MODULE III

GREEDY METHOD

CONTENTS

- INTRODUCTION TO GREEDY METHOD
- GENERAL METHOD
- COIN CHANGE PROBLEM
- ASSIGNMENT

INTRODUCTION TO GREEDY METHOD

- Greedy method is one of the straightforward design technique which is applicable to wide variety of applications

- Greedy algorithms are typically used to solve an optimization problem.

- An Optimization problem is one in which, we are given a set of input values, need to obtain subset of inputs as a solution w. r. t. some constraints or conditions.

- Generally an optimization problem has n inputs (call this set as input domain or Candidate set, C), we are required to obtain a subset of C (call it solution set, S where $S \subseteq C$) that satisfies the given constraints or conditions

INTRODUCTION TO GREEDY METHOD [Contd..]

- An Optimization Problem usually has 2 solutions: **Feasible Solution** and **Optimal Solution**

- Solution which contains a subset of inputs that satisfies the given **constraints** is known as **Feasible solution**.

- A **feasible solution** that either maximizes or minimizes a given **objective function** is known as **optimal solution**

INTRODUCTION TO GREEDY METHOD [Contd..]

- The greedy method suggests to devise an algorithm that works in stages (by considering one input at a time)
- ✓ Where we consider the inputs in an order, based on some selection procedure
- ✓ Use some optimization measure for selection procedure ,at every stage, examine an input to see whether it leads to an optimal solution
- ✓ If the inclusion of input into partial solution yields an infeasible solution, discard the input; otherwise, add it to the partial solution

CONTROL ABSTRACTION

```
Algorithm Greedy(a, n)
// a[1 : n] contains the n inputs.
{
    solution := ∅; // Initialize the solution.
    for i := 1 to n do
    {
        x := Select(a);
        if Feasible(solution, x) then
            solution := Union(solution, x);
    }
    return solution;
}
```

CONTROL ABSTRACTION

- Here, Select is a function which is used to select an input from the set a[1:n] and removes it. The selected input value is assigned to “x”.
- Feasible is a boolean valued function which verifies the constraints and determines whether the “x” can be included into the solution vector or not.
- Union is a function which is used to add solution “x” to the partially constructed set and updates the objective function.

COIN CHANGE PROBLEM

Problem Statement

Given coins of several denominations find out a way to give a customer an amount with fewest number of coins.

Example

- First finding the number of ways of making changes for a particular amount of cents, **n**, using a given set of denominations **C={c1…cd}** (e.g, the US coin system: {1, 5, 10, 25, 50, 100})
- If n = 4,C = {1,2,3}
- The feasible solutions:
  - 4 coins {1,1,1,1},
  - 3 coins {1,1,2}
  - 2 coins {2,2},{1,3}.
- Second, minimizing the number of coins returned for a particular quantity of change :
- The Optimal Solution will be 2 coins i.e., {2,2} or {1,3}

Example

- Available coins are {1, 5, 10, 25} and n= 30 Cents. Then the solutions obtained are shown below:

Solution No	Denomination	No of coins	Amount /Total No of Coins
1	1	30	30/30
2	5	6	30/6
3	10	3	30/3
4	**25**	**1**	**30/2**
	5	**1**	

Solution 1,2 &3 are feasible solution to the problem where as Solution 4 is the optimal solution

Basic principle is: At every iteration for search of a coin, take the largest coin which can fit into remain amount to be changed at that particular time. At the end you will have optimal solution.

Assignment

Solve coin change problem(Identify feasible and Optimal Solution) :

- ✓ Suppose you are given the coins {1,5,10} with N = 10 cents, what are the total number of permutations of the coins you can arrange to obtain 10 cents.
- ✓ Suppose you are given the coins {1,5,10,25} with N = 48 cents, what are the total number of permutations of the coins you can arrange to obtain 48 cents.

Knapsack Problem

Given :

- A knapsack (kind of shoulder bag) with limited weight capacity.
- Some items each having some weight and profit or value.

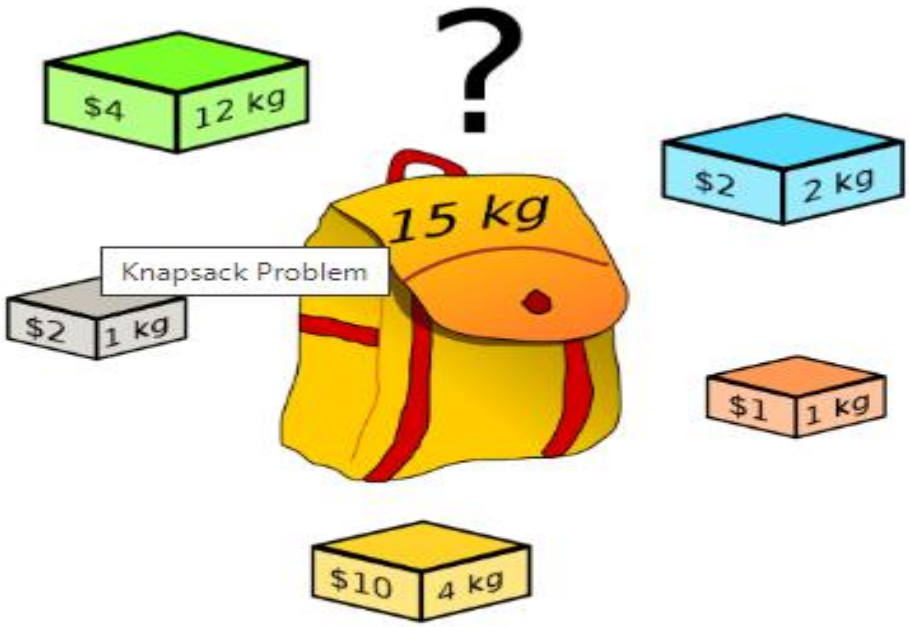
- **The problem states-**

Which items should be placed into the knapsack such

that-

- The value or profit obtained by placing the items into the knapsack is maximum.
- And the weight limit of the knapsack does not exceed.

Knapsack Problem



Knapsack Problem

Knapsack Problem Variants

Knapsack problem has the following two variants-

- Fractional Knapsack Problem (Items can be selected partially)-Greedy Method
- 0/1 Knapsack Problem(Items should be selected completely but not partially)-Dynamic Programming

Fractional Knapsack Problem

Problem Statement:

- Given ***n*** objects each have a ***weight*** w_i and a ***profit*** p_i , and given a knapsack of total ***capacity m***.
- The problem is to pack/fill the knapsack with these objects in order to maximize the total value or profit of those objects packed without exceeding the knapsack's capacity.
- More formally, let x_i denote the fraction of the object ***i*** to be included in the knapsack, $0 \leq x_i \leq 1$, **for** $1 \leq i \leq n$. The problem is to find values for the x_i.
- If a fraction x_i $(0 \leq x_i \leq 1)$ of the object i is placed in the bag. A profit $p_i * x_i$ is made.

Fractional Knapsack Problem

- The objective in the problem is to obtain the maximum profit by filling the bag with given objects.
- If it is not possible to include an object entirely a fraction of the object can be included and accordingly a fraction of the profit is earned.
- The knapsack problem can be stated mathematically as follows.

Maximize $\sum p_i x_i$ $1 \leq i \leq n$ where n is the number of objects. Such that,

$$\sum w_i x_i \leq m$$

For every object x_i : $0 \leq x_i \leq 1$ and $1 \leq i \leq n$

Fractional Knapsack Problem-EXAMPLE

- Consider 3 objects whose profits and weights are defined as:
- Total No of objects : n=3 and Knapsack capacity : m=20
- $(P_1, P_2, P_3) = (25, 24, 15)$ & $(W_1, W_2, W_3) = (18, 15, 10)$

- Problem is to determine the optimum strategy for placing the objects into the knapsack.
- The problem can be solved by the greedy approach where in the inputs are arranged according to selection process (greedy strategy) and solve the problem in stages.
- The various greedy strategies for the problem could be as follows.

Fractional Knapsack Problem-EXAMPLE

We know that n=3, m=20, listing the objects with its corresponding weights and profits as below

Object(i)	1	2	3
Weight(w_i)	18	15	10
Profit(p_i)	25	24	15

Fractional Knapsack Problem-EXAMPLE

I Strategy ::: Greedy about profit: - Select an object which has highest profit i.e arrange the objects in descending order of profits.

Object(i)	1	2	3
Weight(w_i)	18	15	10
Profit(p_i)	25	24	15

Object(i)	1	2	3
Weight(w_i)	18	15	10
Profit(p_i)	25	24	15

Object(i)	Weight(w_i)	Profit(p_i)	x_i	$w_i x_i$	$p_i x_i$	Remaining Capacity
-	-	-	-	-	-	20
1	18	25	1	18*1=18	25*1=25	20-18=2
2	15	24	2/15	15*2/15=2	24*2/15=3.2	2-2=0
3	10	15	0	-	-	-

From the above table, we can observe that

✓$(x_1,x_2,x_3)=(1,2/15,0)$

✓Max Profit= $\sum \mathbf{p_i x_i}$ = 25*1+24*2/15=25+3.2=28.2

✓Max Weight= $\sum \mathbf{w_i x_i}$ = 18*1+15*2/15=20

Fractional Knapsack Problem-EXAMPLE

II Strategy ::: Greedy about weight: - Select an object which has least weight i.e arrange the objects in ascending order of weights.

Object(i)	3	2	1
Weight(w_i)	10	15	18
Profit(p_i)	15	24	25

Object(i)	3	2	1
Weight(w_i)	10	15	18
Profit(p_i)	15	24	25

Object(i)	**Weight(w_i)**	**Profit(p_i)**	**x_i**	**$w_i x_i$**	**$p_i x_i$**	**Remaining Capacity**
-	-	-	-	-	-	20
3	10	15	1	10*1=10	15*1=15	20-10=10
2	15	24	2/3	15*2/3=10	24*2/3=16	10-10=0
1	18	25	0	-	-	-

From the above table, we can observe that

- ✓$(x_1,x_2,x_3)=(0,2/3,1)$
- ✓Max Profit= $\sum \mathbf{p_i\ x_i}$ = 15*1+24*2/3=15+16=31
- ✓Max Weight= $\sum \mathbf{w_i\ x_i}$ = 10*1+15*2/3=20

Fractional Knapsack Problem-EXAMPLE

III Strategy ::: Greedy about profit per unit weight: -

- Obtain the ratio p_i / w_i of all objects
- Select an object which has highest profit/weight ratio i.e arrange the objects in descending order of the ratio profit/weight.

Object(i)	1	2	3
Weight(w_i)	18	15	10
Profit(p_i)	25	24	15
p_i/w_i	1.38	1.6	1.5

Object(i)	2	3	1
Weight(w_i)	15	10	18
Profit(p_i)	24	15	25
p_i/w_i	1.6	1.5	1.38

Object(i)	**Weight(w_i)**	**Profit(p_i)**	**x_i**	**$w_i x_i$**	**$p_i x_i$**	**Remaining Capacity**
-	-	-	-	-	-	20
2	15	24	1	15*1=15	24*1=24	20-15=5
3	10	15	1/2	10*1/2=5	15*1/2=7.5	5-5=0
1	18	25	0	-	-	-

From the above table, we can observe that

✓$(x_1,x_2,x_3)=(0,1,1/2)$

✓Max Profit= $\sum \mathbf{p_i\ x_i}$ = 24*1+15*1/2=24+7.5=31.5

✓Max Weight= $\sum \mathbf{w_i\ x_i}$ = 15*1+10*1/2=20

Greedy Strategy III: **Greedy about profit per unit weight - always yields an optimal solution to the knapsack problem**

Assignment

Solve the knapsack problem using greedy method

- ✓ Number of objects n=4, Knapsack capacity m=10, Weights (w1,w2,w3,w4)=(4,7,5,3) & Profit (p1,p2,p3,p4)= (40,42,25,12) **[Jan 2018]**

- ✓ Given n=7, m=15, Weights= (2,3,5,7,1,4,1) , Profits = (10,5,15,7,6,18,3) **[July 2019]**

MODULE III

GREEDY METHOD

RECAP…

- INTRODUCTION TO GREEDY METHOD
- GENERAL METHOD
- COIN CHANGE PROBLEM
- KNAPSACK PROBLEM
- ASSIGNMENT

AGENDA

- KNAPSACK PROBLEM-ALGORITHM AND ANALYSIS
- JOB SEQUENCING WITH DEADLINES PROBLEM
- ASSIGNMENT

Algorithm- Knapsack Problem

Algorithm Greedy knapsack (m, n)

//p [1:n] and w[1:n] contain the profits and weights

//respectively of the n objects ordered such that p[i]/w[i]=>p[i+1]/w[i+1].

//m is the knapsack size and x [1:n] is the solution vector

```
{
    for i: = 1 to n do
        x[i] = 0.0        // initialize x
    U : = m;
    Profit=0.0
    for i: = 1 to n do
    {
        if (w[i] > U) then   // if w[i] > remaining capacity then stop the selection
            break;
         x[i]: = 1.0;      // if w[i]<= remaining capacity then x[i]=1
        U: = U-w[i];     //update remaining capacity of the knapsack
        Profit+=p[i]*x[i]    // update profit
    }
    if (i ≤ n) then
        x[i]: = U/w[i];      //partial selection of object
         Profit+=p[i]*x[i]
}
```

Analysis- Knapsack Problem

- If we do not consider the time considered for sorting the inputs then all of the three greedy strategies complexity will be O(n).
- If we consider the time considered for sorting the inputs then all of the three greedy strategies complexity will be O(n log n).
  - Complexity of sorting using any better sorting algorithm O(n log n)
  - Complexity of selecting objects O(n)
  - Therefore the time complexity of knapsack problem will be max(O(nlogn),O(n)) i.e., O(n logn)

Job Sequencing with deadlines

Problem Statement:

- We are given a set of “n” jobs.
- Associated with each job there is an integer dead line $d_i \geq 0$ and a profit $p_i > 0$.
- For any job i the profit p_i is earned if and only if the job is completed by its dead line.
- To complete a job one has to process the job on a machine for one unit of time.
- Only one machine is available for processing the jobs.

Job Sequencing with deadlines

- A **feasible solution** for the problem will be a subset “J” if jobs such that each job in this subset can be completed by its deadline. The value of a feasible solution “J” is the sum of the profits of the jobs in “J”.
- An **optimal solution** is a feasible solution with maximum value of the profit.
- The problem involves identification of a subset of jobs which can be completed by its deadline. Therefore the problem suites the subset methodology and can be solved by the greedy method.

Job Sequencing with deadlines - Greedy Strategy

- The greedy strategy for obtaining optimal solution is to consider the jobs in decreasing order of profits and solve using Gantt chart.
- As there are only n jobs and each job takes one unit of time, it is necessary to consider the time slots [i-1,i] $1 \leq i \leq b$,

  where $b = \min \{n, \max \{d_i\}\}$
- The time slots are partitioned into b sets like [0,1][1,2][2,3]…… [b-1,b]
- If job i has not been assigned a processing time, then assign it to slot [α-1, α], where α is the integer such that $1 \leq \alpha \leq d_i$ where d_i is the deadline of the i^{th} job and if the slot [α -1, α] is not free then assign [α -2, α -1] and so on .
- If there is no slot, the new job is not included.

Job Sequencing with deadlines - Problem

Given, n = 5, $(p_1, p_2, p_3, p_4, p_5) = (20,15,10,5,1)$ and $(d_1, d_2, d_3, d_4, d_5) = (2,2,1,3,3)$.

- Write the optimal schedule of jobs that gives maximum profit.
- Are all the jobs completed in the optimal schedule?
- What is the maximum earned profit?

Job Sequencing with deadlines - Problem

Solution-

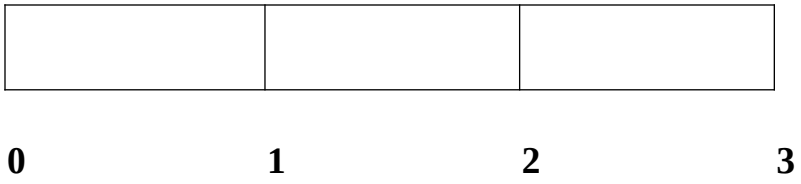
Step 1: Sort all the given jobs in decreasing order of their profit

Job	J1	J2	J3	J4	J5
Deadline(d_i)	2	2	1	3	3
Profit(p_i)	20	15	10	5	1

Job Sequencing with deadlines - Problem

Step 2:

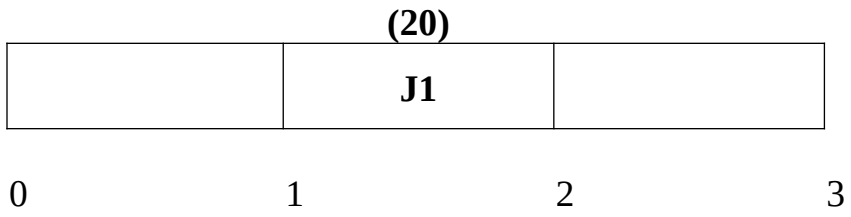
- The maximum number of time slots b=min(n,max(d_i))=(5,3)= 3. Therefore the time slots are[0,1],[1,2],[2,3]
- So, draw a Gantt chart with maximum time slots on Gantt chart = 3 units as shown below-



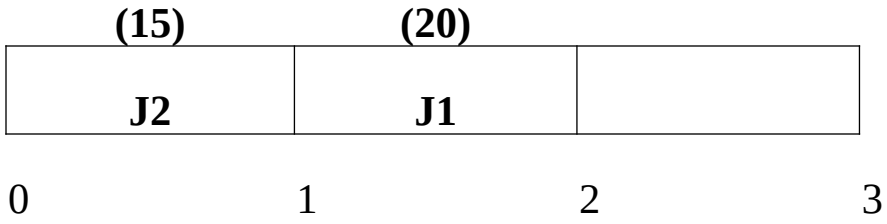
• Take each job one by one in the order they appear in Step 1 and place the job on Gantt chart

Job Sequencing with deadlines - Problem

Step 3: Take job J1, Since its deadline is 2, so we place it in the slot [1,2] as shown below:

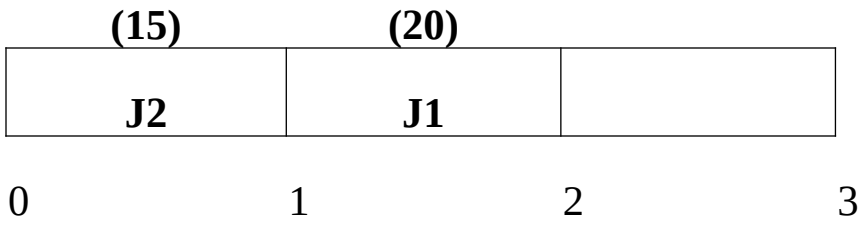


Step 4: Take job J2, Since its deadline is 2, so we need to place it in the slot [1,2], but since the slot is not free(occupied by J1) we can place J2 in the free slot [0,1] as shown below:



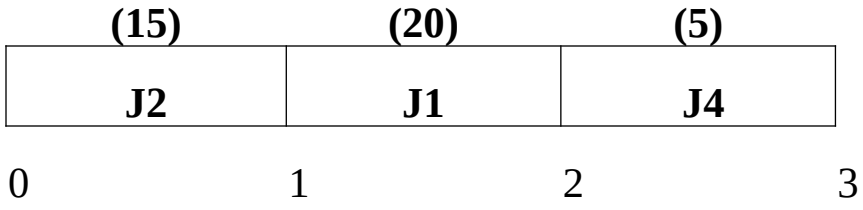
Job Sequencing with deadlines - Problem

Step 5: Take job J3, Since its deadline is 1, so we need to place it in the only one slot [0,1], but since the slot is not free(occupied by J2) and hence we cannot allocate the job J3



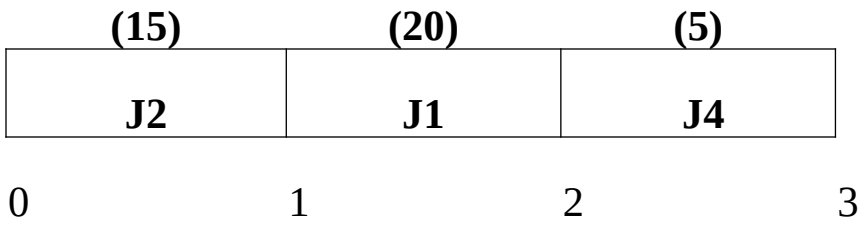
Job Sequencing with deadlines - Problem

Step 6: Take job J4, Since its deadline is 3, so we need to place it in the slot [2,3] which is free as shown below:

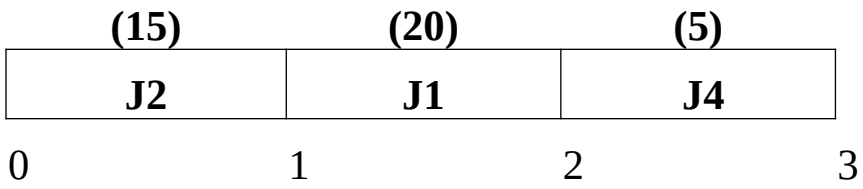


Job Sequencing with deadlines - Problem

Step 5: Take job J5, Since its deadline is 3, so we can place it in the slots [2,3] [1,2],[0,1], but since all the slots are not free(occupied by J4,J2,J1) we cannot allocate the job J5



Job Sequencing with deadlines - Problem



From the above Gantt chart, the optimal solution is:

- ✓Jobs selected= $\{J_1,J_2,J_4\}$
- ✓Processing sequence of jobs=$\{J_2,J_1,J_4\}$
- ✓Profit obtained = 15+20+5=40
- ✓Jobs which are not processed= $\{J_3,J_5\}$

The optimal solution for Job sequencing with deadlines problem is obtained as shown below:

Selected jobs J	Assigned slots	Job considered	Action	Profit
Ø	None	J_1	Assign J_1 to slot [1, 2]	0
{ J_1}	[1,2]	J_2	Assign J_2 to slot [0,1]	20
{J_1,J_2}	[0,1],[1,2]	J_3	Cannot assign, reject job J_3 ***As [0,1] slot is not free***	35
{J_1,J_2}	[0,1],[1,2]	J_4	Assign J_4 to slot [2,3]	35
{J_1,J_2,J_4}	[0,1],[1,2],[2,3]	J_5	Cannot assign, reject job J_5 ***As [0,1],[1,2],[2,3] slots are not free***	40

The optimal solution is:

Jobs selected= {J_1,J_2,J_4}

Processing sequence of jobs={J_2,J_1,J_4}

Profit obtained =40

Job Sequencing with deadlines - Algorithm

Algorithm Greedy Job (d,p,J,n)

//**Input:** n is the number of jobs and d[1:n] is the deadlines of the jobs
//according to their descending order of profits and p[1:n] be the profits
//**Output:** J is a set of jobs that can be completed by their dead lines

```
{
      J: = {1};
      Profit=p[1]
      for i: = 2 to n do
      {
            if (all jobs in J U {i} can be completed by their dead lines)   then
                        J: = J U { i};
                        Profit:=Profit+p[i]
       }
}
```

Analysis- Job Sequencing with deadlines

- If we do not consider the time considered for sorting the inputs then the time complexity will be O(n).
- If we consider the time considered for sorting the inputs then complexity will be O(n log n).
  - Complexity of sorting using any better sorting algorithm O(n log n)
  - Complexity of selecting jobs O(n)
  - Therefore the time complexity of Job Sequencing with deadlines problem will be max(O(nlogn),O(n)) i.e., O(n logn)

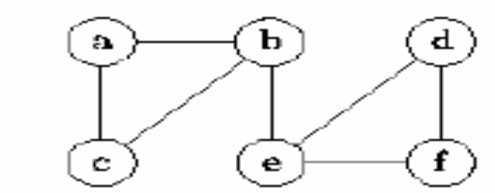
Assignment

Find the optimal solution for the job sequencing with deadlines problem using greedy method:

- Let n=5, profits [10,3,33,11,40] and deadlines [3,1,1,2,2] respectively [Jan 2018]
- Let n=5, profits [60,100,20,40,20] and deadlines [2,1,3,2,1] respectively

What is a Graph?

- Any Graph, G = (V, E) is a collection of set of edges E and Vertices V, it might be a simple, connected, directed/undirected graph that is edge weighted/not edge-weighted.



(a) Graph G

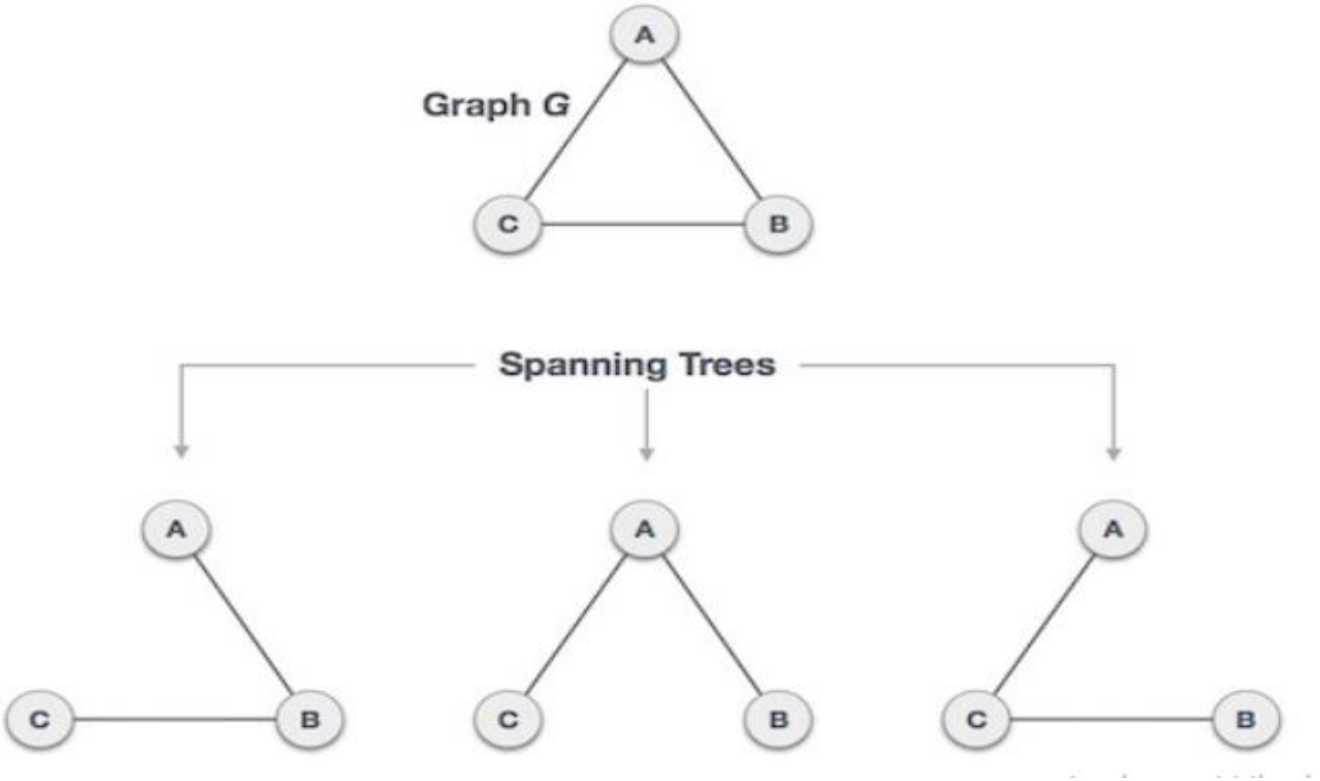
What is a Spanning tree??

- A ***spanning tree*** of any undirected connected graph is a connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph and the number of edges is equal to one less than the number of vertices of the graph.
- Thus a minimum spanning tree T for G is a graph, T = (V', E') with the following properties:
  - ❖V'= V
  - ❖E'=|V|-1
  - ❖T is connected
  - ❖T is acyclic.

Spanning Tree

- A complete undirected graph can have maximum n^{n-2} number of spanning trees, where n is the number of nodes.
- A Non-complete undirected graph can have maximum ${}^{|E|}C_{|V|-1}$-(no of cycles) in the graph

Example of Spanning Tree



Number of Spanning Trees = $n^{n-2}=3^{3-2}=3^1=3$ spanning trees

Minimum Cost Spanning Tree

- For an edge-weighted , connected, undirected graph, G, the total cost of G is the sum of the weights on all its edges.
- A minimum-cost spanning tree for G is a spanning tree of the smallest weight, where the ***weight*** of a tree is defined as the sum of the weights on all its edges.
- For the given non-complete graph with 4 vertices and 4 edges we have 3 spanning trees i.e., ${}^4C_3$ -1= 4-1=3.,

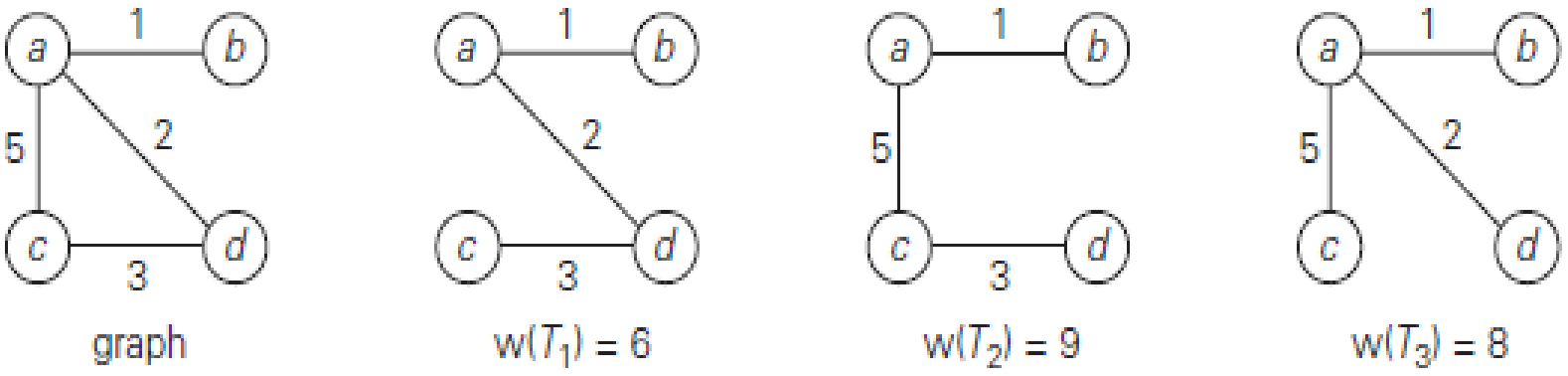


FIGURE : Graph and its spanning trees, with T_1 being the minimum spanning tree.

What is a Spanning tree??

- A ***spanning tree*** of any undirected connected graph is a connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph and the number of edges is equal to one less than the number of vertices of the graph.
- Thus a minimum spanning tree T for G is a graph, T = (V', E') with the following properties:
  - ❖V'= V
  - ❖E'=|V|-1
  - ❖T is connected
  - ❖T is acyclic.

MCST Algorithms

There are two methods to find Minimum Cost Spanning Tree

- Kruskal's Algorithm
- Prim's Algorithm

Kruskal's Algorithm

- Kruskal‟s Algorithm is another greedy algorithm for the minimum spanning tree problem which is an optimal solution.

- It is named ***Kruskal's algorithm*** after Joseph Kruskal, who discovered this algorithm when he was a second-year graduate student.

- Kruskal‟s algorithm looks at a minimum spanning tree of a weighted connected graph $G = (V, E)$ as an acyclic subgraph with $|V| - 1$ edges for which the sum of the edge weights is the smallest.

- Consequently, the algorithm constructs a minimum spanning tree as an expanding sequence of subgraphs that are always acyclic but are not necessarily connected on the intermediate stages of the algorithm.

Kruskal's Algorithm

Below are the steps for finding MST using Kruskal’s algorithm

Step 1: List all the edges along with its weight

Step 2: Sort all the edges in ascending order of their weight.

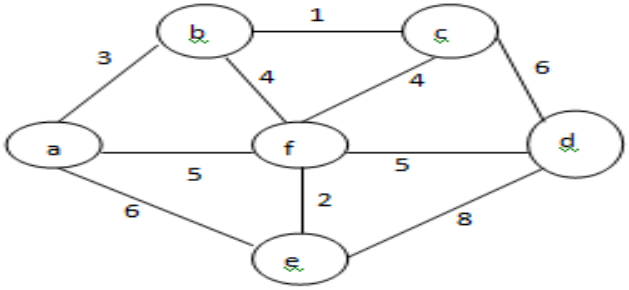
Step 3: Starting with the empty subgraph, Pick the edge from the sorted list. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

Step 4: Repeat step 3 until the requires number of edges are selected for the spanning tree(i.e., |V|-1)

Kruskal's Algorithm

Problem:

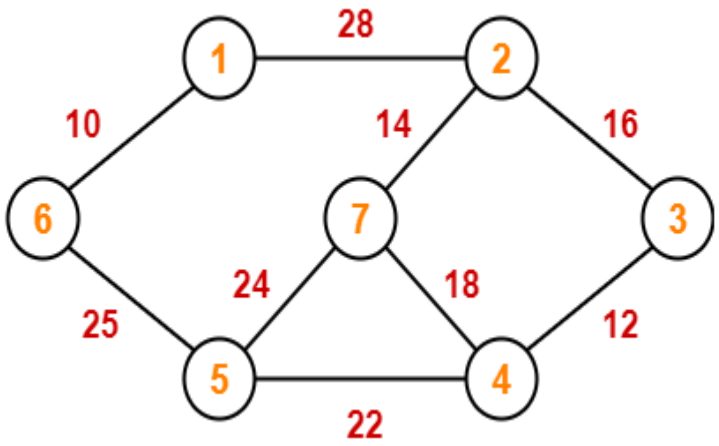
Find minimum spanning tree of the below graph using kruskal's algorithm

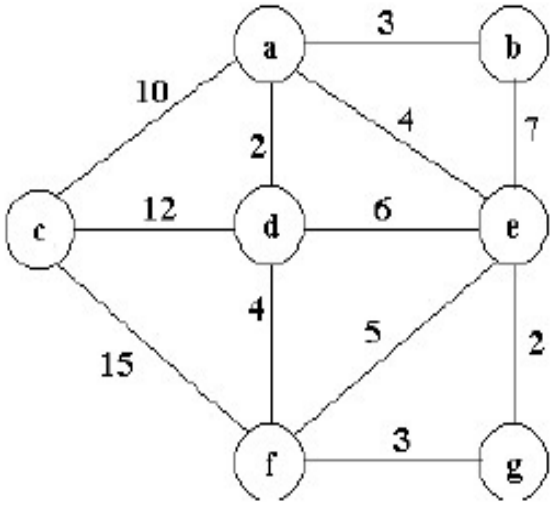


Solution:

Assignment

Apply kruskal's algorithm and Find minimum spanning tree for the below graph





THANK YOU….

MINIMUM COST SPANNNING TREES

- Graph
- Spanning tree
- Minimum Cost Spanning Trees(MCST)
- MCST Algorithms
- Applications

Kruskal's Algorithm

Algorithm Kruskal(G)

//Purpose: Kruskal's Algorithm to construct minimum spanning tree of graph G

//Input: A weighted undirected connected graph G = <V, E>

//Output: E_T --- the set of edges composing MST of G

{

Sort E in ascending order of the edge weight i.e., $w(e_1) \leq w(e_2) \ldots \leq w(e_{|E|})$

$E_T = \varnothing$; ecounter = 0 **//initialize the set of tree edges and its size**

k = 0 **//initialize the number of processed edges**

while ecounter < |V| - 1 **do**

k = k+1

if $E_T \cup \{e_k\}$ is acyclic **then**

$E_T = E_T \cup \{e_k\}$;

ecounter = ecounter + 1

return E_T

}

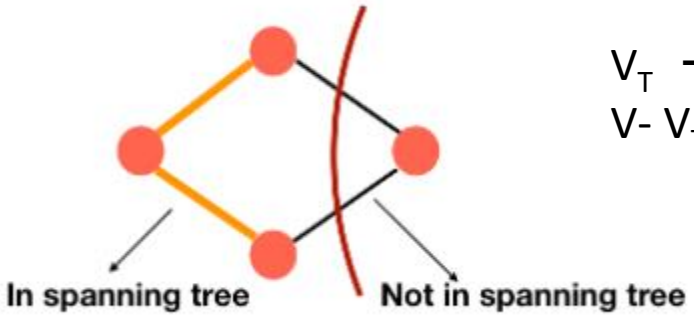
Analysis - Kruskal's Algorithm

With an efficient sorting algorithm, the time efficiency of Kruskal's algorithm will be in O(|E| log |E|).

- Complexity of sorting edges E, using any better sorting algorithm O(|E| log |E|)
- Complexity of selecting edges to form MST O(|V|)
- Therefore the time complexity of Kruskal's Algorithm will be max(O(|E|log|E|),O(|V|)) i.e., O(|E| log |E|)(Assuming |V|≈|E|)

Prim's Algorithm

- We divide the vertices into two parts, one contains the vertices which are in the growing spanning tree and the other part has the rest of the vertices.

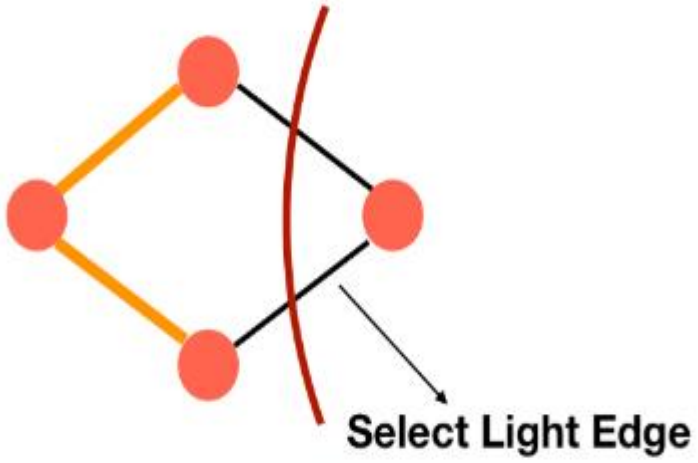


V_T → Tree Vertices

$V - V_T$ → Remaining /Fringe Vertices

Prim's Algorithm

- ✓ We select the least edge each time i.e., **e* = (u*,v*)** among all the edges (u, v) such that u is in V_T and v is in $V - V_T$
- ✓ Add the vertex at the end of the least edge i.e., **v*** into the growing spanning tree
- ✓ Add the edge **e*** into the growing spanning tree.



Prim's Algorithm

Prim's Algorithm.

Step 1: Select an arbitrary vertex from the graph & add to the Spanning tree.

Step 2: Select (the) an edge (u, v) which has least weight such that $u \in V_T$ & $v \in V - V_T$

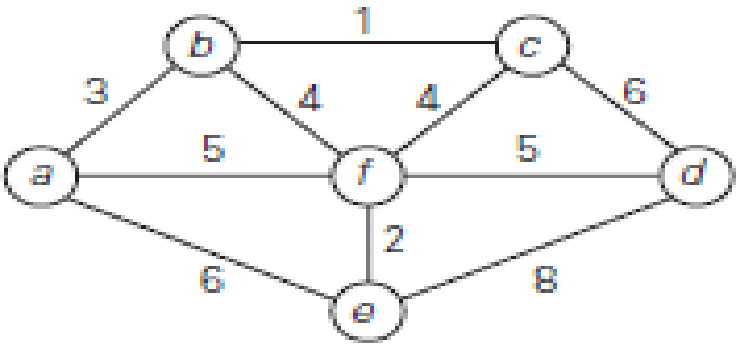
Step 3: Repeat the step 2 until all the vertices are included in the Spanning tree & required no of edges are selected.

Prim's Algorithm

- Prim's algorithm constructs a minimum spanning tree through a sequence of expanding subtrees.
- The initial subtree in such a sequence consists of a single vertex selected arbitrarily from the set V of the graph's vertices.
- On each iteration, the algorithm expands the current tree in the greedy manner by simply attaching to it the nearest vertex not in that tree. (By the nearest vertex, we mean a vertex not in the tree connected to a vertex in the tree by an edge of the smallest weight. Ties can be broken arbitrarily.)
- The algorithm stops after all the graph's vertices have been included in the tree being constructed.
- Since the algorithm expands a tree by exactly one vertex on each of its iterations, the total number of such iterations is $n - 1$, where n is the number of vertices in the graph.
- The tree generated by the algorithm is obtained as the set of edges used for the spanning tree

Prim's Algorithm - Example

Find MST by applying prim's algorithm for the below graph



Solution

Prim's Algorithm

ALGORITHM Prim (G)

//Prim's algorithm for constructing a MST

//**Input:** A weighted connected graph G = { V, E }

//**Output:** E_T the set of edges composing a MST of G

{

$V_T \leftarrow \{v_0\}$ **//the set of tree vertices can be initialized with any vertex**

$E_T \leftarrow \varnothing$ **//the edge set of MST is initially empty**

for i ← 1 to |V| - 1 do

find a minimum-weight edge **e* = (u* , v*)** among all the edges (u, v) such that u is in V_T and v is in $V - V_T$

$V_T \leftarrow V_T \cup \{v^*\}$ **// add v* to V_T**

$E_T \leftarrow E_T \cup \{e^*\}$ **// add (u* , v*) to E_T**

return E_T **// all vertices included the currently constructed tree**

}

Analysis - Prim's Algorithm

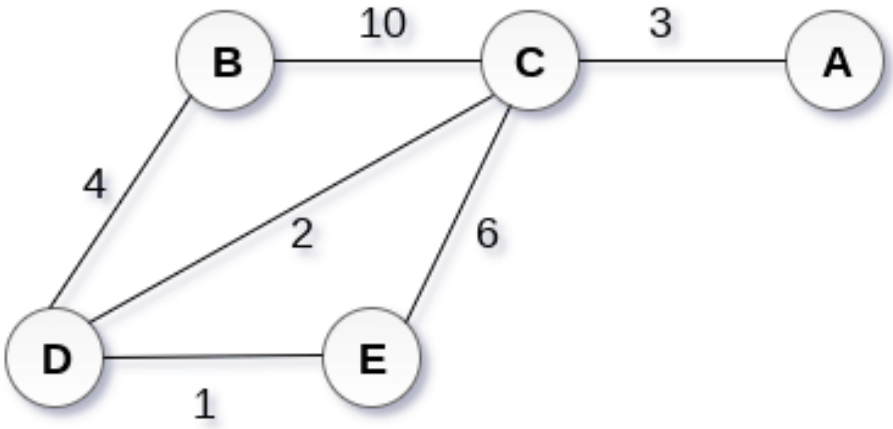
The efficiency of prim's algorithm depends on the data structures used to represent

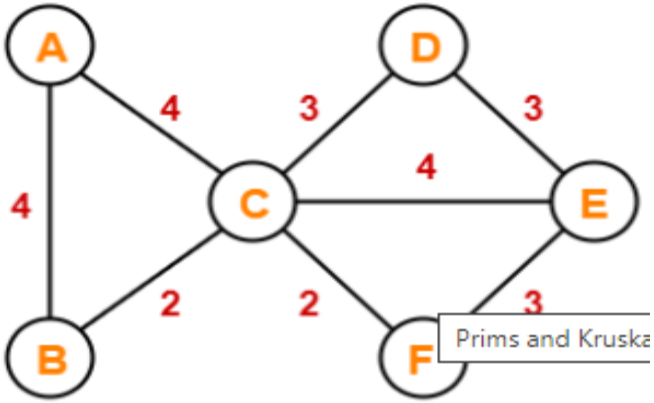
- The graph and
- the priority queue used for the set $V - V_T$ whose vertex priorities are the distances to the nearest tree vertices.

Data structure for Graph	Data structure for Priority queue	Time Efficiency
Weight Matrix	Unordered Array	$O(|V|^2)$
Weight Matrix	Min-heap	$O(|V|^2)$
Adjacency List	Min-heap	$O(|E| \log |V|)$

Assignment

Apply prim's algorithm and Find minimum spanning tree for the below graphs





Applications

Minimum-cost spanning trees have many applications.

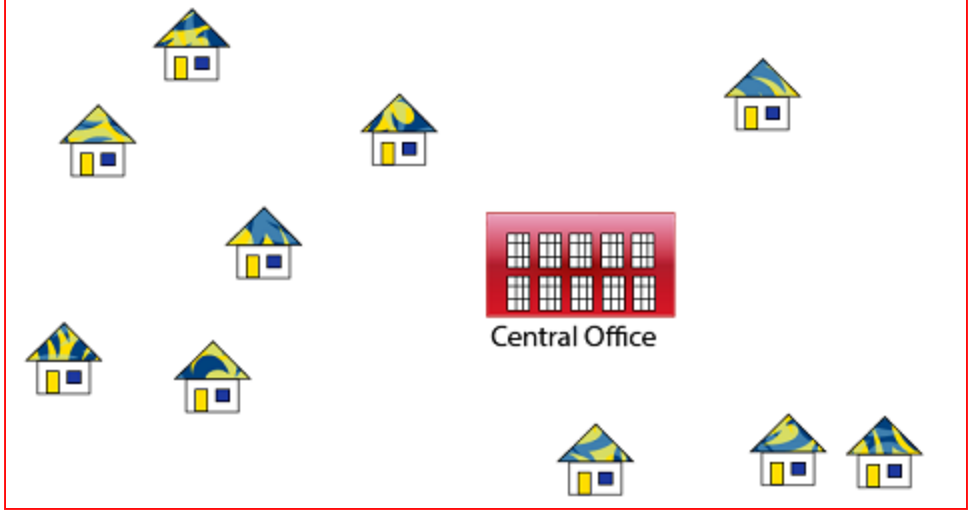
Some are:

- Building cable networks that join n locations with minimum cost.
- Building a road network that joins n cities with minimum cost.
- Obtaining an independent set of circuit equations for an electrical network.
- In pattern recognition minimal spanning trees can be used to find noisy pixels.

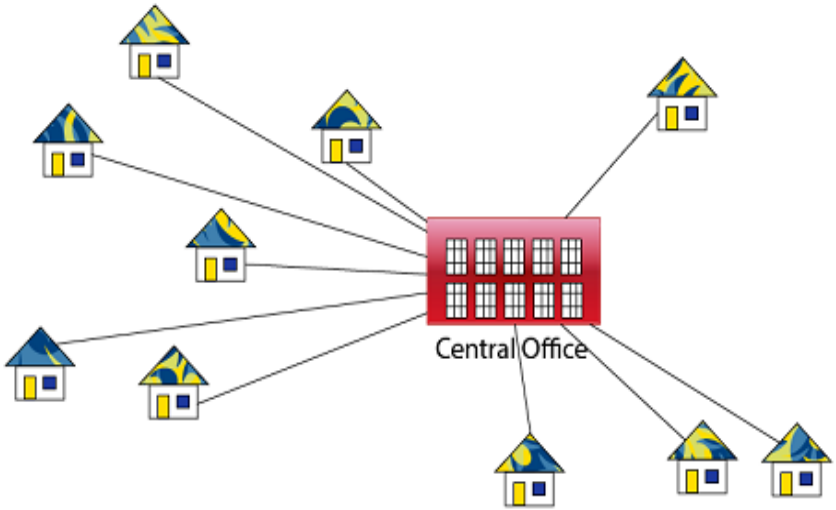
Applications

Consider, city network as a huge graph and now plans to deploy telephone lines in such a way that in minimum lines(wires) we can connect to all city nodes. This is where the spanning tree comes into picture.

For Example, Problem laying Telephone Wire.

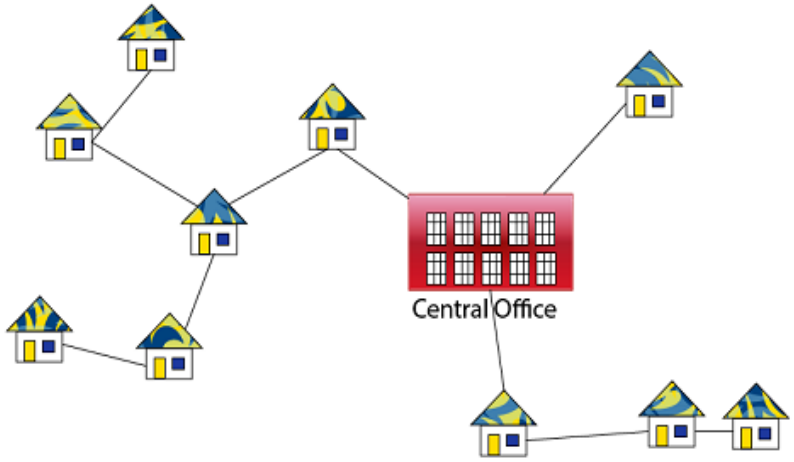


Wiring : Naive Approach



Expensive!

Wiring : Better Approach



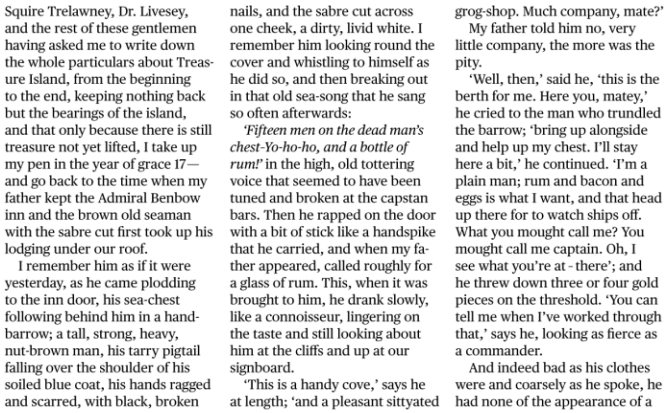
Minimize the total length of wire connecting the customers

Optimal Tree Problem

- ✓ Problem is that, we have to encode a text that comprises symbols from some n-symbol alphabet by assigning to each of the text's symbols some sequence of bits called the **codeword**

- ✓ **There are two ways of encoding:**
  - ✓ Fixed-length encoding and
  - ✓ Variable-length encoding**.**

Optimal Tree Problem



Squire Trelawney, Dr. Livesey, and the rest of these gentlemen having asked me to write down the whole particulars about Treasure Island, from the beginning to the end, keeping nothing back but the bearings of the island, and that only because there is still treasure not yet lifted, I take up my pen in the year of grace 17— and go back to the time when my father kept the Admiral Benbow inn and the brown old seaman with the sabre cut first took up his lodging under our roof.

I remember him as if it were yesterday, as he came plodding to the inn door, his sea-chest following behind him in a hand-barrow; a tall, strong, heavy, nut-brown man, his tarry pigtail falling over the shoulder of his soiled blue coat, his hands ragged and scarred, with black, broken nails, and the sabre cut across one cheek, a dirty, livid white. I remember him looking round the cover and whistling to himself as he did so, and then breaking out in that old sea-song that he sang so often afterwards:

'Fifteen men on the dead man's chest-Yo-ho-ho, and a bottle of rum!' in the high, old tottering voice that seemed to have been tuned and broken at the capstan bars. Then he rapped on the door with a bit of stick like a handspike that he carried, and when my father appeared, called roughly for a glass of rum. This, when it was brought to him, he drank slowly, like a connoisseur, lingering on the taste and still looking about him at the cliffs and up at our signboard.

'This is a handy cove,' says he at length; 'and a pleasant sittyated grog-shop. Much company, mate?'

My father told him no, very little company, the more was the pity.

'Well, then,' said he, 'this is the berth for me. Here you, matey,' he cried to the man who trundled the barrow; 'bring up alongside and help up my chest. I'll stay here a bit,' he continued. 'I'm a plain man; rum and bacon and eggs is what I want, and that head up there for to watch ships off. What you mought call me? You mought call me captain. Oh, I see what you're at - there'; and he threw down three or four gold pieces on the threshold. 'You can tell me when I've worked through that,' says he, looking as fierce as a commander.

And indeed bad as his clothes were and coarsely as he spoke, he had none of the appearance of a

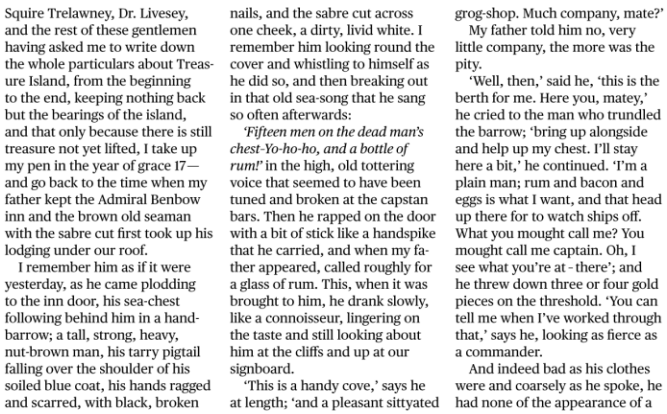
Original file

Encoding

Compressed file

- **A technique to compress data effectively**
  - **Usually between 20%-90% compression**
- **Lossless compression**
  - **No information is lost**
  - **When decompress, you get the original file**

Optimal Tree Problem



Squire Trelawney, Dr. Livesey, and the rest of these gentlemen having asked me to write down the whole particulars about Treasure Island, from the beginning to the end, keeping nothing back but the bearings of the island, and that only because there is still treasure not yet lifted, I take up my pen in the year of grace 17— and go back to the time when my father kept the Admiral Benbow inn and the brown old seaman with the sabre cut first took up his lodging under our roof.

I remember him as if it were yesterday, as he came plodding to the inn door, his sea-chest following behind him in a hand-barrow; a tall, strong, heavy, nut-brown man, his tarry pigtail falling over the shoulder of his soiled blue coat, his hands ragged and scarred, with black, broken nails, and the sabre cut across one cheek, a dirty, livid white. I remember him looking round the cover and whistling to himself as he did so, and then breaking out in that old sea-song that he sang so often afterwards:

'Fifteen men on the dead man's chest-Yo-ho-ho, and a bottle of rum!' in the high, old tottering voice that seemed to have been tuned and broken at the capstan bars. Then he rapped on the door with a bit of stick like a handspike that he carried, and when my father appeared, called roughly for a glass of rum. This, when it was brought to him, he drank slowly, like a connoisseur, lingering on the taste and still looking about him at the cliffs and up at our signboard.

'This is a handy cove,' says he at length; 'and a pleasant sittyated grog-shop. Much company, mate?'

My father told him no, very little company, the more was the pity.

'Well, then,' said he, 'this is the berth for me. Here you, matey,' he cried to the man who trundled the barrow; 'bring up alongside and help up my chest. I'll stay here a bit,' he continued. 'I'm a plain man; rum and bacon and eggs is what I want, and that head up there for to watch ships off. What you mought call me? You mought call me captain. Oh, I see what you're at - there'; and he threw down three or four gold pieces on the threshold. 'You can tell me when I've worked through that,' says he, looking as fierce as a commander.

And indeed bad as his clothes were and coarsely as he spoke, he had none of the appearance of a

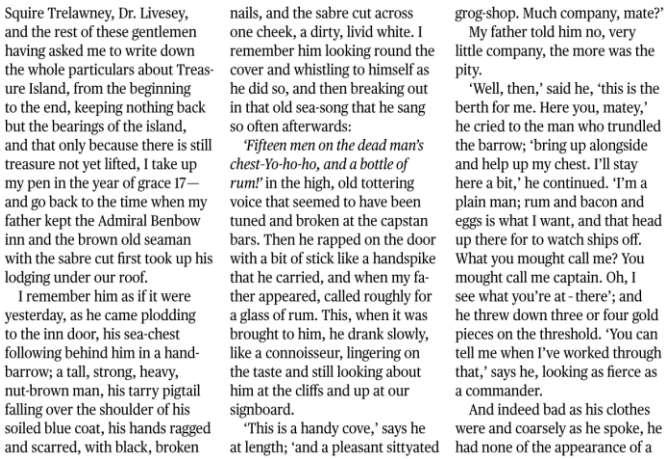
Original file

Huffman coding

Compressed file

- **Saving space**
  - **Store compressed files instead of original files**
- **Transmitting files or data**
  - **Send compressed data to save transmission time and power**
- **Encryption and decryption**
  - **Cannot read the compressed file without knowing the "key"**

Optimal Tree Problem



Squire Trelawney, Dr. Livesey, and the rest of these gentlemen having asked me to write down the whole particulars about Treasure Island, from the beginning to the end, keeping nothing back but the bearings of the island, and that only because there is still treasure not yet lifted, I take up my pen in the year of grace 17— and go back to the time when my father kept the Admiral Benbow inn and the brown old seaman with the sabre cut first took up his lodging under our roof.

I remember him as if it were yesterday, as he came plodding to the inn door, his sea-chest following behind him in a hand-barrow; a tall, strong, heavy, nut-brown man, his tarry pigtail falling over the shoulder of his soiled blue coat, his hands ragged and scarred, with black, broken nails, and the sabre cut across one cheek, a dirty, livid white. I remember him looking round the cover and whistling to himself as he did so, and then breaking out in that old sea-song that he sang so often afterwards:

'Fifteen men on the dead man's chest-Yo-ho-ho, and a bottle of rum!' in the high, old tottering voice that seemed to have been tuned and broken at the capstan bars. Then he rapped on the door with a bit of stick like a handspike that he carried, and when my father appeared, called roughly for a glass of rum. This, when it was brought to him, he drank slowly, like a connoisseur, lingering on the taste and still looking about him at the cliffs and up at our signboard.

'This is a handy cove,' says he at length; 'and a pleasant sittyated grog-shop. Much company, mate?'

My father told him no, very little company, the more was the pity.

'Well, then,' said he, 'this is the berth for me. Here you, matey,' he cried to the man who trundled the barrow; 'bring up alongside and help up my chest. I'll stay here a bit,' he continued. 'I'm a plain man; rum and bacon and eggs is what I want, and that head up there for to watch ships off. What you mought call me? You mought call me captain. Oh, I see what you're at - there'; and he threw down three or four gold pieces on the threshold. 'You can tell me when I've worked through that,' says he, looking as fierce as a commander.

And indeed bad as his clothes were and coarsely as he spoke, he had none of the appearance of a

- Assume in this file only 6 characters appear
  - E, A, C, T, K, N
- The frequencies are:

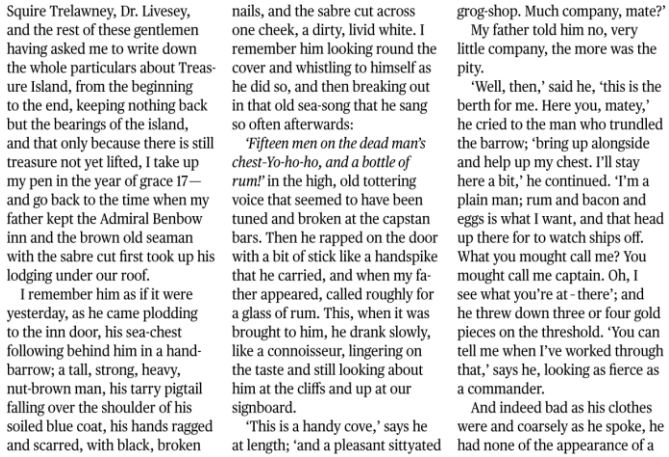
Character	Frequency
E	10,000
A	4,000
C	300
T	200
K	100
N	100

- Option I (No Compression)
  - Each character = 1 Byte (8 bits)
  - Total file size = 14,700 * 8 = 117,600 bits

- Option 2 (Fixed length encoding- compression)
  - We have 6 characters, so we need 3 bits to encode them
  - Total file size = 14,700 * 3 = 44,100 bits

Character	Fixed Encoding
E	000
A	001
C	010
T	100
K	110
N	111

Optimal Tree Problem



Squire Trelawney, Dr. Livesey, and the rest of these gentlemen having asked me to write down the whole particulars about Treasure Island, from the beginning to the end, keeping nothing back but the bearings of the island, and that only because there is still treasure not yet lifted, I take up my pen in the year of grace 17— and go back to the time when my father kept the Admiral Benbow inn and the brown old seaman with the sabre cut first took up his lodging under our roof.

I remember him as if it were yesterday, as he came plodding to the inn door, his sea-chest following behind him in a hand-barrow; a tall, strong, heavy, nut-brown man, his tarry pigtail falling over the shoulder of his soiled blue coat, his hands ragged and scarred, with black, broken nails, and the sabre cut across one cheek, a dirty, livid white. I remember him looking round the cover and whistling to himself as he did so, and then breaking out in that old sea-song that he sang so often afterwards:

'Fifteen men on the dead man's chest-Yo-ho-ho, and a bottle of rum!' in the high, old tottering voice that seemed to have been tuned and broken at the capstan bars. Then he rapped on the door with a bit of stick like a handspike that he carried, and when my father appeared, called roughly for a glass of rum. This, when it was brought to him, he drank slowly, like a connoisseur, lingering on the taste and still looking about him at the cliffs and up at our signboard.

'This is a handy cove,' says he at length; 'and a pleasant sittyated grog-shop. Much company, mate?'

My father told him no, very little company, the more was the pity.

'Well, then,' said he, 'this is the berth for me. Here you, matey,' he cried to the man who trundled the barrow; 'bring up alongside and help up my chest. I'll stay here a bit,' he continued. 'I'm a plain man; rum and bacon and eggs is what I want, and that head up there for to watch ships off. What you mought call me? You mought call me captain. Oh, I see what you're at - there'; and he threw down three or four gold pieces on the threshold. 'You can tell me when I've worked through that,' says he, looking as fierce as a commander.

And indeed bad as his clothes were and coarsely as he spoke, he had none of the appearance of a

- Assume in this file only 6 characters appear
  - E, A, C, T, K, N
- The frequencies are:

Character	Frequency
E	10,000
A	4,000
C	300
T	200
K	100
N	100

- Option 3 (Variable length encoding - compression)
  - Variable-length compression
  - Assign shorter codes to more frequent characters and longer codes to less frequent characters
  - Total file size:

(10,000 x 1) + (4,000 x 2) + (300 x 3) + (200 x 4) + (100 x 5) + (100 x 5) = 20,700 bits

Char.	Variable Length Encoding
E	0
A	10
C	110
T	1110
K	11110
N	11111

Optimal Tree Problem

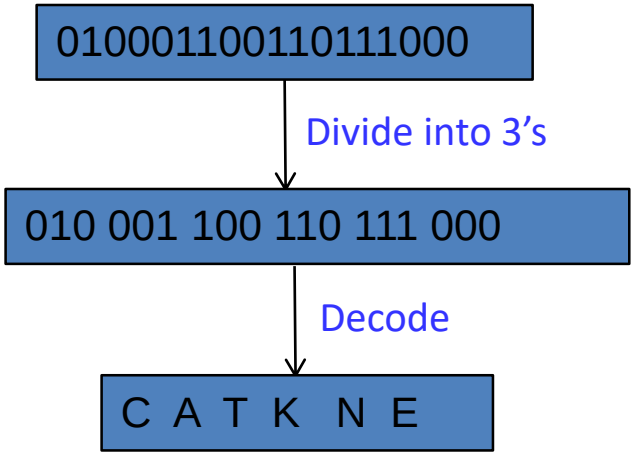
- **A variable-length coding for characters**
  - **More frequent characters ➔ shorter codes**
  - **Less frequent characters ➔ longer codes**

- **It is not like ASCII coding where all characters have the same coding length (8 bits)**

- **Two main questions**
  - **How to assign codes *(Encoding process)?***
  - **How to decode (from the compressed file, generate the original file) *(Decoding process)?***

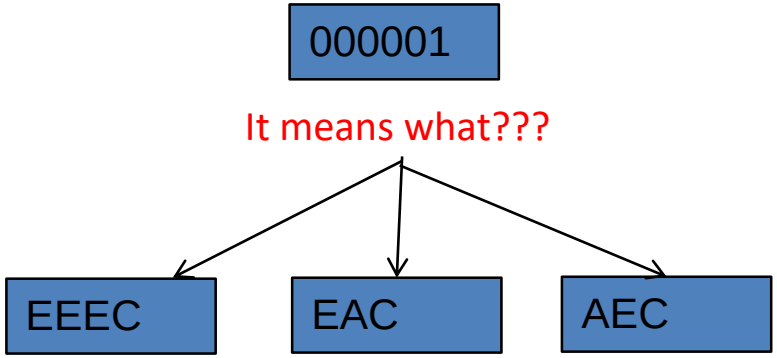
Decoding for fixed-length codes is much easier

Character	Fixed-length Encoding
E	000
A	001
C	010
T	100
K	110
N	111



Decoding for variable-length codes is not that easy…

Character	Variable-length Encoding
E	0
A	00
C	001
…	…
…	…
…	…



Optimal Tree Problem

- To avoid this complication, we can limit ourselves to the so-called ***prefix-free (or simply prefix) codes.***

- In a prefix code, no codeword is a prefix of a codeword of another symbol.

- Hence, with such an encoding, we can simply scan a bit string until we get the first group of bits that is a codeword for some symbol, replace these bits by this symbol, and repeat this operation until the bit string‟s end is reached.

- Example: a = 0, b = 101, c = 100
  - Decode 00100
    - Translates to “aac”

Optimal Tree Problem

- The solution to the problem to create a binary prefix code for some alphabet, is to associate the alphabet's symbols with leaves of a binary tree in which

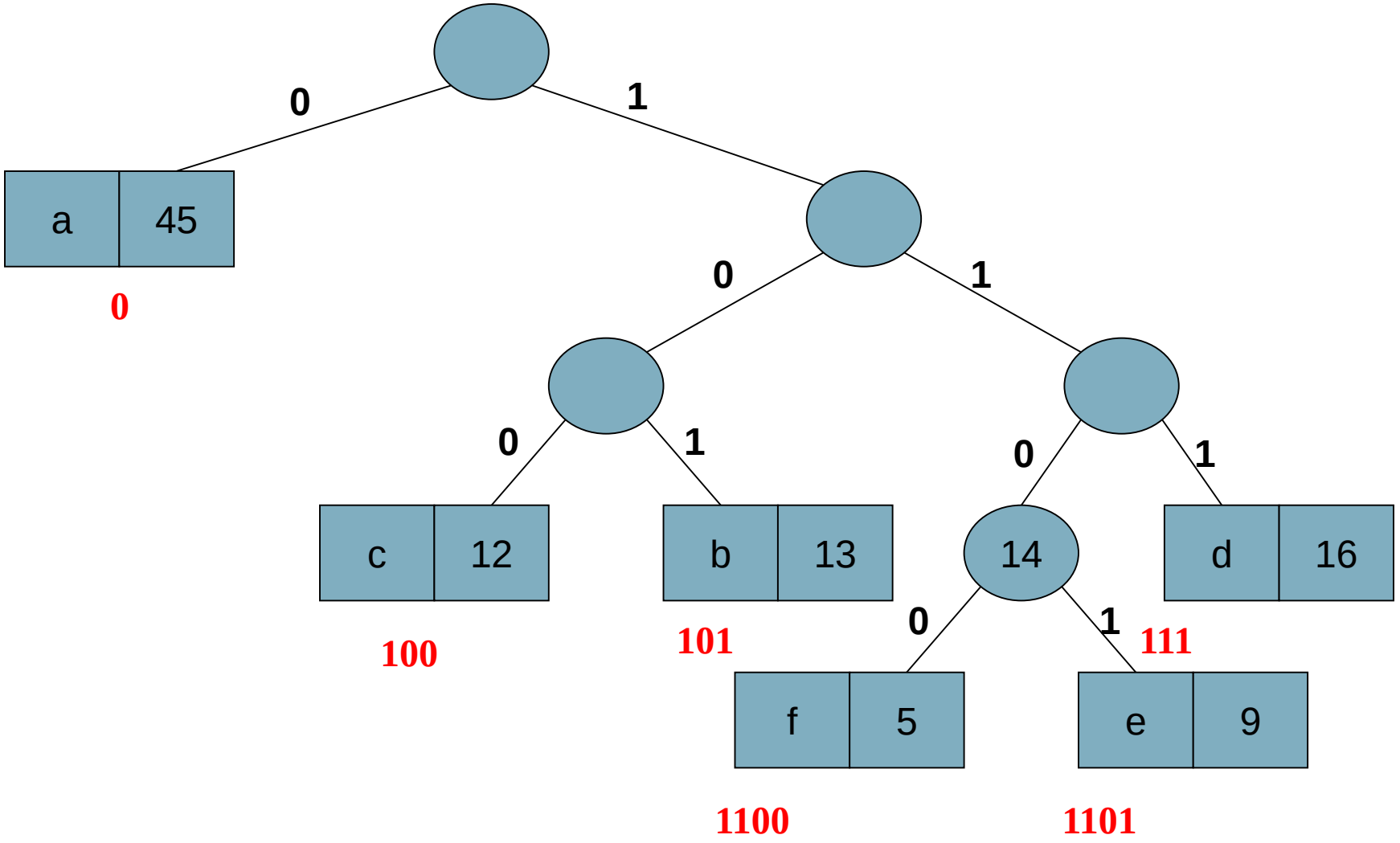
  - ✓all the left edges are labeled by 0
  - ✓all the right edges are labeled by 1.

Optimal Tree Problem

- The codeword of a symbol can then be obtained by recording the labels on the simple path from the root to the symbol's leaf.

- Among the many trees that can be constructed to solve this problem – The optimal one can be done by the greedy algorithm, invented by David Huffman known as Huffman algorithm.

Example



Huffman Algorithm

Step 1: Get Frequencies

- Scan the file to be compressed and count the occurrence of each character
- Sort the characters based on their frequency

Step 2: Build Tree & Assign Codes

- Build a Huffman tree (binary tree)
- Traverse the tree to assign codes

Step 3: Encode (Compress)

- Scan the file again and replace each character by its code

Step 4: Decode (Decompress)

- Huffman tree is the key to decompress the file

How to build Huffman tree ??

Step 1: Initialize ‘n’ one-node trees and label them with the symbols of the alphabet given. Record the frequency of each symbol in its tree’s root to indicate the tree’s weight.

Step 2: Repeat the following operation until a single tree is obtained.

- ✓ Find two trees with the smallest weight.
- ✓ Make them the left and right subtree of a new tree and record the sum of their weights in the root of the new tree as its weight.
- ✓ Rearrange the nodes again in ascending order of weights

- ❖ A tree constructed by the above algorithm is called as **Huffman tree.**
- ❖ The codeword defined using this tree is called as **Huffman code.**

Huffman Code- Example

Construct the Huffman Code for the following data:

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

Encode the text **DAD** and decode the text **100110110111101**

Solution

Assignment

Construct the Huffman Code for the following data: **[July 2018]**

Symbol	A	B	C	D	_
Probability	0.4	0.1	0.2	0.15	0.15

Encode the text **ABACABAD** and decode the text **100010111001010**

MODULE III

GREEDY METHOD

Heaps and Heap Sort

- Why to study Heap Sort?
- What is a Heap?
- Examples
- Types of Heaps
- Representation of Heaps
- Properties of Heaps
- Construction of Heaps

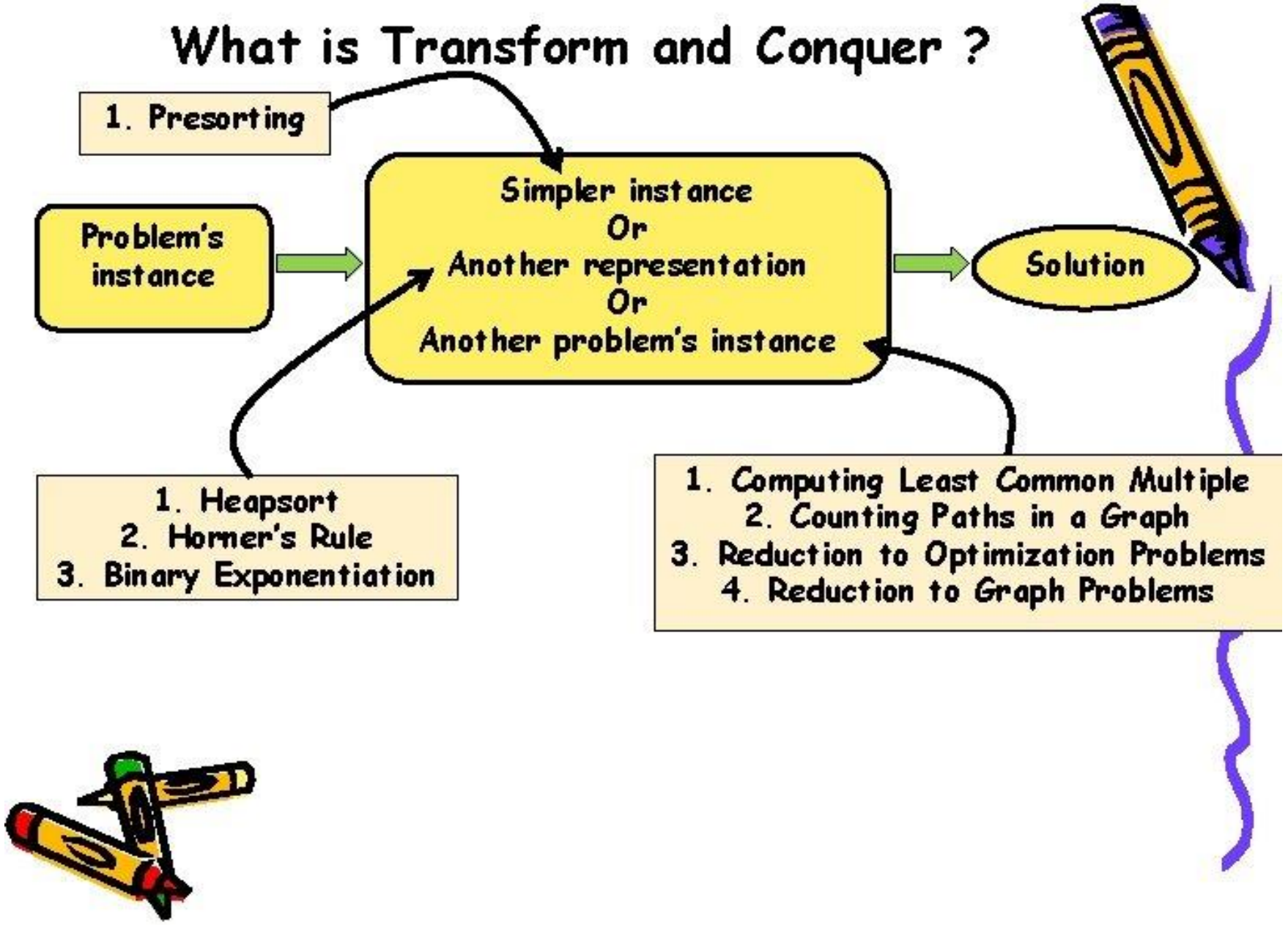
Why study HeapSort?

- It is a well-known, traditional sorting algorithm

- Heapsort is *always* O(n log n)
  - Quicksort is usually O(n log n) but in the worst case slows to $O(n^2)$
  - Quicksort is generally faster, but Heapsort is better in time-critical applications

- Heapsort is an sorting algorithm used for real-time (or time-bound where QuickSort doesn’t fit) embedded systems where less space is available (MergeSort doesn’t fit).!

What is Transform and Conquer ?



Why study HeapSort?

- It uses the algorithm design approach so called **Transform-and-conquer** approach which work as two-stage procedures.
  - First, in the transformation stage, the problem‟s instance is modified to be, for one reason or another, more amenable to solution.
  - Then, in the second or conquering stage, it is solved.

- Hence in heap sort, the instance is first transformed to heap and then sorted

What is a “Heap”?

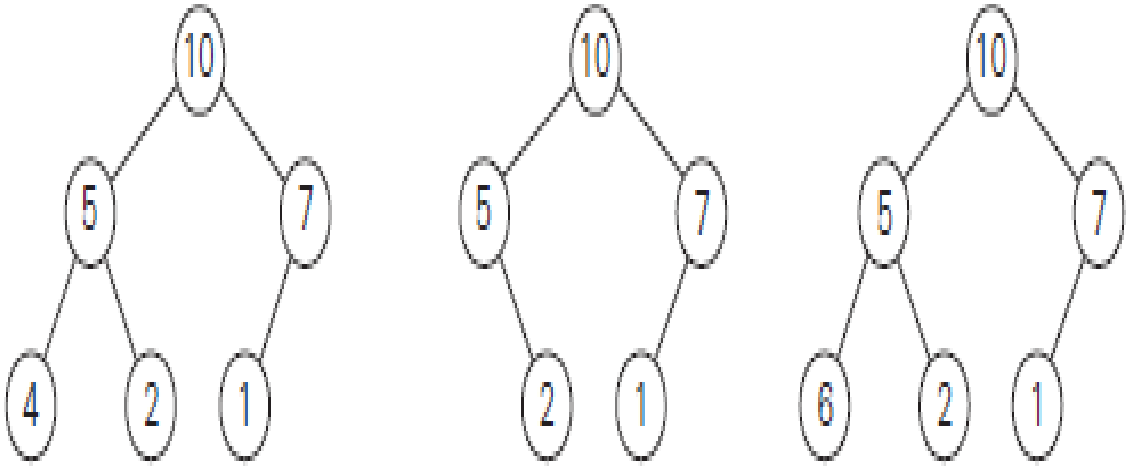
Definitions of heap:

1. A large area of memory from which the programmer can allocate blocks as needed dynamically, and deallocate them (or allow them to be garbage collected) when no longer needed

2. A heap is a **binary tree** with keys assigned to its nodes, one key per node, provided the following two conditions are met:
   - **The shape property**—the binary tree is essentially complete (or simply complete), i.e., all its levels are full except possibly the last level, where only some rightmost leaves may be missing.
   - **The parental dominance or heap property**—the key in each node is greater than(lesser than) or equal to the keys in its children.

- Heapsort uses the second definition

Heap Examples

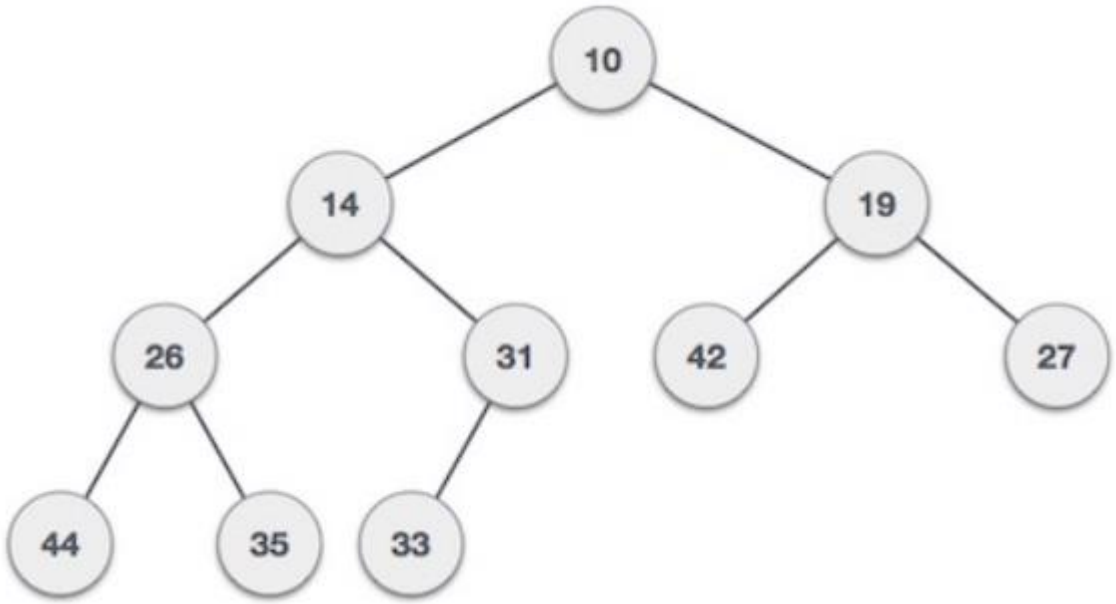


Property	Heap	Not a Heap	Not a Heap
Shape	Yes	No	Yes
Heap	Yes	Yes	No

Types of Heaps

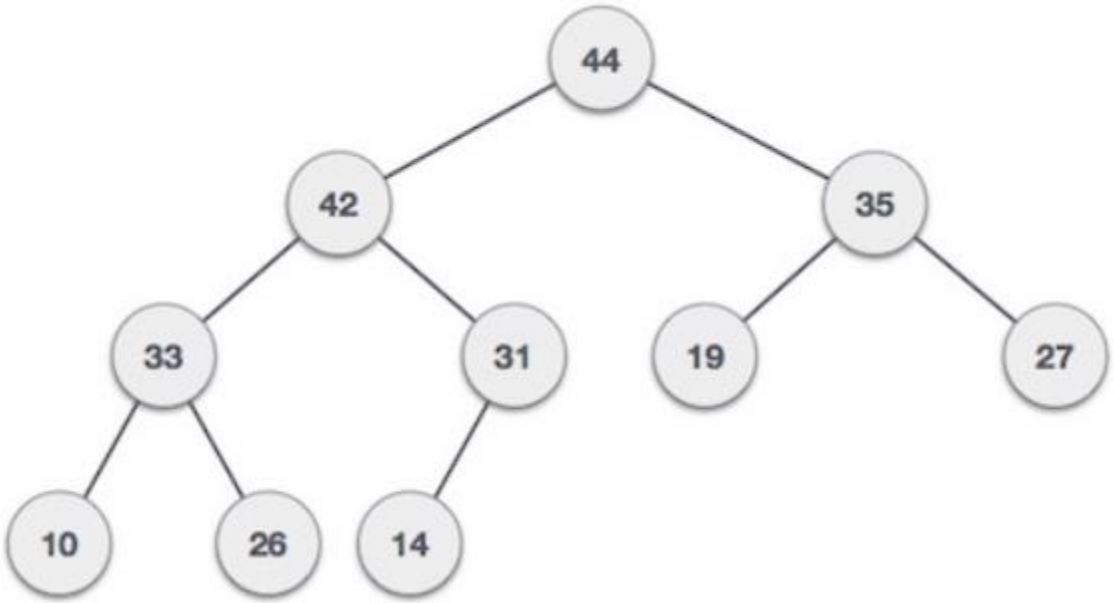
Two Types:

Min-Heap − Where the key in each node is lesser than or equal to the keys in its children.



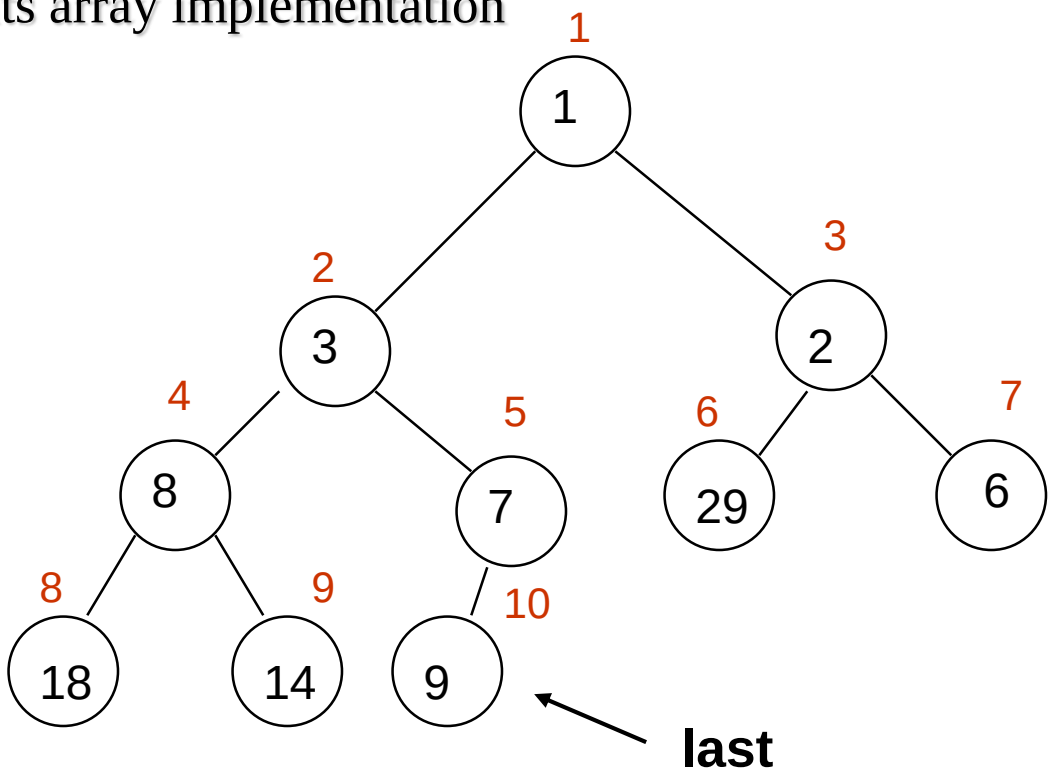
Types of Heaps

Max-Heap − Where the key in each node is greater than or equal to the keys in its children.



Representation of Heaps

Heap and its array implementation



root = 1

Parent(*i*) = $\lfloor i/2 \rfloor$

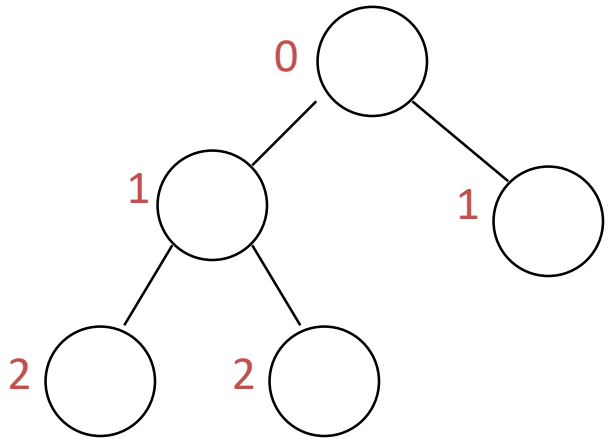
Left(*i*)=2*i*

Right(*i*)=2*i*+1

H	1	3	2	8	7	29	6	18	14	9
	1	2	3	4	5	6	7	8	9	10

Depth of tree nodes

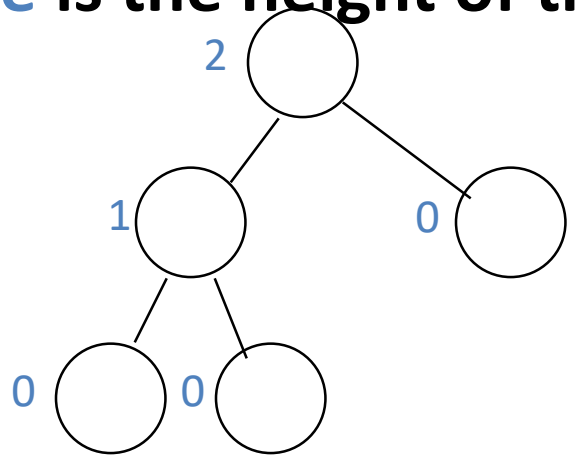
- **Depth of a node is:**
  - **If node is the root - 0**
  - **Otherwise - (depth of its parent + 1)**
- **Depth of a tree is maximum depth of its leaves.**



A tree of depth 2

Height of tree nodes

- **Height of a node is:**
  - **If node is a leaf - 0**
  - **Otherwise - (maximum height of its children +1)**
- **Height of a tree is the height of the root.**



A tree of height 2

Properties of Heaps

- There exists exactly one essentially complete binary tree with *n* nodes. Its height is equal to $\lfloor \log_2 n \rfloor$
- The root of a heap always contains its largest element.
- A node of a heap considered with all its descendants is also a heap.
- A heap can be implemented as an array of size ‘n’ where n is the number of nodes

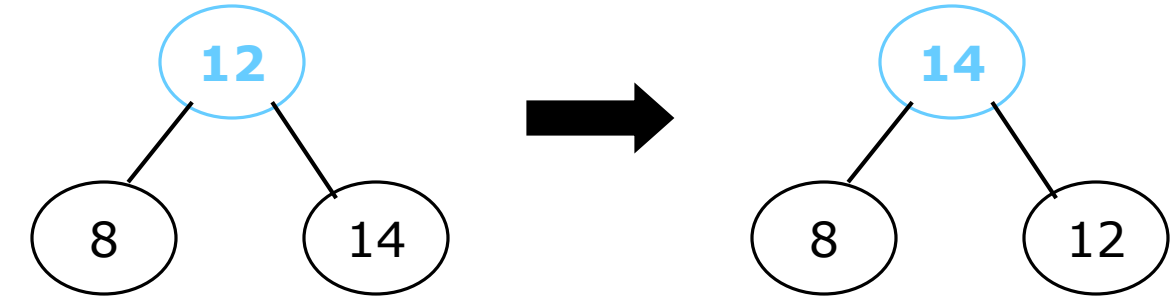
Constructions of Heap

There are two principal alternatives for constructing Heap.

- Top-down heap construction
- Bottom-up heap construction

siftUp

- Given a node that does not have the heap property, you can give it the heap property by exchanging its value with the value of the larger child



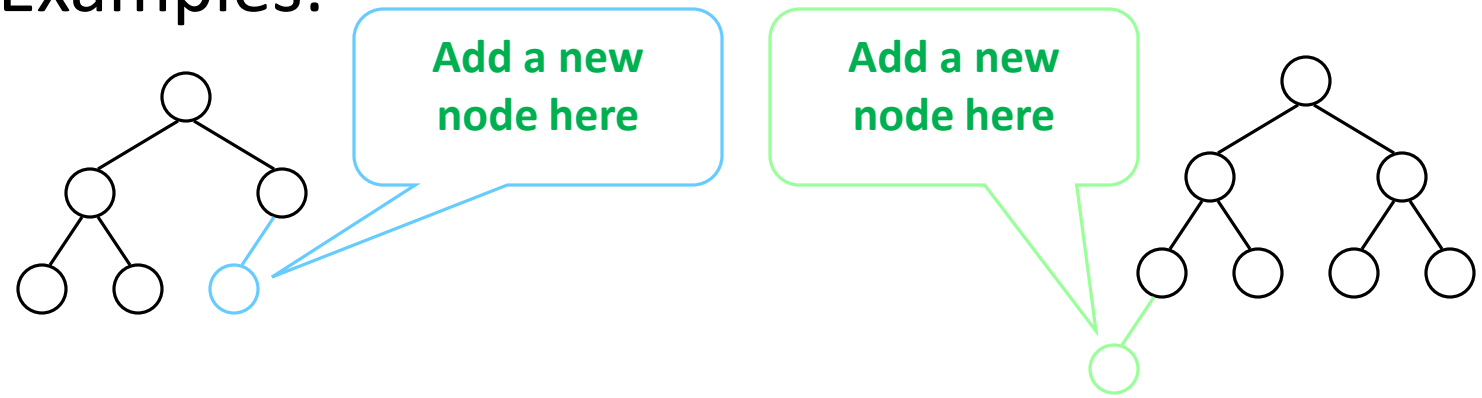
Blue node does not have heap property

Blue node has heap property

- This is sometimes called sifting up
- Notice that the child may have *lost* the heap property

Constructing a heap-Top down Approach

- A tree consisting of a single node is automatically a heap
- We construct a heap by adding nodes one at a time:
  - Add the node as last leaf node in the deepest level
  - If the deepest level is full, start a new level
- Examples:

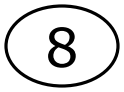


Constructing a heap

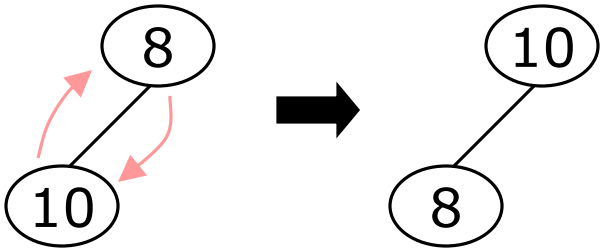
- Each time we add a node, we may destroy the heap property of its parent node
- To fix this, we sift up
- But each time we sift up, the value of the topmost node in the sift may increase, and this may destroy the heap property of *its* parent node
- We repeat the sifting up process, moving up in the tree, until either
  - We reach nodes whose values don't need to be swapped (because the parent is *still* larger than both children), or
  - We reach the root

Constructing a heap

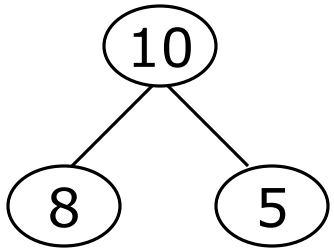
Construct the heap using top-down construction method: 8,10,5,12,14



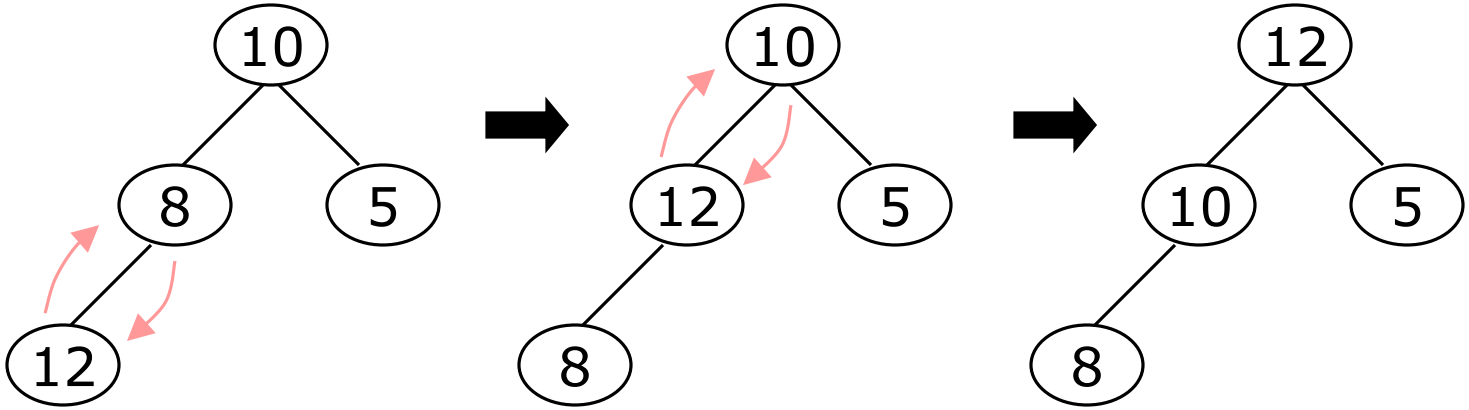
1



2

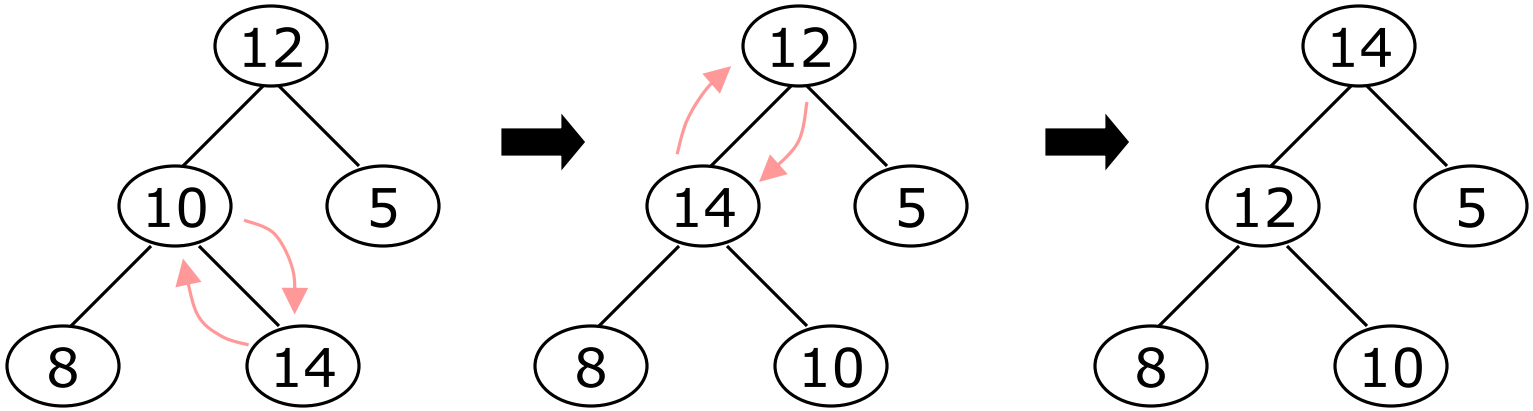


3



4

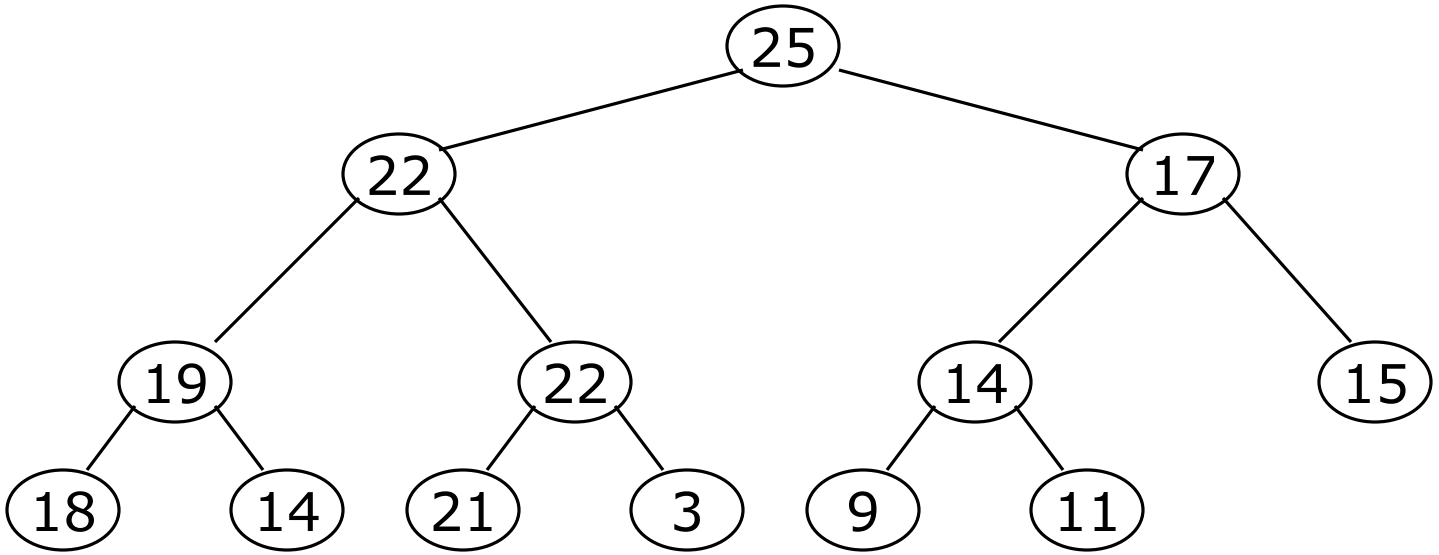
Constructing a heap



- The node containing 8 is not affected because its parent gets larger, not smaller
- The node containing 5 is not affected because its parent gets larger, not smaller
- The node containing 8 is still not affected because, although its parent got smaller, its parent is still greater than it was originally

A sample heap

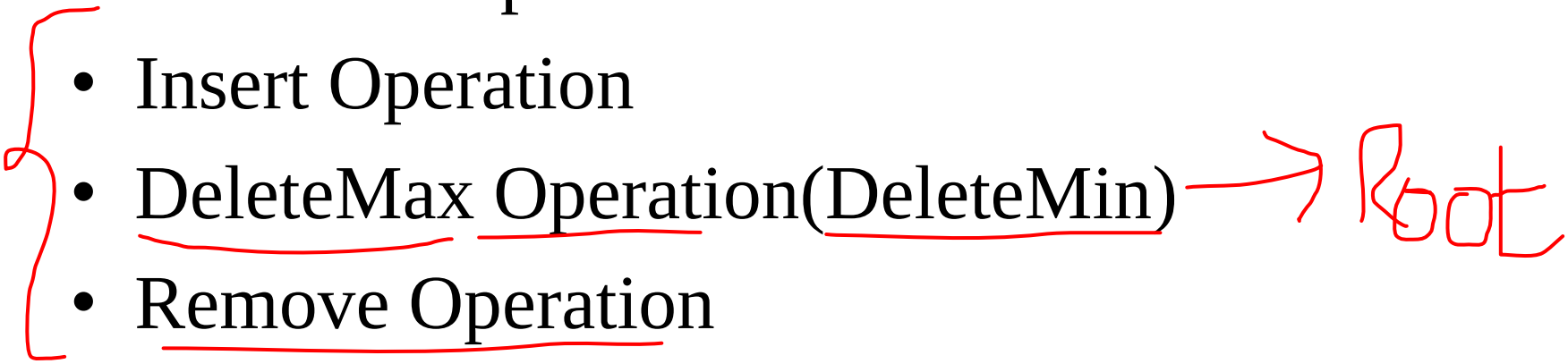
- Here's a sample binary tree after it has been heapified



- Notice that heapified does *not* mean sorted
- Heapifying does *not* change the shape of the binary tree; this binary tree is balanced and left-justified because it started out that way

Operations on Heap

- Build a Heap
- Insert Operation
- DeleteMax Operation(DeleteMin)
- Remove Operation



Insert Operation

The algorithm constructs a heap by inserting a new key **K** into a previously constructed heap

- First, attach a new node with key *K* in it after the last leaf of the existing heap.

- Then shift *K* up to its appropriate place in the new heap as follows.
  - ✓ Compare *K* with its parent's key: if the parent is greater than or equal to *K*, stop (the structure is a heap); otherwise, swap these two keys and compare *K* with its new parent.
  - ✓ This swapping continues until *K* is not greater than its last parent or it reaches the root.

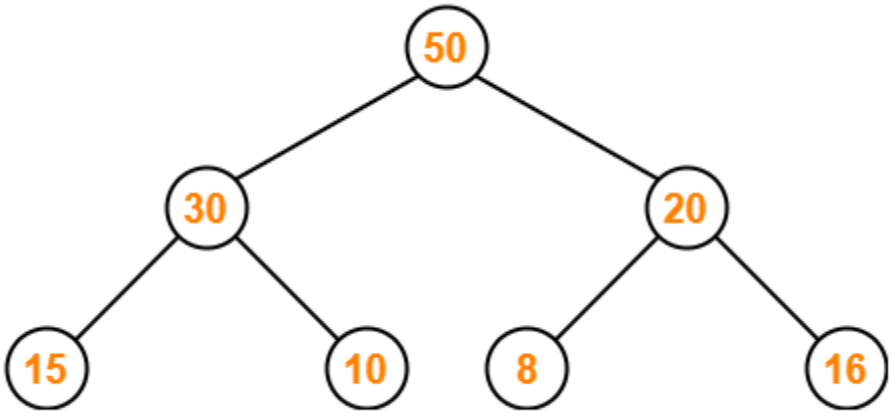
Insert Operation - Example

Consider the following max heap- 50, 30, 20, 15, 10, 8, 16. Insert a new node with value 60.

Solution:

Step-1:

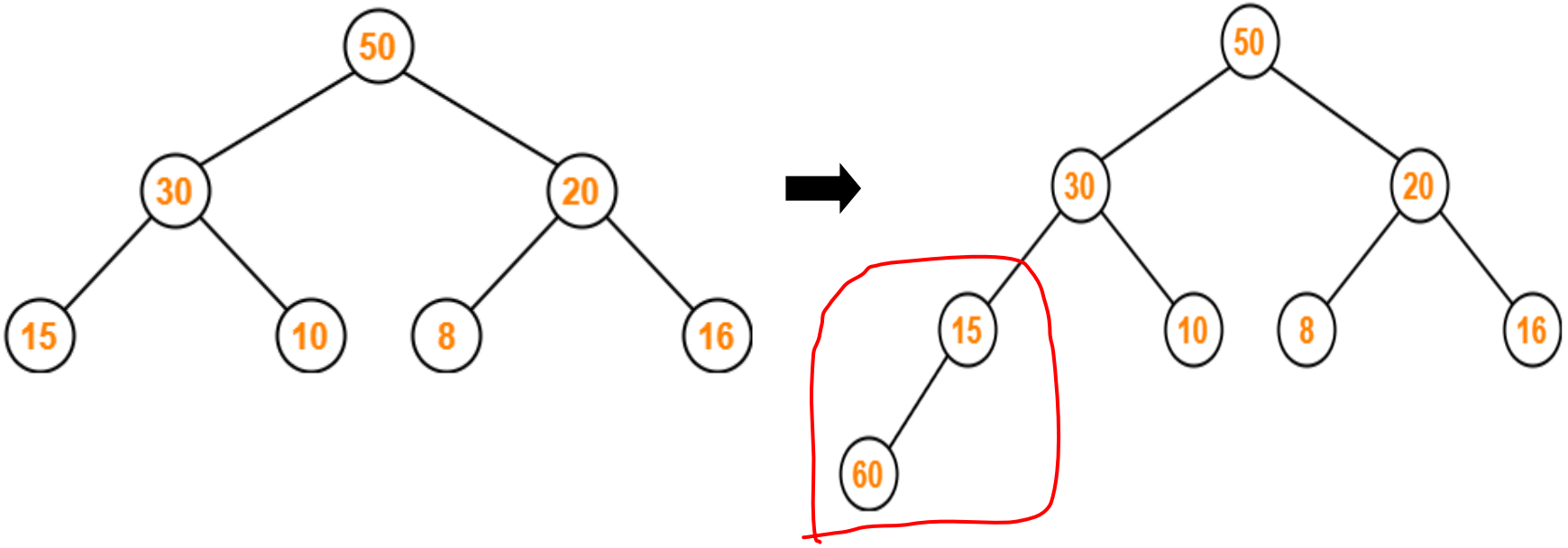
We convert the given array of elements into a heap tree



Insert Operation - Example

Step-2:

We insert the new element 60 as the last leaf node in the above tree. The resulting tree after adding the new node is-

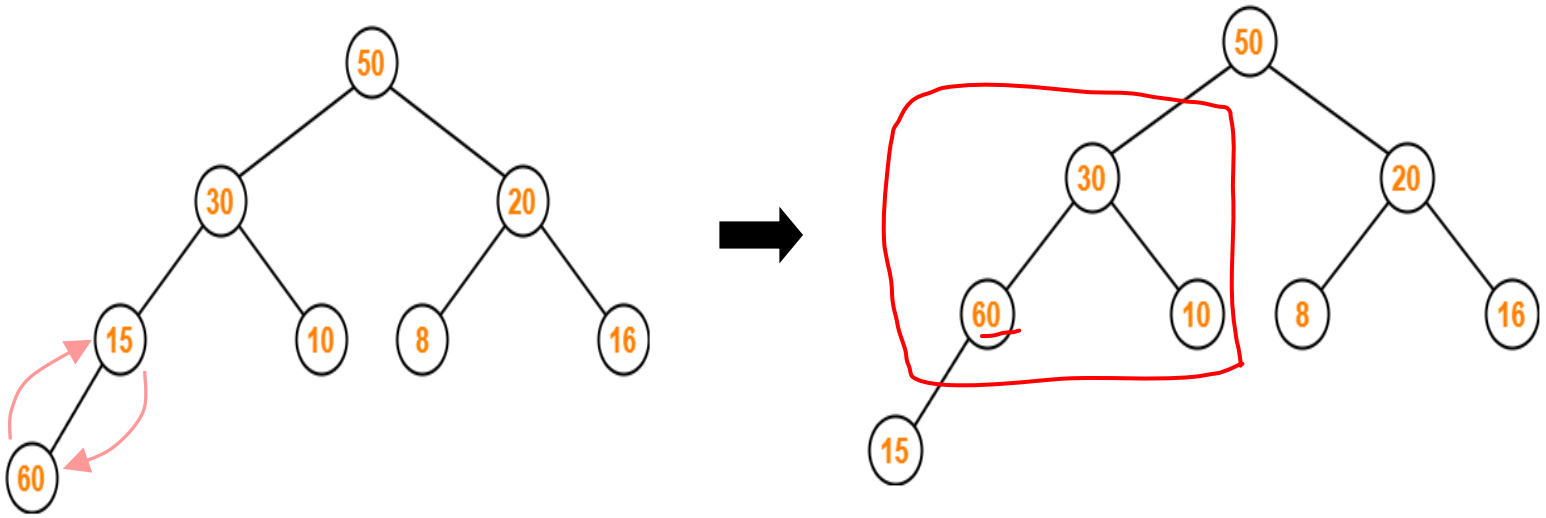


Insert Operation - Example

Step-3:

- We must ensure that the resultant tree is a max heap.
- Node 15 contains greater element in its left child node.
- So, we swap node 15 and node 60.

The resulting tree is-

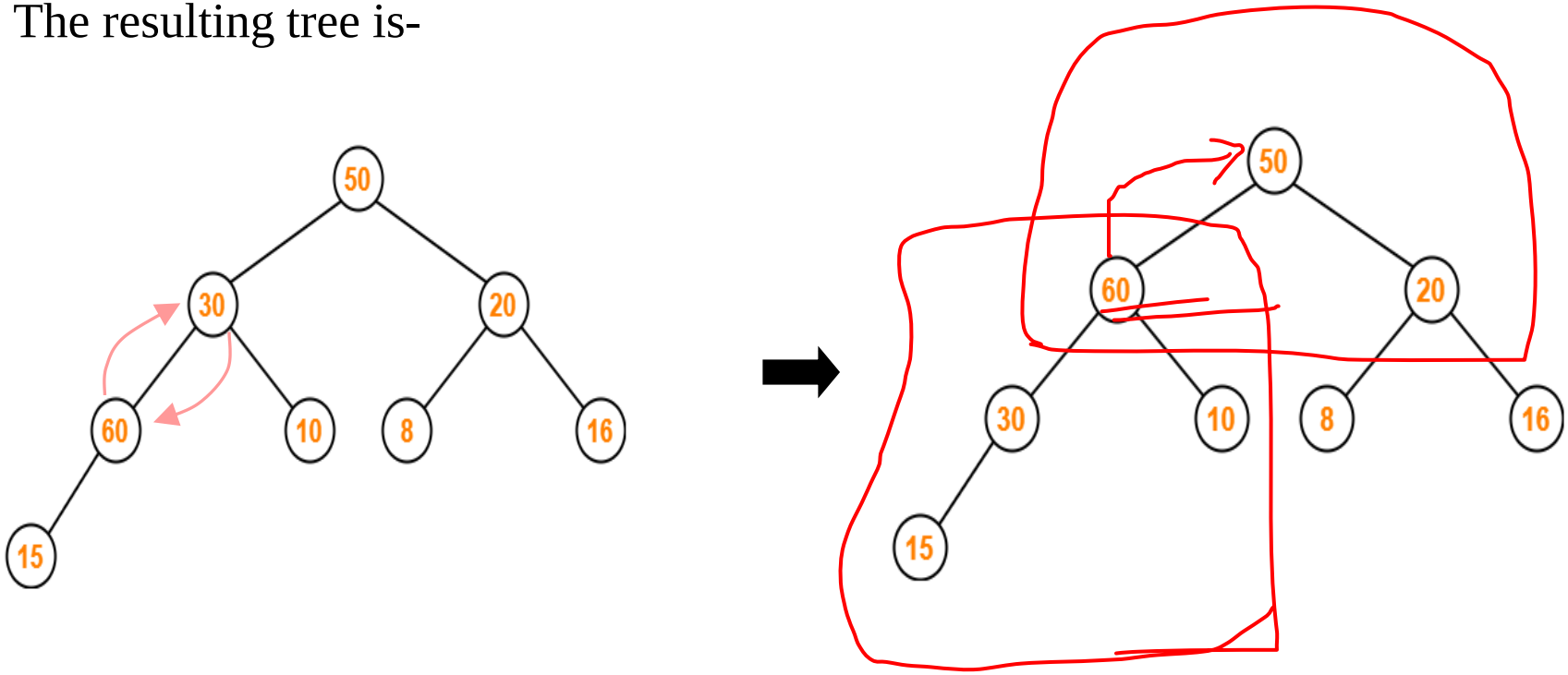


Insert Operation - Example

Step-4:

- Node 30 contains greater element in its left child node.
- So, we swap node 30 and node 60.

The resulting tree is-

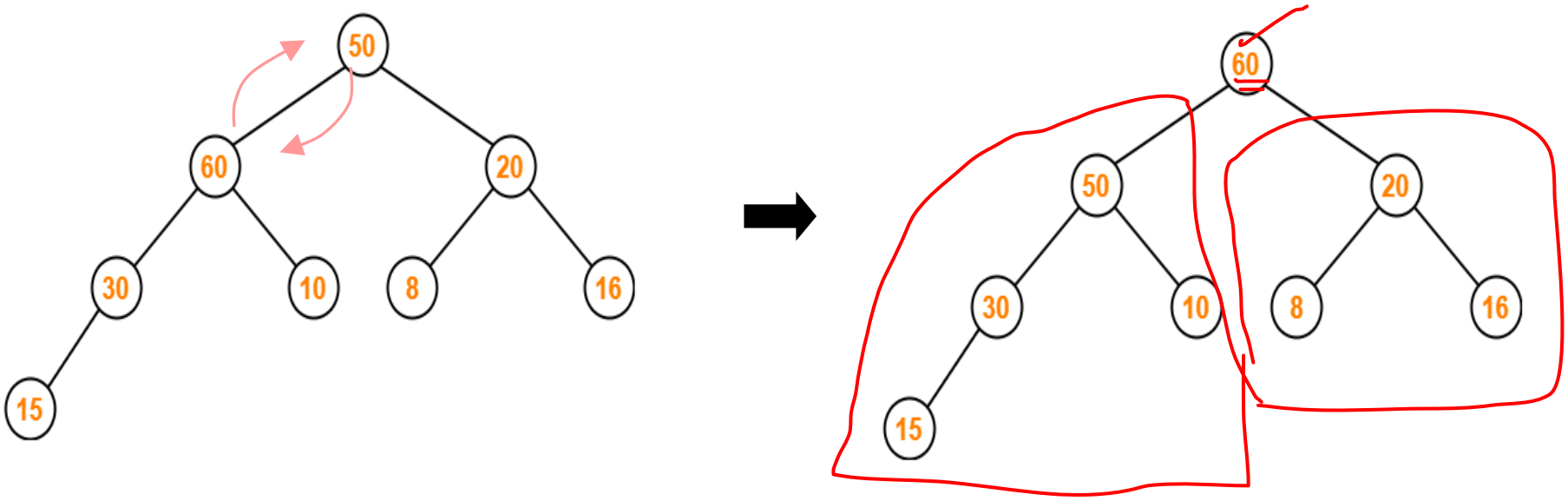


Insert Operation - Example

Step-5:

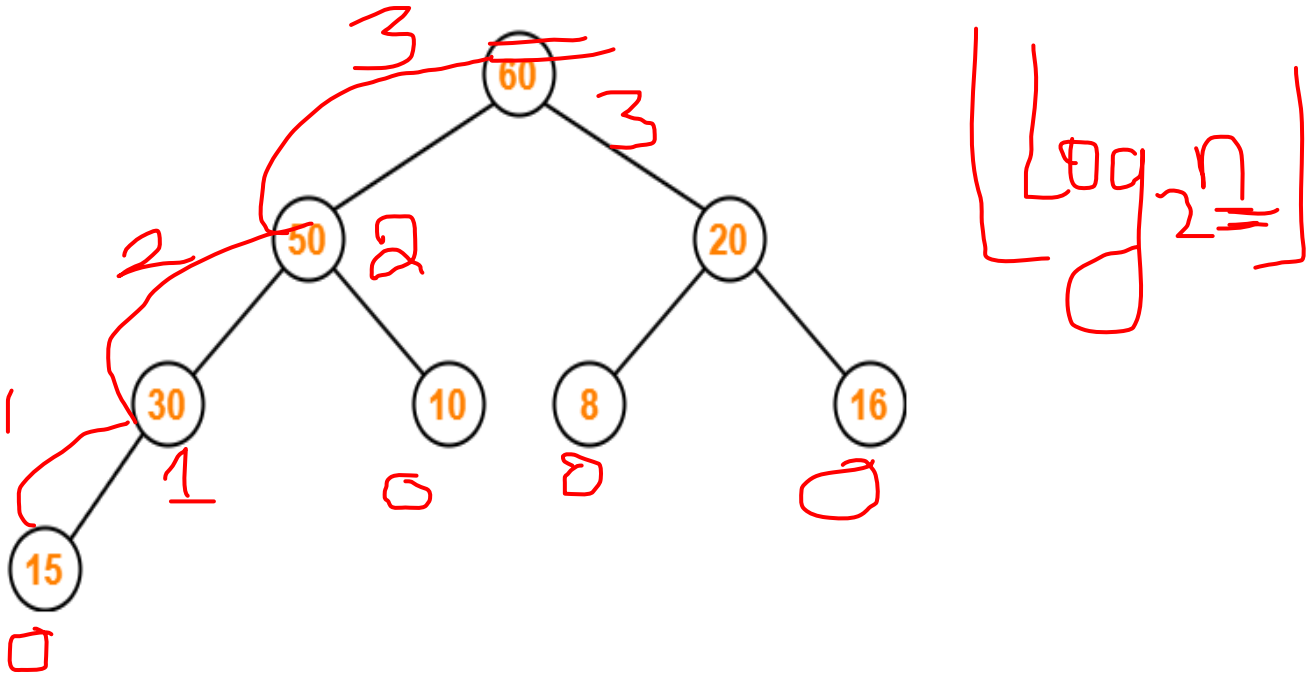
- Node 50 contains greater element in its left child node.
- So, we swap node 50 and node 60.

The resulting tree is-



Insert Operation - Example

The resultant max-heap after insertion is 60,50,20,30,10,8,16,15 and its tree representation is shown below:

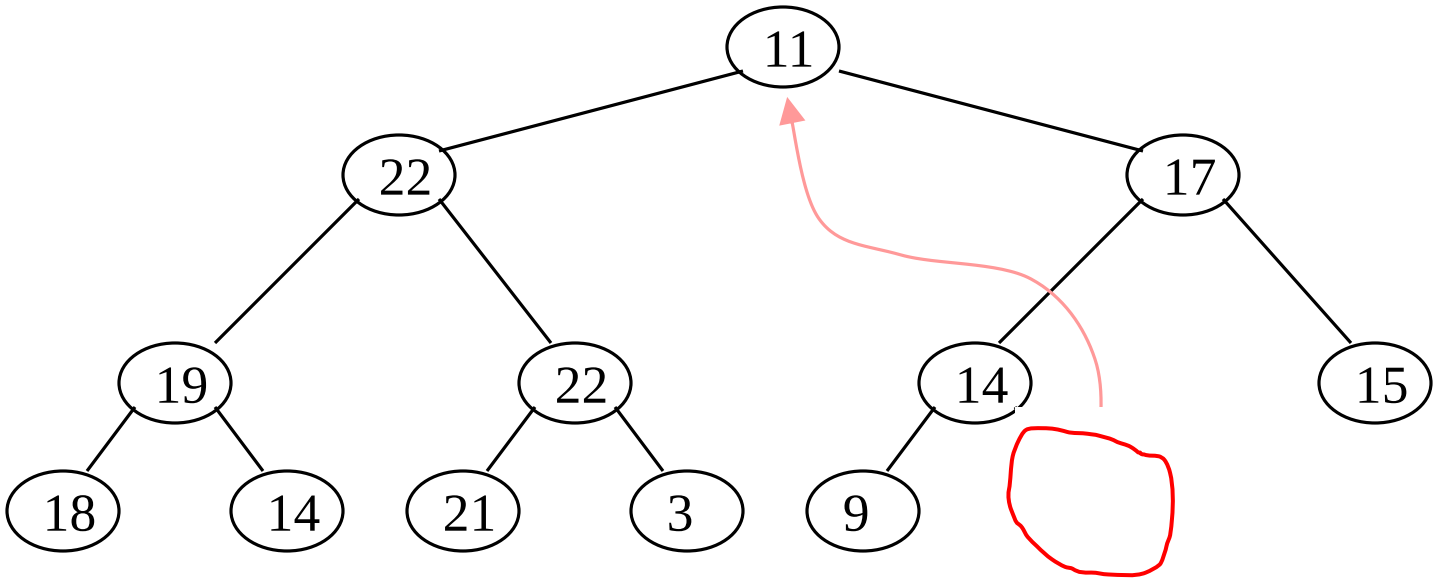


Insert Operation - Analysis

- ✓ Insertion operation cannot require more key comparisons than the heap's height.
- ✓ Since the height of a heap with n nodes is about $\log_2 n$
- ✓ The time efficiency of insertion operation is O(log n).

Delete Max Operation-Removing the root

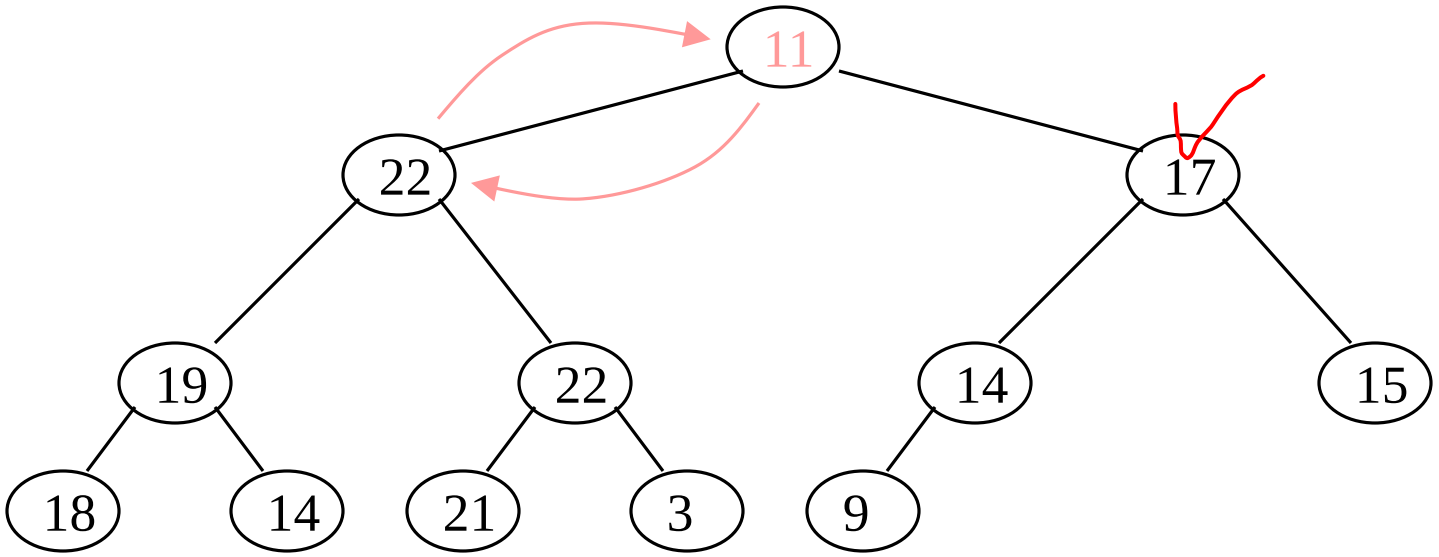
- Notice that the largest number is now in the root
- Suppose we *discard* the root:



- How can we fix the problem after removing the root?
- Solution: remove the rightmost leaf at the deepest level and use it for the new root

Delete Max Operation-Removing the root

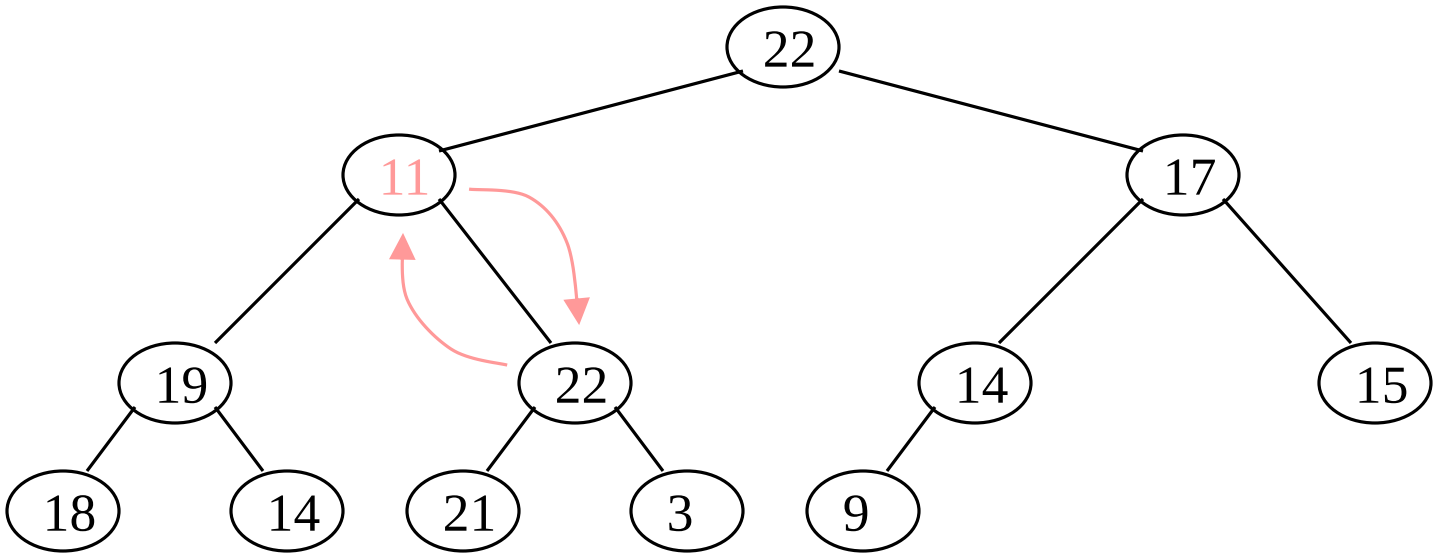
- Our tree is complete binary tree, but no longer a heap
- However, *only the root* lacks the heap property



- We can **siftUp()** the root
- After doing this, one and only one of its children may have lost the heap property

Delete Max Operation-Removing the root

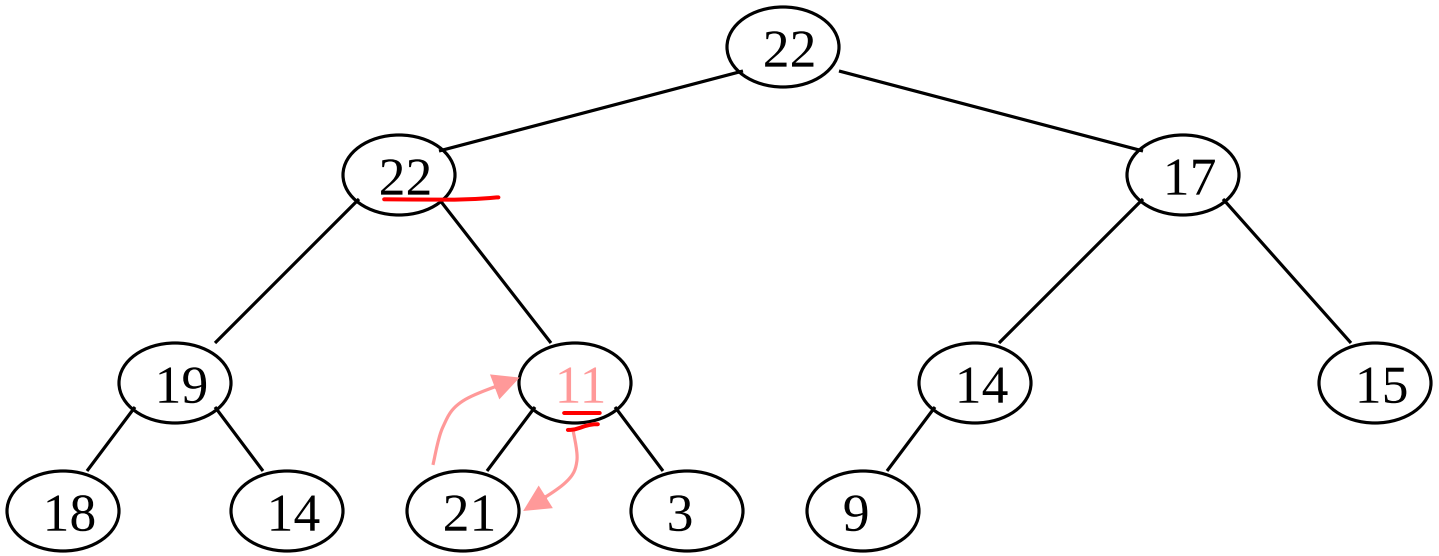
- Now the left child of the root (still the number **11**) lacks the heap property



- We can **siftUp()** this node
- After doing this, one and only one of its children may have lost the heap property

Delete Max Operation-Removing the root

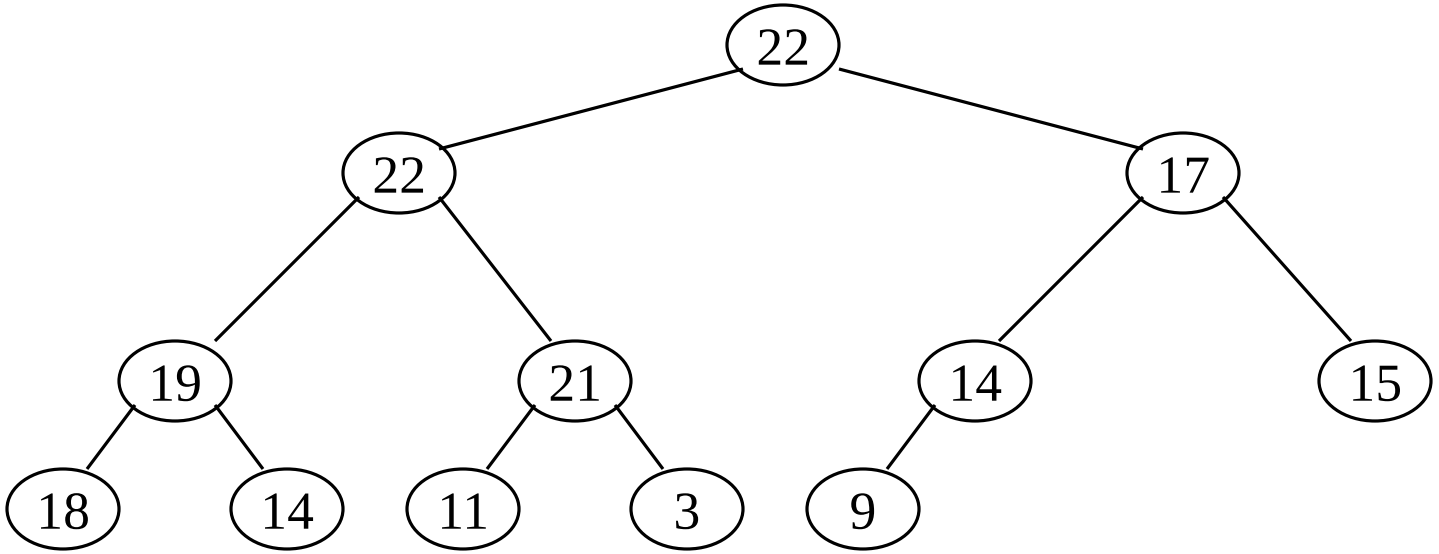
- Now the right child of the left child of the root (still the number **11**) lacks the heap property:



- We can **siftUp()** this node
- After doing this, one and only one of its children may have lost the heap property —but it doesn't, because it's a leaf

Delete Max Operation-Removing the root

- Our tree is once again a heap, because every node in it has the heap property



- Once again, the largest (or *a* largest) value is in the root
- We can repeat this process until the tree becomes empty
- This produces a sequence of values in order largest to smallest

Assignment

- Consider the heap 9,6,8,2,5,7. Insert 10
- Consider the heap 10,6,9,2,5,7,8.Insert 20
- Construct a max heap for the given array of elements-1, 5, 6, 8, 12, 14, 16
- Consider the following max heap-50, 30, 20, 15, 10, 8, 16. Delete the root node. After heapifying delete the node 20.

Constructions of Heap

There are two principal alternatives for constructing Heap.

- Top-down heap construction
- Bottom-up heap construction

Bottom-up heap construction

The bottom-up heap construction algorithm is illustrated bellow.

- It initializes the essentially complete binary tree with n nodes by placing keys in the order given and then “heapifies” the tree as follows.

- Starting with the last parental node, the algorithm checks whether the parental dominance holds for the key in this node. If it does not, the algorithm exchanges the node’s key K with the larger key of its children and checks whether the parental dominance holds for K in its new position. This process continues until the parental dominance for K is satisfied. (Eventually, it has to because it holds automatically for any key in a leaf.)

- After completing the “heapification” of the subtree rooted at the current parental node, the algorithm proceeds to do the same for the node’s immediate predecessor

- The algorithm stops after this is done for the root of the tree.

Bottom-up heap construction-Example

Bottom-up construction of a heap for the list 2, 9, 7, 6, 5, 8 is shown below.

Note: The double headed arrows show key comparisons verifying the parental dominance.

Solution:

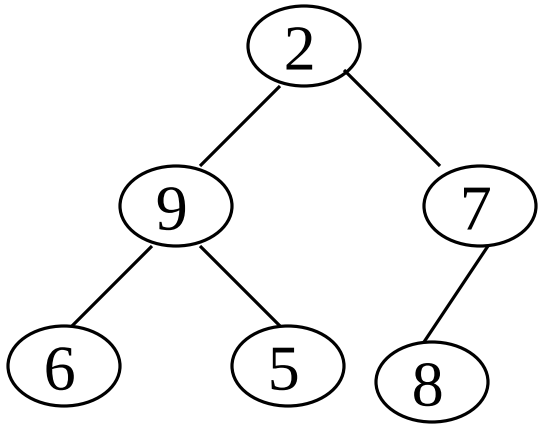
Solution:

Step 1:Construct the array for the given list of elements and then construct the binary tree by placing the keys in the order given

Array:

[1]	[2]	[3]	[4]	[5]	[6]
2	9	7	6	5	8

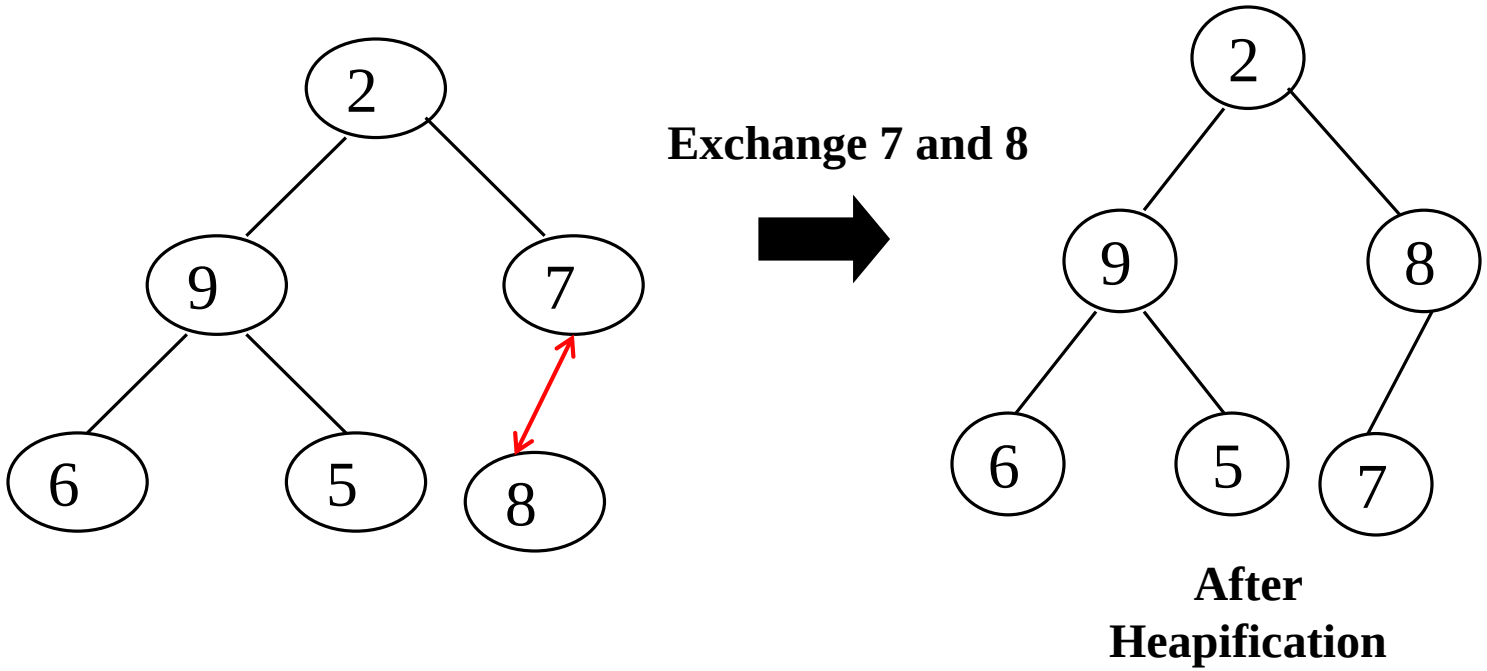
Binary Tree:



Solution:

Step 2: Starting with the last parental node, the algorithm checks whether the parental dominance holds for the key in this node is satisfied. If it does not, the algorithm heapifies the subtree rooted at the current parental node. This continues until we reach the root

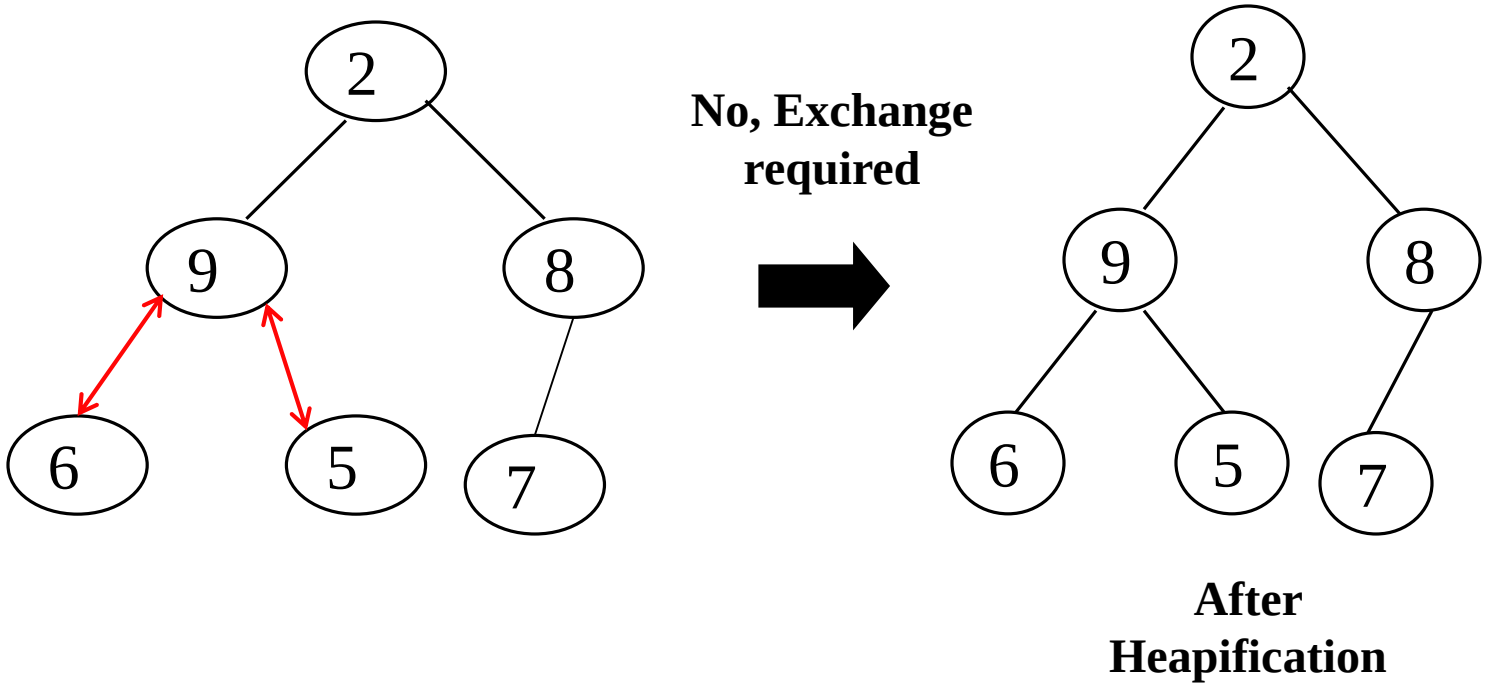
❖ Here, the last parental node is 7



Solution:

Step 2 (Continued):

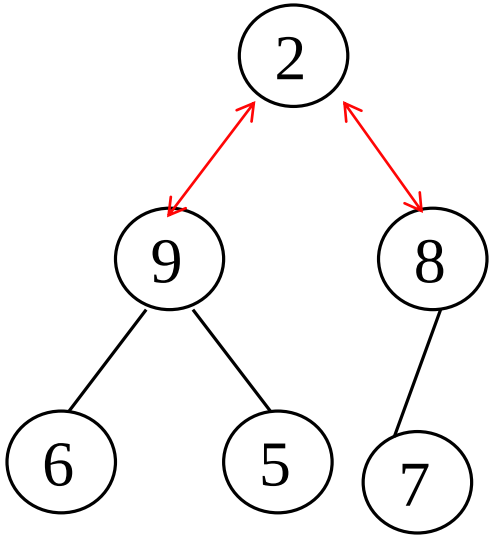
❖ The next last parental node is 9



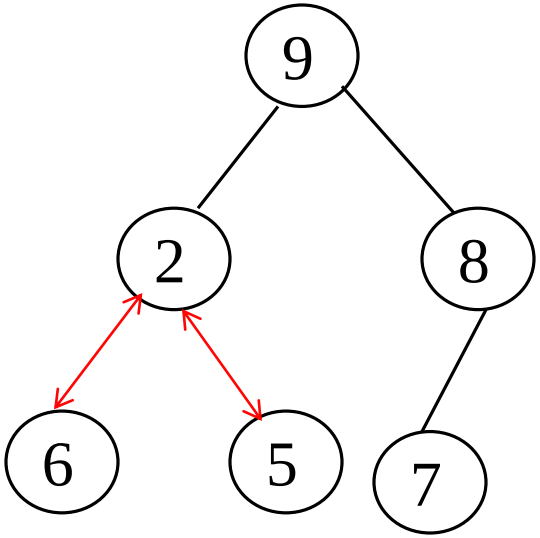
Solution:

Step 2 (Continued):

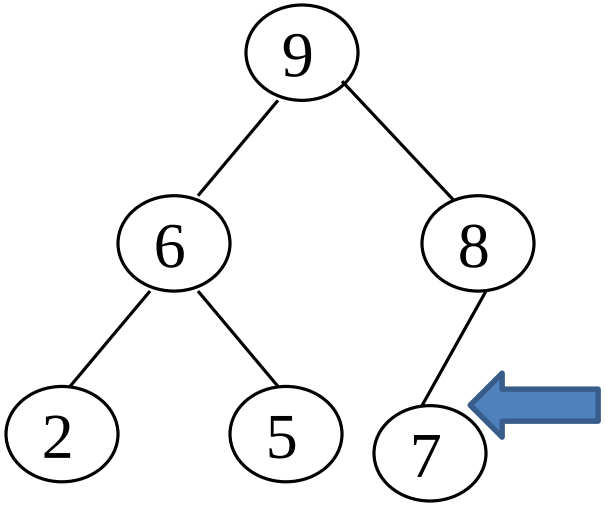
❖ The next last parental node is 2, which is the root



Exchange 2 and 9



Exchange 2 and 6



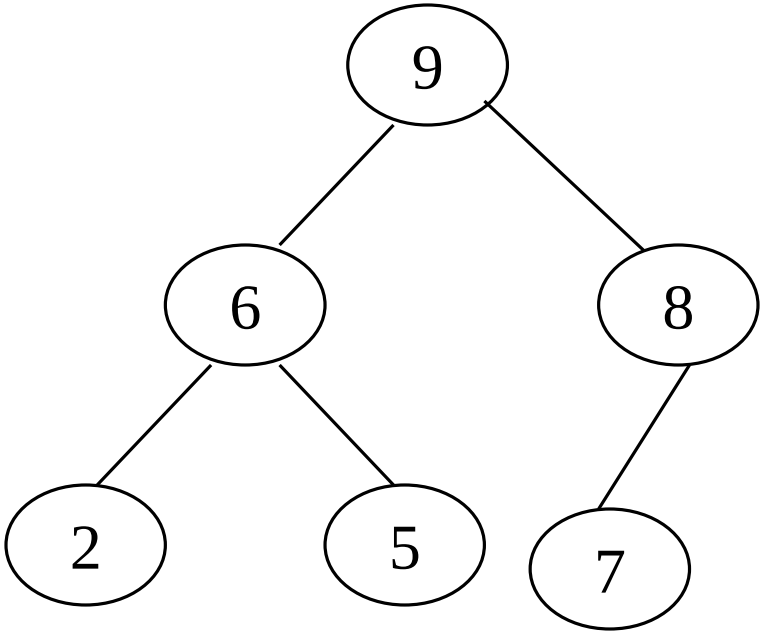
After Heapification

Solution:

Step 2(Continued):

The algorithm stops as we have reached the root.

Step 3: Therefore the heap that is constructed is as below



Assignment

- Construct the heap for the list 3,2,4,1,6,5, by using bottom-up algorithm
- Construct the heap for the list 1, 5, 6, 8, 12, 14, 16,10, by using bottom-up algorithm

Heap Sort

✓ **Heapsort** is an interesting sorting algorithm is discovered by J.W. J. Williams.

✓ This is a two- stage algorithm that works as follows.

- **Stage 1 (heap construction):** Construct a heap(max-heap) for a given array.
- **Stage 2 (maximum deletions):** Apply the root-deletion operation n−1 times to the remaining heap.

✓ As a result, the array elements are eliminated in decreasing order.

✓ But since under the array implementation of heaps an element being deleted is placed last, the resulting array will be exactly the original array sorted in increasing order.

Heap Sort- Example

Assignment

- Sort the given list of elements using heap sort 2,9,7,6,5,8 [July 2017]
- Sort the given list of elements using heap sort 3,2,4,1,6,5 [Jan 2019]

THANK YOU….