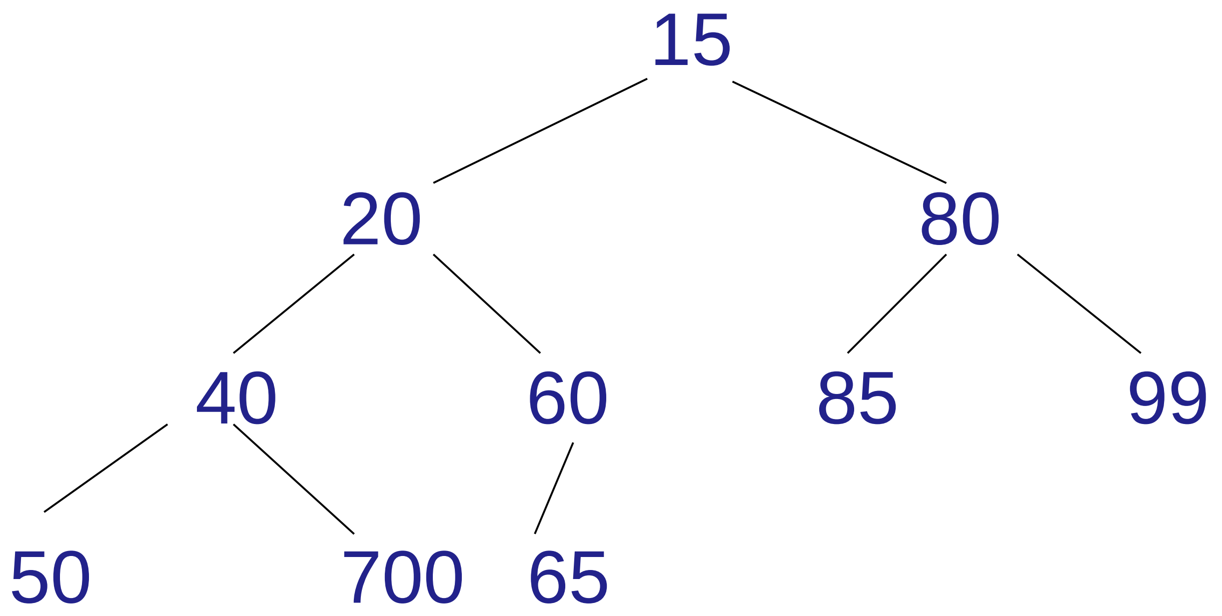
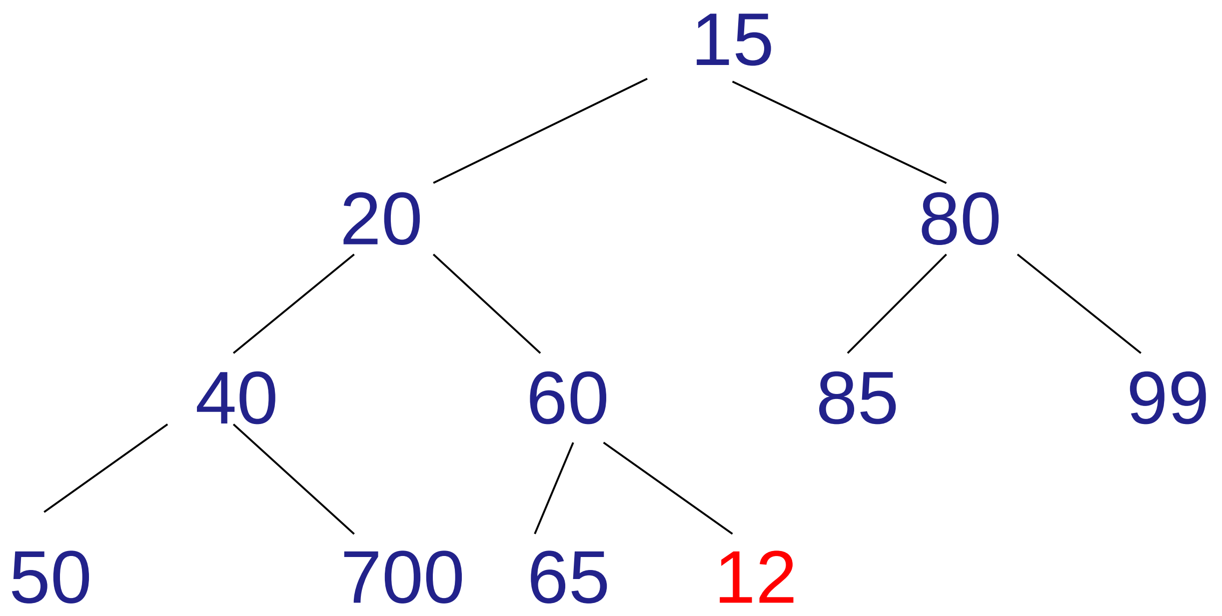
INSERTION AND DELETION IN HEAP TREES

insert 12 on this tree

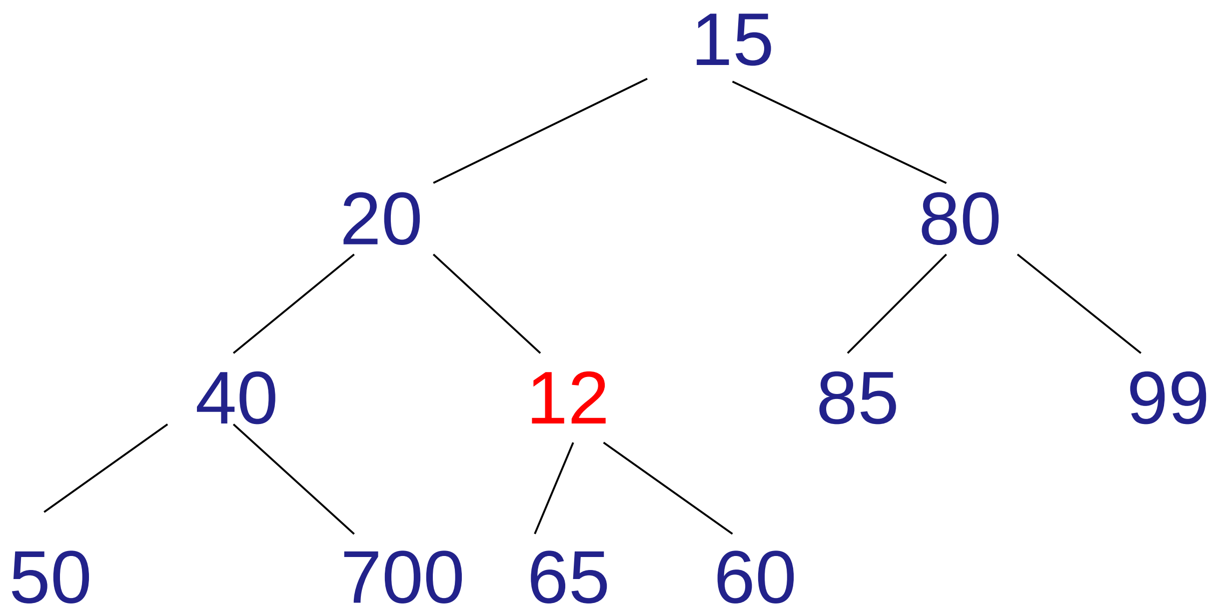


12 inserted, violates heap property



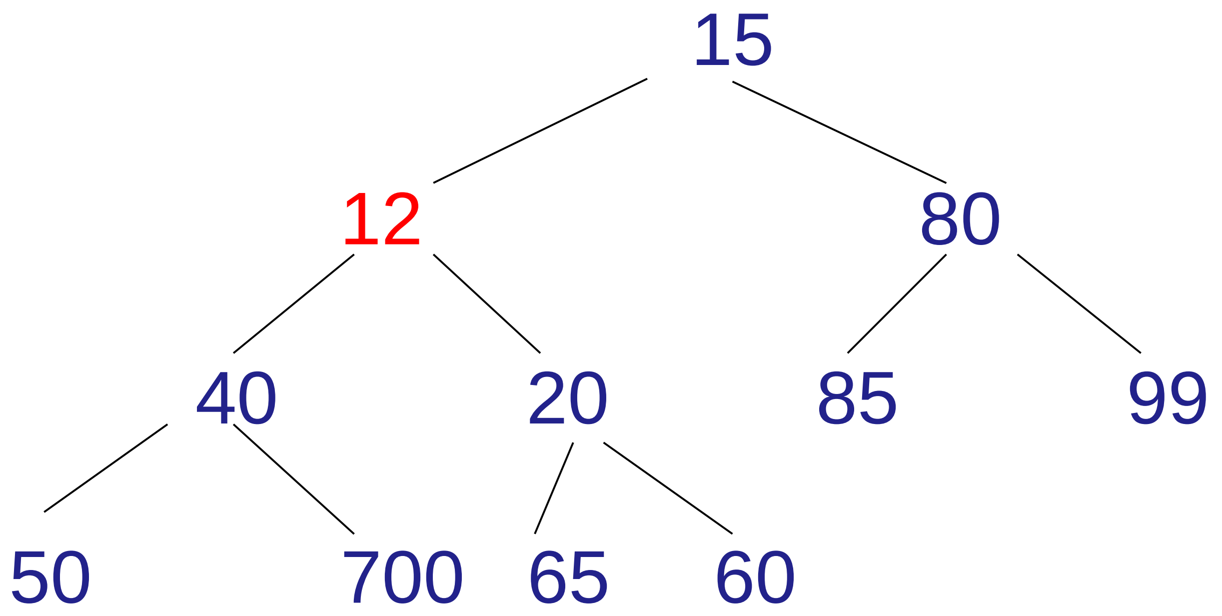
Percolate 12 up the tree

12 moved up, still violates heap property

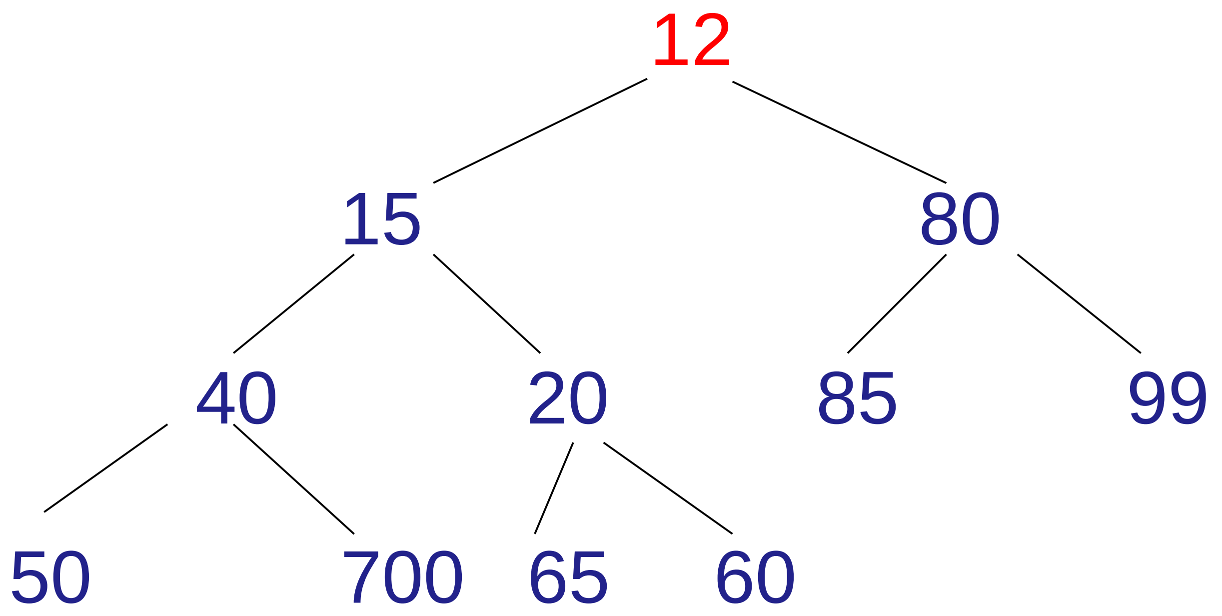


Percolate 12 up the tree

12 moved up, still violates heap property



Still there is need to Percolate 12 up the tree



Insertion complete, tree satisfies heap property.
3 changes made, complexity O(log n)

DELETION IN HEAP TREE

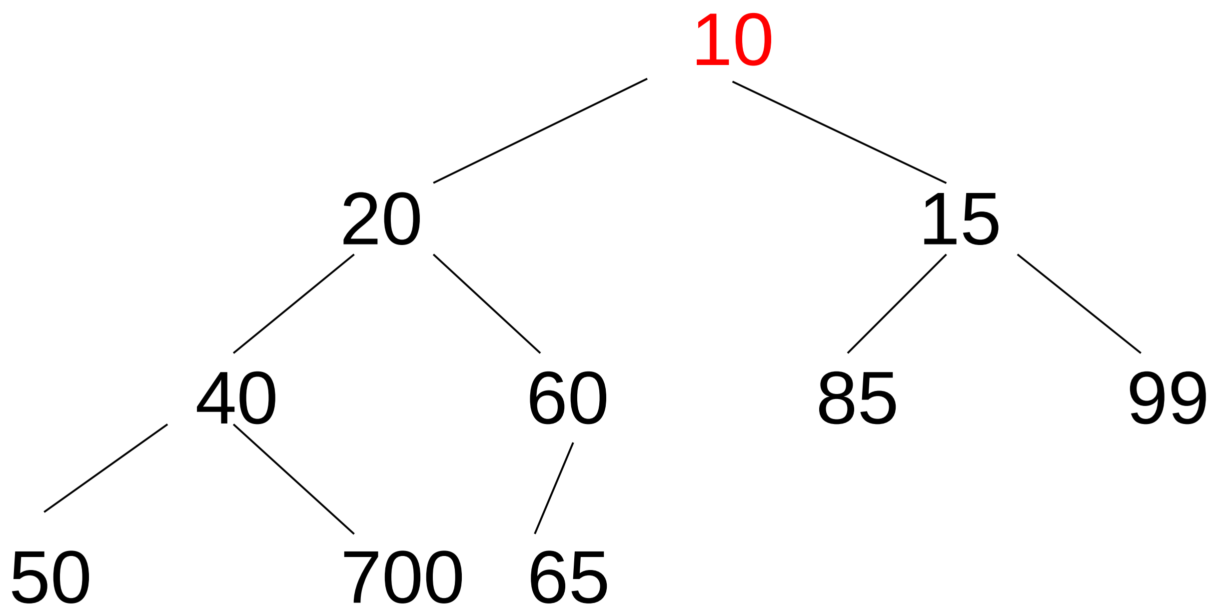
DELETE-MIN

Heap – Deletemin

Basic Idea:

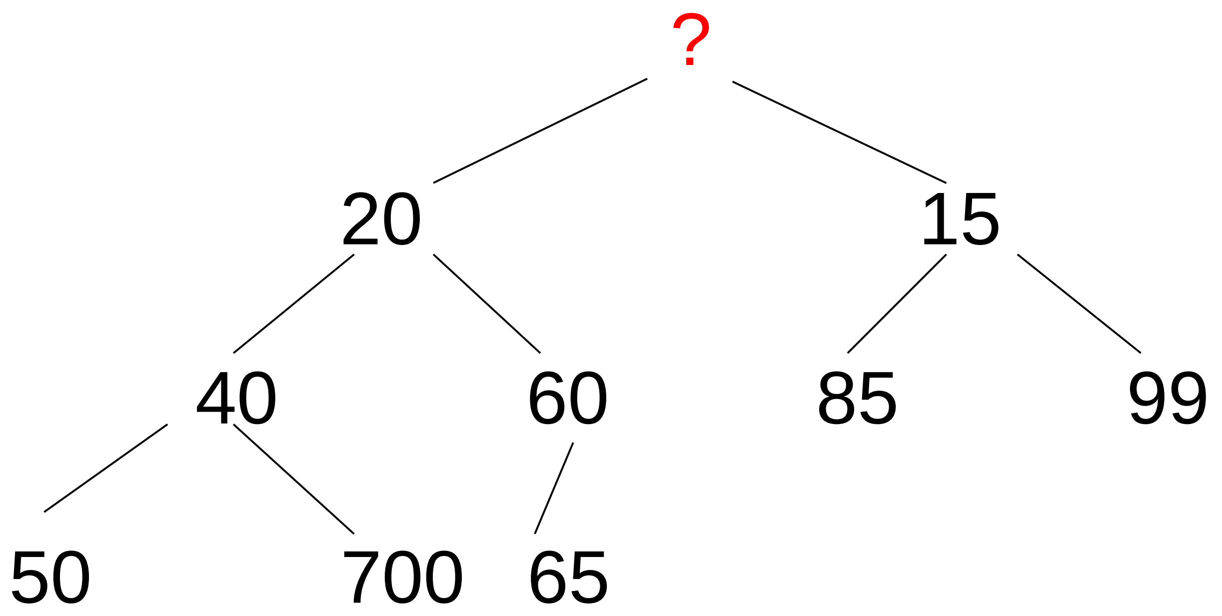
1. Remove root (that is always the min!)
2. Creates a hole
3. Put “last” leaf node at this hole
4. Compare its value with two children
5. If needed, Swap node with its smaller child
6. Repeat steps 3 & 4 until no swaps needed.

DeleteMin from this tree



	10	**20**	**15**	**40**	**60**	**85**	**99**	**50**	**700**	**65**			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Deletion creates a hole in the tree

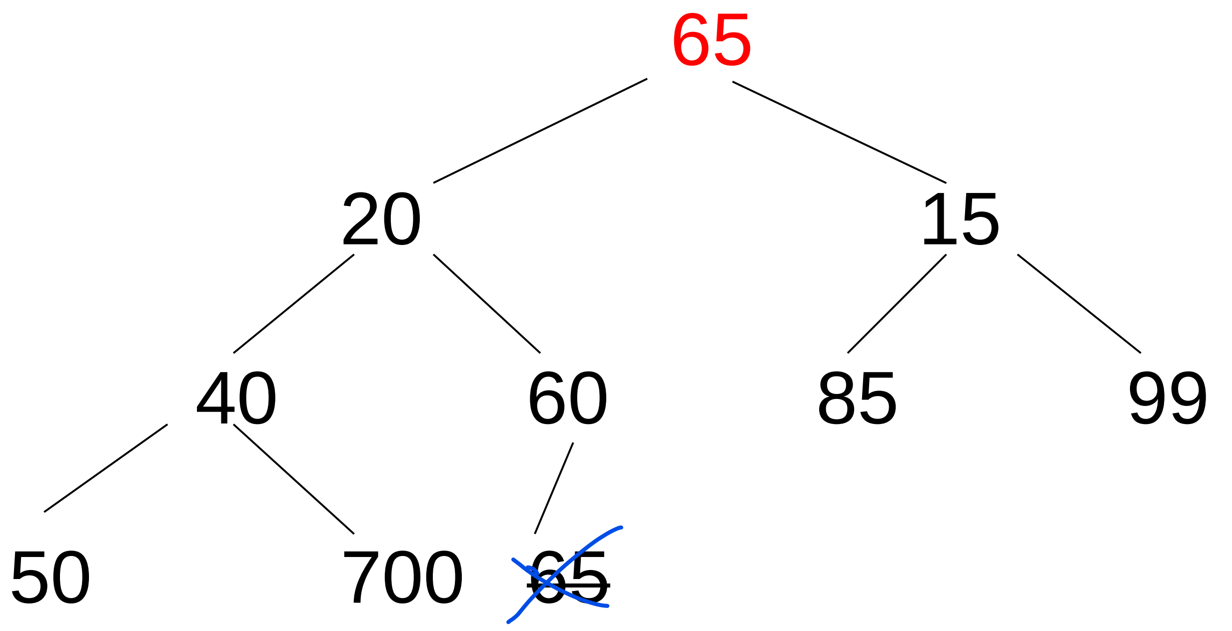


	--	**20**	**15**	**40**	**60**	**85**	**99**	**50**	**700**	**65**			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

After deletion, there will be 9 elements left
Position 1 can not be left blank.
Shift last element 65 to position 1 (root)

0	1	2	3	4	5	6	7	8	9	10	11	12	13
	65	**20**	**15**	**40**	**60**	**85**	**99**	**50**	**700**	**--**			

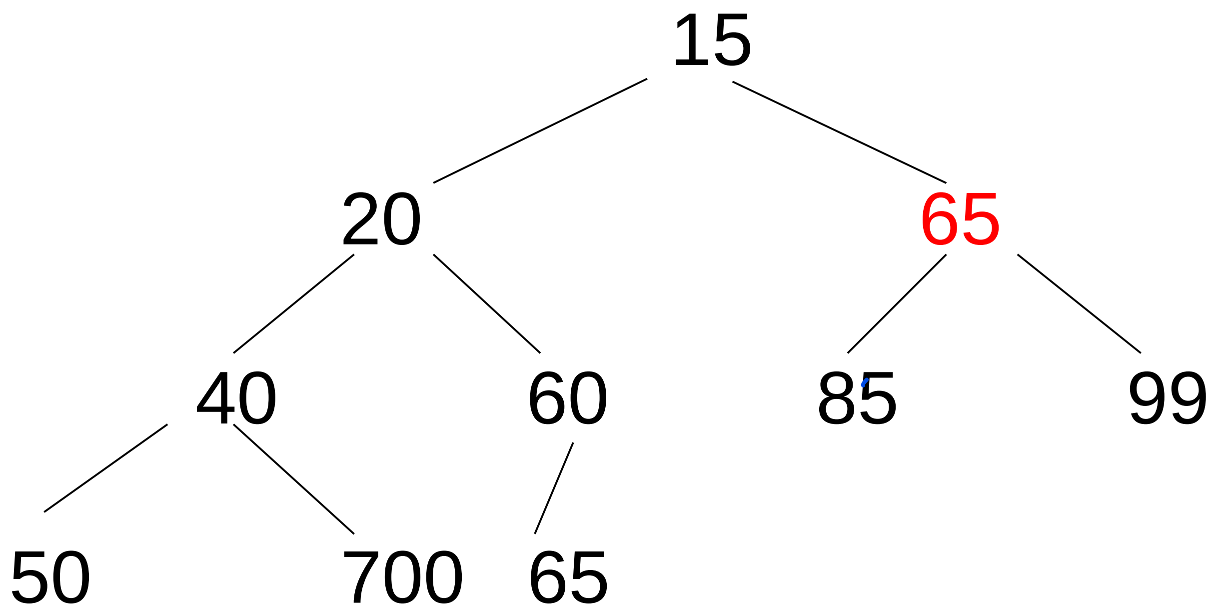
Last element moved to root



65 will be replaced by 20 or 15?

	--	**20**	**15**	**40**	**60**	**85**	**99**	**50**	**700**	**65**			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

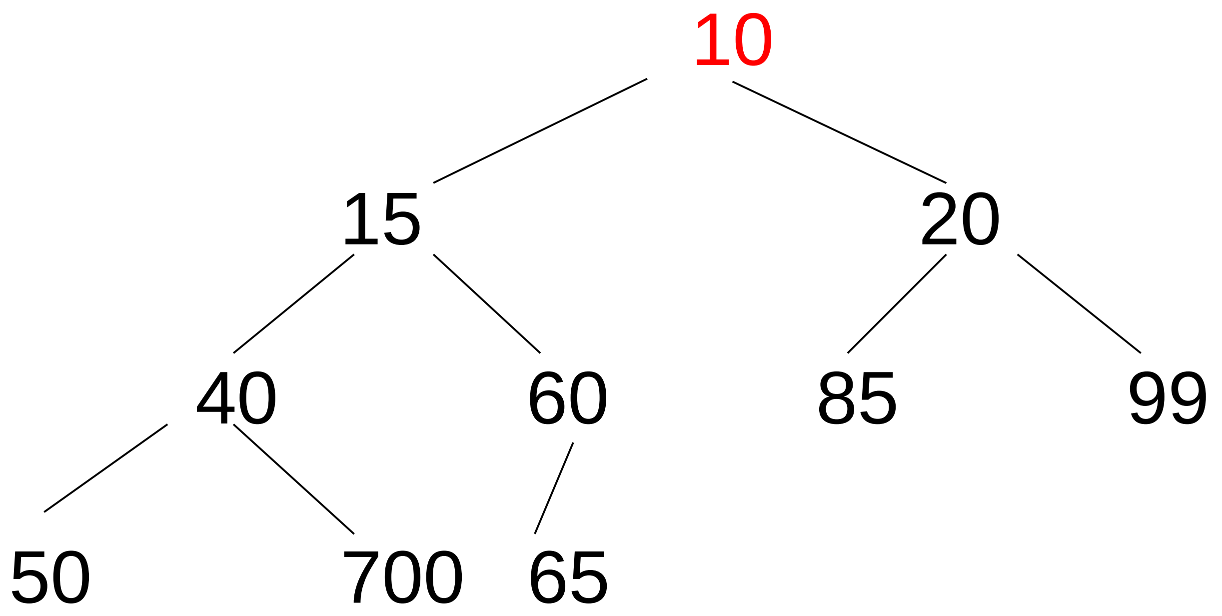
Exchange 65 with smaller child



	15	**20**	**65**	**40**	**60**	**85**	**99**	**50**	**700**				
0	1	2	3	4	5	6	7	8	9	10	11	12	13

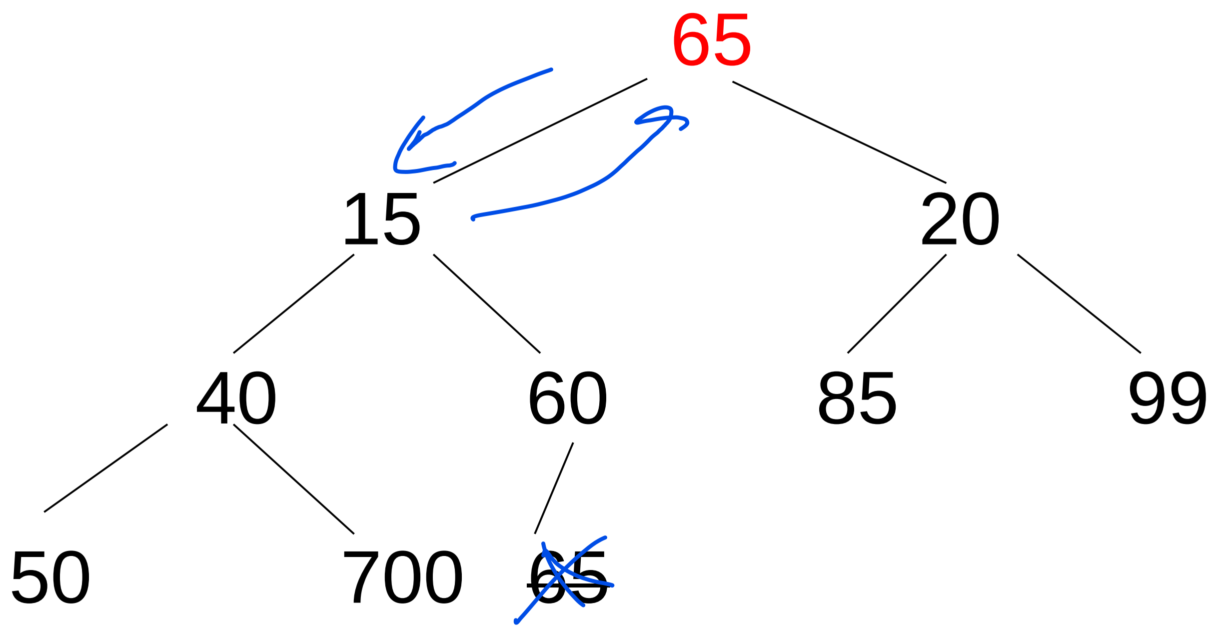
Suppose we need to DeleteMin from this tree

new tree



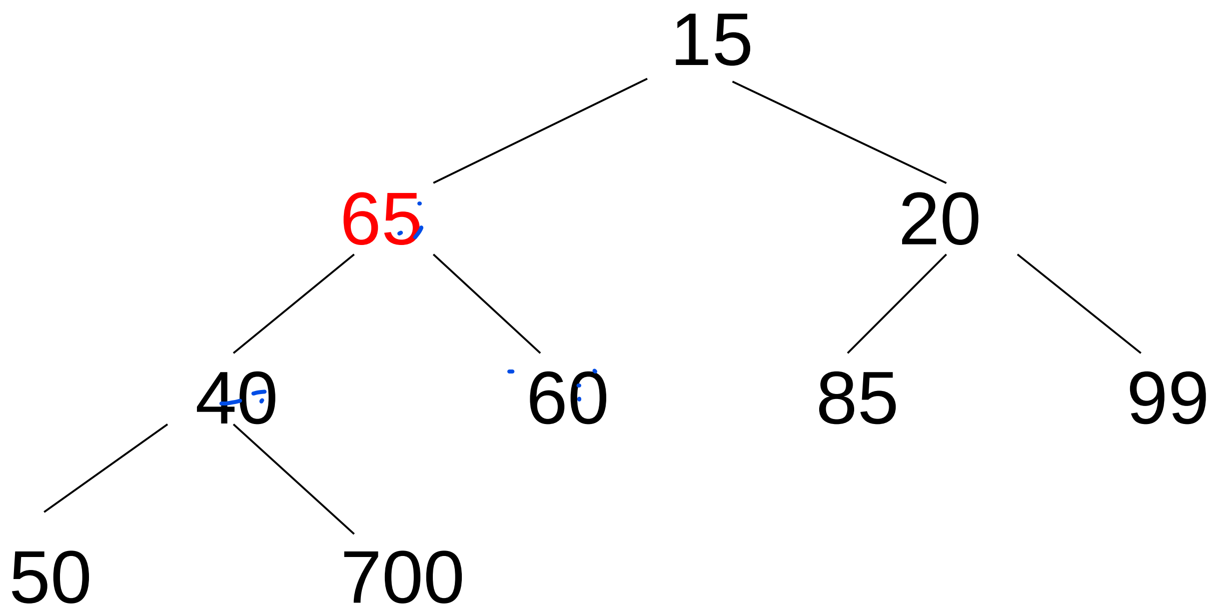
	10	**20**	**15**	**40**	**60**	**85**	**99**	**50**	**700**	**65**			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Last element moved to root

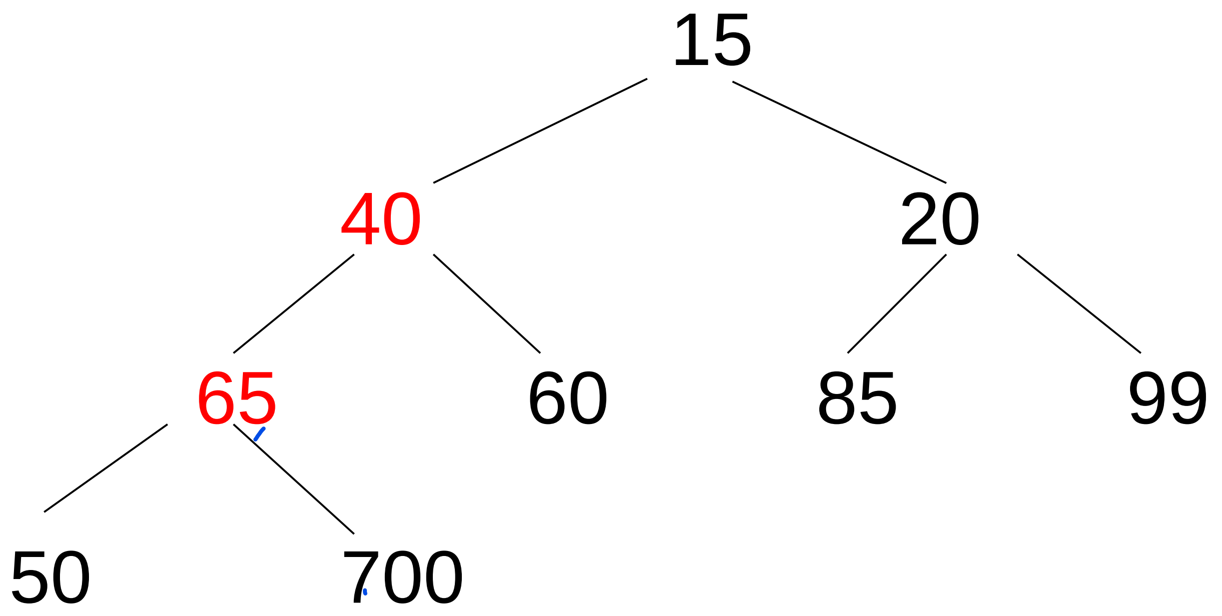


65 will be replaced by 20 or 15?

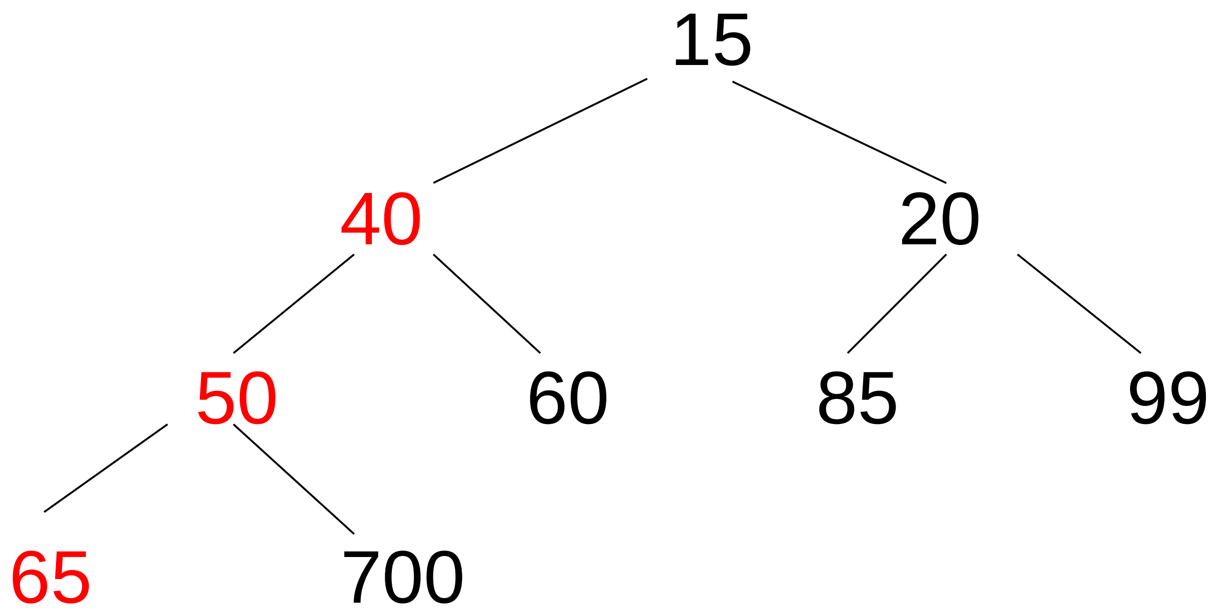
Exchange 65 with smaller child



Exchange 65 with smaller child



Exchange 65 with smaller child



Every deletion needs max (log n) operations.
Thus deletions of k smallest items would need k log n

3. BuildHeap
Heapify

Heapify

Building a Heap

- Inserting an item on Heap tree needs O(Log n)
- To create a tree of n elements, insert one element at a time
- O(n log n) in the worst case

- Time complexity is same as that of sorting an array. So what is the advantage of heap structure?

- Can we do it in O(n)?

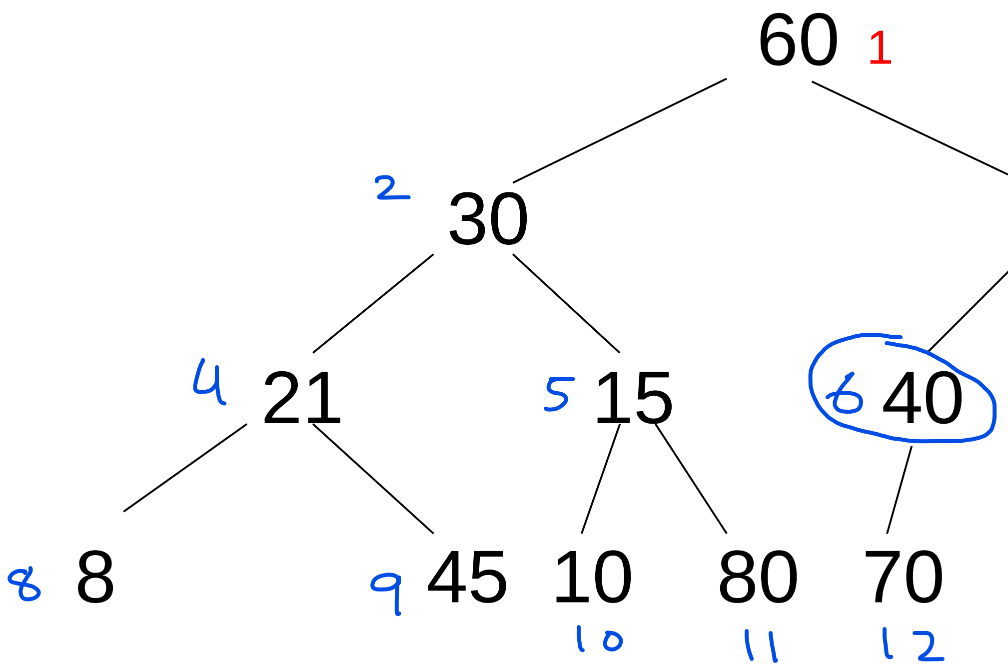
Alternative approach

- Given a set of n elements
  - Do not insert the elements one by one
  - put all of them randomly on a heap tree
  - *We need not check each element to figure out if the tree meets heap structure*
  - *We can leave the bottom most elements (leaf elements) as it is*
  - *Heapify* the remaining elements on the tree
- .

Heapify approach

- Note:
  - On the heap tree with randomly placed elements, no need to consider each node
  - Bottom half elements need not be examined.
  - They are not violating heap property
  - Start examining the nodes from position n/2

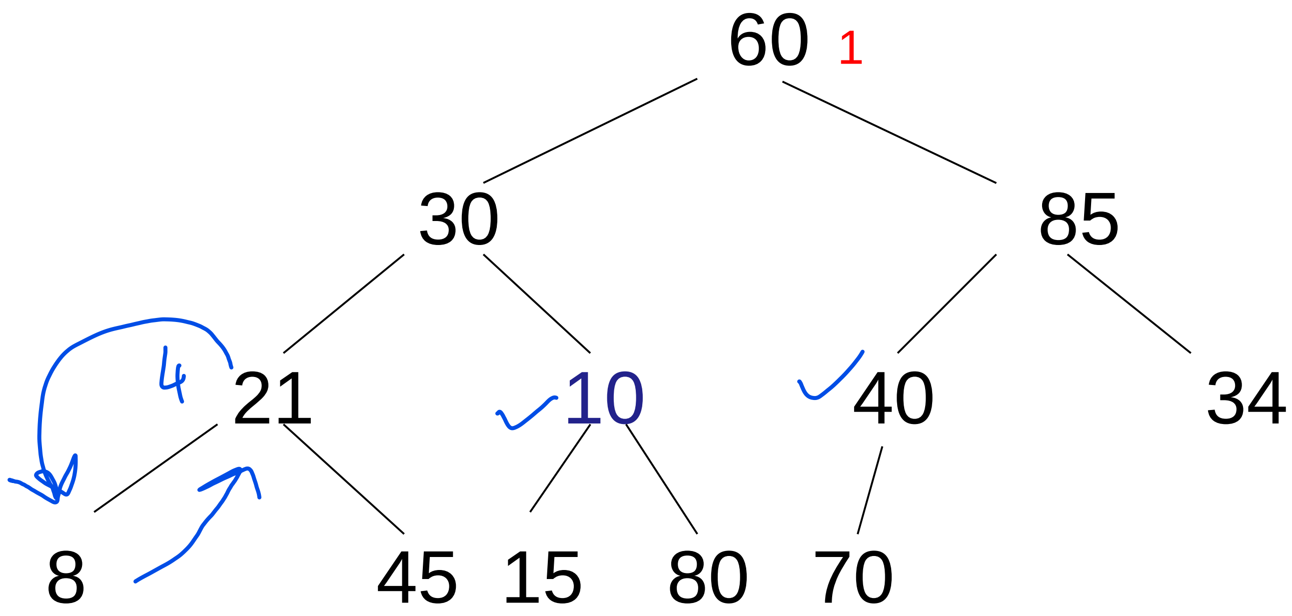
Put random elements of array on heaptree



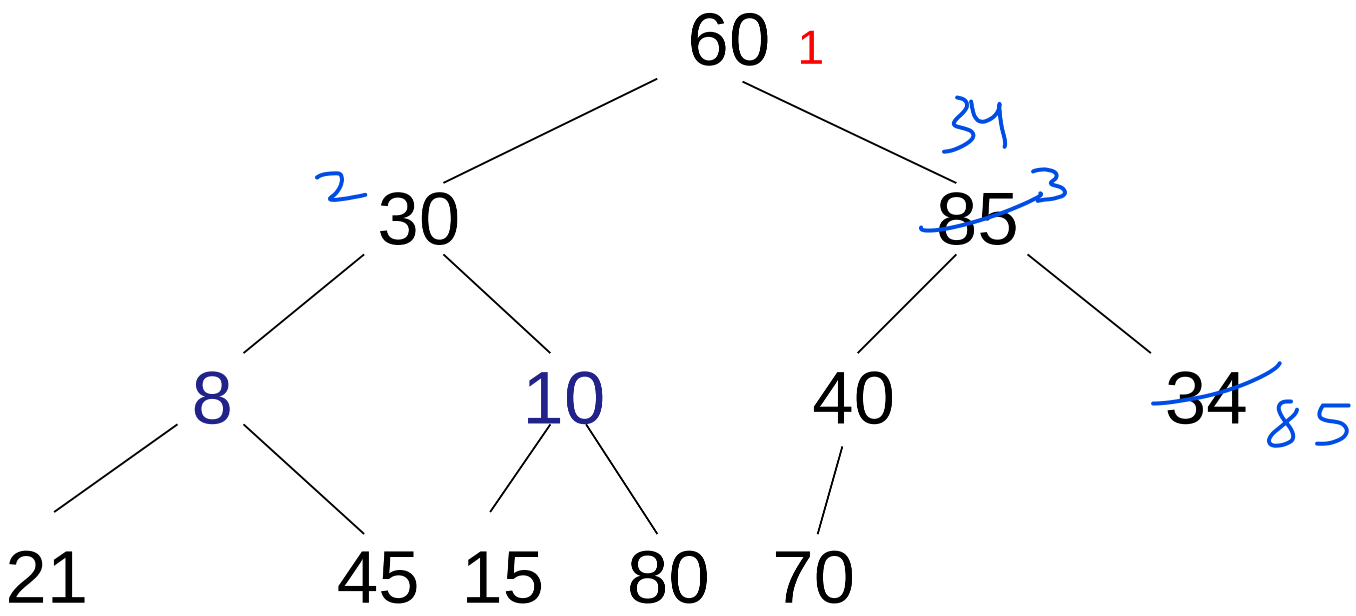
12 elements

Drop Leaf elements

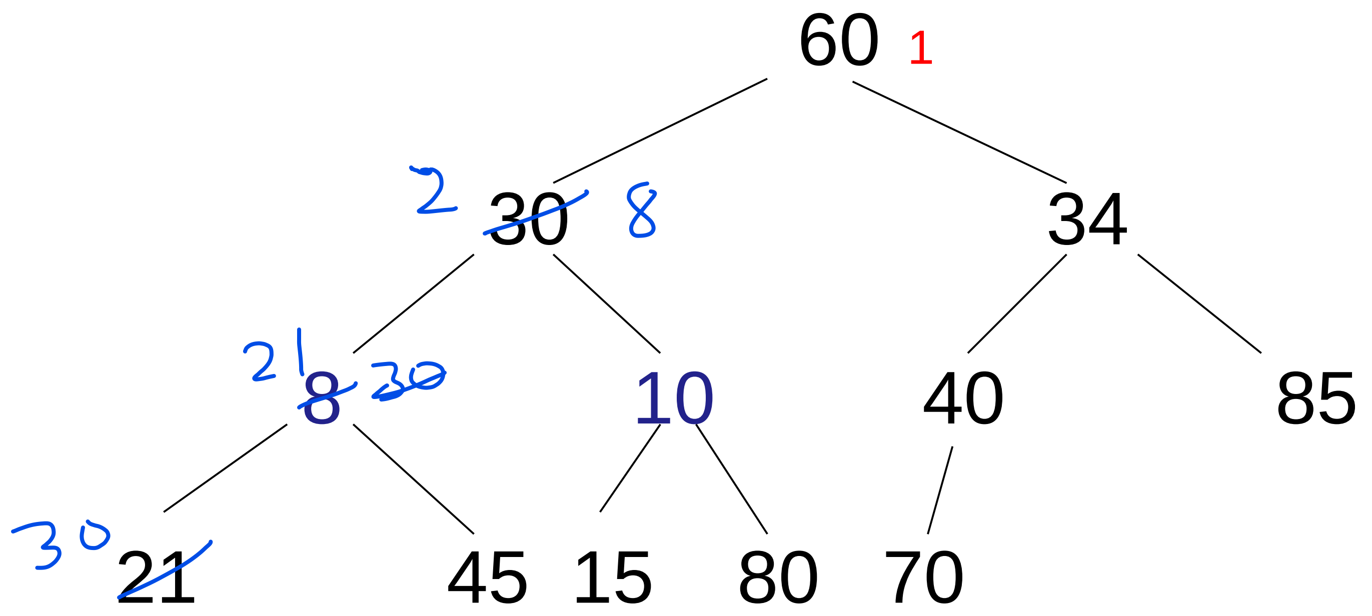
Start checking from element 6 and move up the tree.
Element 6: 40 in right place
Element 5: 15 needs to be exchanged with 10



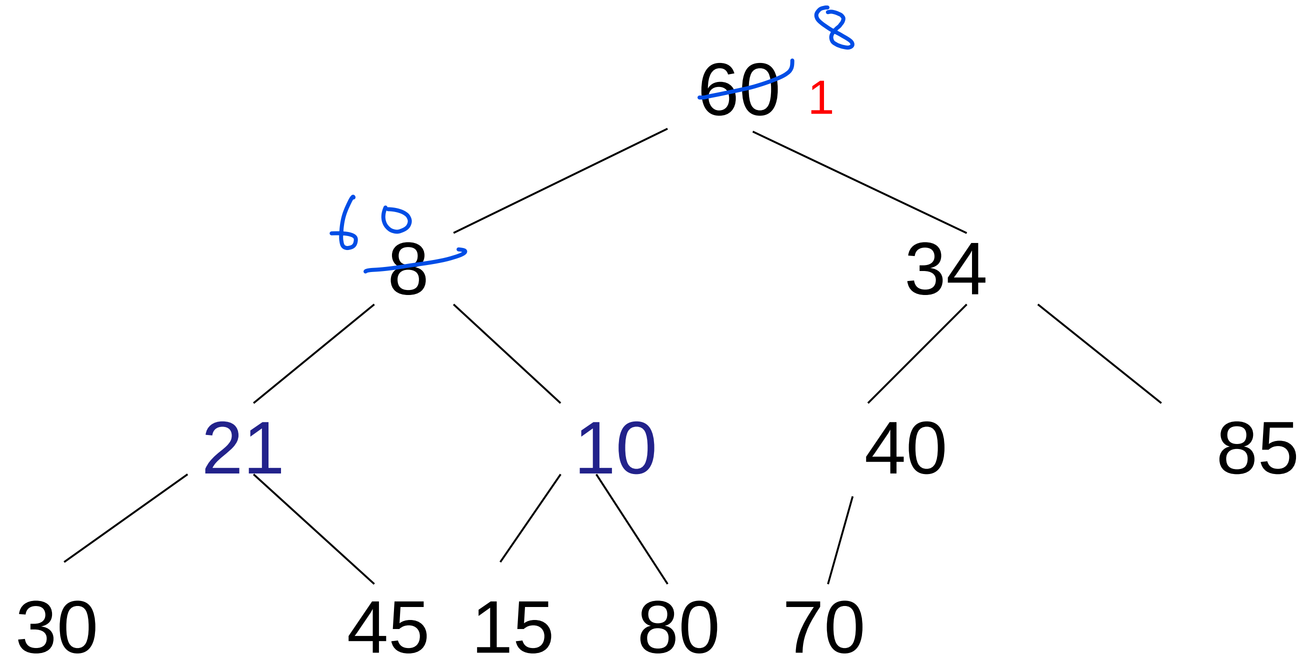
Element 4: 21 needs to be exchanged with 8



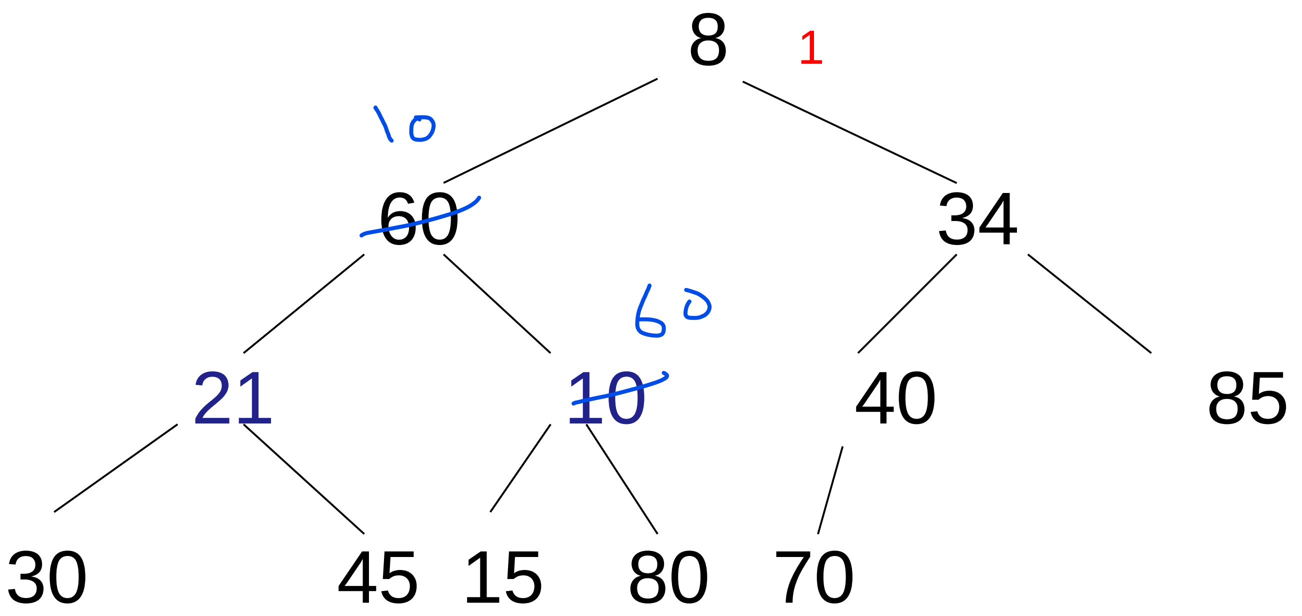
Element 3: 85 needs to be exchanged with smaller value 34



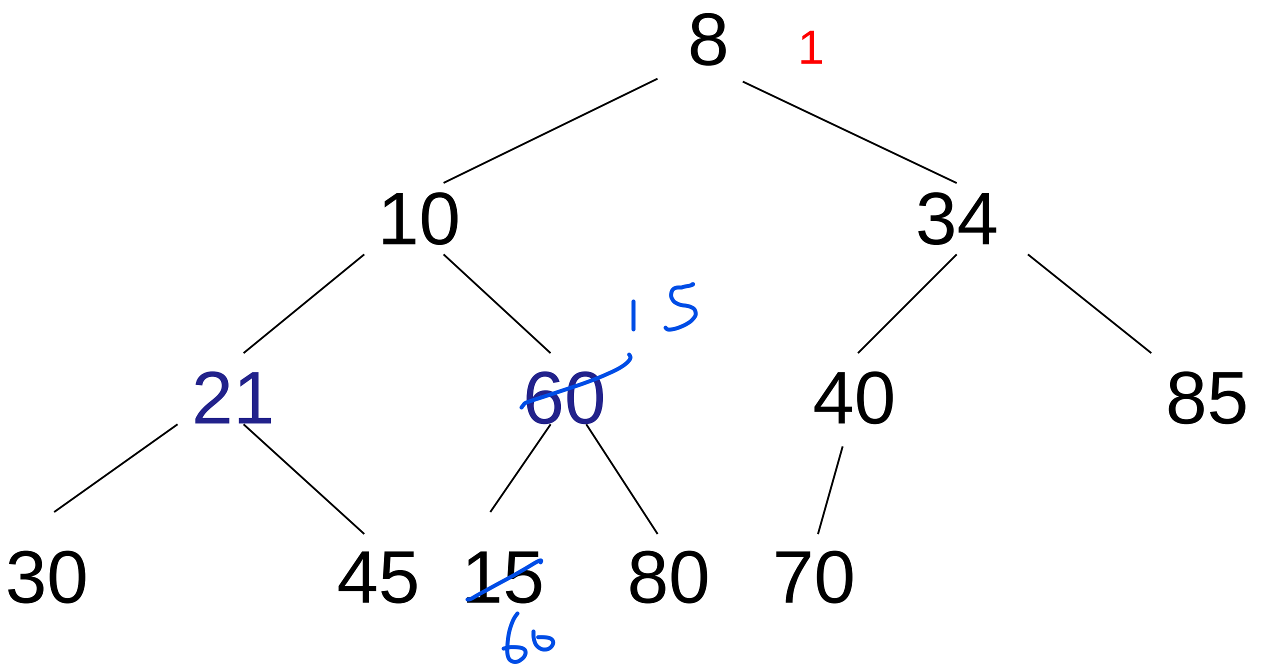
Element 2: 30 needs to be exchanged with smaller value 8
Further another exchange needed with 21 as well



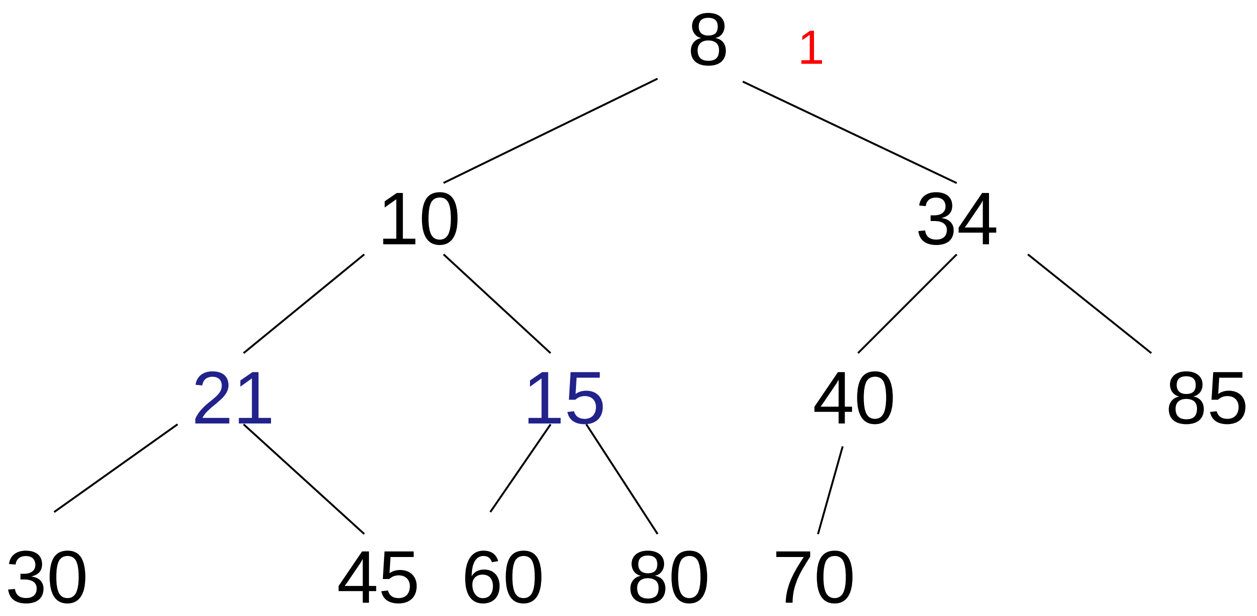
Element 1: exchange 60 with 8



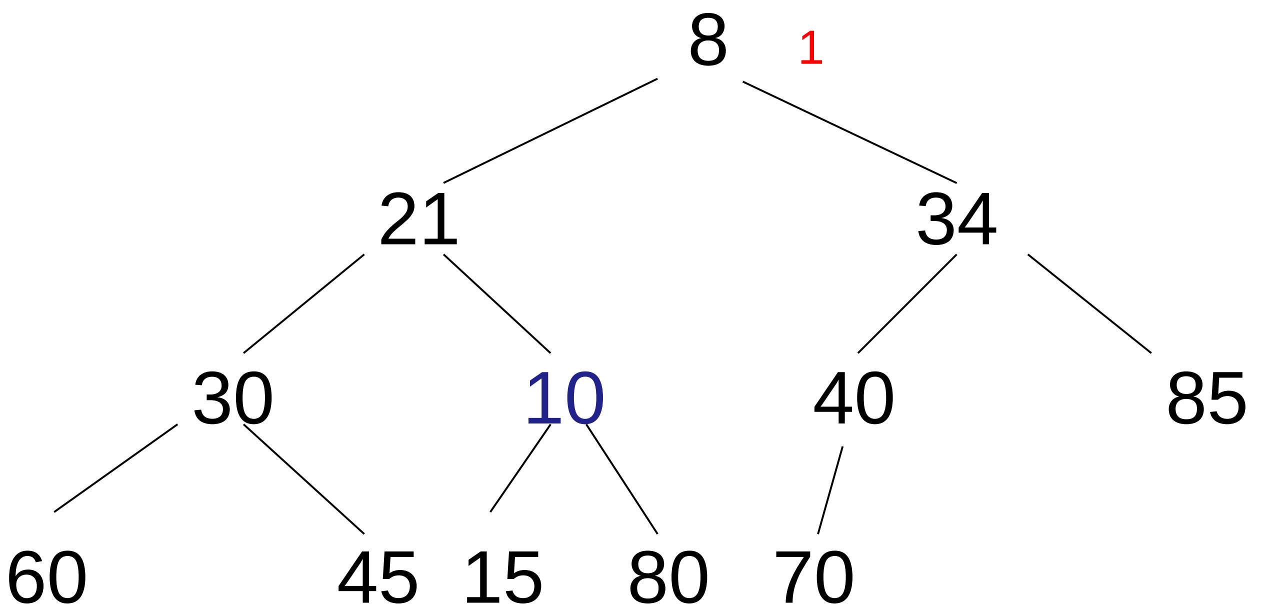
Element 1: exchange 60 with smaller of 10 and 21



Element 1 continue: exchange 60 with 15



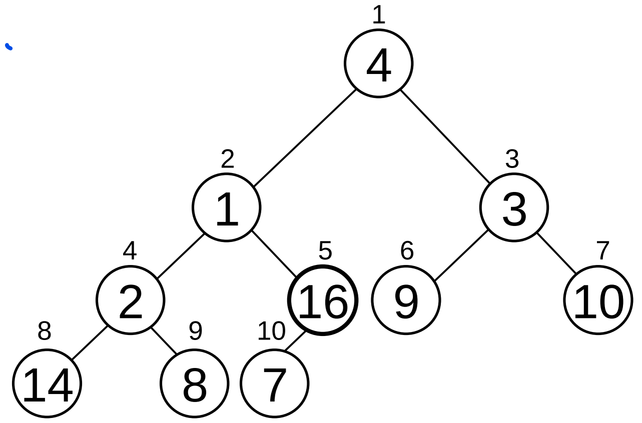
The heap tree has been created successfully. So in all only 6 elements had to be examined and moved around.



Now Heap Tree is ready.

Heapify process

- Consider elements shown
- Note elements 9, 10, 14, 8 and 7 need not be examined.
- Start by examining element 16 and upwards.

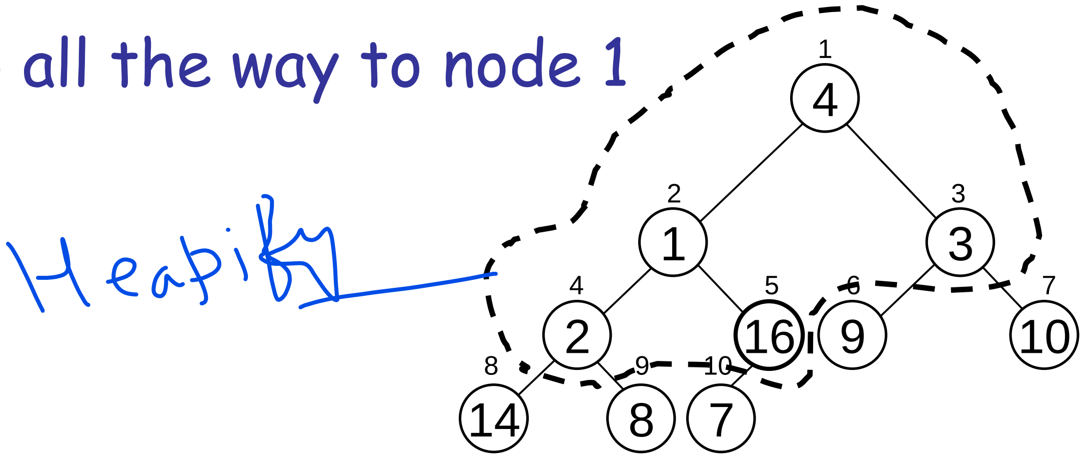


A:

4	1	3	2	16	9	10	14	8	7

Heapify n element tree

- Convert a random array $A[1 \dots n]$ into a heap
- The elements in the subarray $A[(\lfloor n/2 \rfloor+1) \,..\, n]$ are leaves
- No need to examine them,
- Apply HEAPIFY on elements between 1 and $\lfloor n/2 \rfloor$
- *Start from node $\lfloor n/2 \rfloor$ and go all the way to node 1*

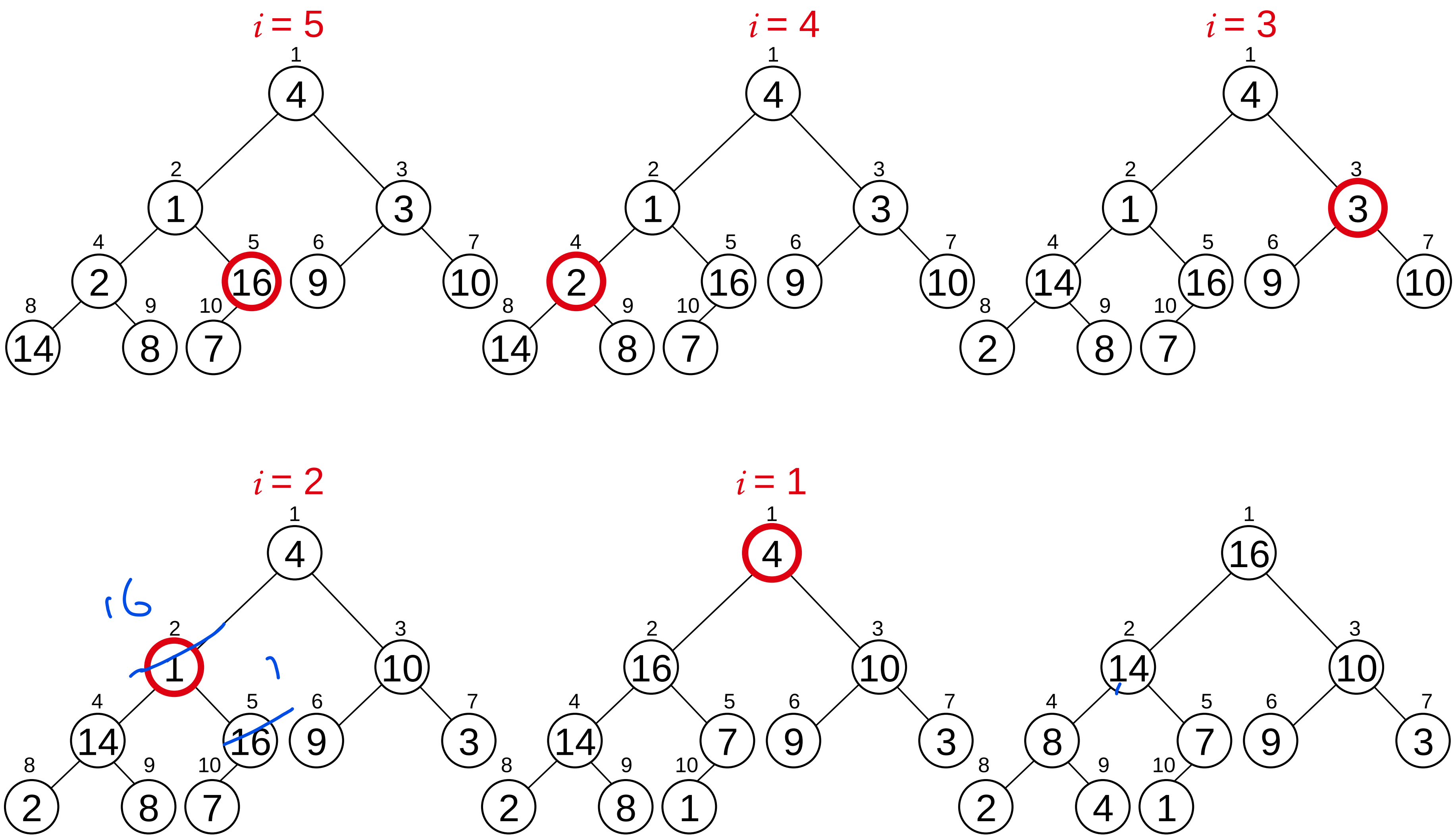


A:

4	1	3	2	16	9	10	14	8	7

Heapifying as maxHeap

4	1	3	2	16	9	10	14	8	7



Running Time of Heapify

Alg: BUILD-MAX-HEAP(*A*)

1. n = length[A]
2. **for** $i \leftarrow \lfloor n/2 \rfloor$ **downto** 1
3. **do** MAX-HEAPIFY(A, i, n) $O(\lg n)$ } $O(n)$

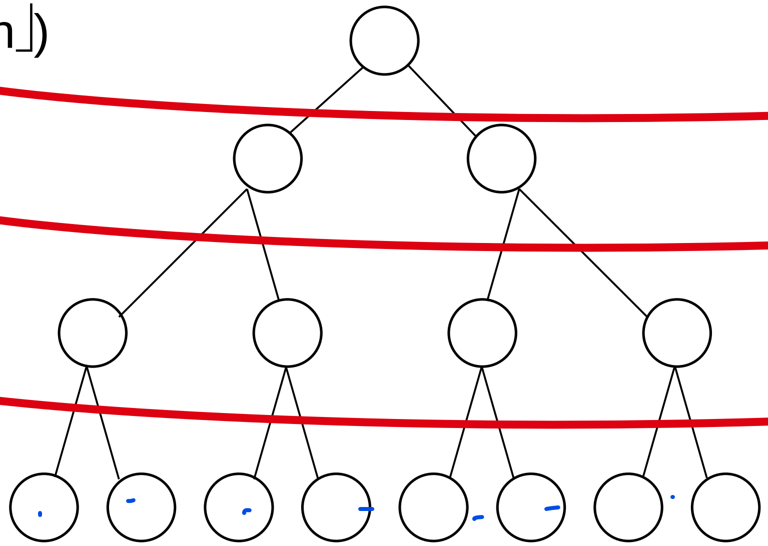
$\Rightarrow$ Running time: $O(n)$

Running Time of BUILD MAX HEAP

- Each node HEAPIFY takes $O(h)$ $\Rightarrow$ the cost of HEAPIFY on a node *i* is proportional to the height of the node *i* in the tree

$$\Rightarrow T(n) = \sum_{i=0}^{h} n_i h_i = \sum_{i=0}^{h} 2^i (h-i) = O(n)$$

Height	Level	No. of nodes
$h_0 = 3$ ($\lfloor \lg n \rfloor$)	$i = 0$	2^0
$h_1 = 2$	$i = 1$	2^1
$h_2 = 1$	$i = 2$	2^2
$h_3 = 0$	$i = 3$ ($\lfloor \lg n \rfloor$)	2^3



$h_i = h - i$ height of the heap rooted at level i

$n_i = 2^i$ number of nodes at level i

Running Time of BUILD MAX HEAP

$$T(n)=\sum_{i=0}^{h} n_i h_i$$

Cost of HEAPIFY at level i $*$ number of nodes at that level

$$=\sum_{i=0}^{h} 2^i (h-i)$$

Replace the values of n_i and h_i computed before

$$=\sum_{i=0}^{h} \frac{h-i}{2^{h-i}} 2^h$$

Multiply by 2^h both at the nominator and denominator and write 2^i as $\frac{1}{2^{-i}}$

$$=2^h \sum_{k=0}^{h} \frac{k}{2^k}$$

Change variables: k = h - i

$$\le n \sum_{k=0}^{\infty} \frac{k}{2^k}$$

The sum above is smaller than the sum of all elements to ∞ and h = lgn

$$=O(n)$$

The sum above is smaller than 2

Running time of BUILD-MAX-HEAP: $T(n) = O(n)$

- 1 node at height h : 2^0 nodes at ht. h − 0
- 2 nodes at height h – 1 : 2^1 nodes at height h – 1
- 4 nodes at height h – 2 : 2^2 nodes at height h - 2
- 2^i nodes at height h - i
- Sum of node heights = $\sum 2^i (h - i)$ with i going from 0 to h

- $S = h + 2(h-1) + 4(h-2) + 8(h-3) + 16(h-4) + \ldots\ldots\ldots + 2^{h-1}$ (1)
- $S = h + 2h - 2 + 4h - 8 + 8h - 24 + 16h - 64 + \ldots\ldots\ldots + 2^{h-1}$ (1)

- $S = h + 2h - 2 + 4h - 8 + 8h - 24 + 16h - 64 + \ldots\ldots.. + 2^{h-1}$ (1)
- $2S = \cancel{2h} + \cancel{4h} - 4 + \cancel{8h} - 16 + 16h - 48 + 32h - 4) + \ldots\ldots. + 2^{h}$ (1) .
- $S = h + \cancel{2h} - 2 + \cancel{4h} - 8 + \cancel{8h} - 24 + 16h - 64 + \ldots\ldots.. + 2^{h-1}$ (1)

- Subtract
- $S = -h + 2 + 4 + 8 + 16 + \ldots\ldots\ldots + 2^{h-1} + 2^{h}$.
- Add 1 and subtract 1
- $S = 1 + 2 + 4 + 8 + 16 + \ldots\ldots\ldots + 2^{h-1} + 2^{h} - 1 - h$
- $S = 2^{h+1}$ – something
- Which is O(n)

Deleting K items from a random array

Building a heap from random array : $O(n)$

Deletion of one item: $O(\log n)$

Deletion of K items: $O(K \log n)$

Total time: $O(n + K \log n)$

Compare with Sorting : $O(n \log n)$

Compare with direct search: $O(K n)$

Deleting K items from a random array

Consider example with : n = 16,000 k = 100

$\log n = 16.1000$
$= 2^4 \cdot 2^{10}$
$= 2^{14}$
14

Building a heap from random array : O(n) = 16,000

16000 + 1400

Deletion using Heap tree: n + K log n = 17,400

Compare with Sorting : O(n log n) = 224,000

Compare with direct search: O(K n) = 1600, 000

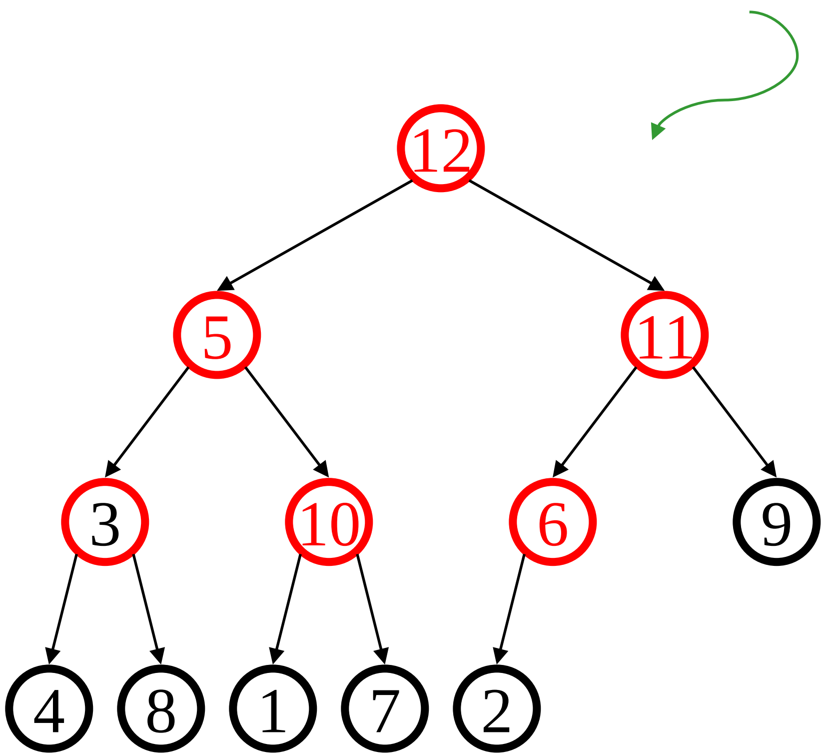
Heapifying as MinHeap

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

Store the elements *randomly* starting from position 1 of an array

BuildHeap

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



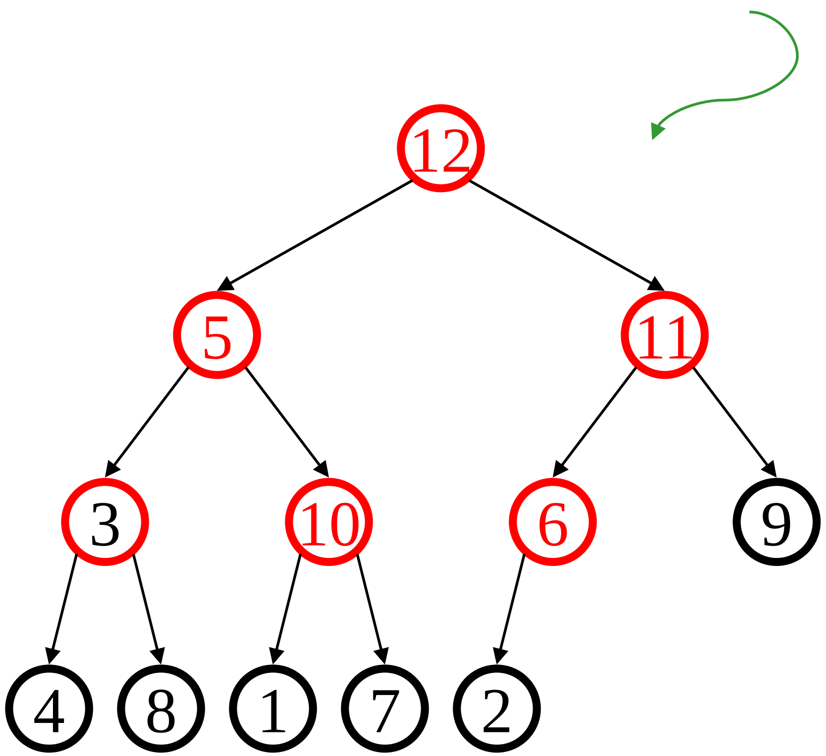
Heapify

- How do we heapify a given array?
  - Heap structure is already taken care of
  - Fix the heap order
  - Start from node at position n/2

BuildHeap: Floyd's Method

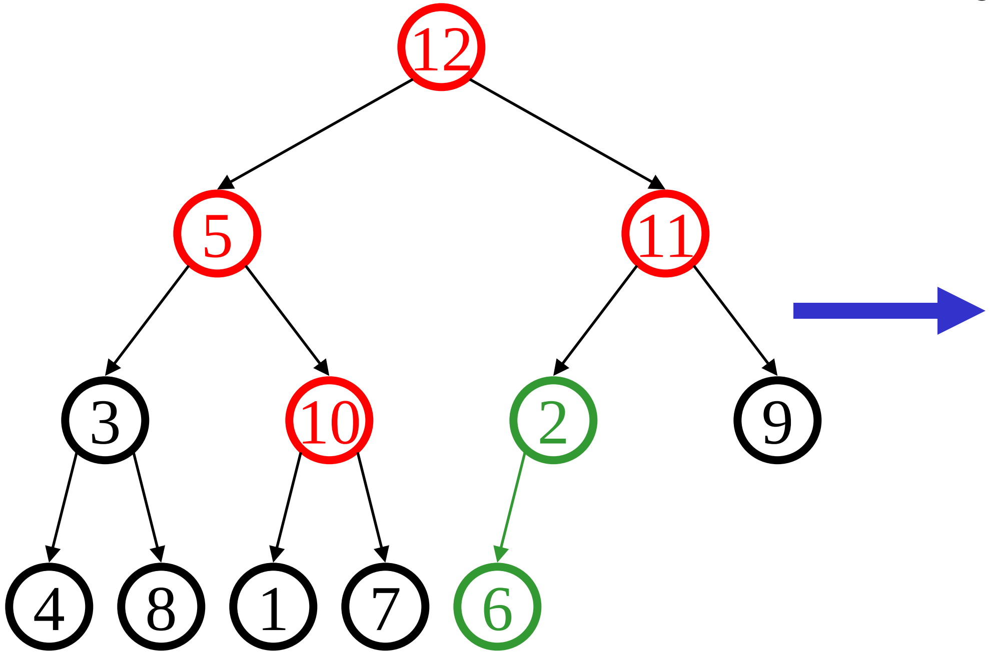
12	5	11	3	10	6	9	4	8	1	7	2

Pretend it's a heap and fix the heap-order property!



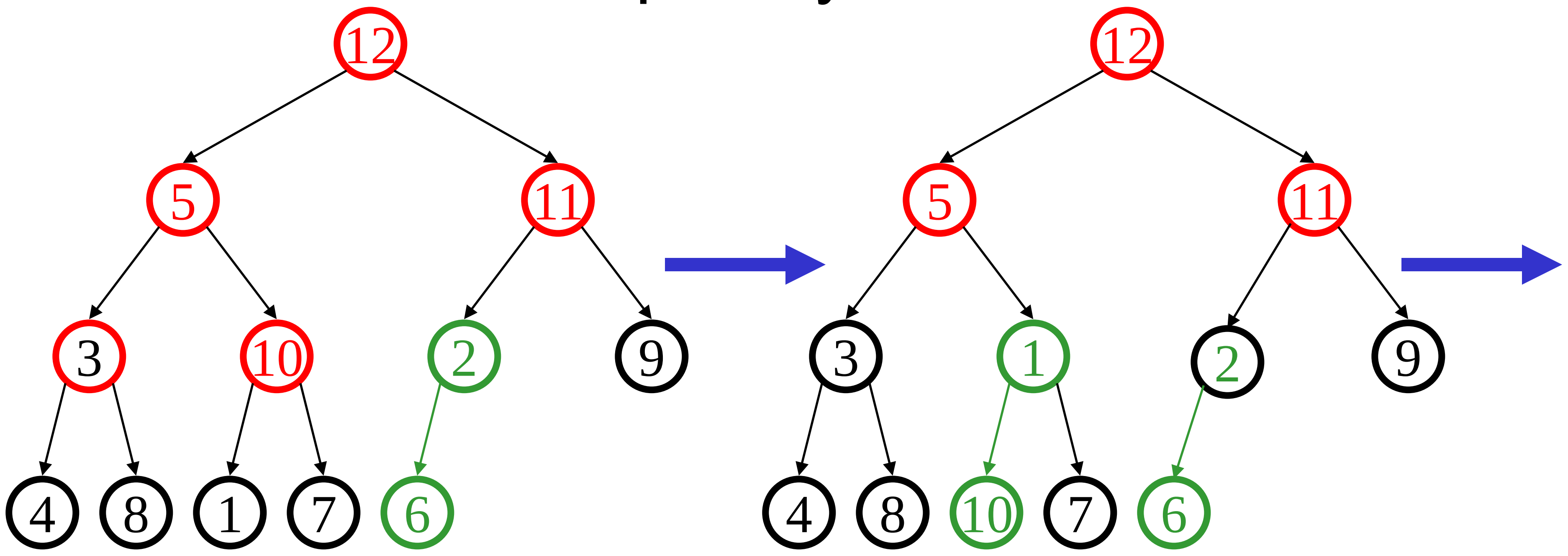
- There are 12 elements
- Start from n/2 that is 6^{th} element
- In this example it is element 6 itself
- This violates heap order property
- Fix it

BuildHeap: Floyd’s Method

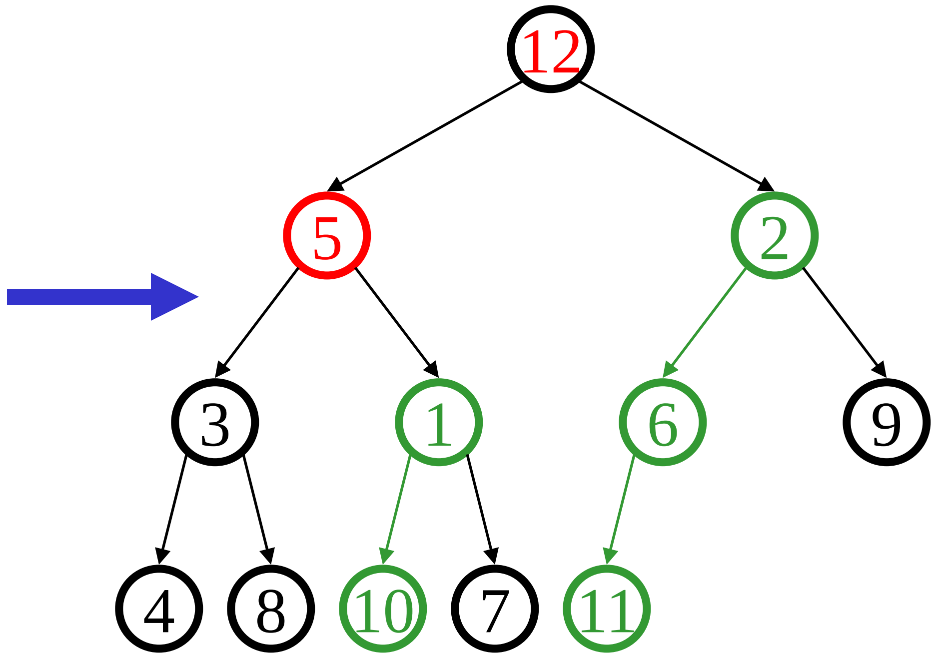


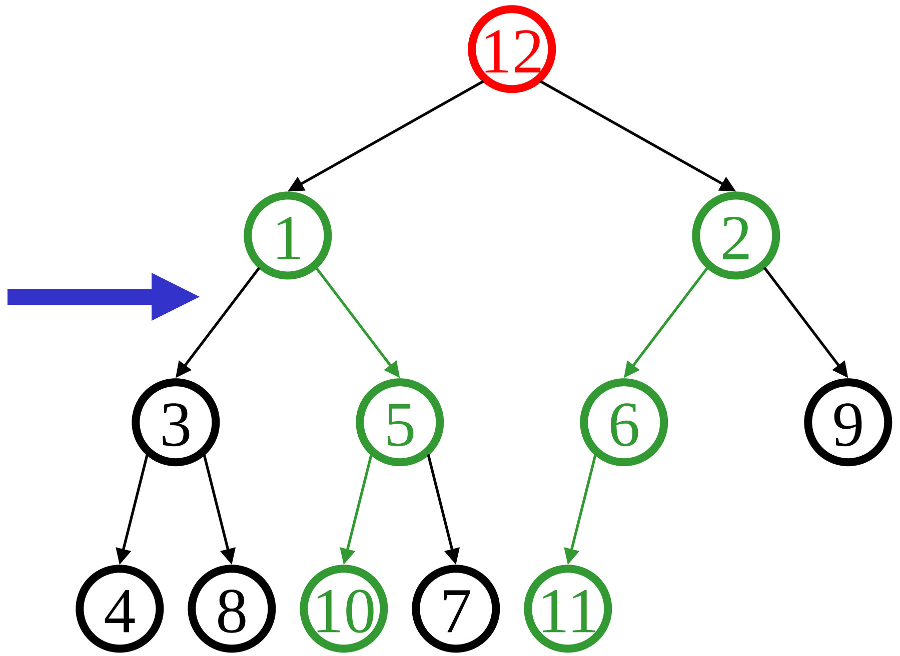
1 change

BuildHeap: Floyd's Method

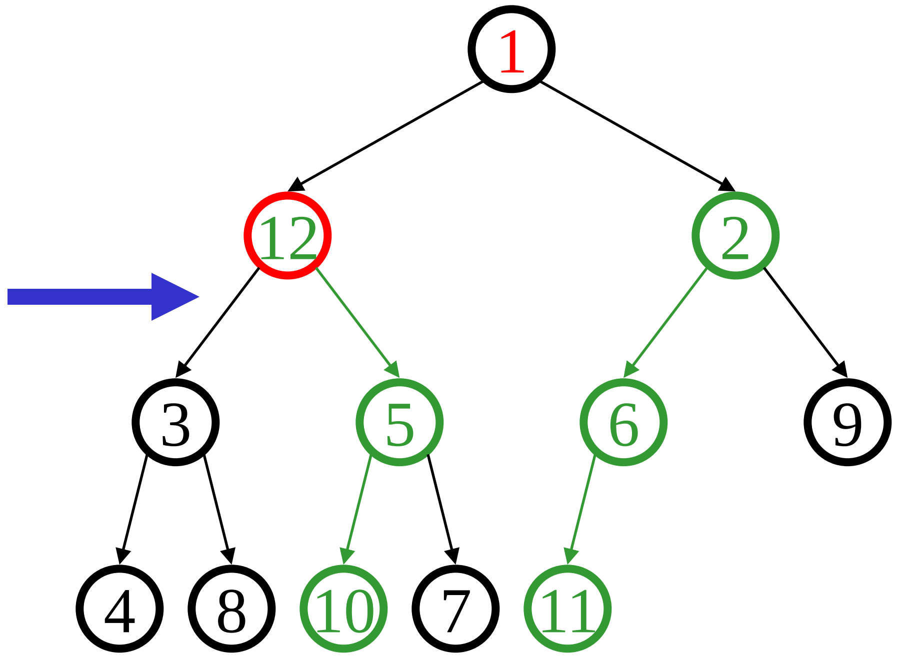


2 changed

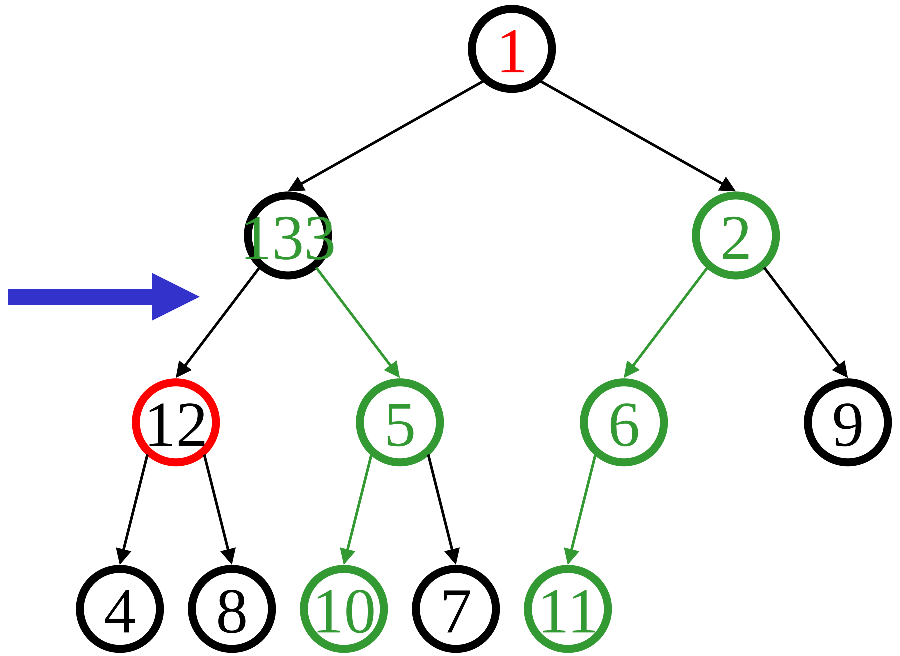

12
5
2
3
1
6
9
4
8
10
7
11


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

3 changed

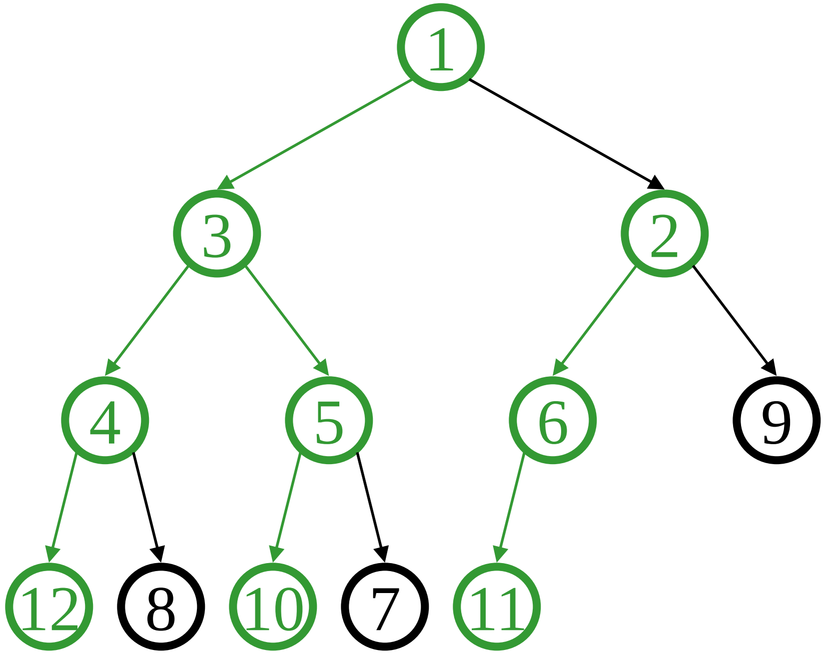


4 changed



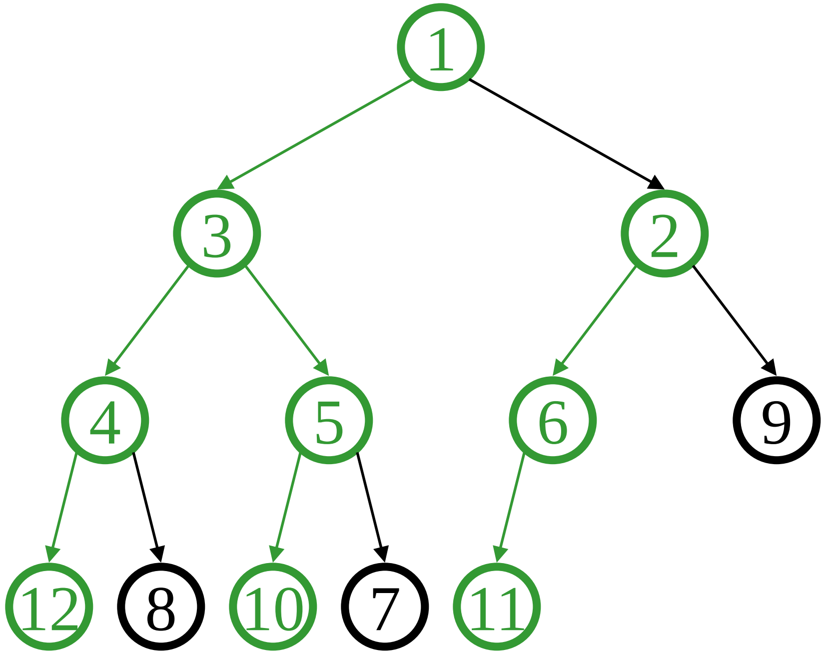
Finally…

5 changed



runtime:

Finally…



runtime: O(n)

Heapify time complexity

- First of all note maximum of N/2 elements need to be changed
- Out of that some are already in correct position
- Others take 1,2,3,.. logN operations to correct the Heap
- Total time is O(N).

kth smallest element

- Time complexity for kth smallest element
- O(N) to build heap
- (k log N) to delete k items from the heap
- total : (N+ k log N)
- Let N= 16,000,   k = 50
- using heap : 16,000 +50 (14) =16,700
- Using any type of sorting: (16000) (log 16000)= (16000) 14 = 2,24,000
- k = 8

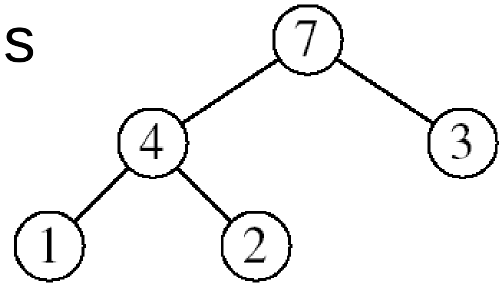
n + k logn

- using heap : 16,000 +8 (14)
- Using any type of sorting: (16000) (log 16000)= (16000) 14 = 2,24,000
- Using direct search ~~128,000~~ 80 90 00

HEAP SORT

Heapsort

- Goal:
  - Sort an array using heap representations



- Idea:
  - Build a **max-heap** from the array
  - The largest element will be at root of the tree.
  - Delete the root and swap with the last element of the array.
  - “Discard” this last node by decreasing the heap size
  - Call MAX-HEAPIFY on the new root
  - Repeat this process until only one node remains

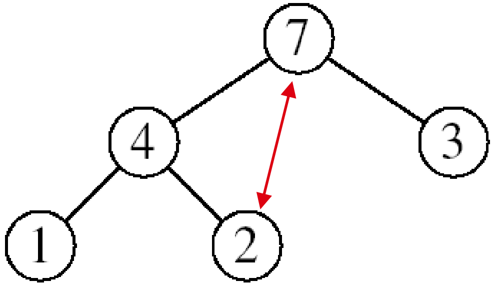
Heapsort

- …
  - Repeat this for all elements on heap tree.
  -
  - Time requirements: O(n log n)
  - Also it does not need an extra array of size n
  - (as needed by mergesort)

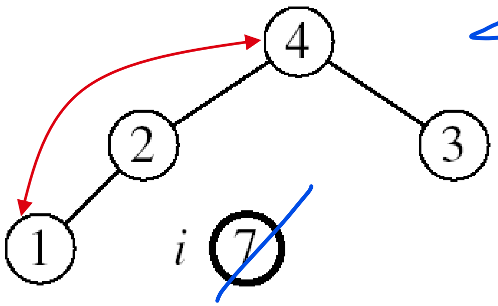
Example: A=[7, 4, 3, 1, 2]

[4, 3, 1, 2, 7]

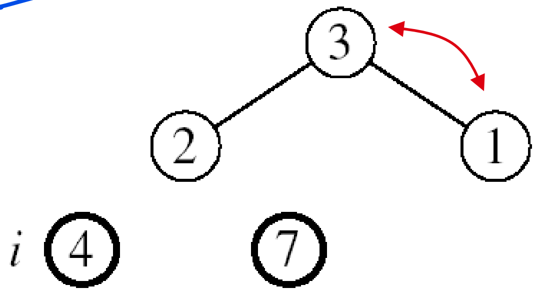
[3, 1, 2, 4, 7]



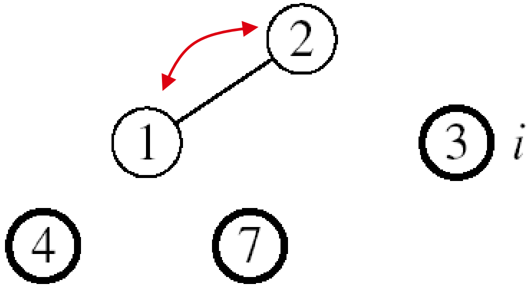
MAX-HEAPIFY(A, 1, 4)



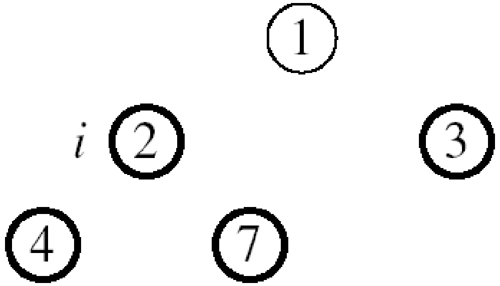
MAX-HEAPIFY(A, 1, 3)



MAX-HEAPIFY(A, 1, 2)



MAX-HEAPIFY(A, 1, 1)



A					
A	1	2	3	4	7

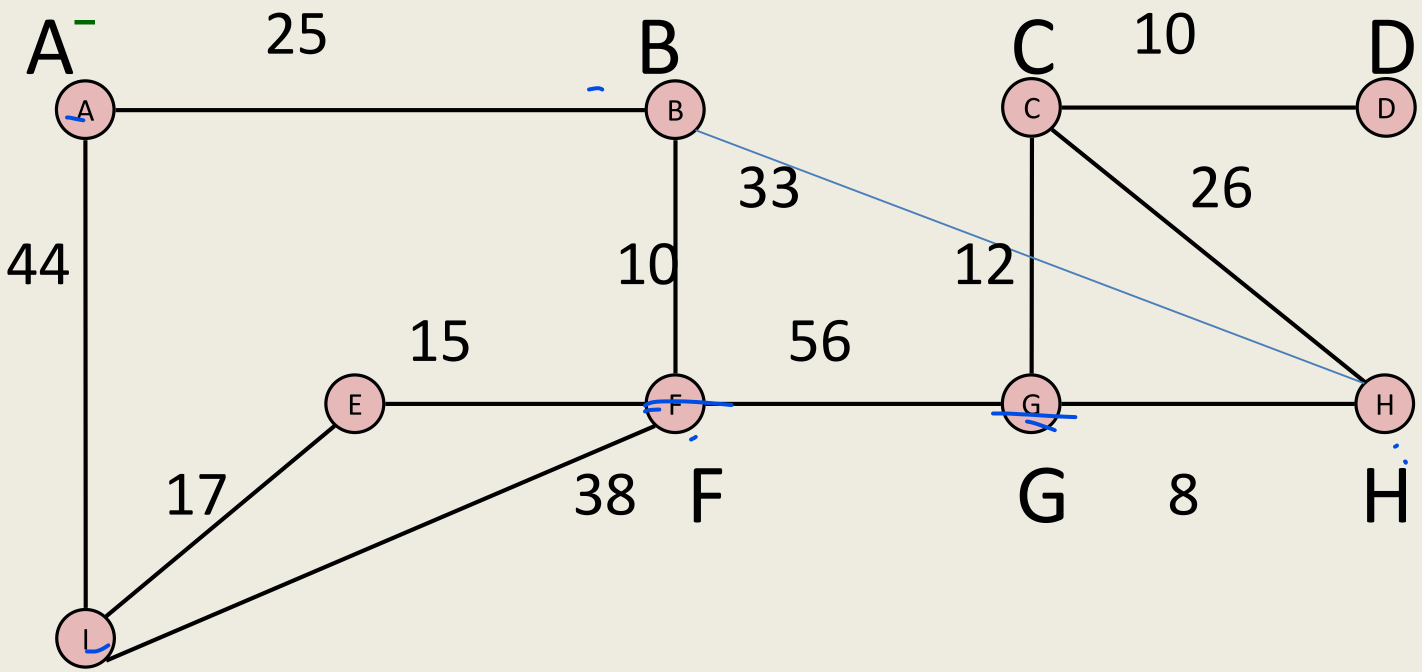
Graph Data Structure

Finding shortest route between cities

Given a network of **roads** connecting various cities,

compute the <u>shortest route</u> between any two **cities**.

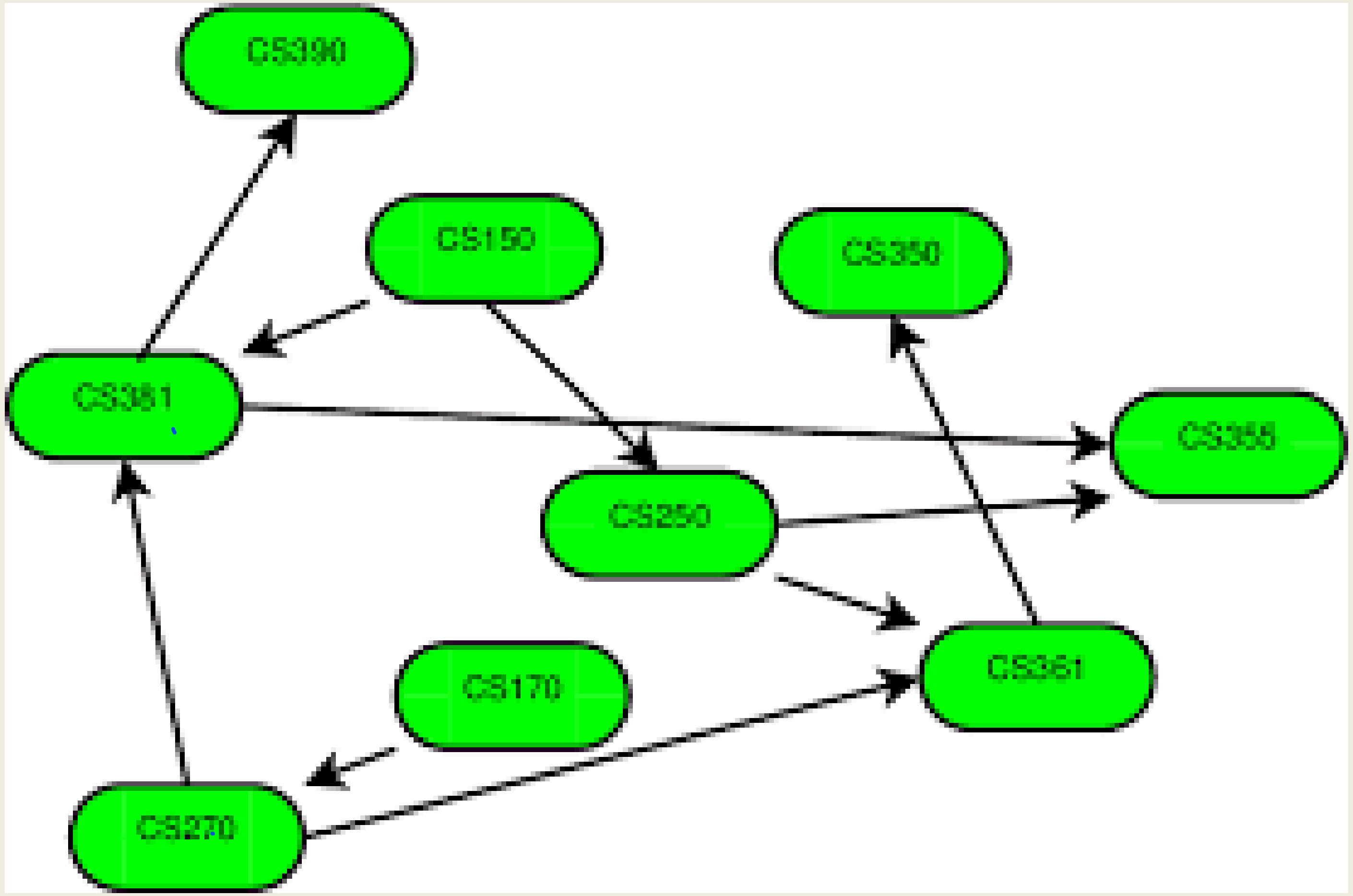
Find shortest path from A to H



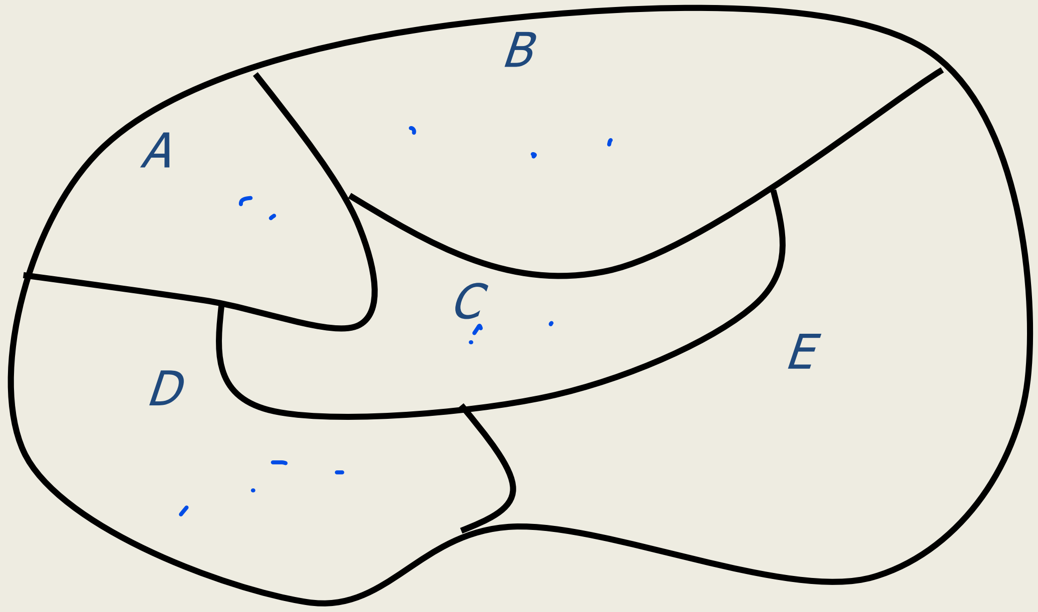
A, B, C, ... , H, I each is called a VERTEX

Lines AB, GH etc. are called EDGES

Directed Graph: Each edge has got a direction. Course Ordering



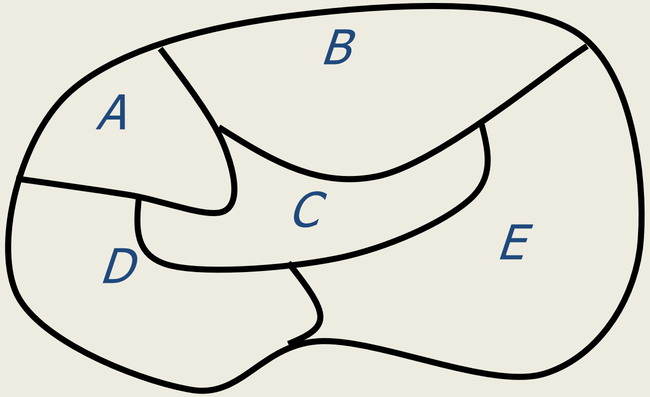
Map coloring



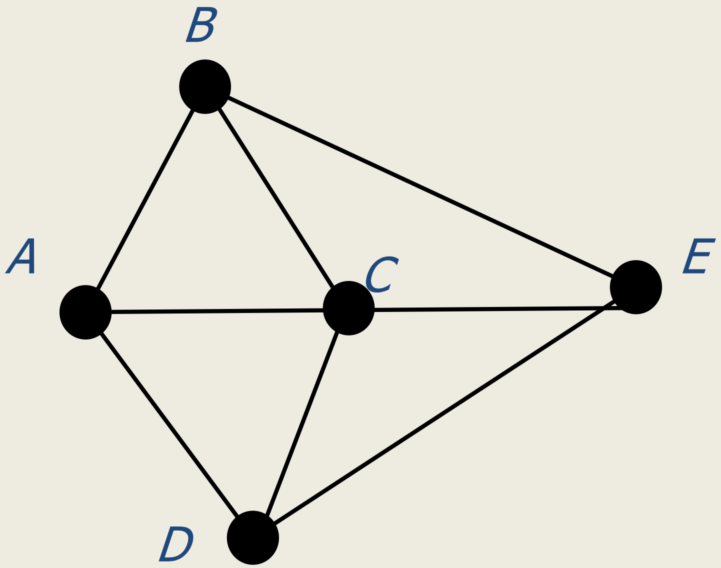
Map Coloring

- Map coloring is a graph problem
- each region represented by a vertex
- neighboring regions represented by an edge
- Two regions with a common border are assigned different colors.
- We want to use as few colors as possible, instead of just assigning every region its own color.

Corresponding Graph



How many colors needed?

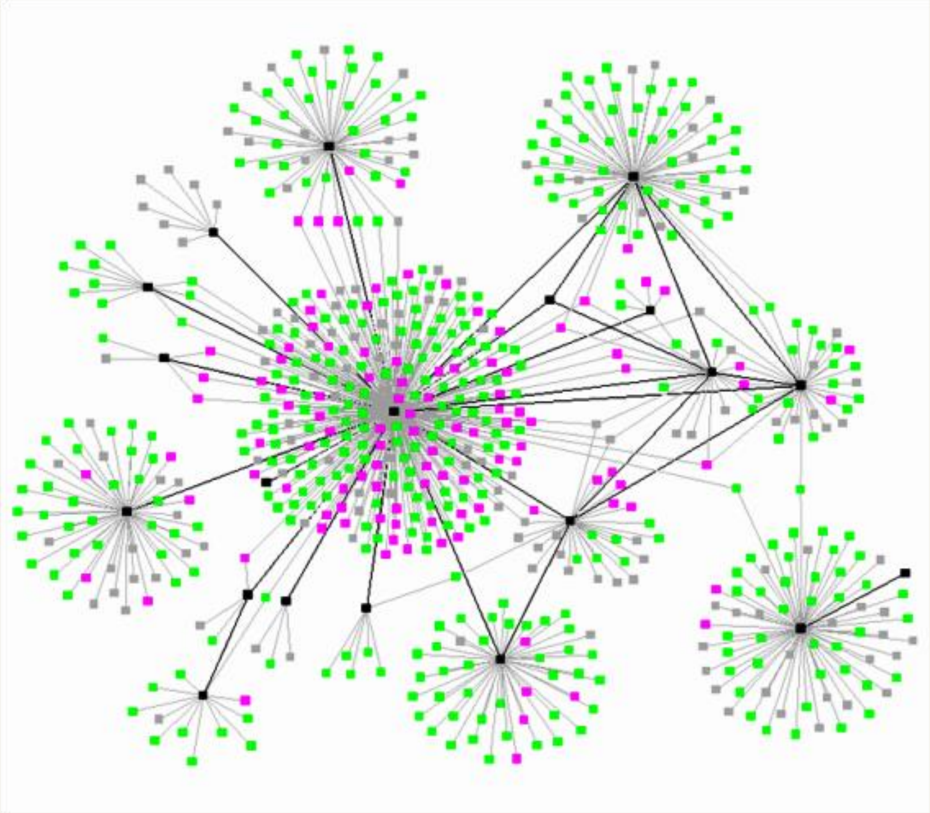


Map with 4 colors

Other problems of similar nature

A social network or world wide web (WWW)

Can we make some useful observations about such networks ?

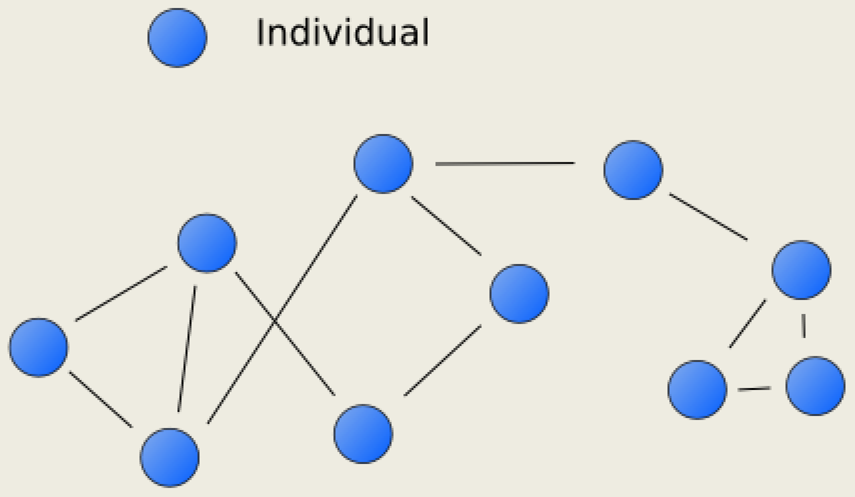


diameter

degree distribution

Social Network Analysis

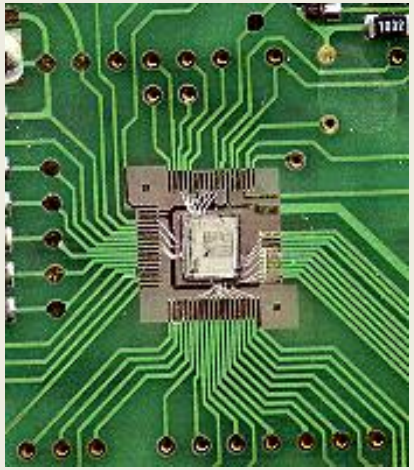
- mapping and measuring of relationships and flows between people, groups, organizations, computers……
- The nodes are the people and groups while the links show relationships or flows between the nodes.



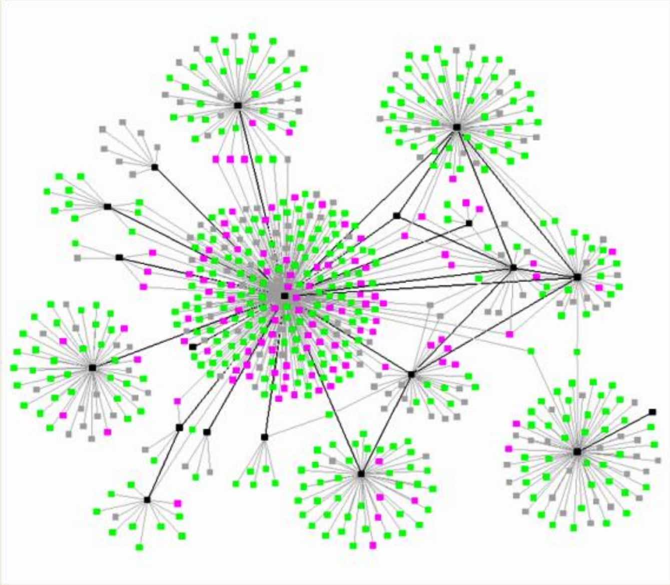
Common issues in all such environments

Interconnected nodes

I



II



III

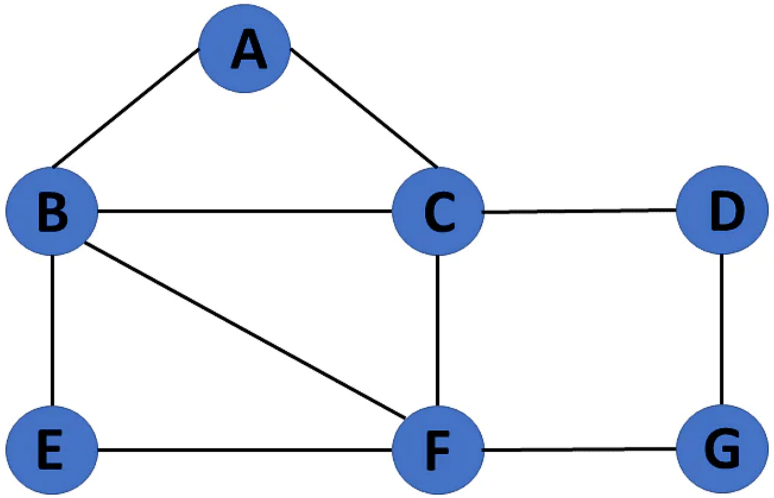
Graph Data Structure

Definitions, notations, and terminologies

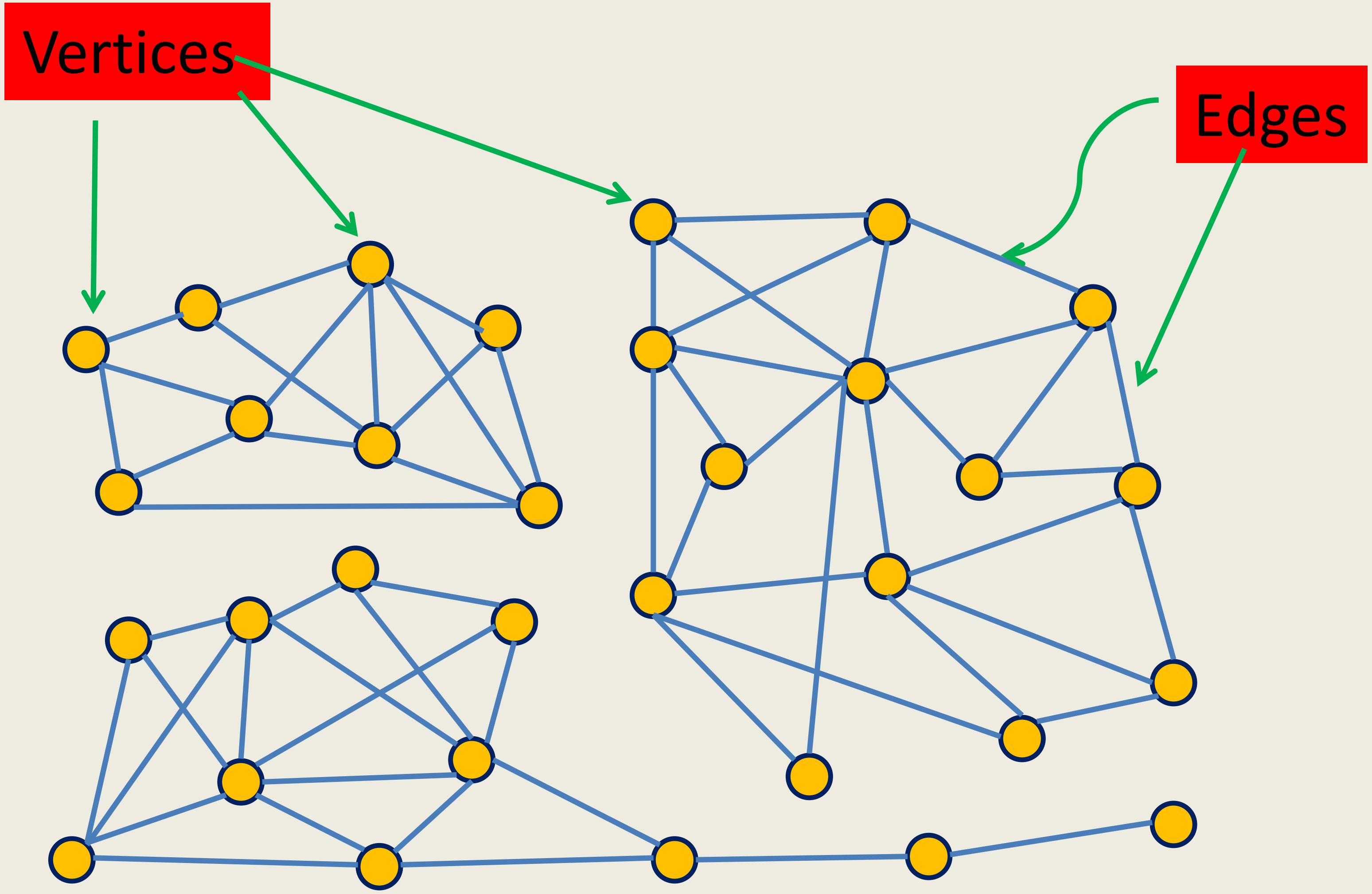
Graph Data Structure

- Graph is a structure where vertices are linked by edges
- In a road network, we can view cities as *vertices*, Nodes
-
- While distances can be represented as *weights of edges* linking the relevant vertices

- A, B, C, D,… are Vertices/ nodes
- Lines joining the nodes are the EDGES



All nodes need not be connected



Graph

A graph ***G*** is defined by two sets

- ***V*** : set of vertices
- ***E*** : set of edges

Notation:

- A graph ***G*** consisting of vertices ***V*** and edges ***E*** is denoted by

$$\boxed{G(V,E)}$$

Notations

Notations:

- **Number of Vertices** $n = |V|$
- ***Number of Edges*** $m = |E|$

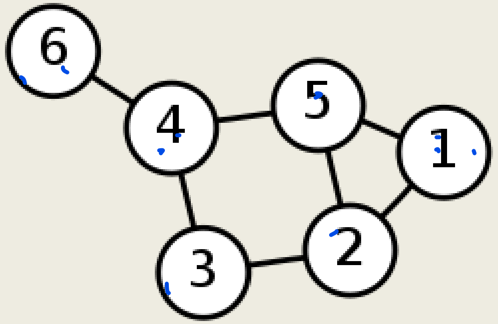
Numbering Terminology

Vertices are always numbered

$\mathbf{1, \ldots, n}$ Or $\mathbf{0, \ldots, n-1}$

Types of graphs

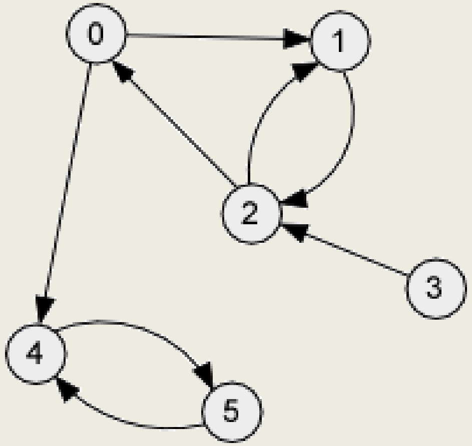
Undirected Graph



V= {1,2,3,4,5,6}

E= {(1,2), (1,5),
(2,5), (2,3),
(3,4),
(4,5), (4,6)}

Directed Graph



V= {0,1,2,3,4,5}

E= { (0,1), (0,4),
(1,2),
(2,0), (2,1),
(3,2),
(4,5),
(5,4) }

Cycle:

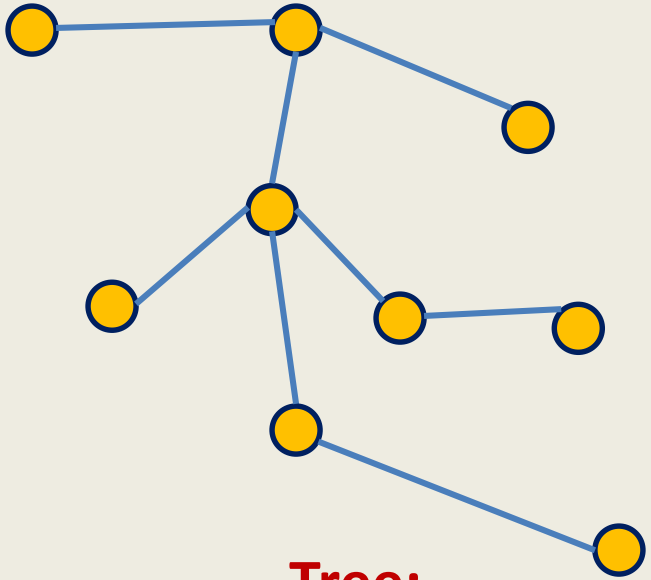
A path whose start and end vertices are same,

and no **intermediate** vertex gets repeated

tree:

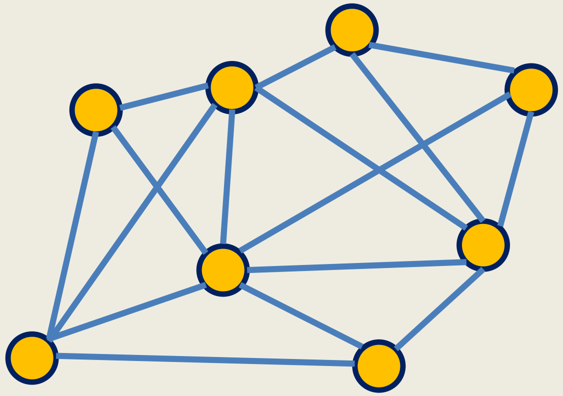
A tree is a connected graph without cycles

(**acyclic**)



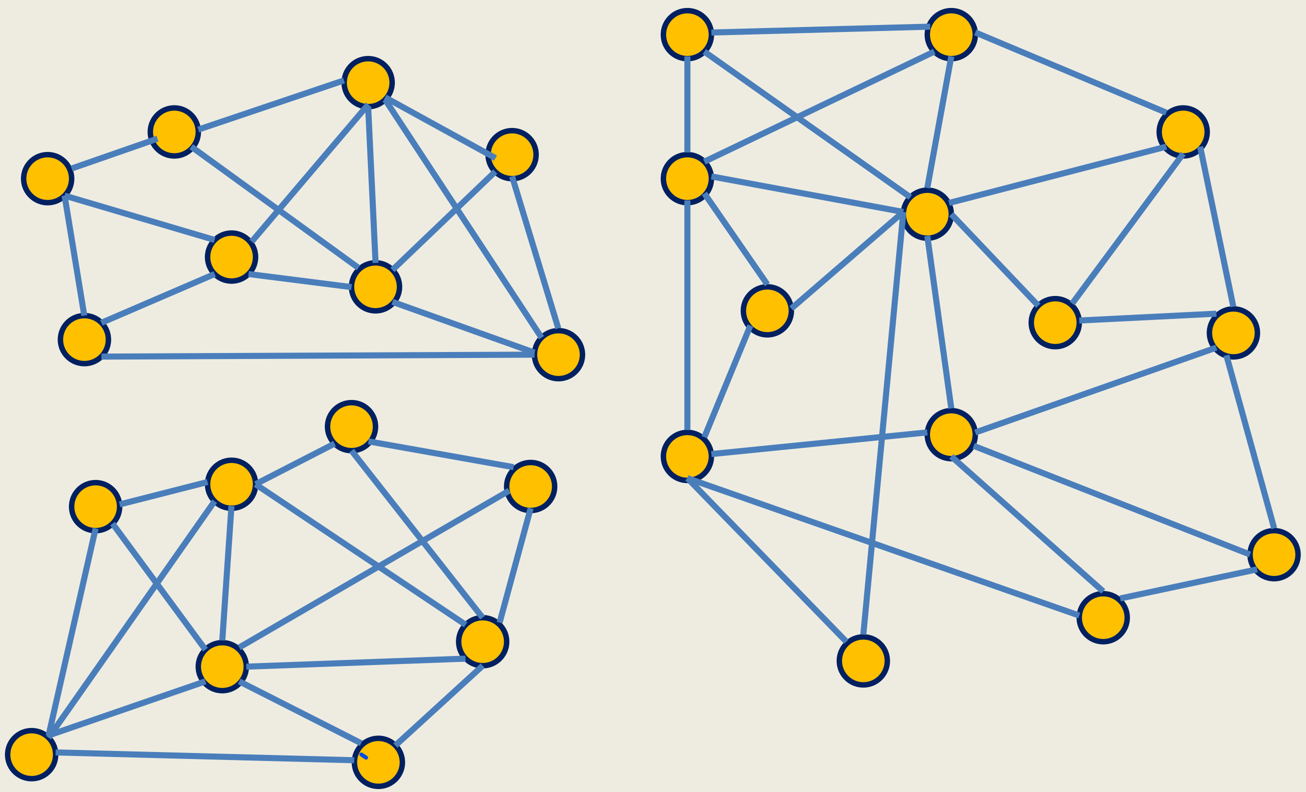
Tree:

No cycles

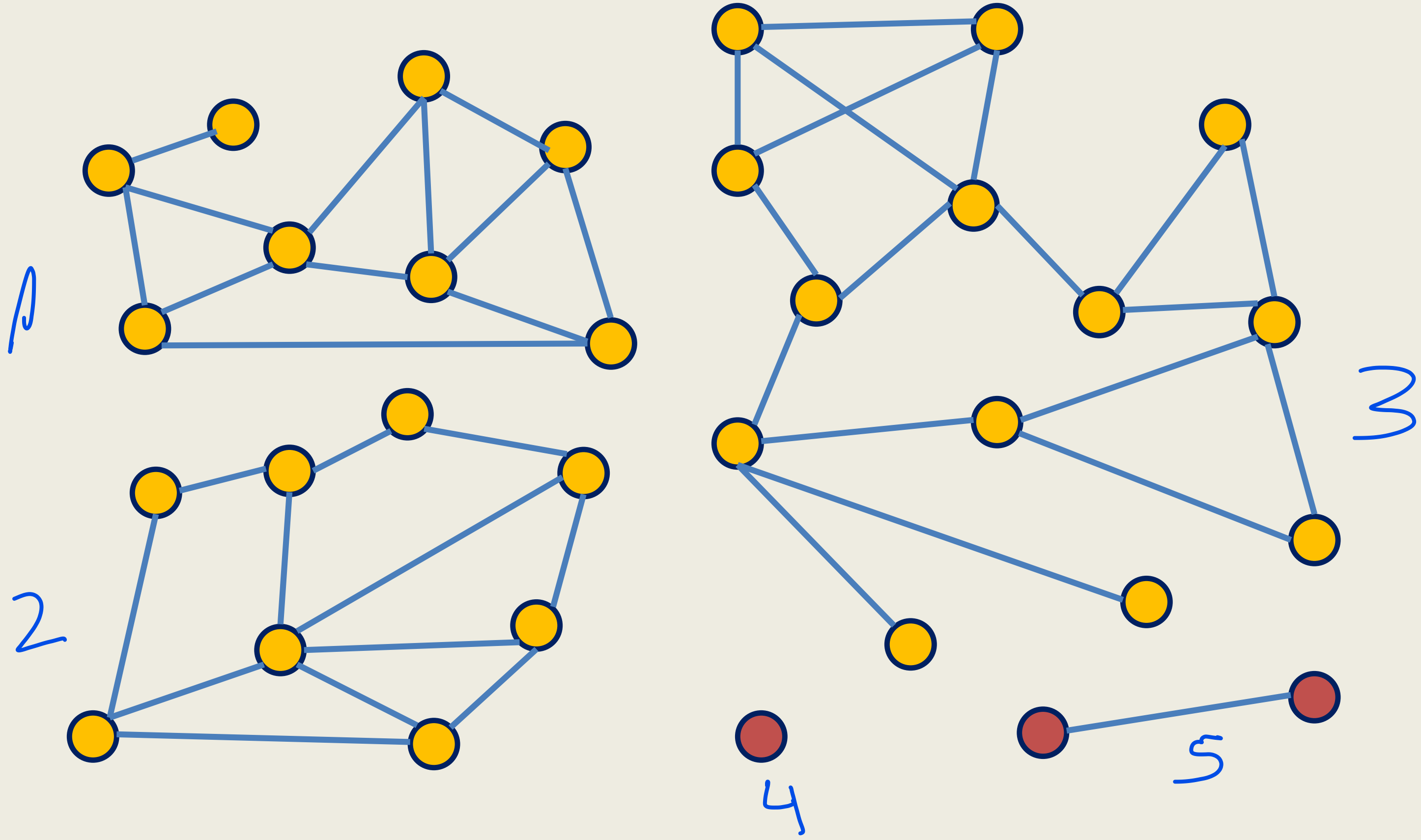


connected component

Any subset of connected vertices



A Graph with 3 Connected components



A Graph with 5 Connected components

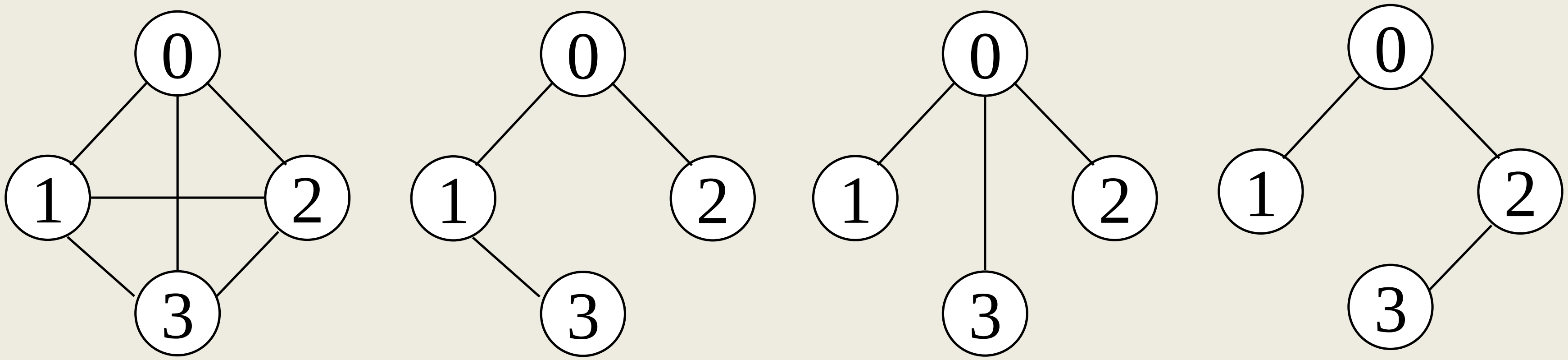
Spanning Tree

Tree formed of graph edges which connect all the vertices of the graph

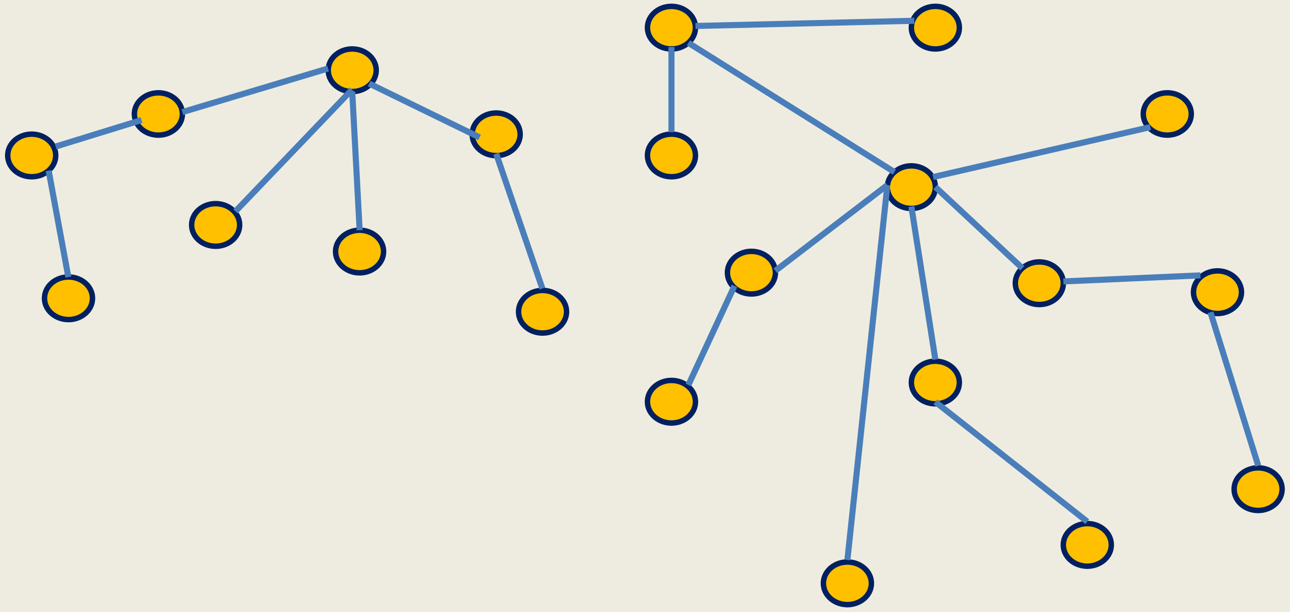
Complete Graph:

A fully connected graph : Every vertex is having an edge to all other vertices

Spanning Tree examples

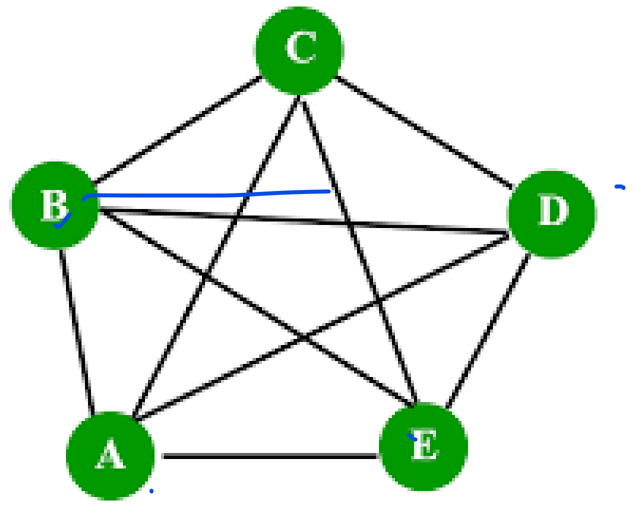


G_1 Possible spanning trees



Spanning Trees

Complete graph



Complete Graph

Graph ADT

- Are two cities connected?
- Distance from one city to all other cites
- Possible paths between two cities
- Shortest path between a pair of cities
- Cheapest possible road network to connect n cities
- How many connected road segments?
- Handling different weights on directed edges