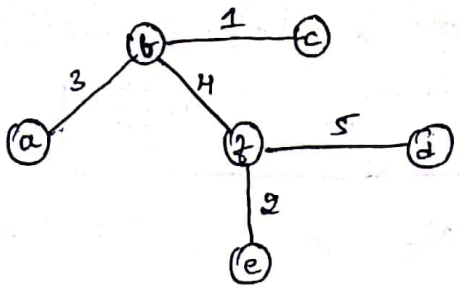






'cf' results in cycle

* df

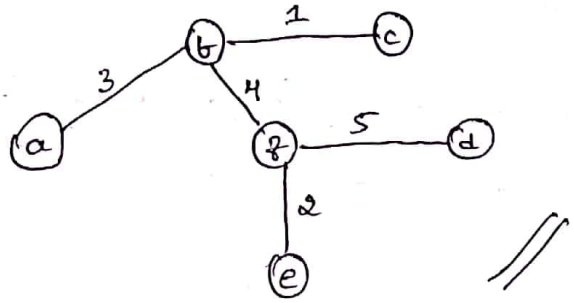


* ac — "ac" results in cycle

* cd — "cd" results in cycle

* de — "de" results in cycle

∴ The Minimum Spanning tree is



* Dijakstra's Algorithm (Single Source Shortest path)

Finding a Shortest path from a particular Vertex to all other Vertices Such a problem is known as "Single - Source Shortest path problem".

→ Dijakstra's Algorithm is Applied on given Source Vertex to find Shortest path to all other Vertices. as Shown

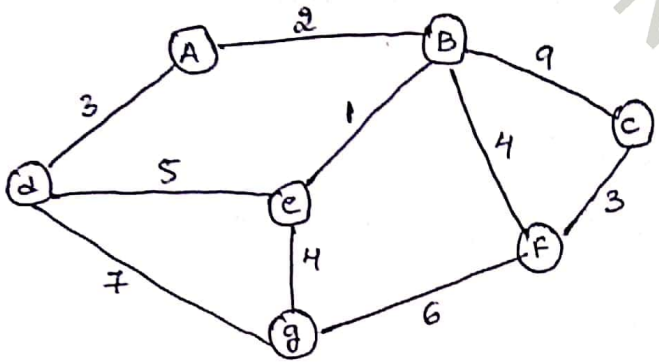
* Given a Source, firstly find the Vertex which is nearest to it.

* then, find the Second nearest & so on.

Thus at i^{th} Iteration, we will find the Shortest dista-nce between the Source to all other Vertices.

→ The Main theme of Dijakstrean Algo is a path that Starts at Any Arbitary Vertex (Source) & Visits all other Vertices of a graph with Min Cost.

→ Consider the foll Example graph with Source = A



Tree Vertices	Remaining Vertices
* a(-, 0)	B(a, 2) D(a, 3), C(-, ∞), E(-, ∞) F(-, ∞) e(-, ∞)
* B(a, 2)	C(B, 2+9), D(a, 3), E(B, 2+1), F(B, 2+4) e(-, ∞)
* D(a, 3)	C(B, 11), E(B, 2+1), F(B, 6), e(D, 3+7)
* E(B, 3)	F(B, 6), e(E, 3+4), C(B, 11)
* F(B, 6)	C(F, 6+3), e(E, 7)
* e(e, 7)	C(F, 9)
* C(F, 9)	

→ The Shortest paths set is given as

From a to b : A → b dist = 2

a to c : A → B → F → C dist = 9

a to d : A → D dist = 3

a to e : A → B → E dist = 3

a to f : A → B → F dist = 6

a to g : A → B → E → G dist = 7 //

* Dijakstra's Algorithm (G, s)

I/P:- A weighted connected graph G = (V, E)

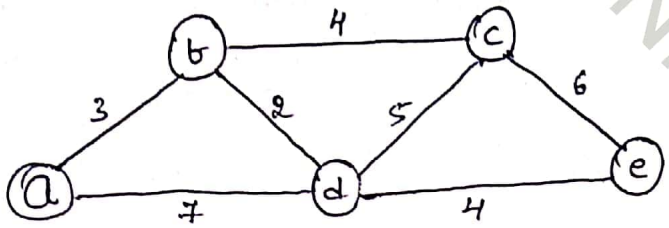
O/P: Shortest path from source to all other vertices.

```
for i ← 0 to n-1 do
    d[i] ← cost [source, i]
    S[i] ← 0
End for
S[source] ← 1
for i ← 1 to n-1 do
  find Adjacency vertex 'u' & distance 'd[u]' such that
d[u] is minimum
  add u to S
  S[u] ← 1
  for Every v ∈ V-S do
    if (d[u] + w(u,v) < d[v])
      d[v] ← d[u] + w[u,v)
    End if
  End for
End for
```

→ The time complexity of Dijakstra's Algorithm is

$$\boxed{\Theta(|V|^2)}$$ //

* Apply Dijakstra's Algo for the foll graph & Source vertex is 'a'



⇒

Tree vertices	Remaining vertices
* (a (-, 0)	b (a, 3), c (-, ∞), d (a, 7), e (-, ∞)
* b (a, 3)	c (b, 3+4), d (b, 3+2), e (-, ∞)
* d (b, 5)	c (b, 7), e (d, 5+4)
* c (b, 7)	e (d, 9)
* e (d, 9)	— × —

→ The Shortest path Set is given as

From A to B : A → B dist = 3

A to C : A → B → C dist = 7

A to D : A → B → D dist = 5

A to E : A → B → D → E dist = 9 //

* Huffman Trees :- (Huffman code)

In the field of Computer Science, Encoding & Decoding a particular text / information is an important Aspect to Save the Space.

Usually, Encoding is done by assigning some sequence of bits called "codeword" to Each character of the text.

→ There are Mainly two Methods of Encoding

* Fixed-length Encoding → Each character is Assigned a bit String of Same length.

* Variable-length Encoding → Each character is Assigned a bit String of diff length.

→ A "prefix code" also known as "prefix free code" is a Vari-able length Encoding Scheme that has the important property that no code word is a prefix of another.

Ex: If the character "a" is Encoded as "0", then no other Character has a code that Starts with "0".

"B" is Encoded as "10", then no other code word Start-s with "10".

→ "Huffman Algorithm" is used to Construct Such a tree which will give Shorter code for frequent characters & longer code for less frequent Characters.

→ The Steps to be followed in Huffman coding

* Step 1 :- Initialize 'n' One-node trees & label them with characters of text & frequency(occurance). Arr-ange the characters in non-decreasing order.

* Step 2 :- Find two trees/nodes with smallest weights make them left (lesser) & right (greater) subtree of a new tree.
put the sum of their weights as a root.

Repeat Step (2) till a single tree is obtained.

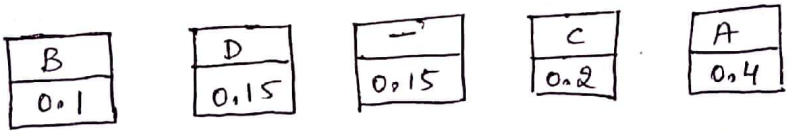
→ The Tree Constructed using the above Algorithm is called "Huffman Tree" & Codewords we get is "Huffman code".

* Example :- Construct a Huffman code for the following data

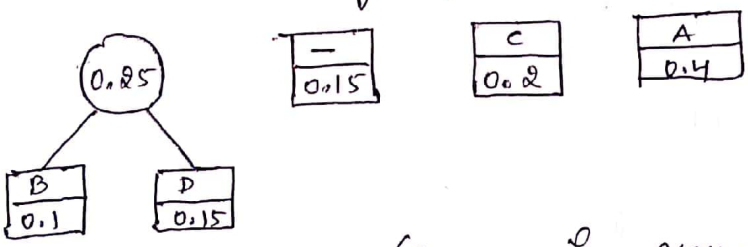
character	A	B	C	D	-
probability	0.4	0.1	0.2	0.15	0.15

i) Encode the text ABACABAD using the code
ii) Decode the code 100010111001010

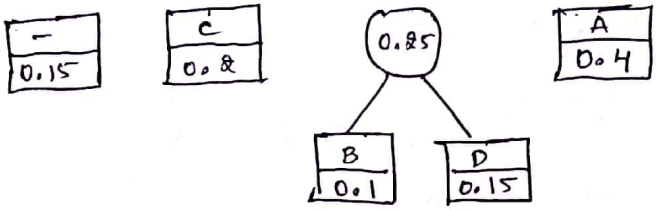
⇒ In the first step, we create 5 one-node trees as shown below in Ascending order of weight.



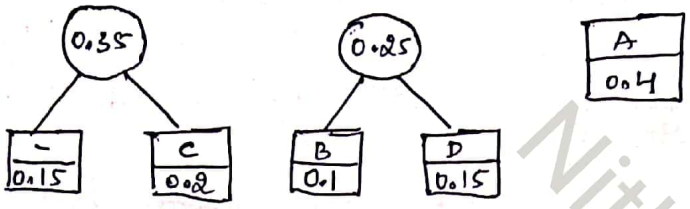
Choose smallest two nodes/subtree & construct a new tree with root node consisting of their sum.



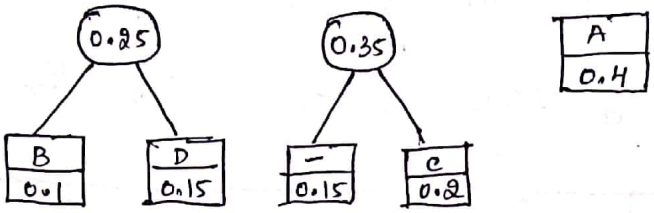
→ Again sort the nodes/subtree in non-decreasing order



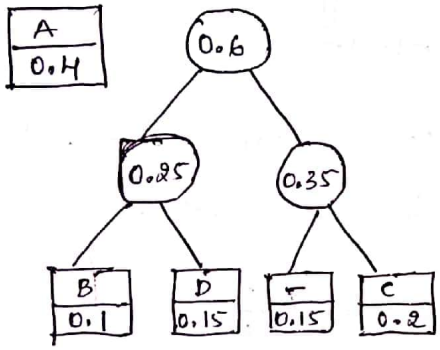
→ Choose Smallest two node/Subtree & Construct a new tree.



→ Again, sort it & choose Smallest two node/Subtree & Construct a tree.

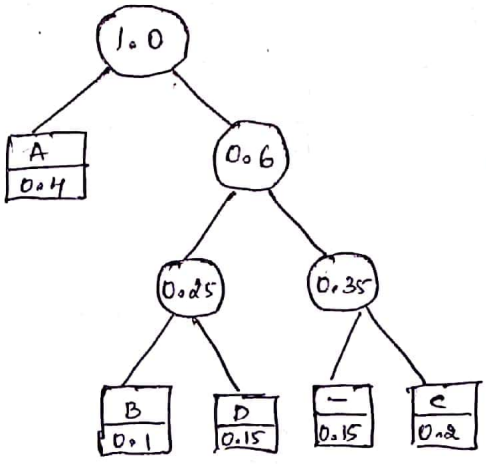


→ Again, choose two Smallest node/Subtree & Construct new one

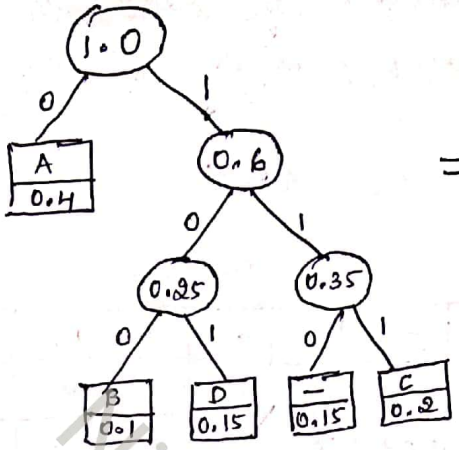


please note :-
while constructing tree place lesser value node/subtree on left & higher value node/subtree on right

→ Again repeat the procedure to get final tree



→ Finally place value '0' on left Edge of the tree & '1' on Right Edge of the tree.



⇒

A → 0
B → 100
C → 111
D → 101
- → 110

→ Encode = A B A C A B A D = 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0 1

Decode = 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 = B A D – A D A

The Time complexity of Heap sort is $O(n \log n)$

* Knapsack problem Using Greedy Method :-

Knapsack problem is a problem in "combinatorial optimization" in which, Given a set of items, Each with a weights $(w_1, w_2, w_3 \ldots w_n)$ & profits $(p_1, p_2 \ldots p_n)$ & Capacity of knapsack (bag) "M".

→ The Main objective is to place the objects into the knapsack so that Maximum profit is obtained & the weights of objects chosen should not Exceed the Capacity of knapsack.

→ The knapsack problem can be Stated as

$$\text{Maximize} \sum_{i=1}^{n} p_i x_i$$

where p_i → profit
x_i → fraction of object added

$$\text{Constraint} \sum_{i=1}^{n} w_i x_i \leq M$$

weights added to the bag should be less than or Equal to bag capacity 'M'

* Consider the foll Example of knapsack problem for $n = 7$

$m = 15$ $\quad p = (10, 5, 15, 7, 6, 18, 3)$ $\quad w = (2, 3, 5, 7, 1, 4, 1)$

⇒ given m (capacity) $= 15$ $\quad n = 7$

P_i	10	5	15	7	6	18	3
w_i	2	3	5	7	1	4	1
P_i/w_i	5	1.67	3	1	6	4.5	3

→ Arrange the Above table based on P_i/w_i (non-decreasing) & Add Extra row "X_i" to Mark the added objects.

P_i	6	10	18	15	3	5	7
w_i	1	2	4	5	1	3	7
P_i/w_i	6	5	4.5	3	3	1.67	1
X_i	1	1	1	1	1	0.66	0

$m = 15$

object(i)	w_i	P_i	$X = 1$ or m/w_i	profit $X * P_i$	Remaining capacity (m) $m - w_i$
1	1	6	1	$1*6 = 6$	$15 - 1 = 14$
2	2	10	1	$1*10 = 10$	$14 - 2 = 12$
3	4	18	1	$1*18 = 18$	$12 - 4 = 8$
4	5	15	1	$1*15 = 15$	$8 - 5 = 3$
5	1	3	1	$1*3 = 3$	$3 - 1 = 2$
6	3	5	$2/3 = 0.666$	$0.666*5 = 3.33$	$2 - 2 = 0$

profit $= 6 + 10 + 18 + 15 + 3 + 3.33$
$= 55.35$

→ Fraction $(X_1, X_2, X_3, X_4, X_5, X_6, X_7) = (1, 1, 1, 1, 1, 0.67, 0)$

Weights $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (1, 2, 4, 5, 1, 3, 7)$ //

* Algorithm for Knapsack problem using greedy Method

Knapsack (m, n, w, p, x)

I/p:- m is the capacity of knapsack
n is the no of objects
w & p are weights & profits.

O/p:- Maximum profit

```
for i ← 1 to n do
    x[i] ← 0
    weight ← 0
    while weight < W do
        i ← the best remaining object
        if weight + w[i] ≤ W then
            x[i] ← 1
            weight ← weight + w[i]
        else
            x[i] ← (W - weight) / w[i]
            weight ← W
    return x.
```

* Solve the Greedy knapsack problem where $m = 10$ $n = 4$
$P = (40, 42, 25, 12)$ & $w = (4, 7, 5, 3)$

⇒ given M (capacity) = 10 $n = 4$

P_i	40	42	25	12
w_i	4	7	5	3
P_i/w_i	10	6	5	4

→ Arrange the Above table based on P_i/w_i (Decreasing) & Add Extra row "x_i" to Mark the added objects

P_i	40	42	25	12
w_i	4	7	5	3
P_i/w_i	10	6	5	4
X_i	1	6/7	0	0

$w = 10 = w_c$

object (i)	w_i	P_i	$x = 1$ or w_c/w_i	profit $x * P_i$	remaining capacity
–	–	–	–	–	10
1	4	40	1	$1 * 40$	$10 - 4 = 6$
2	7	42	$6/7$ (w_c/w_i)	$6/7 * 42 = 36$	$6 - 6 = 0$

profit = $40 + 36$
$= 76$ //

→ Fraction $(x_1, x_2, x_3, x_4) = (1, 6/7, 0, 0)$
Weights $(w_1, w_2, w_3, w_4) = (4, 7, 5, 3)$ //

* Job Sequencing with deadlines :- The Sequencing of Jobs on a Single processor with deadline Constraints is named as "Job Sequencing with deadlines".

→ You are given a set of Jobs where Each Job has a def -ined deadline & some profit Associated with it. The profit of that is given only when that Job is completed within its deadline.

→ The Main objective is to Maximize the total profit by Completing only one Job at a time within a given dead -line.

→ The Constraints Associated with Job Sequencing with deadline are
* only one processor is Available for processing all Jobs
* processor take one unit of time to complete a Job.

* All the Jobs have to be Completed within their respective deadline to obtain the profits Associated with them.

* Job Sequencing with deadline Algorithm

→ Step 1 :- Sort all the given Jobs in decreasing order of their profit.

→ Step 2 :- Check the value of Max deadline. Then, draw a chart where Max time on chart is the value of Max deadline.

→ Step 3 :- pick up the Jobs one by one & then put them on chart as far as possible from Starting point. Such that Jobs get Completed before its deadline. Repeat Step 3.

* Conider the foll Jobs, their deadlines & Associated profit as Shown.

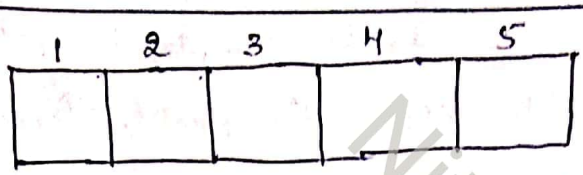
Jobs	J_1	J_2	J_3	J_4	J_5	J_6
deadline	5	3	3	2	4	2
profits	200	180	190	300	120	100

Write the optimal Schedule that gives max profit ? & what is the Max Earned profit.

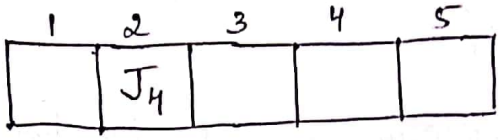
⇒ Firstly, Sort all given Jobs in the decreasing order of their profits.

Jobs	J_4	J_1	J_3	J_2	J_5	J_6
Deadline	2	5	3	3	4	2
profits	300	200	190	180	120	100

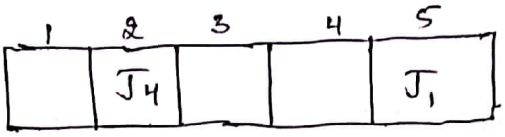
The Max deadline = 5 so draw a chart with Max time on chart = 5 units.



→ Now, take Each Job one by one in order & place them as far as possible within deadline, Finally we take Job J_4 place it in 4th deadline '2'

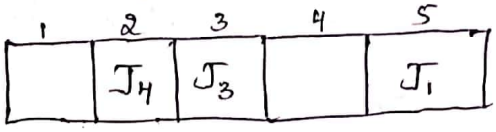


→ Next J_1, 4th deadline is '5'

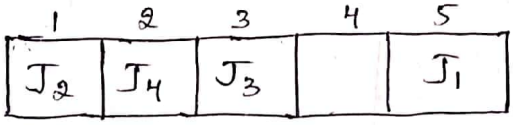


Note: If deadline slot is filled check towards LHS of deadline for slots

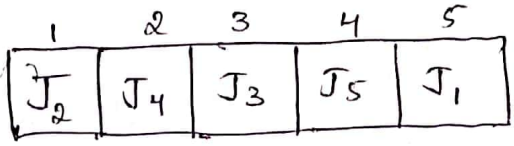
→ Next J_3, 3th deadline is '3'



→ Next J_2, 2th deadline is '3', which is already filled so check towards "left side of chart"
only Empty slot within deadline is "Slot 1", place it.



→ Next J_5, 5th deadline is '4'. slot is Empty, place it on that slot.



//

∴ The optimal Schedule is

$(J_2, J_4, J_3, J_5, J_1)$ //

∴ Total profit Earned is

$180 + 300 + 190 + 120 + 200$.

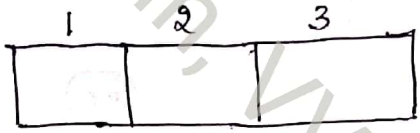
$= 900$ //

* Let n=5, profits [10, 3, 33, 11, 40] & deadlines [3, 1, 1, 2, 2] respectively. Find the optimal Soln using greedy Algo.

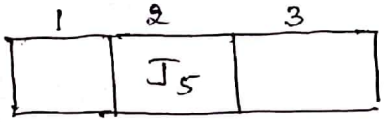
⇒ Sort, all the Job in decreasing order of their profit

Jobs	J_5	J_3	J_4	J_1	J_2
deadlines	2	1	2	3	1
profit	40	33	11	10	3

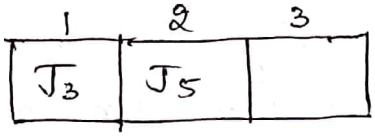
Max deadline = 3, so draw a chart with Max time on chart is 3 units



→ Select first Job J_5, place it as far as possible within deadline, so place it on Slot '2'

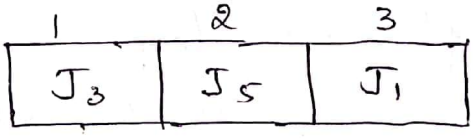


→ Next J_3, its deadline is '1' & slot '1' is Empty place it



→ Next J_4, its deadline is '2', which is filled & Slots on leftside of '2' are also filled so ignore it.

→ Next J_1, its deadline is '3' & Slot 3 is Empty, place it.



∴ The optimal Schedule is

J_3, J_5, J_1 //

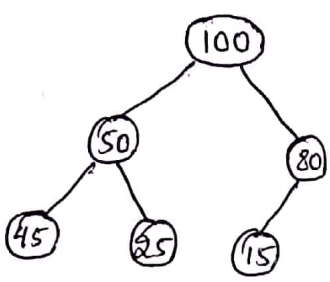
∴ Total profit is

$40 + 33 + 10$

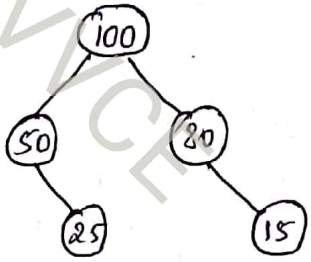
$= 83$ //

* Heap Sort :- A Heap is a Complete Binary Tree or Almost Complete Binary Tree Satisfying the foll two Conditions

  * All its levels are full Except possibly the last level. If the last level is not full, all the nodes should be filled only from left-to-right.
  * The key at Each node Should be greater than or Equal to key of its children.



Heap



Not-Heap

→ Heap sort is a Sorting technique used to Arrange numbers in Ascending or Descending orders.
This Sorting Technique uses Creating a Max-Heap so that root is having greater value than its children's. The Same Condition Must be true for Each Subtree of a Heap.

→ Heap Sort Technique Consists of two phases

* Heap Creation phase → Heap can be Created using Bottom-up Approach
* Sorting phase → Consists of Two Steps
  * Exchange the root item with last Element of Heap
  * Decrement the Size of Heap by 1 & Heapify (Heap Construction).

* Heap Sort Algorithm (n, a[])

I/p:- list of nos a[]

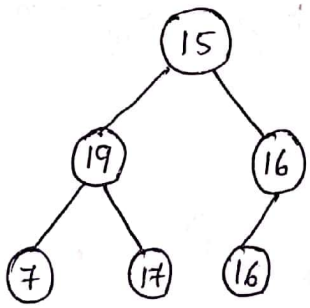
O/p:- Sorted list of nos

```
Bottom_up_Heapify (n, a)        // Construct a Heap
    For i = n-1 down to 0
        Exchange (a[0], a[i])
        Bottom_up_Heapify ();
    End For
```

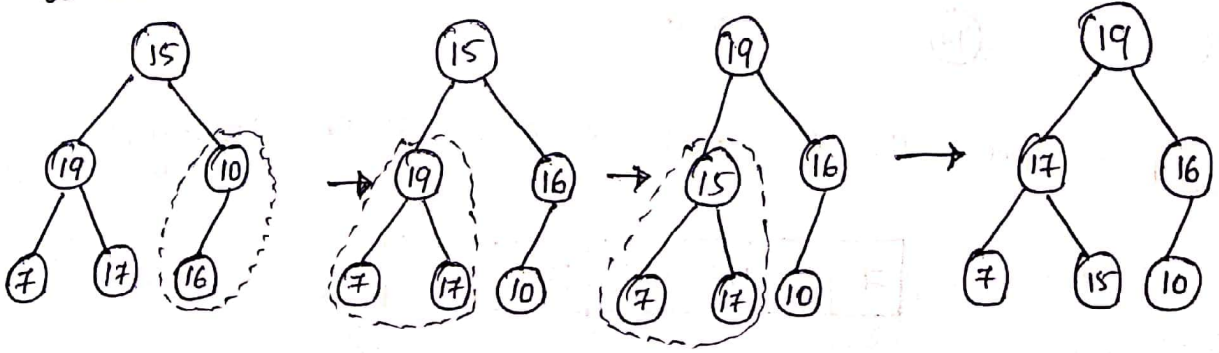
→ The Time Complexity of Heap Sort Algo is $O(n \log n)$ //

* Construct the Heap for the list 15, 19, 10, 7, 17, 16 by Bottom-up-Approach.

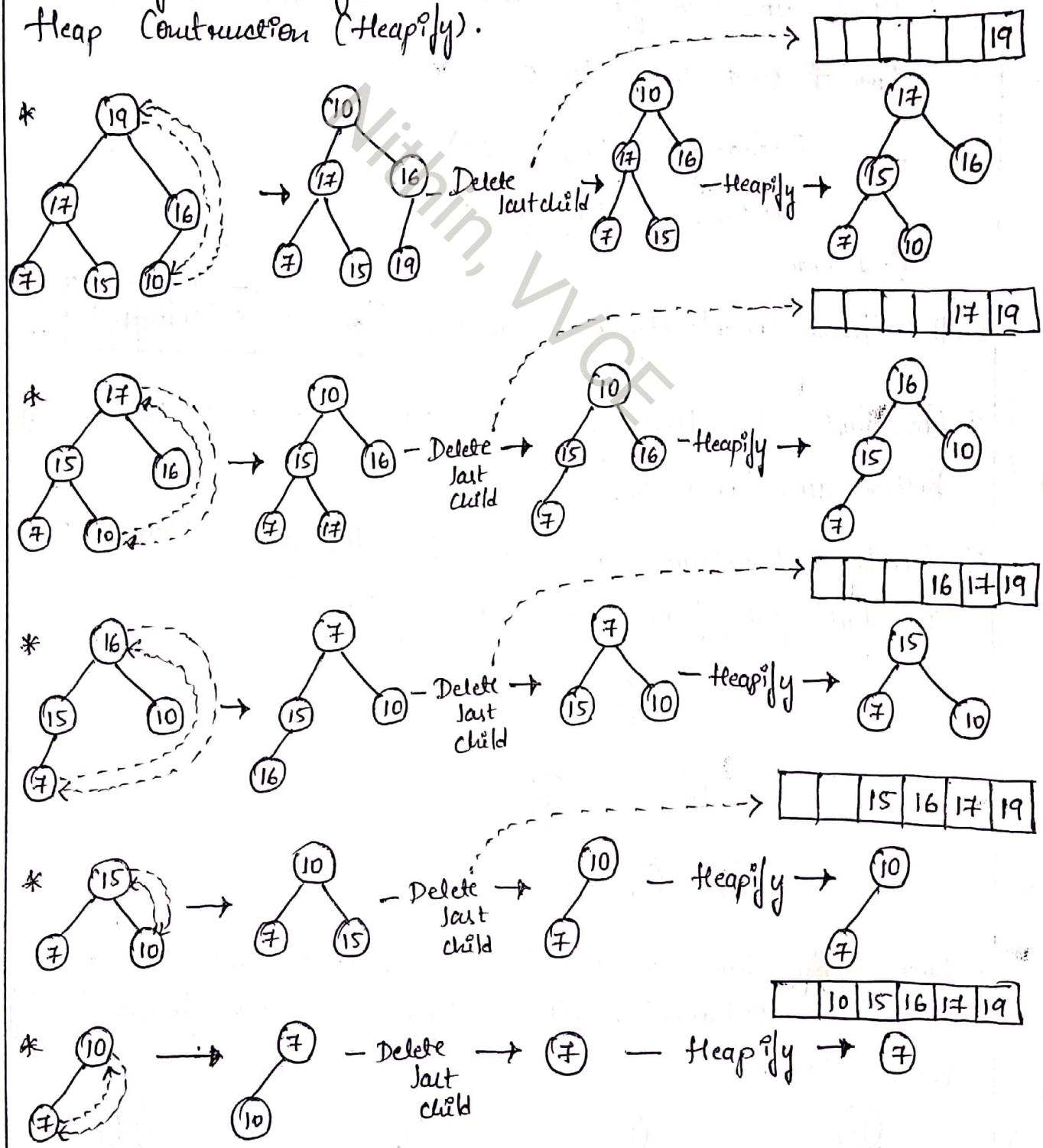
⇒ Firstly, Construct a Heap for the given values As it is, but fill from left-to-right.



→ Now Apply Heap Construction rule, Root value Should be greater than children's. First Heapify Sub-tree followed by whole tree.



→ Once after Completion of Heap Construction (Heapify), In Every Individual Step, Exchange Root value with last child value & Delete the last child & Store the value in Array.
Once After Deleting the last child (root) Again Apply the Heap Construction (Heapify).



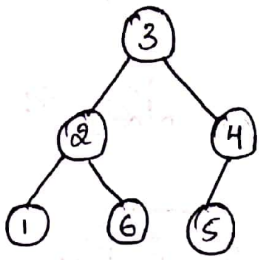
* (7) → Delete the last node & place it on Array.

7	10	15	16	17	19

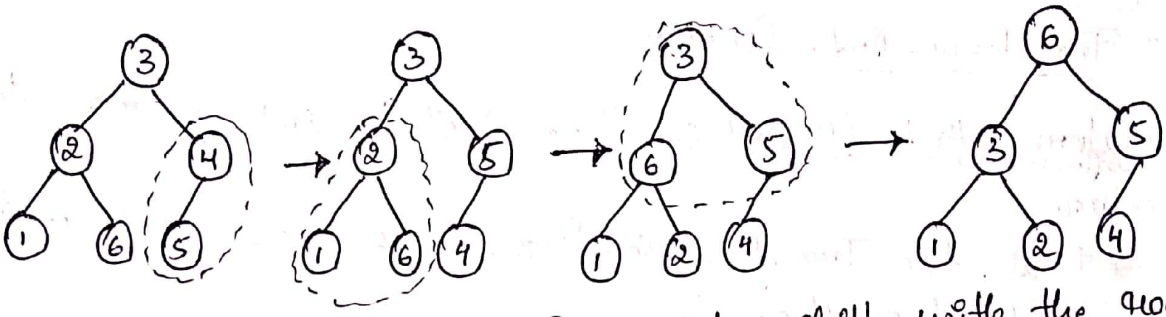
* Sort the below list using heap sort

3, 2, 4, 1, 6, 5

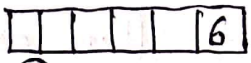
⇒ Firstly, Construct a heap for the given values As it is, but fill the values from left-to-right

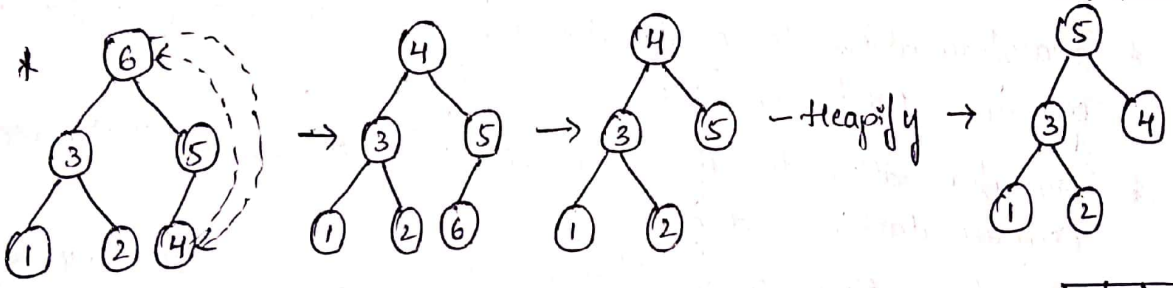


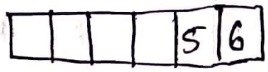
→ Now Apply Heap Construction rule, root value should be greater than children

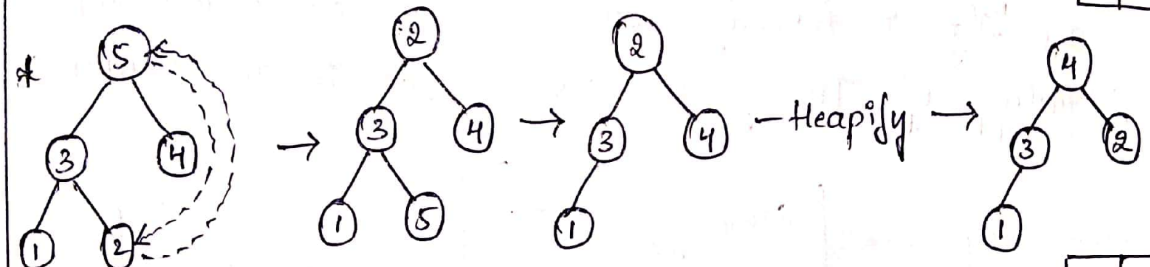


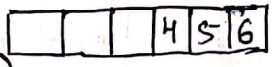
→ Now, Apply HeapSort by Swaping last child with the root. Once After Swaping delete last child & place it in result. Again Heapify

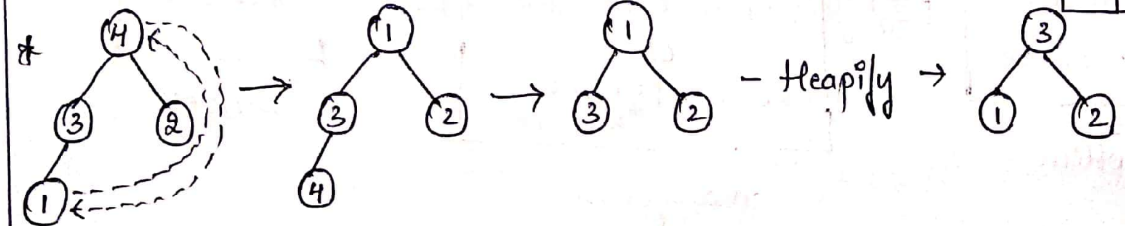


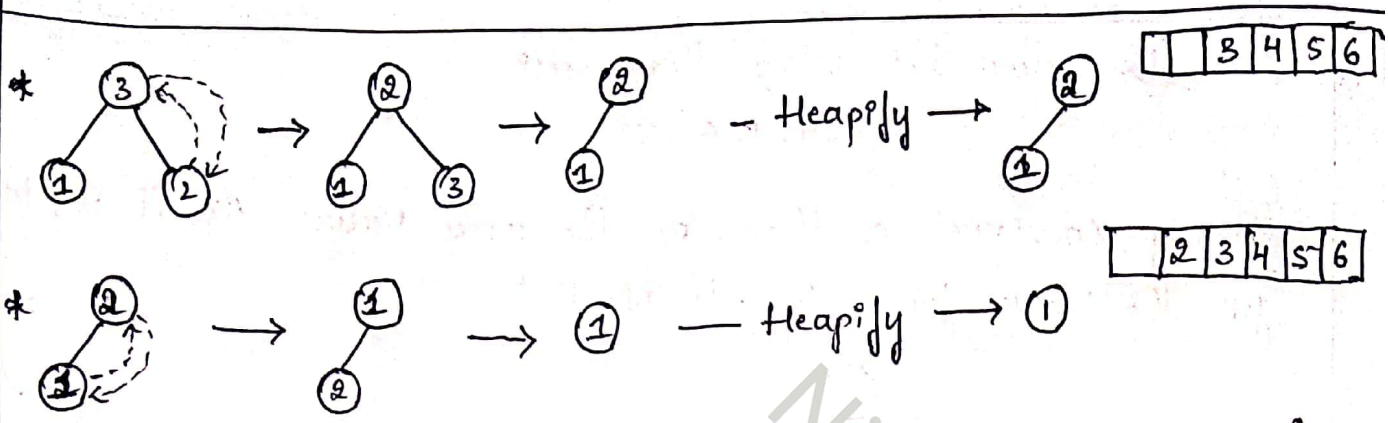












* (1) → Delete the last node & place it on resultant Array

1	2	3	4	5	6

The procedure that we Applied to Solve the HeapSort problem is "Transform - And - Conquer"

* Transform - And - Conquer :- these Method work as two - Stage procedure
  * First, the Transformation Stage, the problem's Instance is Modified or changed.
  * Second, The Conquering Stage, it is Solved.

→ There are 3 Variation in Transforming from Instance
  * Transformation to a Simpler or More Convenient Instance of Same problem. (Instance Simplification).
  * Transformation to diff representation of Same Instance (Representation change)
  * Transformation to an Instance of a diff problem for which an Algo is Already Available. (problem reduction)

→ The Strategy of Transform - And - Conquer is as Shown