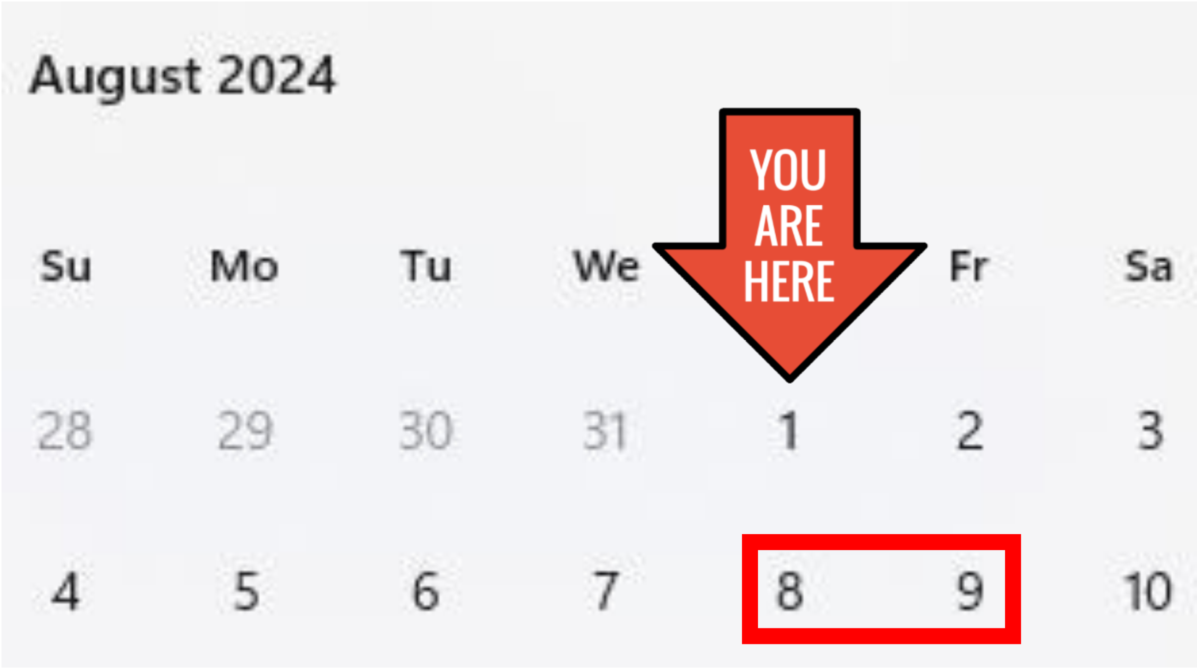
CSE 3318

Week of 07/29/2024

Instructor : Donna French

FEQ – August 8th – August 9th

- FEQ1
  - Radix and Heap
- FEQ2
  - Binary Search and Recursion and Asymptotic Notation
- FEQ3
  - Merge Sort and Quick Sort
- FEQ4
  - Dijkstra and Prim and Kruskal
- FEQ5
  - Hashing
- FEQ Extra
  - TBA on the last day of class



UTA Course Evaluations

You will receive daily emails with links to the surveys. These reminders will cease once you have completed all surveys in your dashboard.

A pop-up screen will appear any time you log into Canvas encouraging and directing you to the surveys.

This screen will disappear once you have completed the surveys. You can also find a link to the surveys under the Course Evaluations tab on the navigation menu.

Your responses are completely confidential!

Instructors will only have access to aggregated results AFTER grades are released.

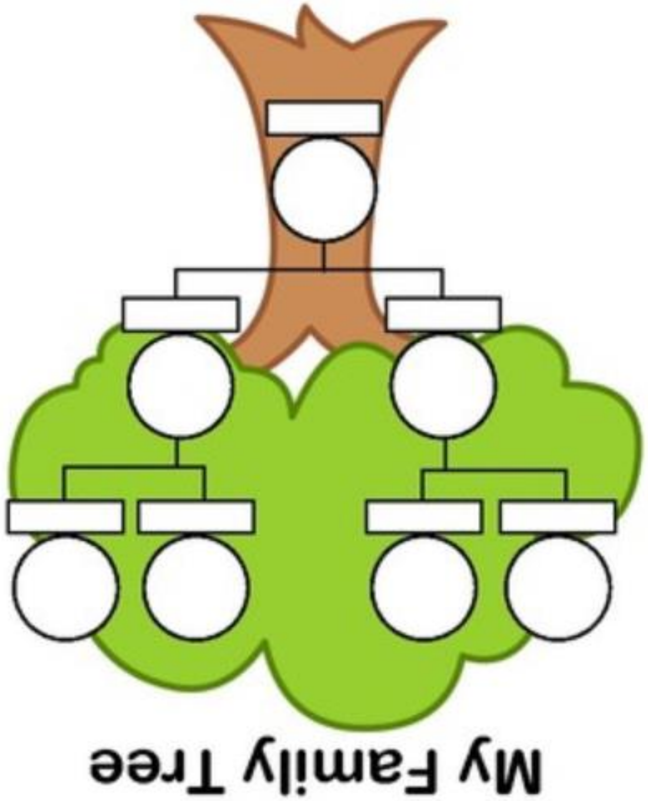
Students' information IS NOT available to the instructors ever.

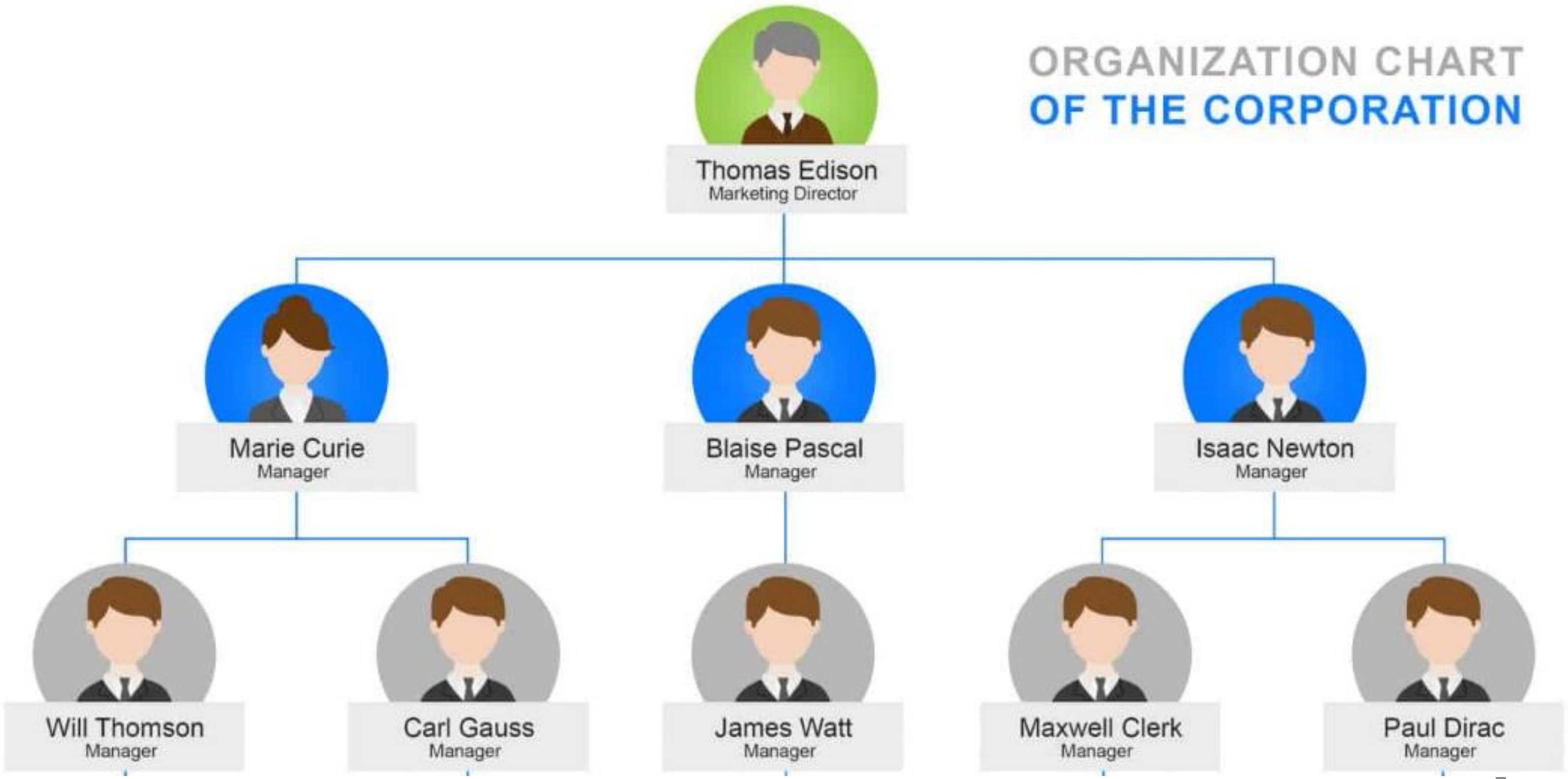
Tree Data Structure

- Linked lists, stacks, and queues are all **linear** structures
  - one node follows another
  - each node contains a pointer to the next node

- Trees are **non-linear** structures
  - more than one node can follow another
  - each node contains pointers to an arbitrary number of nodes
  - the number of nodes that follow can vary from one node to another.

- Trees organize data hierarchically instead of linearly.


My Family Tree


ORGANIZATION CHART
OF THE CORPORATION
Thomas Edison
Marketing Director
Marie Curie
Manager
Blaise Pascal
Manager
Isaac Newton
Manager
Will Thomson
Manager
Carl Gauss
Manager
James Watt
Manager
Maxwell Clerk
Manager
Paul Dirac
Manager

Binary Tree

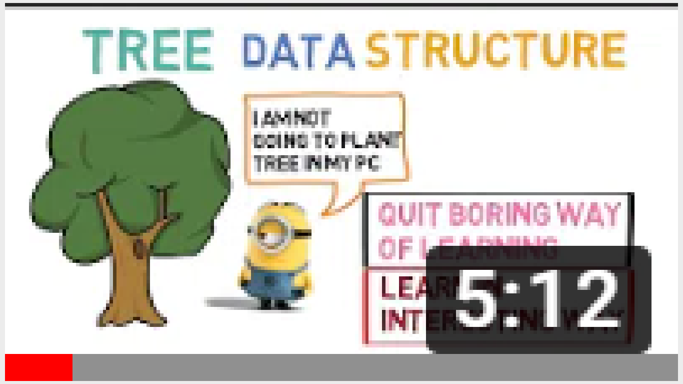
What is a binary tree?

A binary tree is a non-linear tree-like data structure consisting of nodes where each node has up to two child nodes, creating the branches of the tree.

The two children are usually called the left and right nodes.

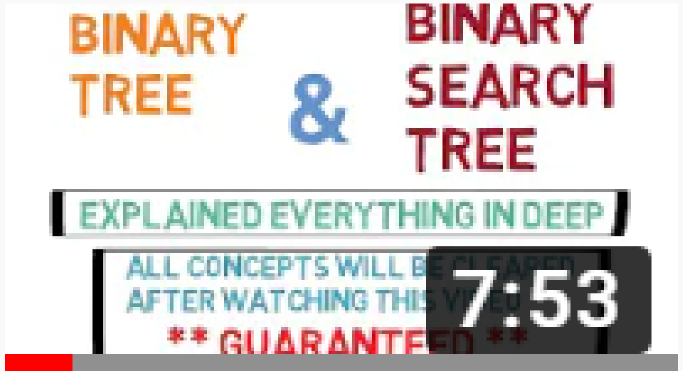
Parent nodes are nodes with children. Parent nodes can be child nodes themselves.

Binary trees are used to implement binary search trees and binary heaps. They are also often used for sorting data as in a heap sort.



Introduction to Tree Data Structure

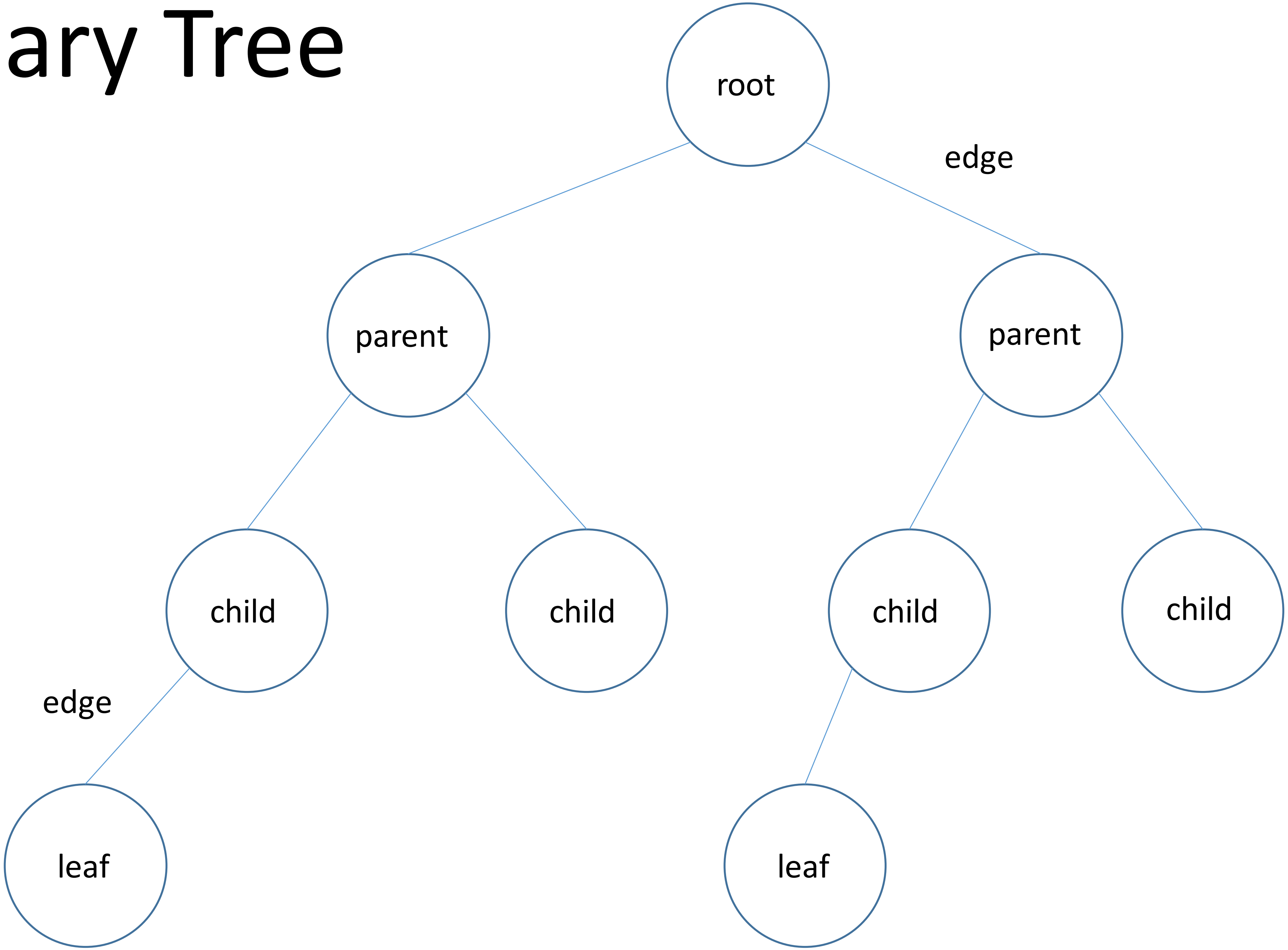
Codearchery



Binary Tree and Binary Search Tree in Data Structure

Codearchery

Binary Tree

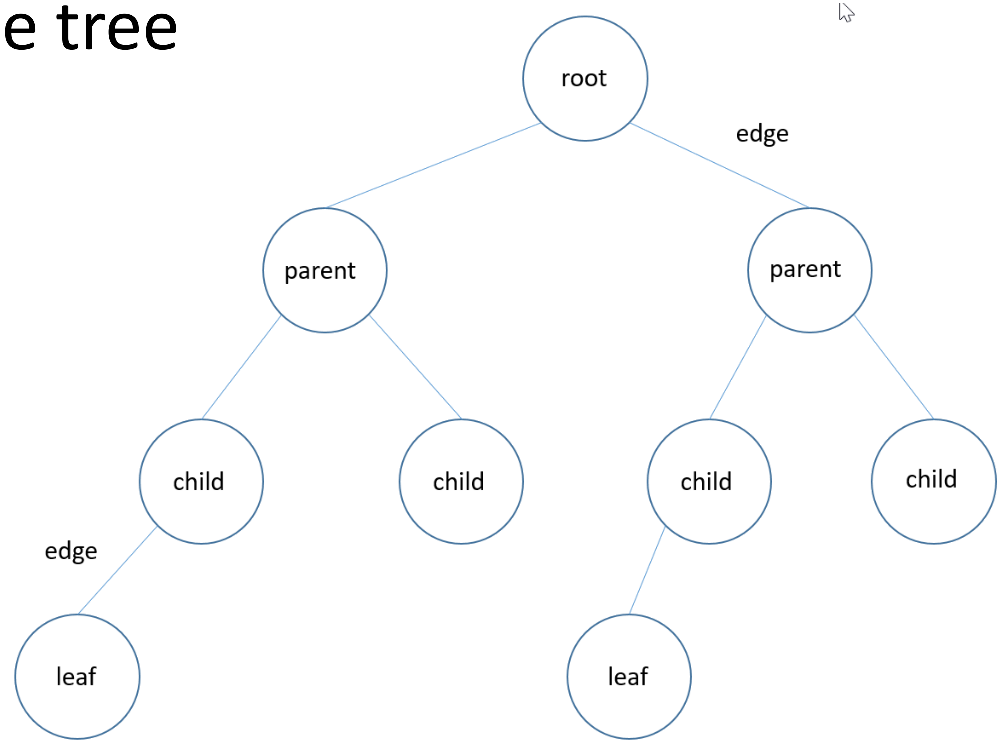


root
edge
parent
parent
child
child
child
child
edge
leaf
leaf

Binary Tree

Tree Vocabulary

topmost node	root of the tree
node directly under another node	child
node directly above another node	parent
node with no children	leaf
link between two nodes	edge
length of the path from the root	depth
length of the path from the node to the deepest leaf reachable from it	height



Binary Trees

Before we talk about heaps, let's talk about some versions of binary trees.

Perfect Binary Tree

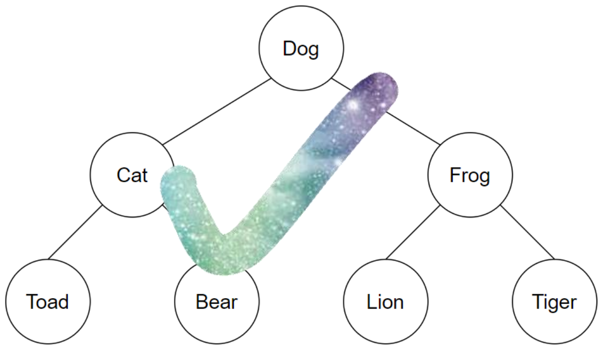
Full Binary Tree

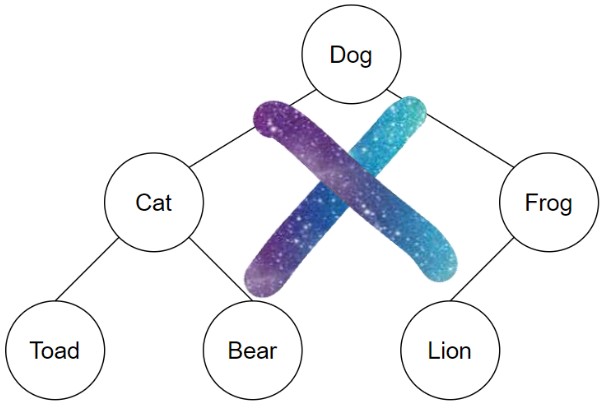
Complete Binary Tree

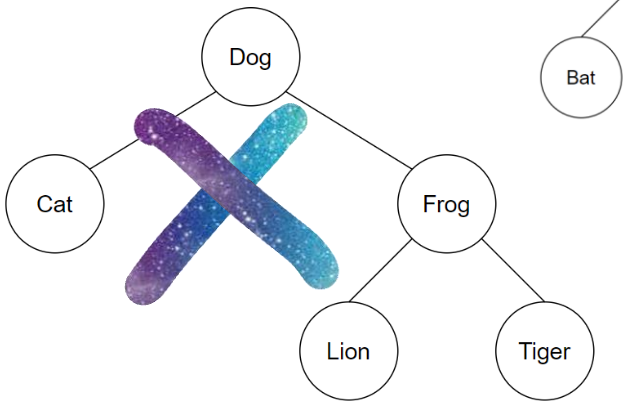
Binary Trees

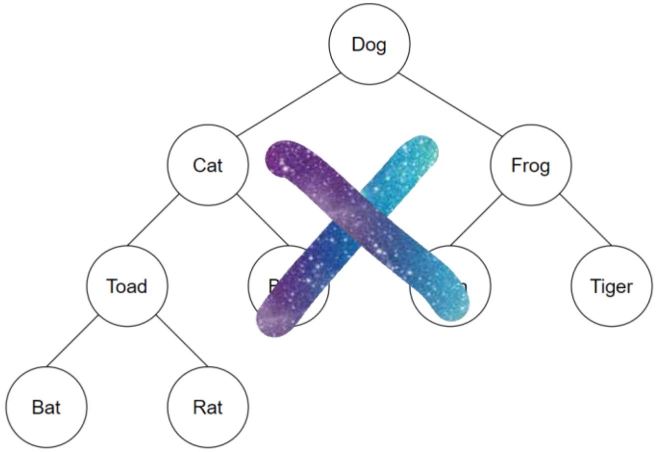
Perfect Binary Tree

A Binary tree is a **Perfect Binary Tree** if all the internal nodes have two children and all leaf nodes are at the same level.







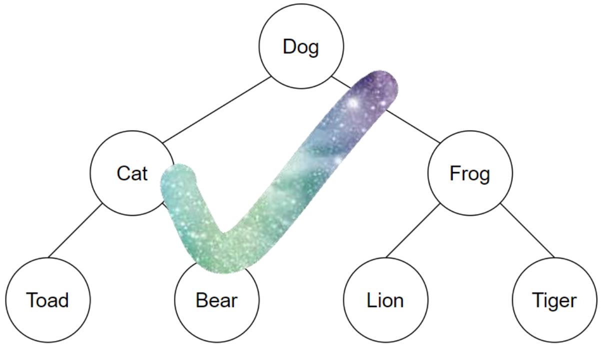


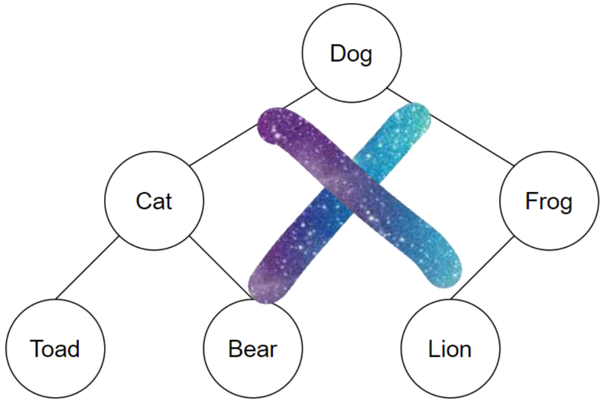
Binary Trees

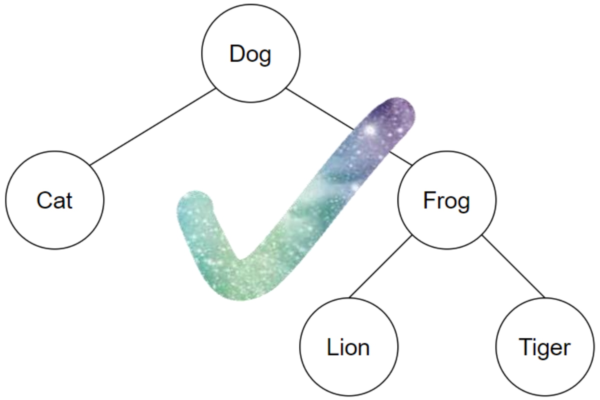
Full Binary Tree

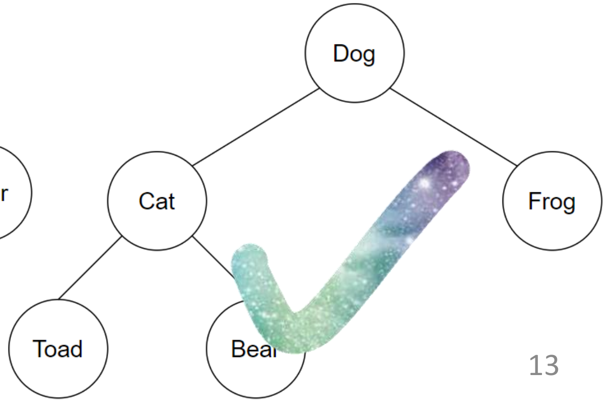
A Binary Tree is a **full binary tree** if every node has 0 or 2 children.

We can also say a full binary tree is a binary tree in which all nodes except leaf nodes have two children.





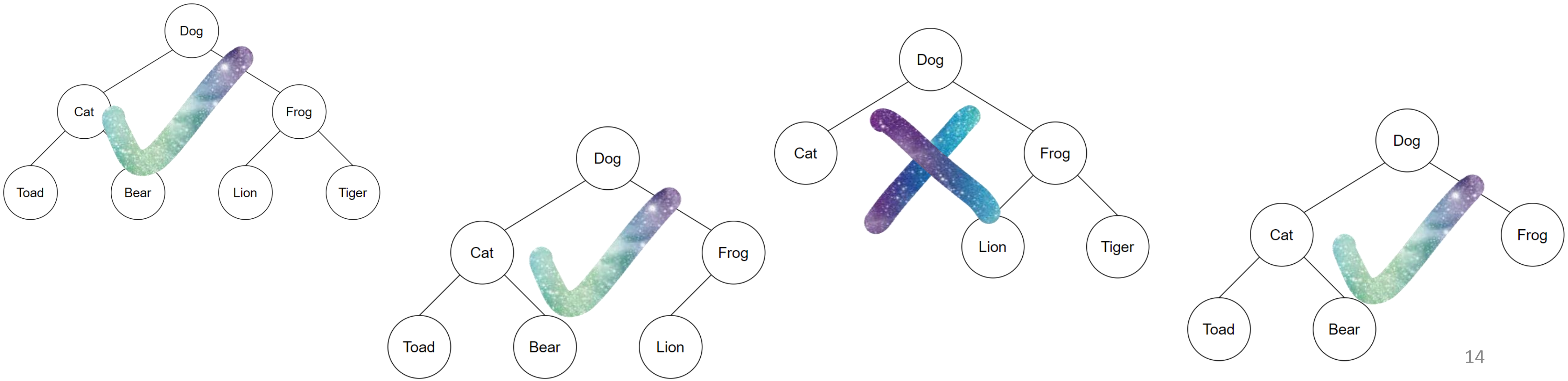




Binary Trees

Complete Binary Tree

A Binary Tree is a **Complete Binary Tree** if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible



Binary Trees

A Binary Tree is a **complete binary tree** if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible

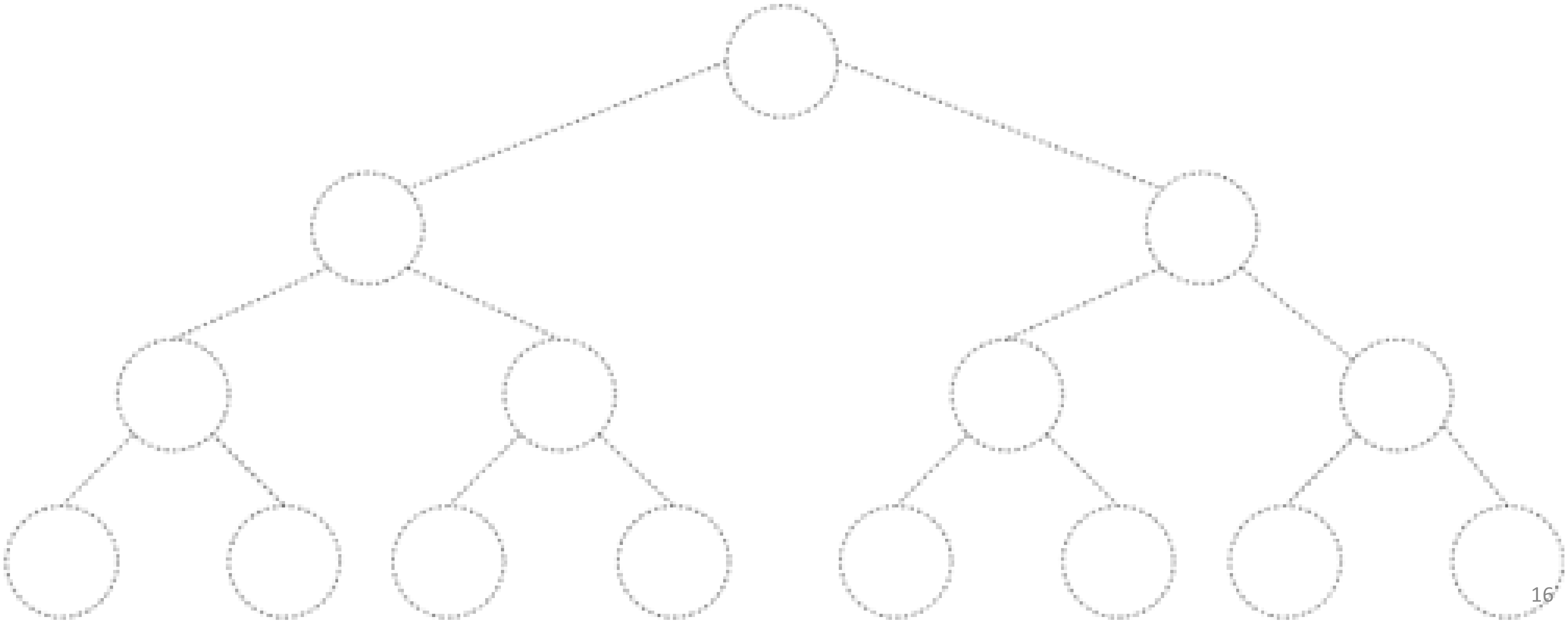
A Binary Tree is a **full binary tree** if every node has 0 or 2 children.

A Binary tree is a **Perfect Binary Tree** if all the internal nodes have two children and all leaf nodes are at the same level.

Complete binary tree if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible

Full binary tree if every node has 0 or 2 children.

Perfect Binary Tree if all the internal nodes have two children and all leaf nodes are at the same level.



Binary Heap

A binary **heap** is a complete binary tree that satisfies the heap property.

Complete Binary Tree - all the levels are completely filled except possibly the last level and the last level has all keys as left as possible

Heap is one of the most efficient implementations of an abstract data type called a priority queue.

A priority queue is an abstract data type similar to regular queue data structure in which each element additionally has a "priority" associated with it.

In a priority queue, an element with high priority is served before an element with low priority.

Priority Queue

Imagine you are an Air Traffic Controller (ATC) working in the control tower of an airport.

Aircraft X is ready to land and the runway is free, so you give them permission to land.

Aircraft Y radios in that they will be arriving at the airport within the next 5 minutes.

You also know that the runway will be occupied/unavailable for any other plane for at least 15 minutes while a plane is landing.

You tell Aircraft Y to go into a holding pattern when they arrive because the runway will be occupied for at least another 10 minutes after their arrival.

You have Aircraft X and Y in a queue.

Priority Queue

If another aircraft arrives, they will be put in the queue behind Aircraft Y and told to maintain a holding pattern while waiting their turn.

Five minutes later, Aircraft Z radios in that they are 7 minutes away from the airport but critically low on fuel due to going around a big storm system.

They can't land immediately because the runway will still be occupied by Aircraft X.

So you put them in the queue because they have to wait on Aircraft X (which is already landing) but they have priority over Aircraft Y even though Aircraft Y arrived first.

Binary Heap

A binary heap is a binary tree with the following properties…

1. Complete tree

2. Heap property which means the binary heap is either a

   Min Heap
   or
   Max Heap

Binary Heap

Min Heap

The value at root of the tree must be smallest value present in the Binary Heap.

This property must be recursively true for all nodes in the binary tree.

Any parent node should be smaller than its child nodes.

Binary Heap

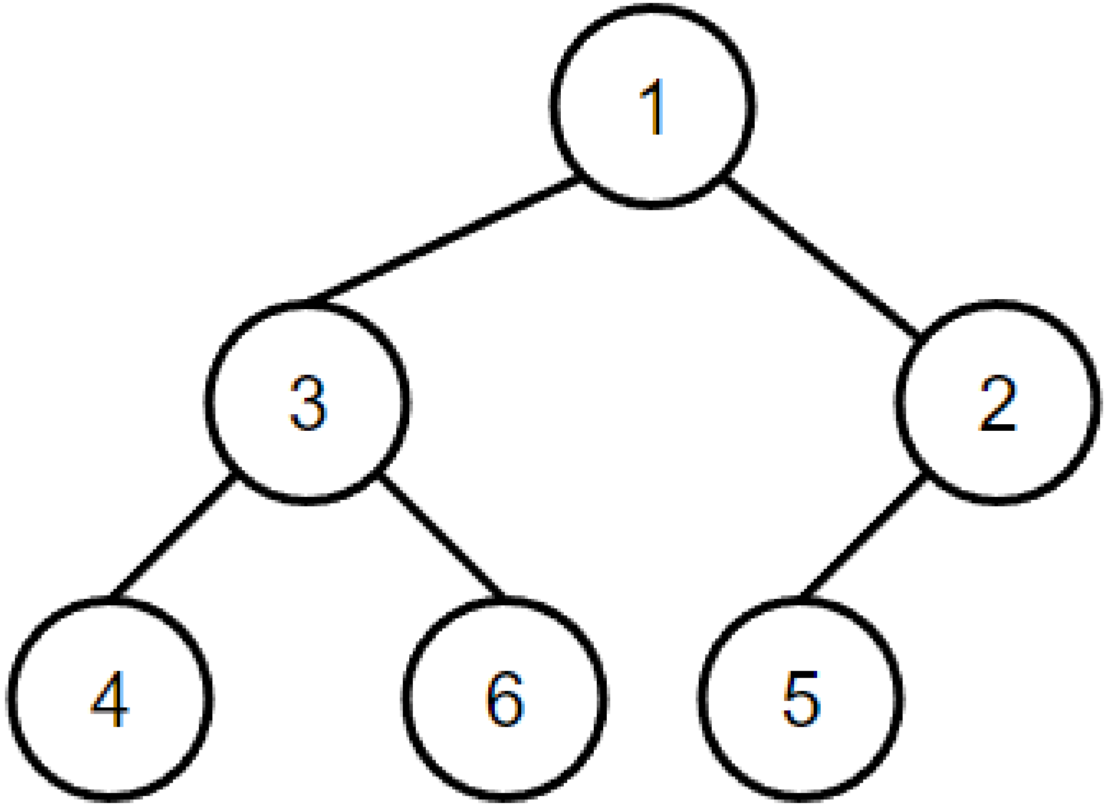
Is this a Min Heap?

1. Is it a complete tree?

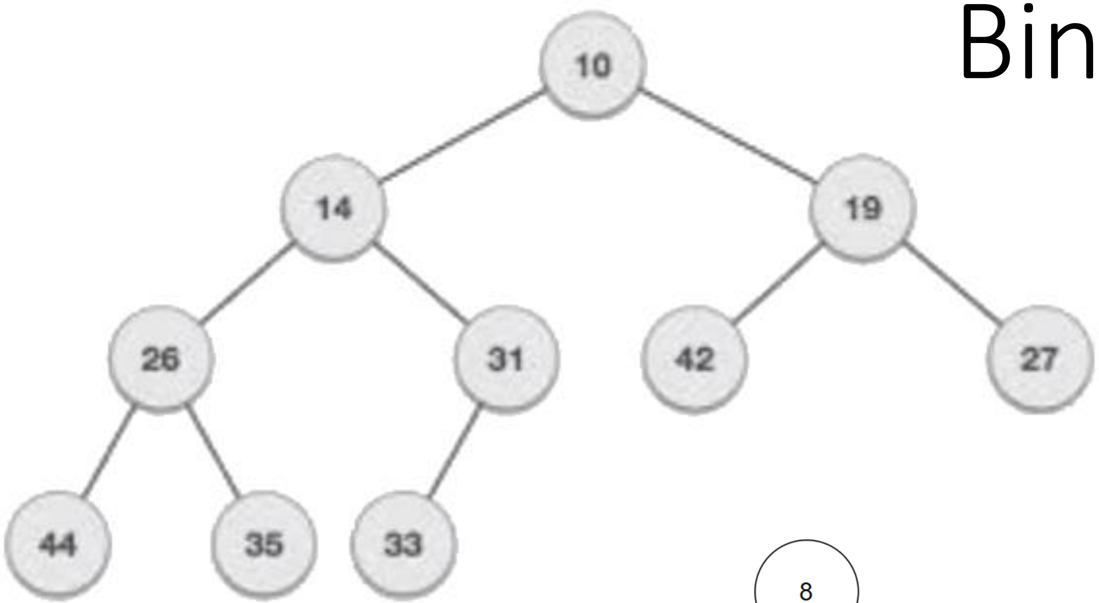
    all the levels are completely filled except possibly the last level and the last level has all keys as left as possible

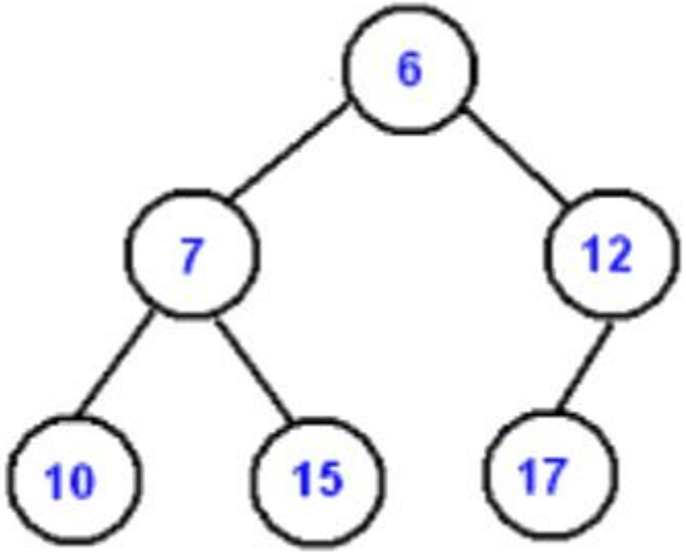
2. Is the value at the root of the tree the smallest value in the heap?

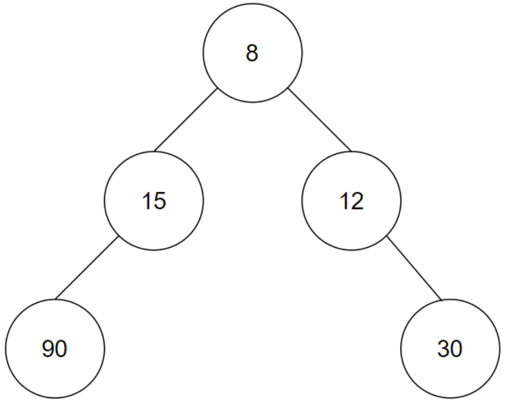
3. Are all parent nodes smaller than their children?

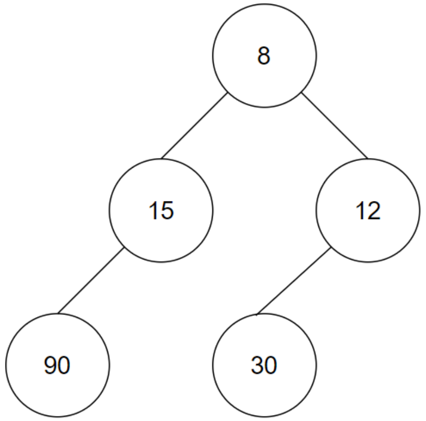


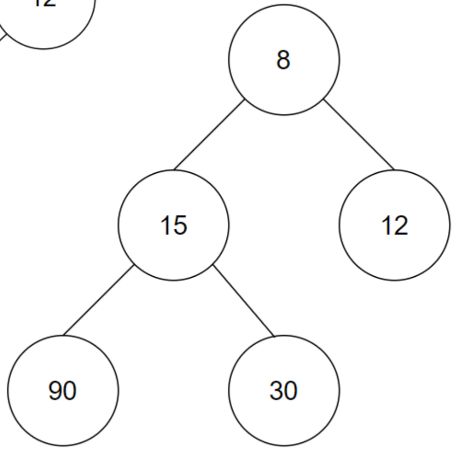
Binary Heap

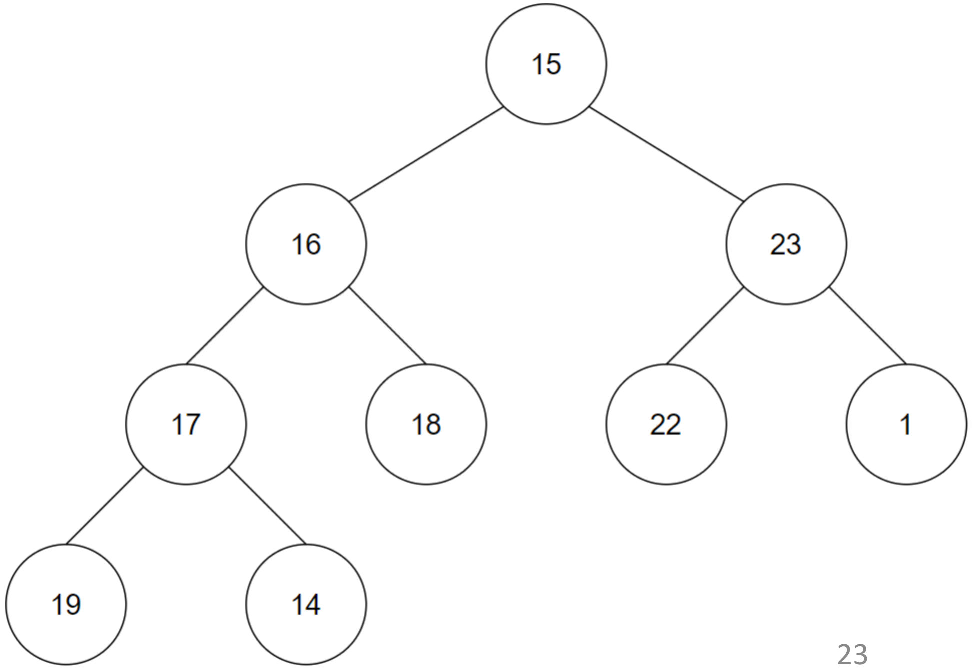

10
14
19
26
31
42
27
44
35
33


6
7
12
10
15
17


8
15
12
90
30


8
15
12
90
30


8
15
12
90
30


15
16
23
17
18
22
1
19
14

Binary Heap

Max Heap

The value at root of the tree must be largest value present in the Binary Heap.

This property must be recursively true for all nodes in the binary tree.

Any parent node should be larger than its child nodes.

Binary Heap

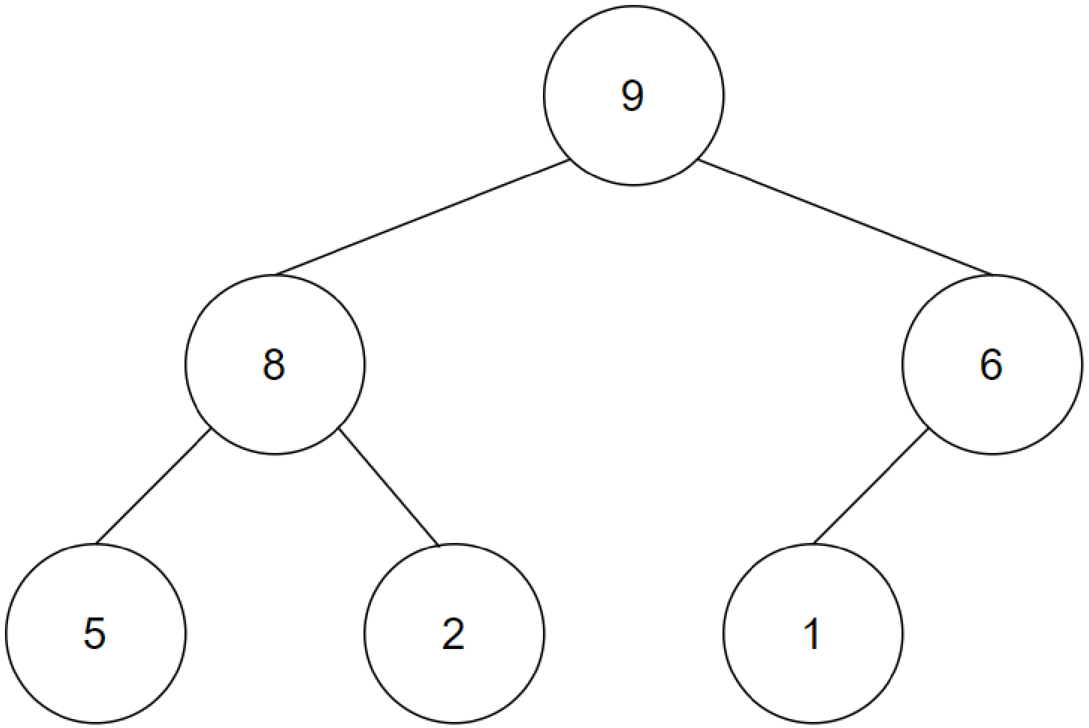
Is this a Max Heap?

1. Is it a complete tree?

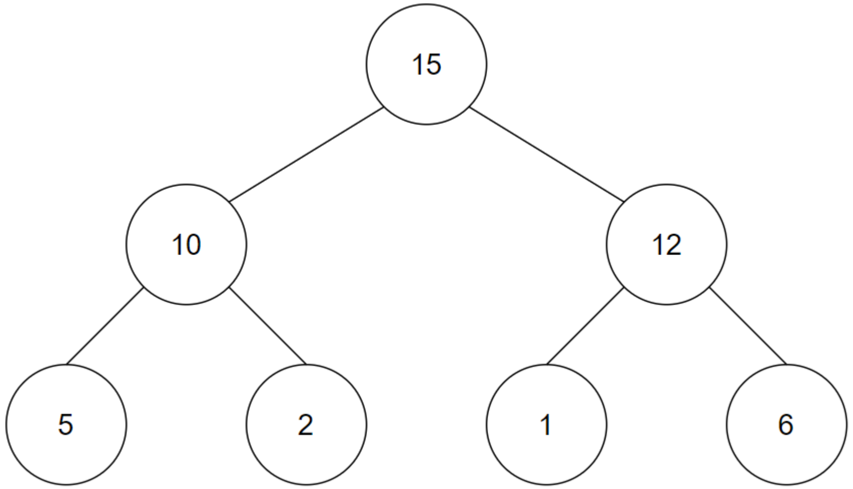
   all the levels are completely filled except possibly the last level and the last level has all keys as left as possible

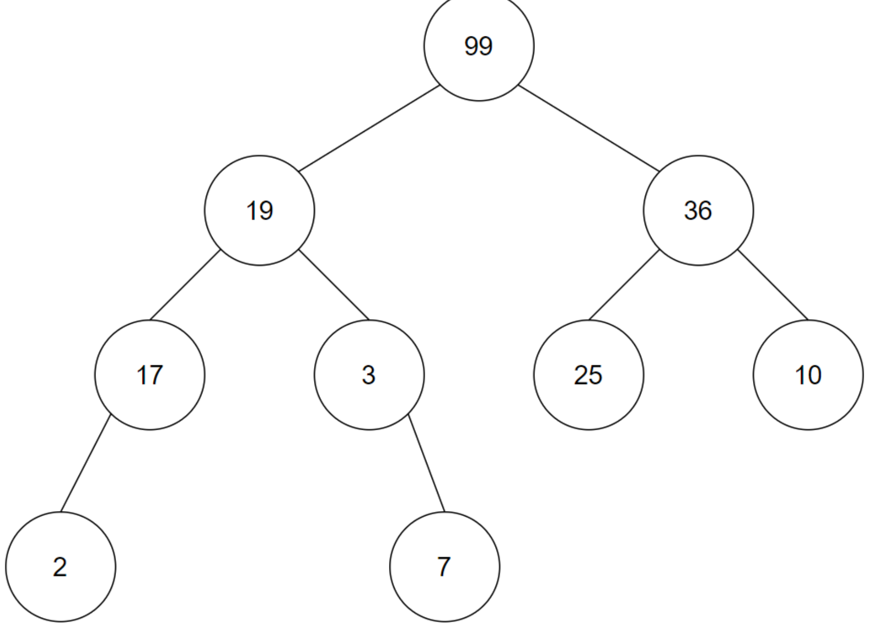
2. Is the value at the root of the tree the largest value in the heap?

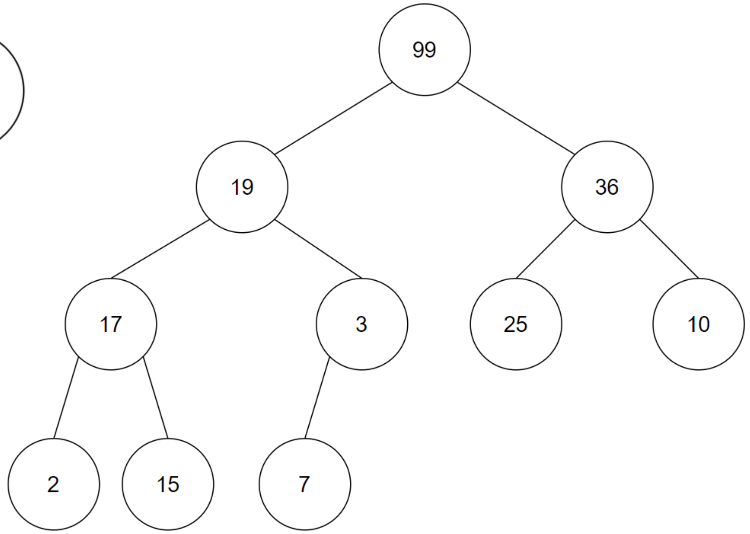
3. Are all parent nodes larger than their children?

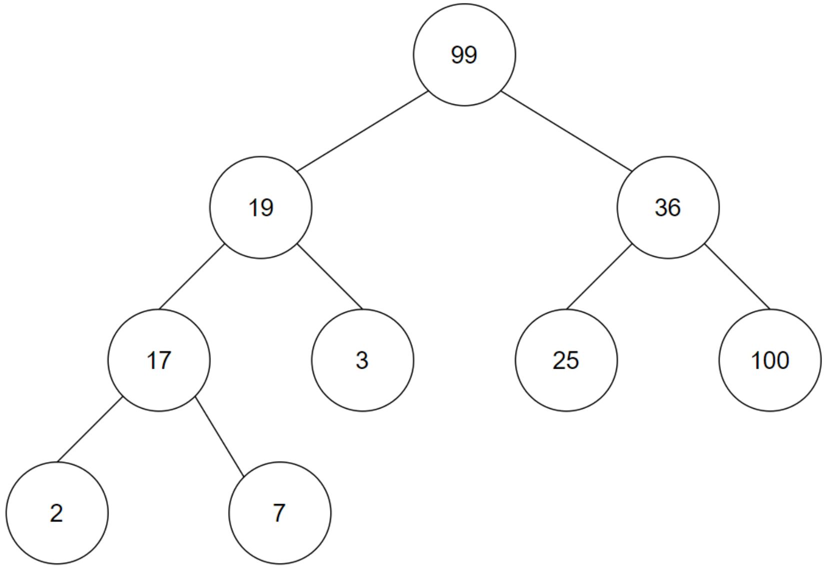


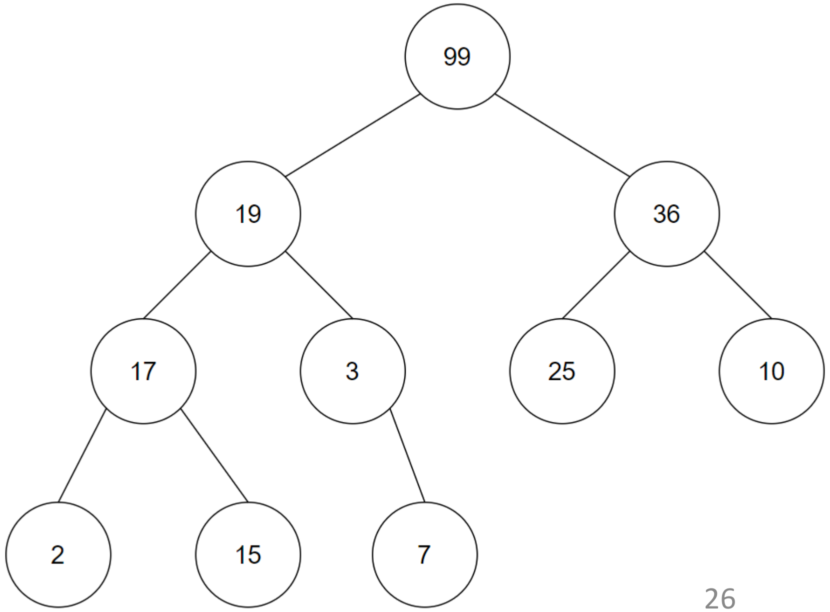
Binary Heap


15
10
12
5
2
1
6


99
19
36
17
3
25
10
2
7


99
19
36
17
3
25
10
2
15
7


99
19
36
17
3
25
100
2
7


99
19
36
17
3
25
10
2
15
7

Binary Heap

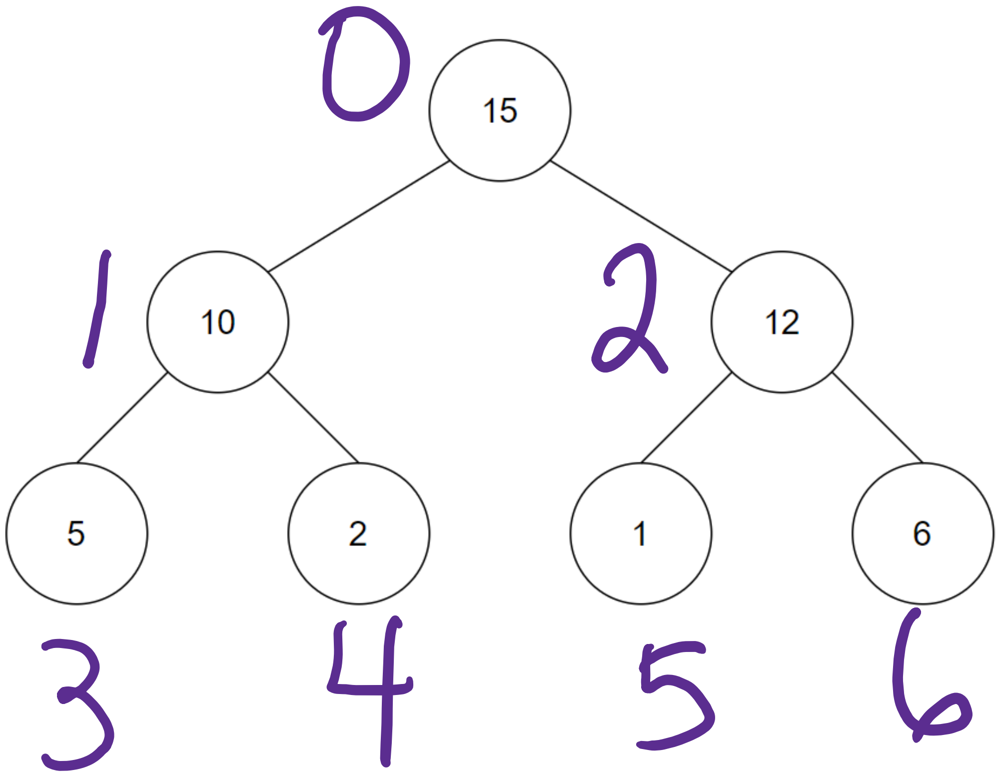
Now, here's the good news.

A binary heap is typically represented by an array.

So how do we turn a tree into an array?

Binary Heap

Let's look at it graphically and then we'll formalize the process.

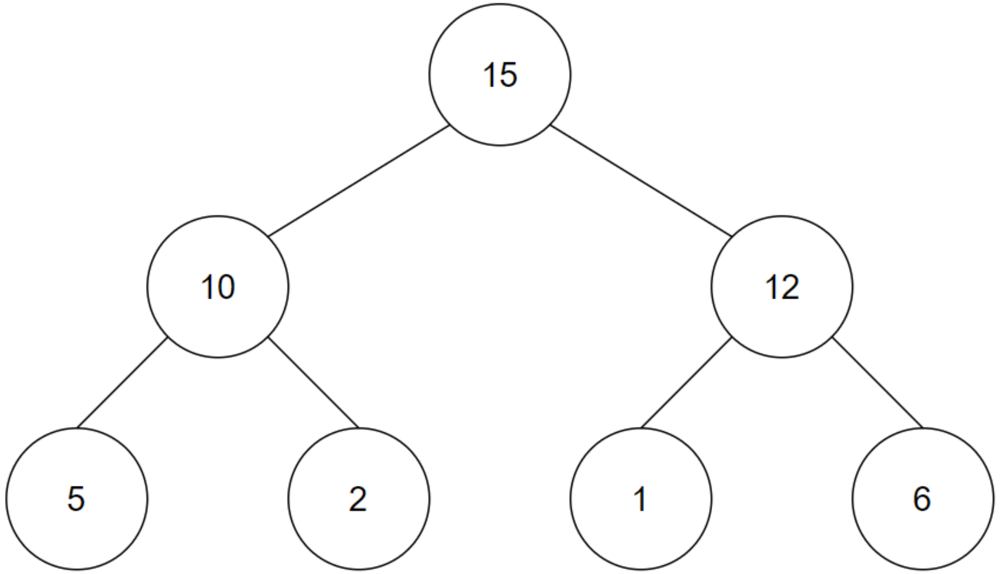


0	1	2	3	4	5	6

Binary Heap

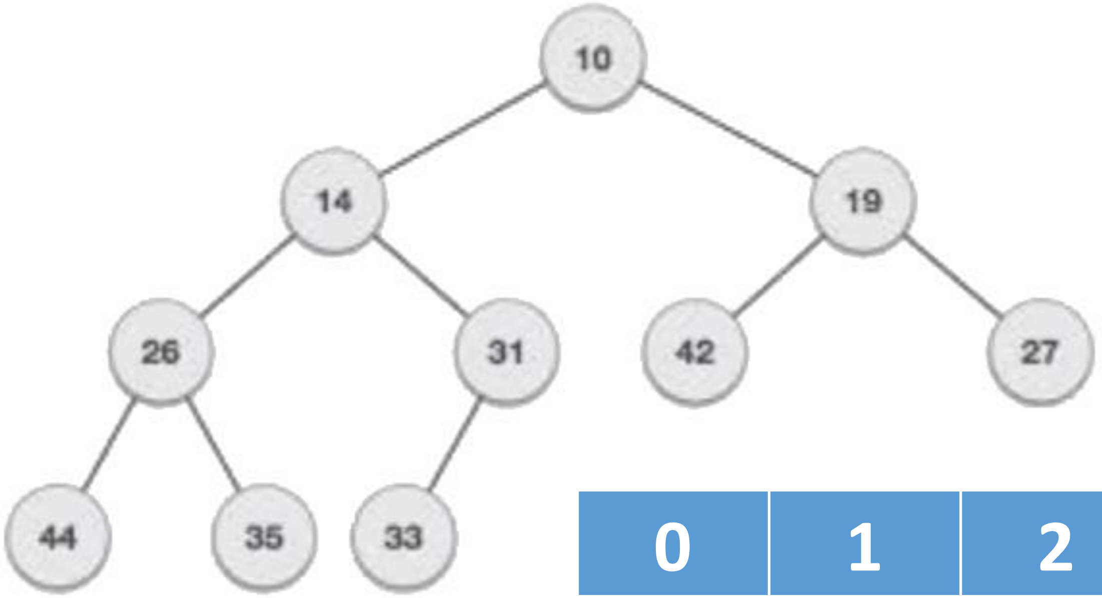
Let's start with the array and see if we correctly recreate the binary heap.

0	1	2	3	4	5	6
15	10	12	5	2	1	6



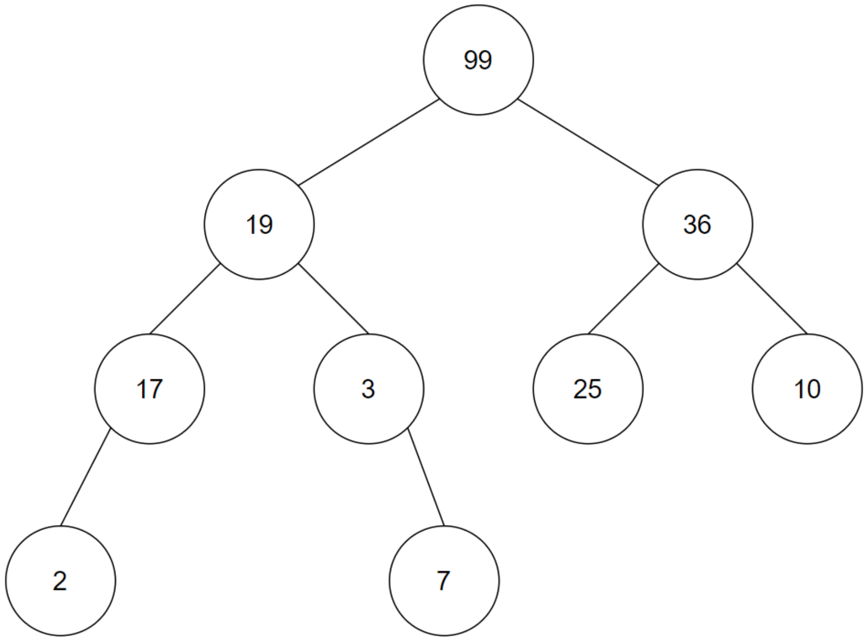
Binary Heap

That was for a max heap – let's try a min heap...



0	1	2	3	4	5	6	7	8	9

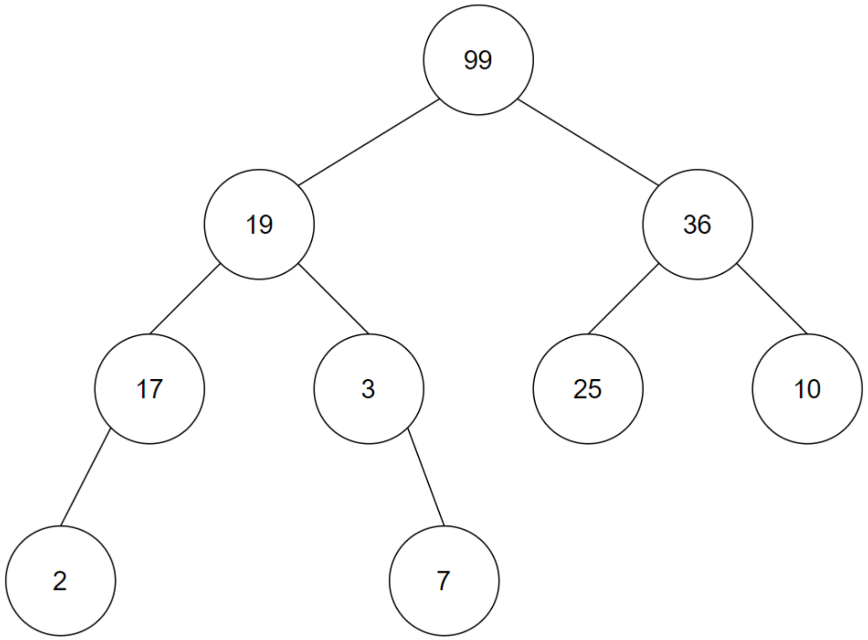
Binary Heap



What happens if we follow this process with a tree that has the values in the right locations (max parents/root) but was not a complete tree?

all the levels are completely filled except possibly the last level and the last level has all keys as left as possible

Binary Heap



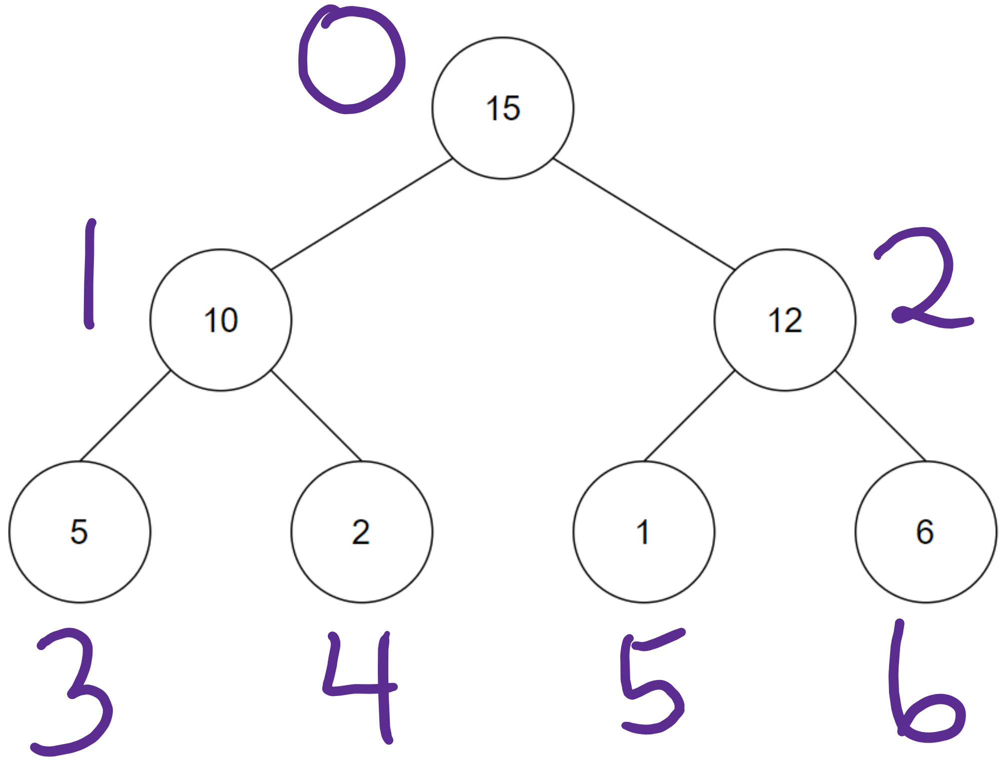
What if we just left blanks?

0	1	2	3	4	5	6	7	8	9	10

Binary Heap

This traversal method is called Level Order. We go through each level in order.

0	1	2	3	4	5	6
15	10	12	5	2	1	6



That order corresponds to the array indexes.

Binary Heap

So root always goes to ARRAY[0].

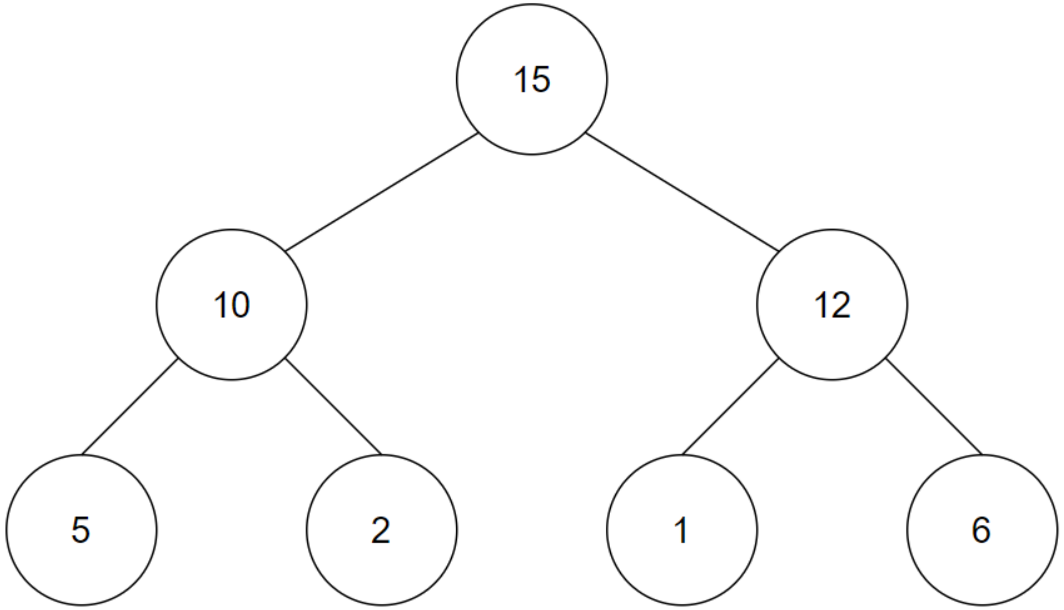
When using zero indexed arrays, we can state that for a index of i

left child = 2i + 1

right child = 2i + 2

Left child of root (i=0) = 2i + 1 = 1

Right child of root (i=0) = 2i + 2 = 2



0	1	2	3	4	5	6
15	10	12	5	2	1	6

Binary Heap

left child = 2i + 1

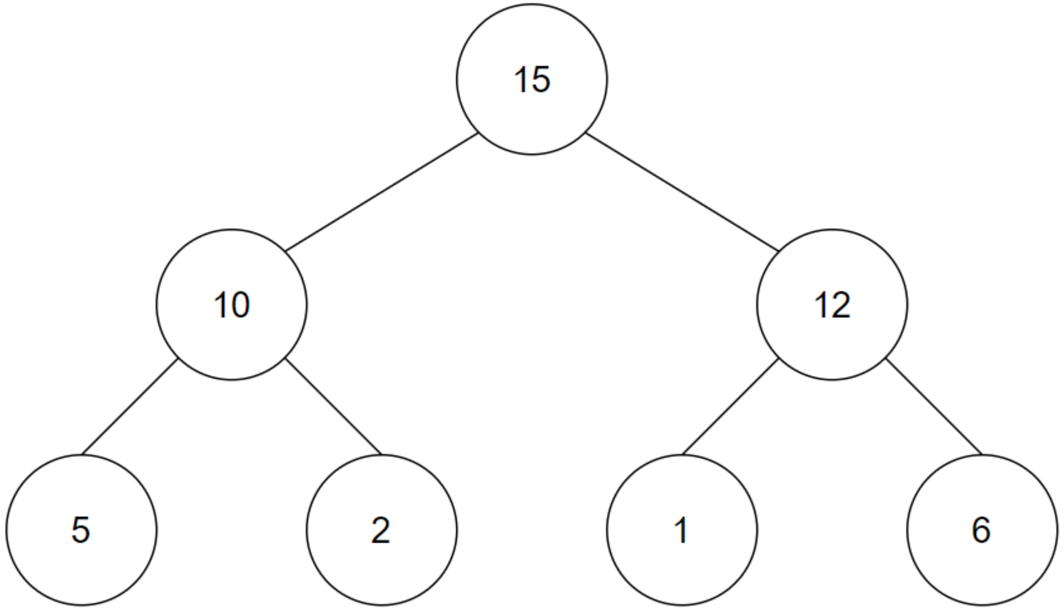
right child = 2i + 2

Left child of (10) (i=1) = 2i + 1 = 3

Right child of (10) (i=1) = 2i + 2 = 4

Left child of (12) (i=2) = 2i + 1 = 5

Right child of (12) (i=2) = 2i + 2 = 6



0	1	2	3	4	5	6
15	10	12	5	2	1	6

Adding to a Max Heap

How do we add a value?

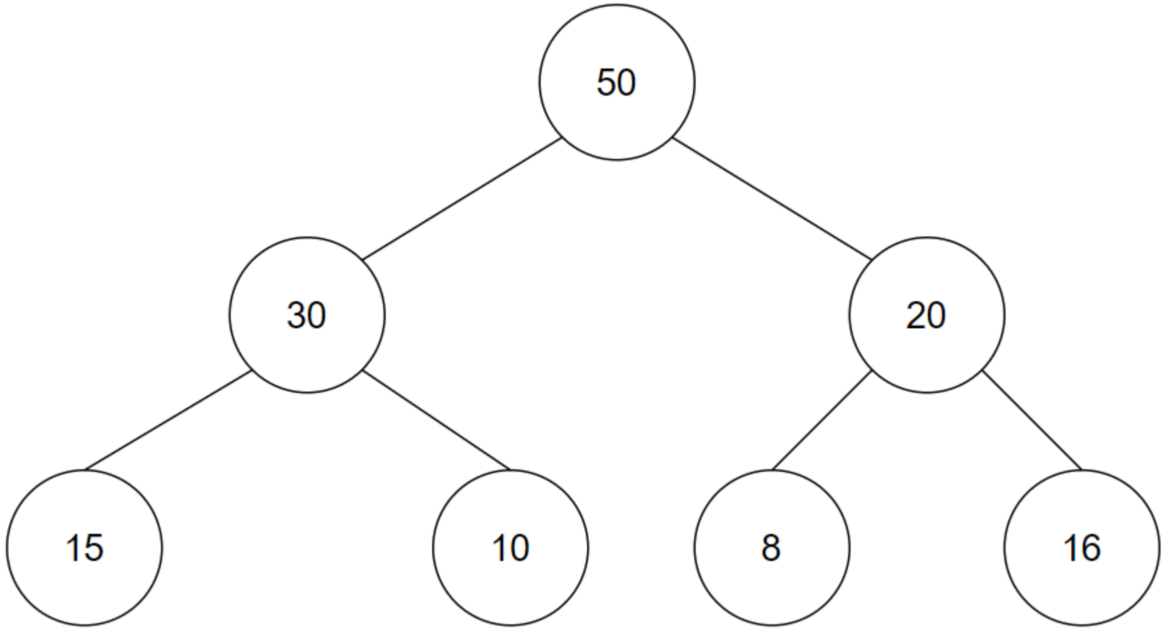
Let's say we want to add 60?

60 is greater than all of the other values in the tree so it should become the root.

But how do we make it the root?

Where does 50 go?

How do we shift around the parents and children?



0	1	2	3	4	5	6
50	30	20	15	10	8	16

Adding to a Max Heap

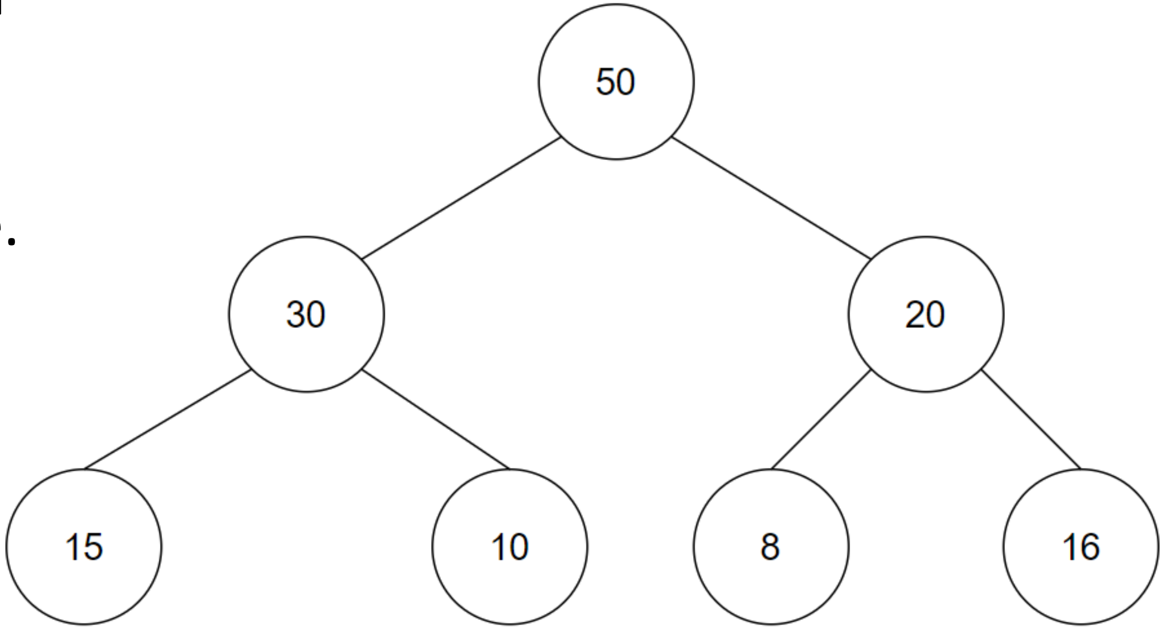
Adding a value to heap is called inserting or pushing.

So let's look at the array instead of the tree.

Where would it make sense to add an element to the array?

At the end would be the easiest

should be room
won't have to move anyone else



0	1	2	3	4	5	6
50	30	20	15	10	8	16

Adding to a Max Heap

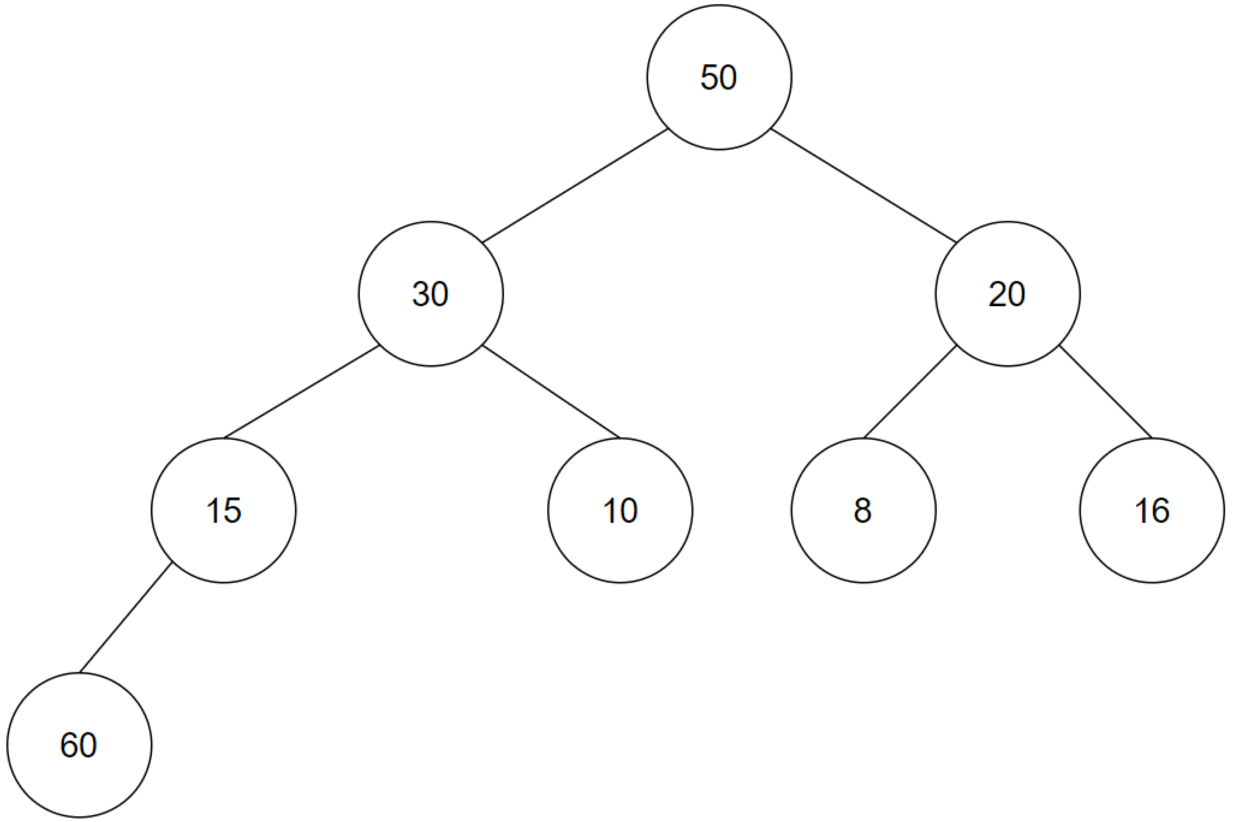
What does that do to our tree?

Where would the 60 go according to the array?

Once we put 60 there, we have violated the heap property

60 is largest value in the tree but it is not the root.

So we need to move it…



0	1	2	3	4	5	6	7
50	30	20	15	10	8	16	60

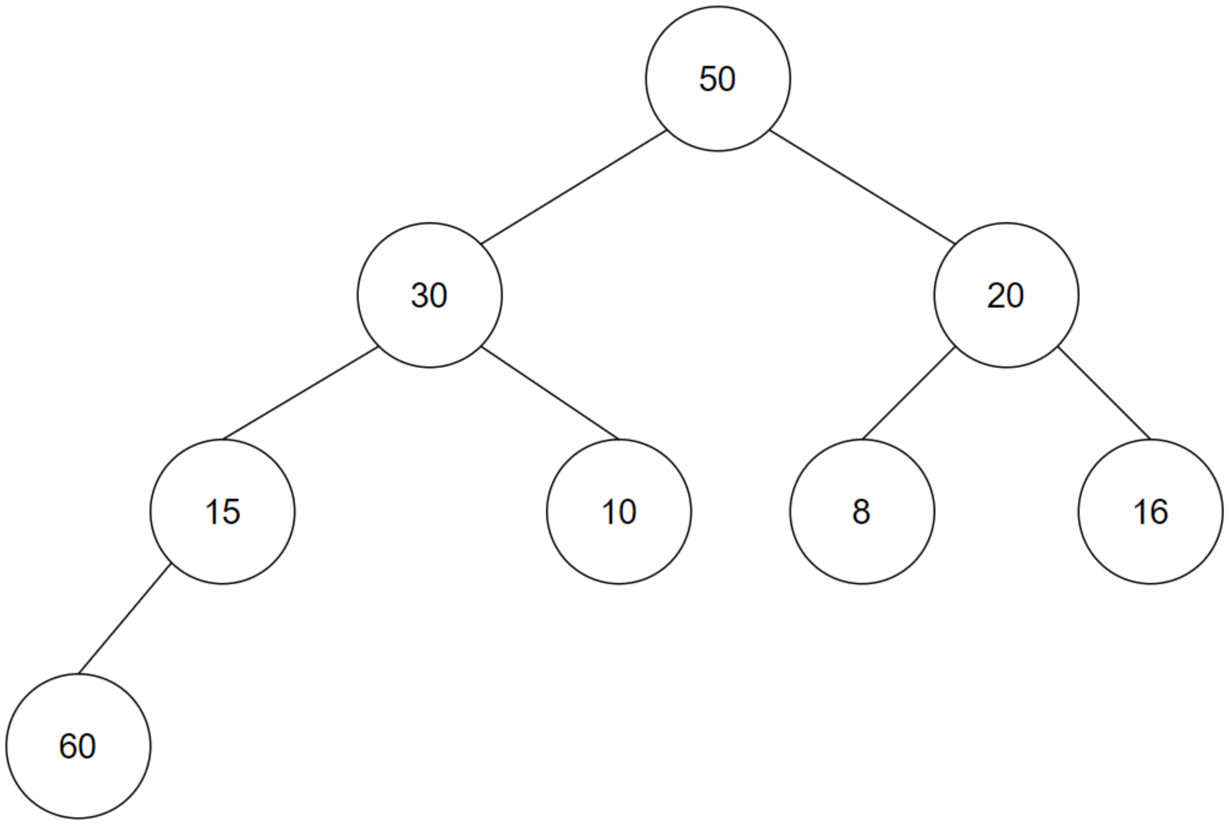
Adding to a Max Heap

We could swap 15 and 60.

And then swap 30 and 60.

And then swap 50 and 60.

Have we restored the heap property?

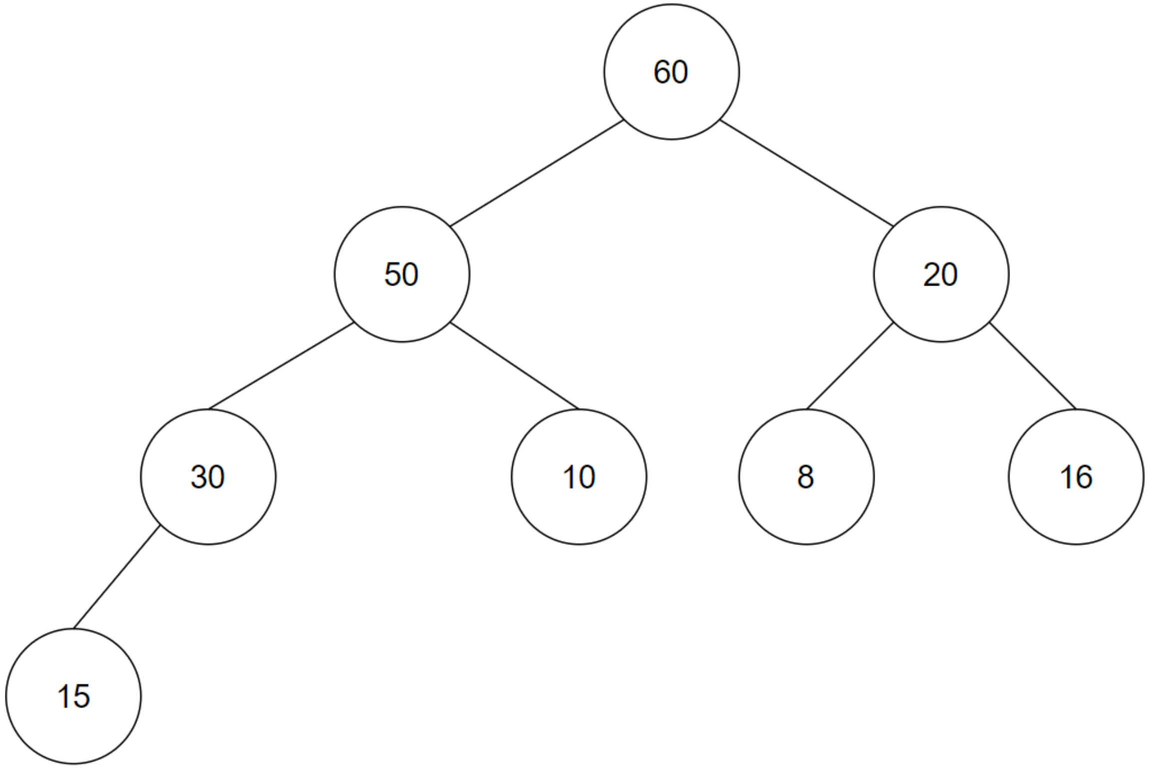


0	1	2	3	4	5	6	7
50	30	20	15	10	8	16	60

Adding to a Max Heap

Yes!

We have a complete tree and the root is the greatest value and every child is less than its parent.



0	1	2	3	4	5	6	7
60	50	20	30	10	8	16	15

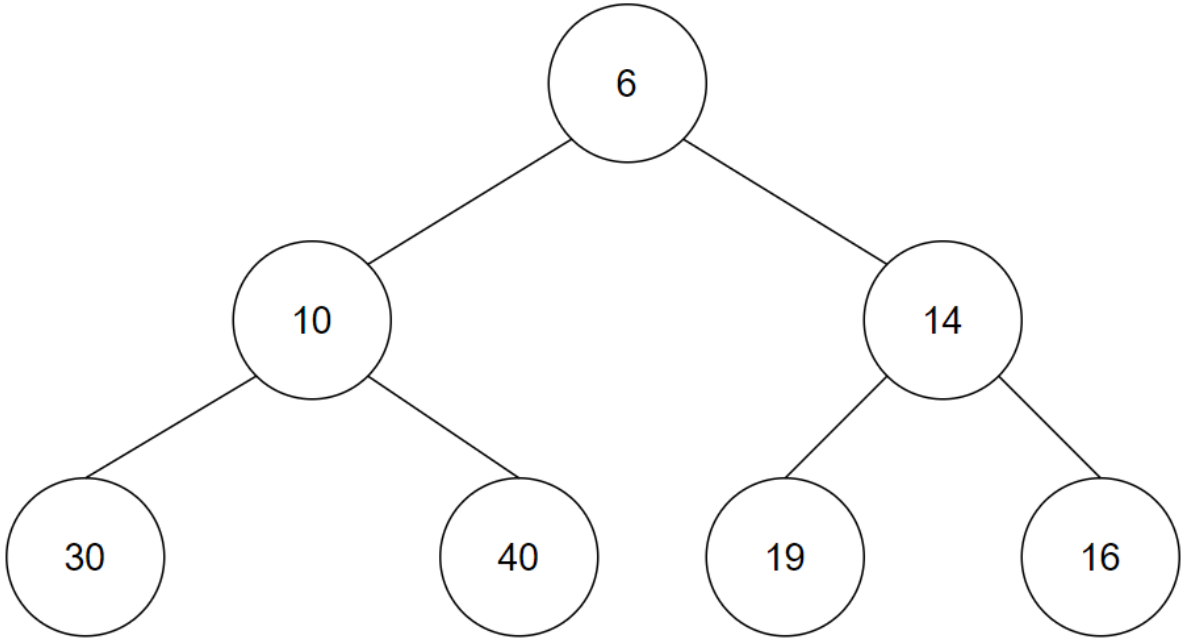
Adding to a Min Heap

How do we add a value?

Let's say we want to add 2?

2 is smaller than all of the other values in the tree so it should become the root.

Do we follow the same process as we did with max heap?



0	1	2	3	4	5	6
6	10	14	30	40	19	16

Adding to a Min Heap

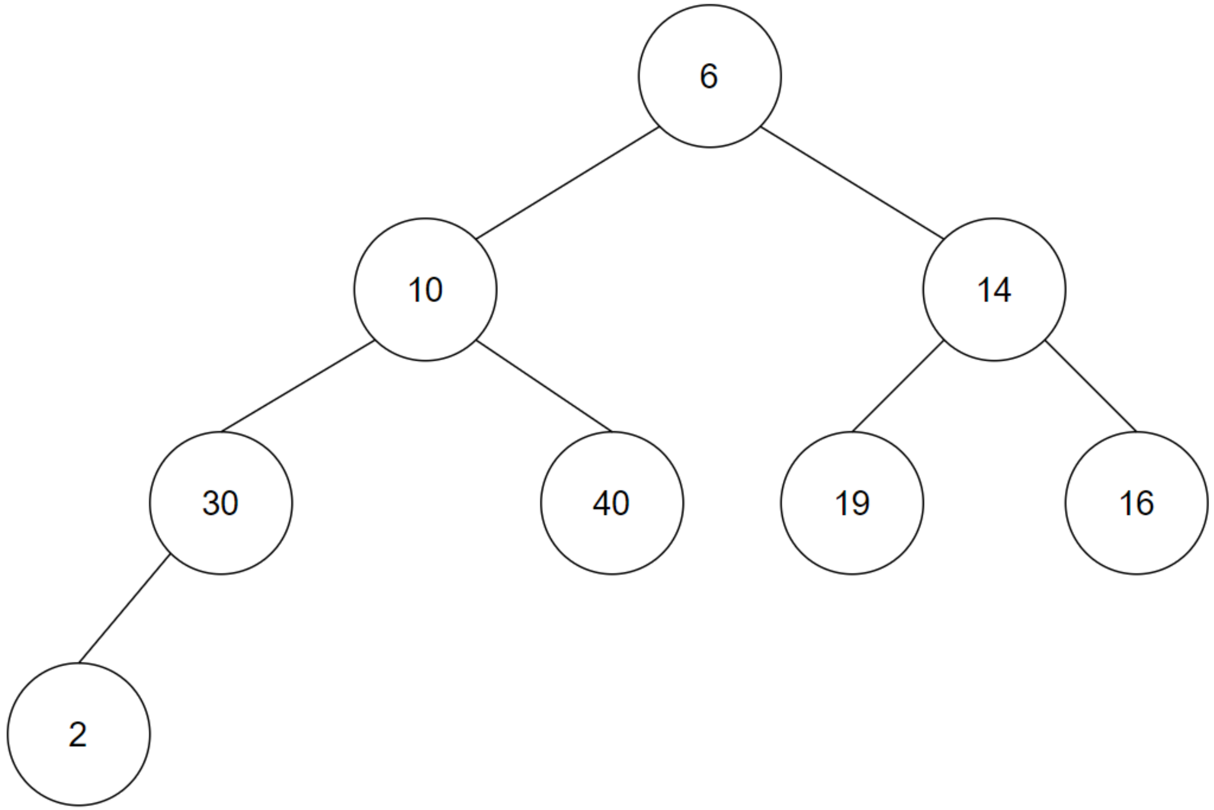
Let's add the 2 just like we did 60.

Now we need to swap 2 and 30.

Now swap 2 and 10.

Now swap 6 and 2.

Have we restored the heap property?

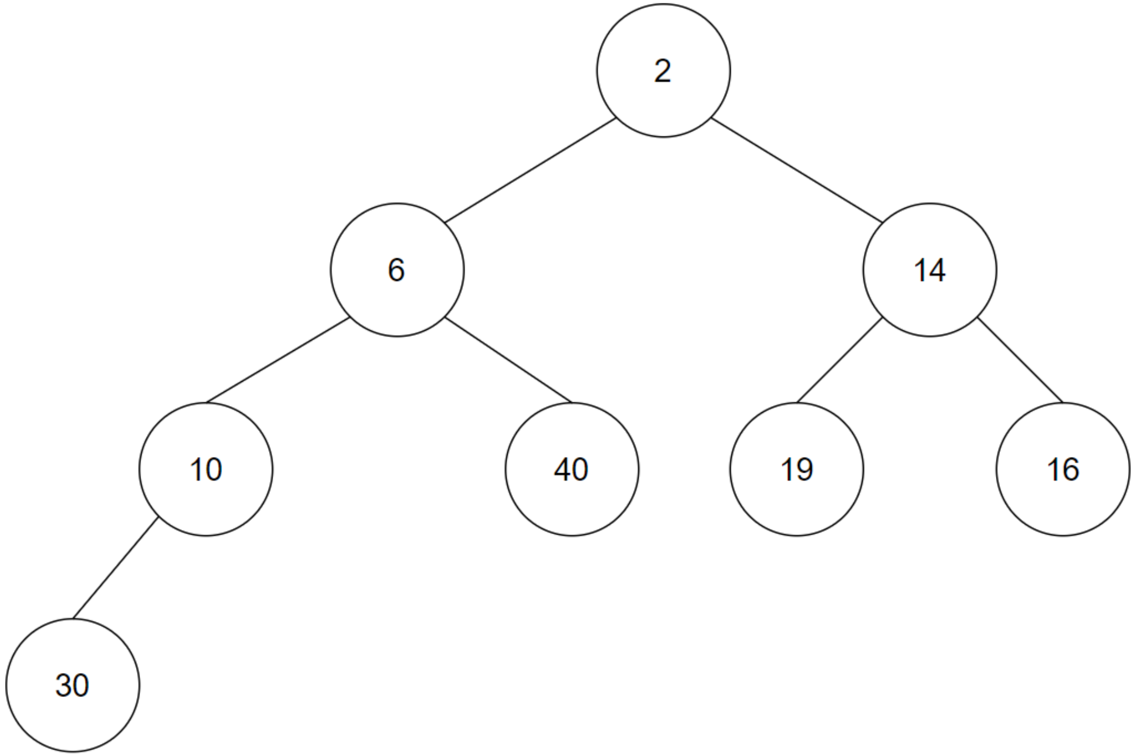


0	1	2	3	4	5	6	7
6	10	14	30	40	19	16	2

Adding to a Min Heap

Yes!

We have a complete tree and the root is the smallest value and every child is greater than its parent.



0	1	2	3	4	5	6	7
2	6	14	10	40	19	16	30

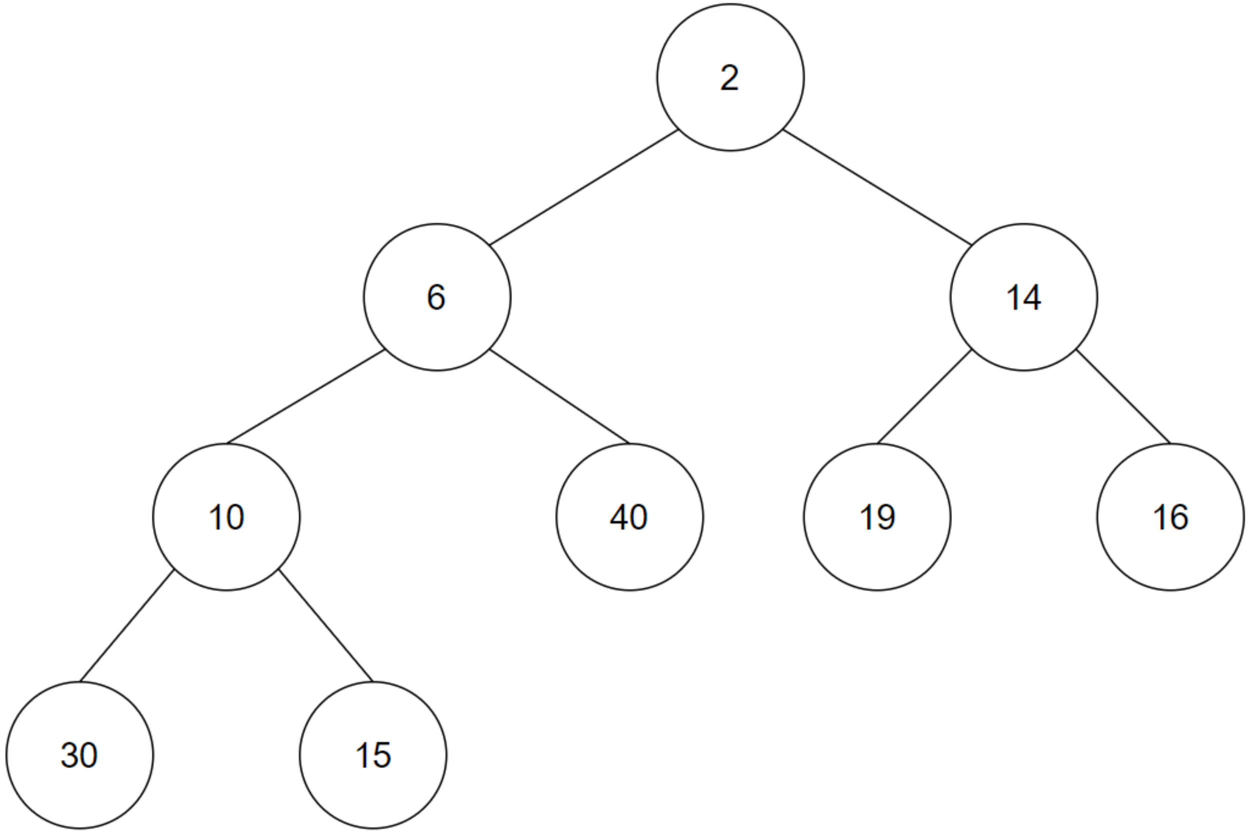
Adding/Inserting with Heap

What if my heap looks like this and I want to add 1.

Where do I add it?

Follow the same process as before…

Add the new value at the end of the array.



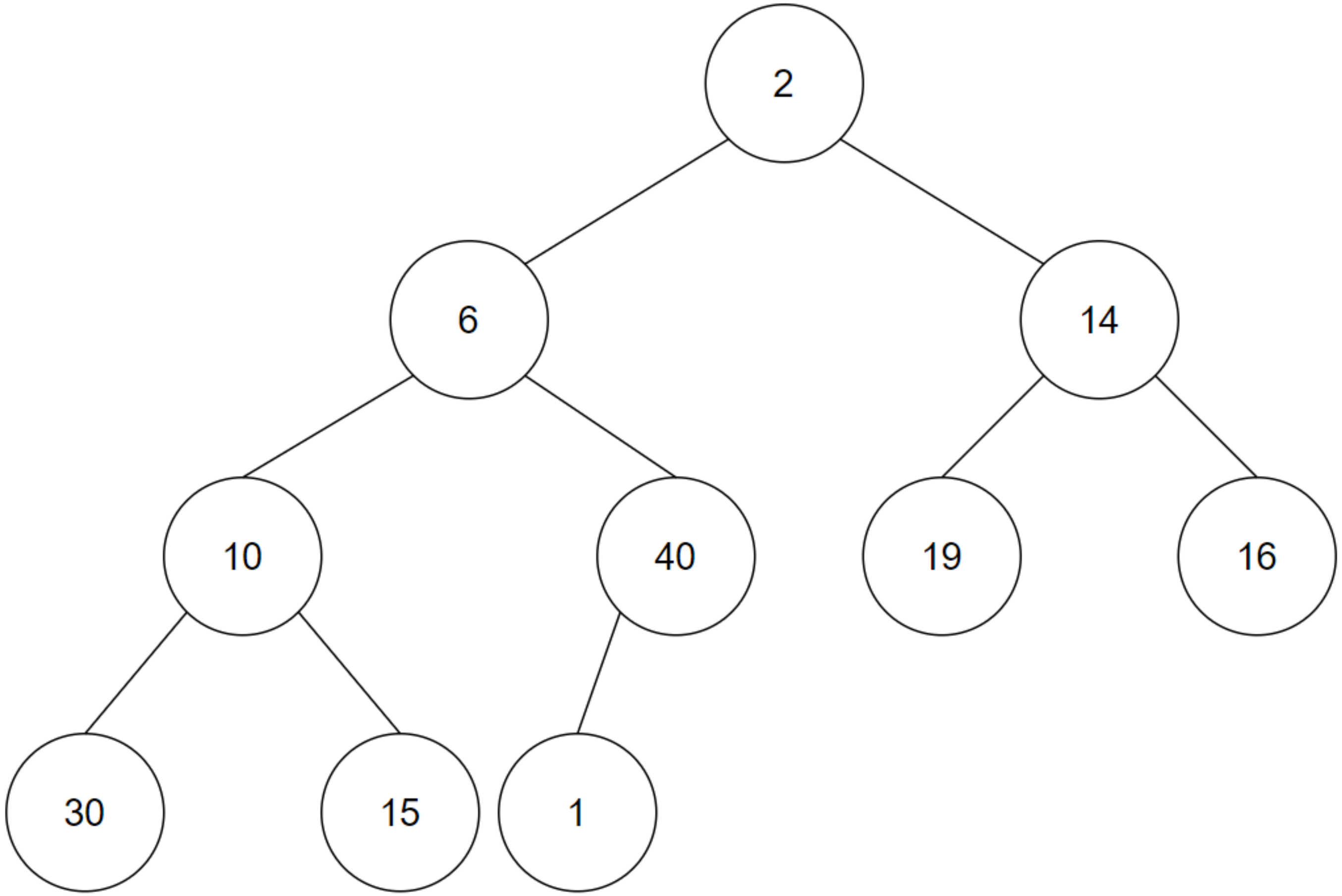
0	1	2	3	4	5	6	7	8	9
2	6	14	10	40	19	16	30	15	

Adding/Inserting with Heap

Now swap 40 and 1

Swap 1 and 6

Swap 1 and 2



0	1	2	3	4	5	6	7	8	9
2	6	14	10	40	19	16	30	15	1

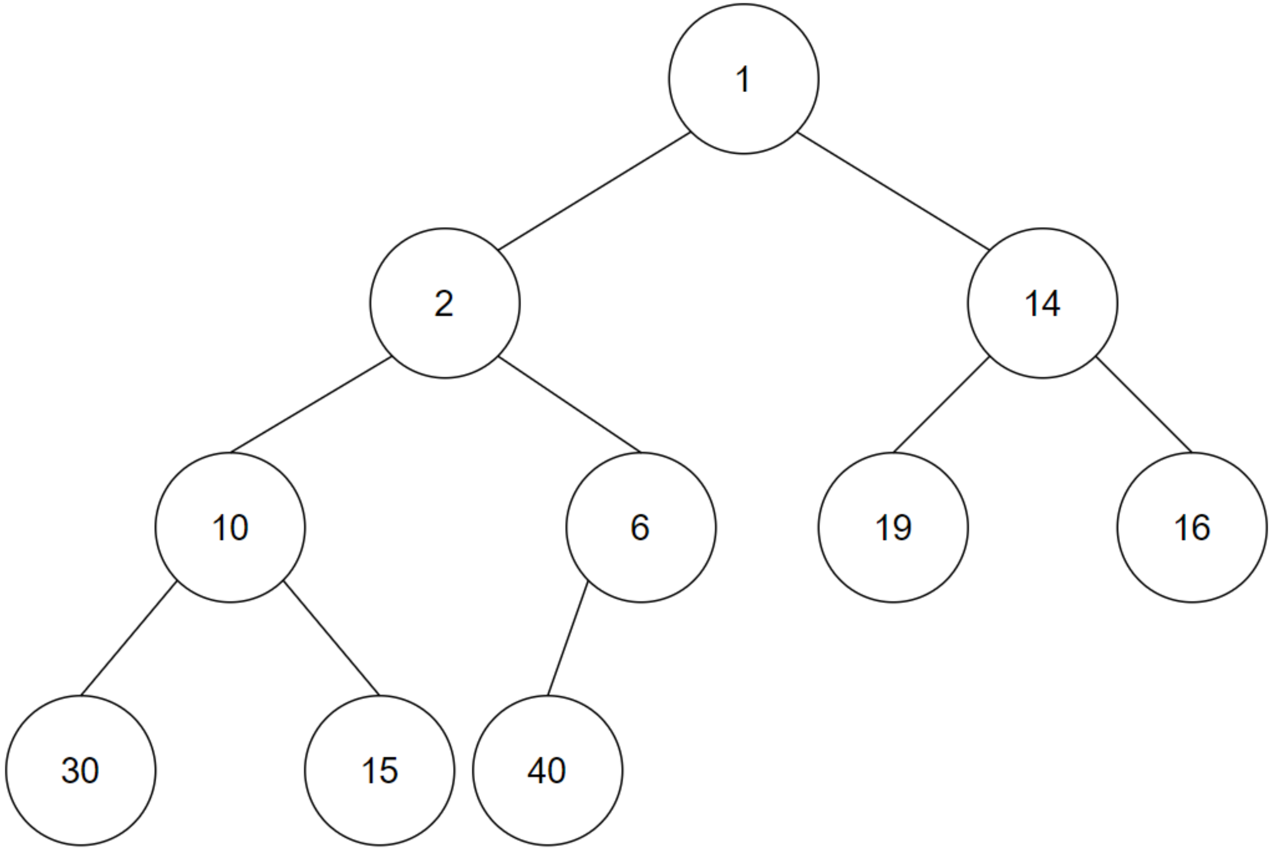
Adding/Inserting with Heap

Do we still have a min heap?

Complete tree and every parent is less than its children.

Root is the smallest.

Yes!



0	1	2	3	4	5	6	7	8	9
1	2	14	10	6	19	16	30	15	40

Deleting from a Heap

If you had a stack of cans like this one, how would you remove a can?

Without making a mess or knocking them over?

This is not a game of Jenga

so we would take the top can.

Deleting from a Heap

We delete from a heap the same way – we take the top element.

For a max heap, the top element/root is the biggest element in the heap.

For a min heap, the top element/root is the smallest element in the heap.

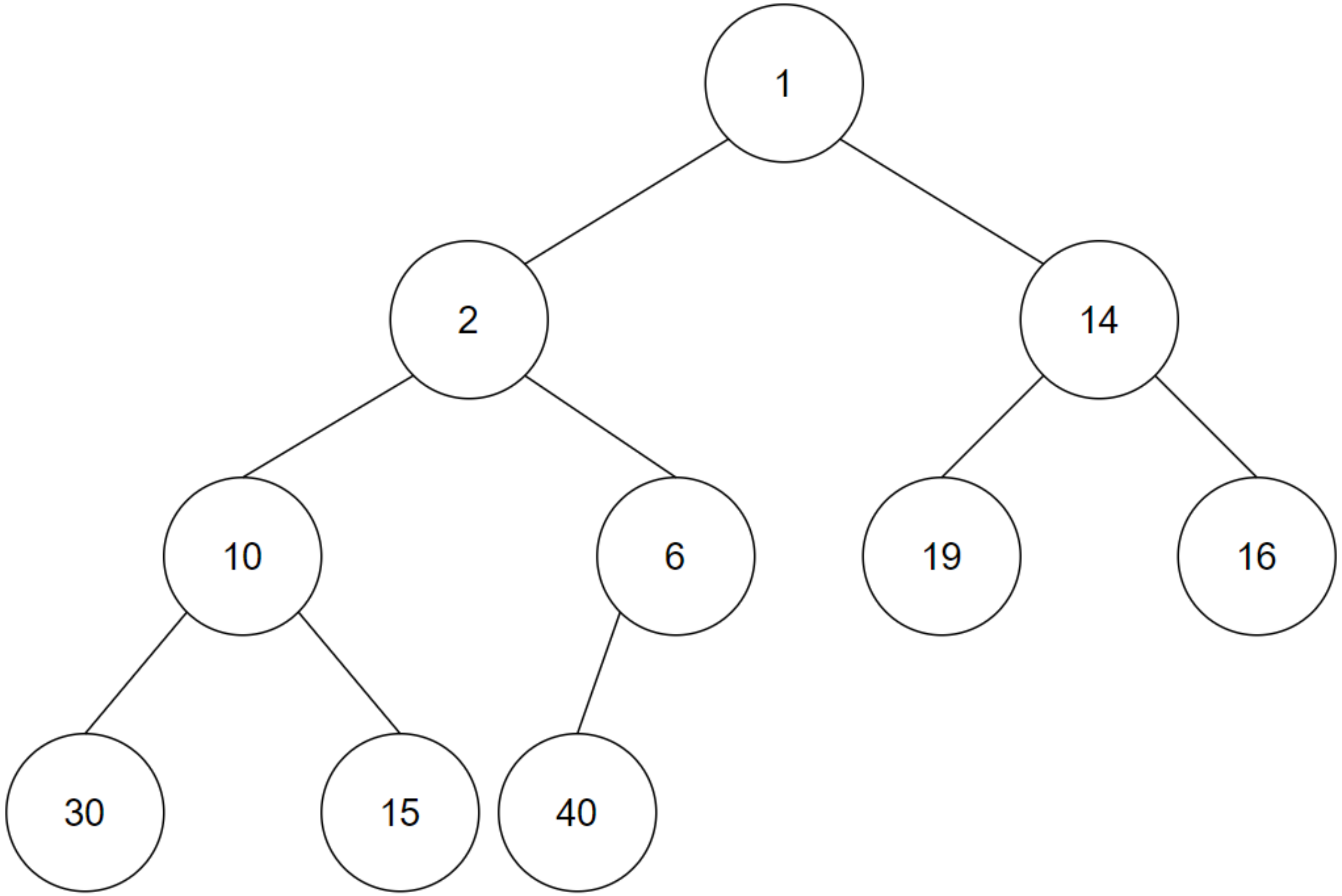
Deleting from a Min Heap

So what happens when we remove the root?

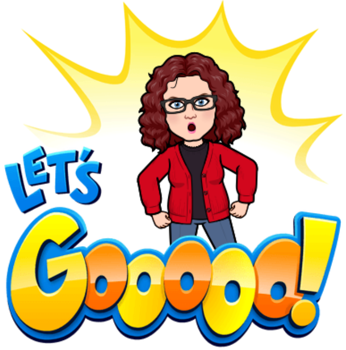
We need a new root?

Do we pick one of the root's children?

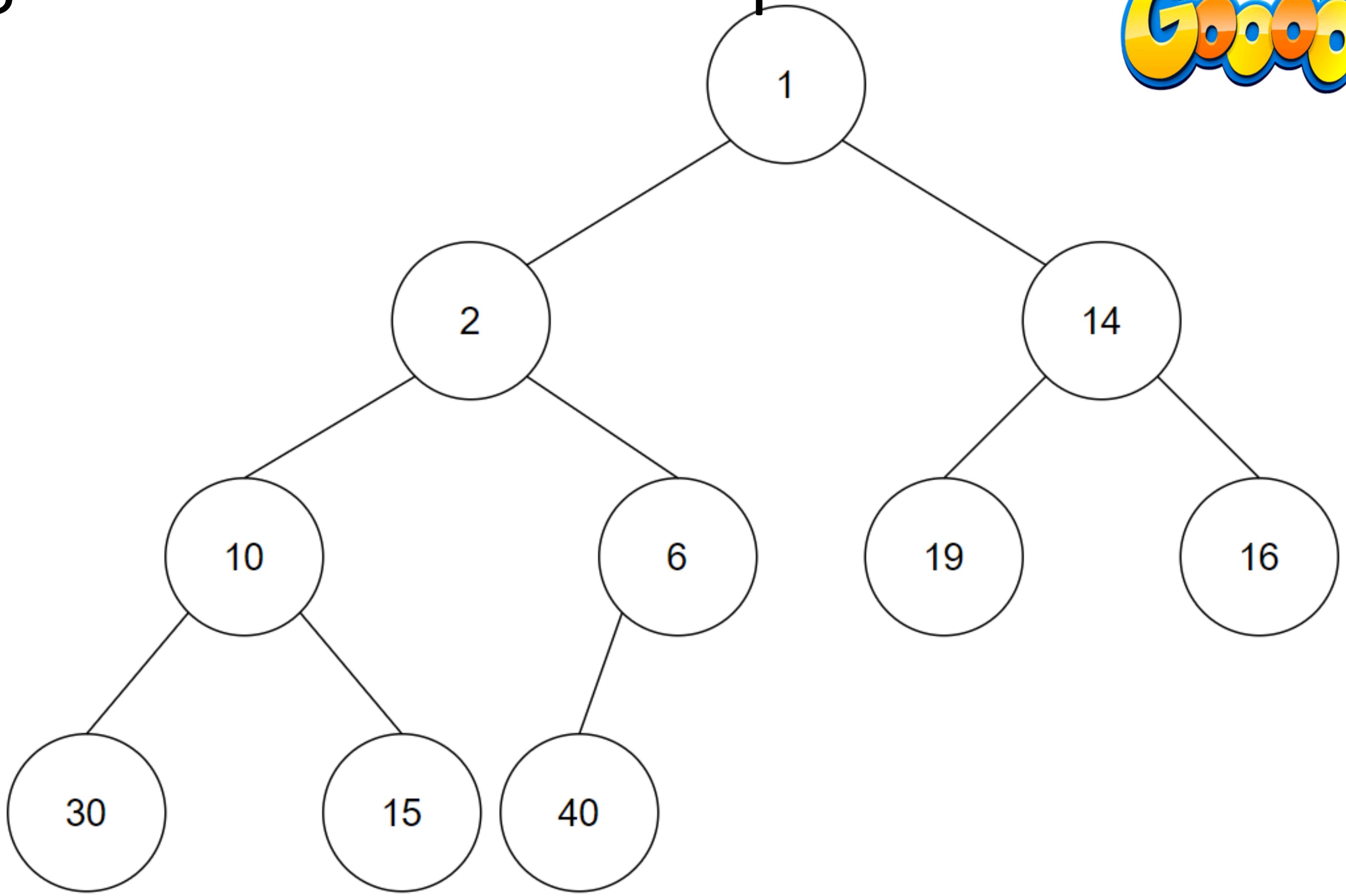
No.



A



Deleting from a Min Heap



Let's look at the array again.

If we take the value out of the 0^{th} element, what is the easiest replacement value from the array?

If we take element 1, then we'll just have another gap.

We've already said that we don't like gaps in our arrays.

0	1	2	3	4	5	6	7	8	9
	2	14	10	6	19	16	30	15	40

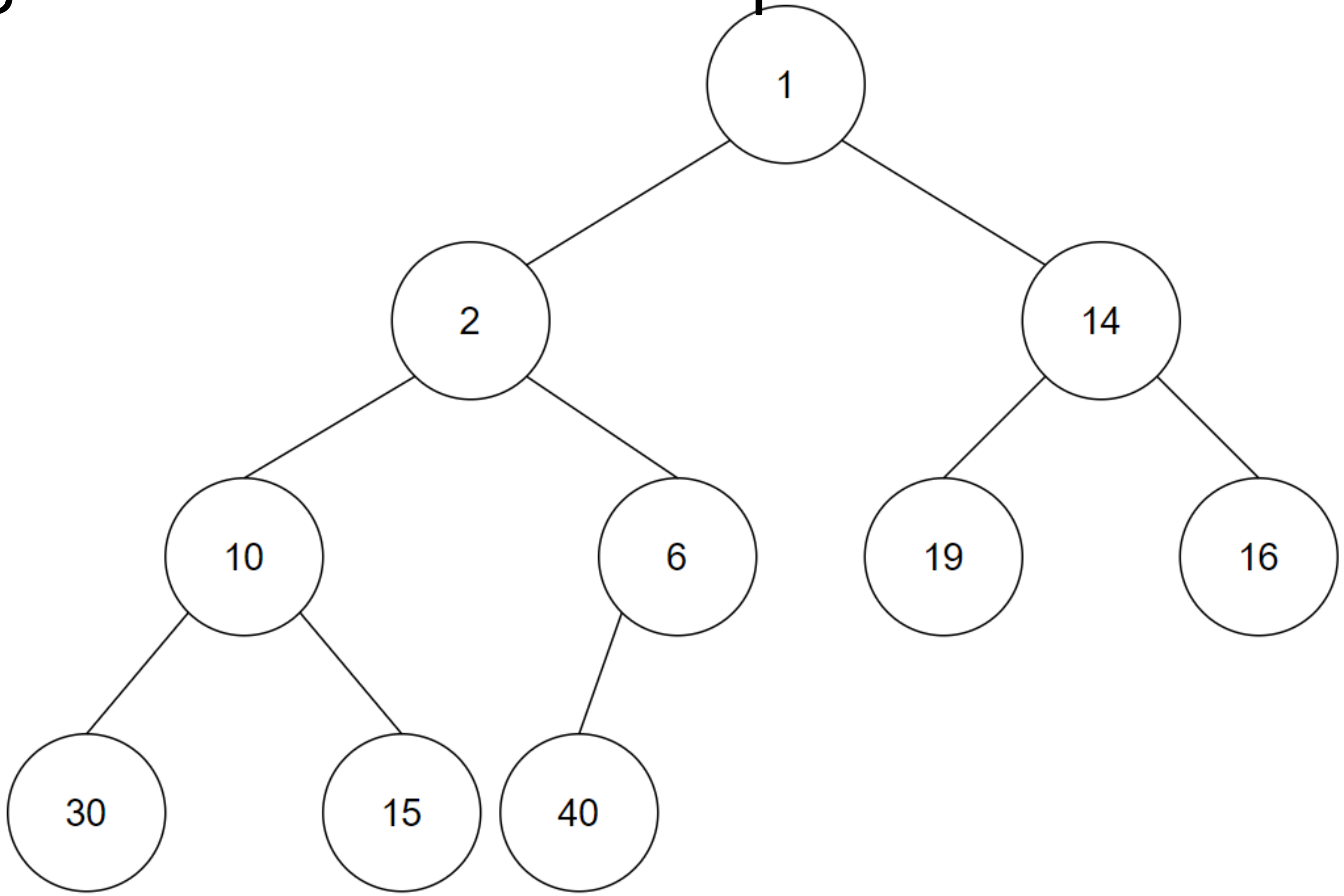
Deleting from a Min Heap

What if we move the last element to the 0^{th} spot?

It's OK to have spaces at the END of the array.

What will the heap look like?

Do we have a problem?



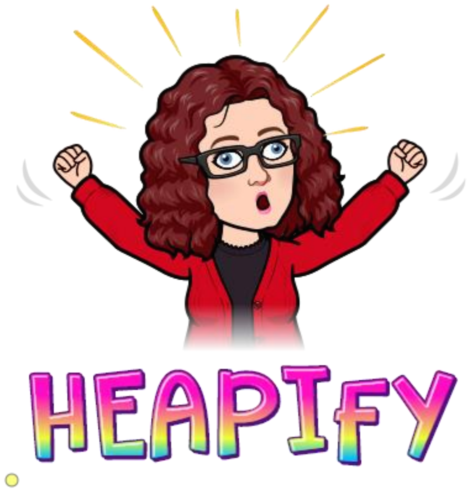
0	1	2	3	4	5	6	7	8	9
40	2	14	10	6	19	16	30	15	

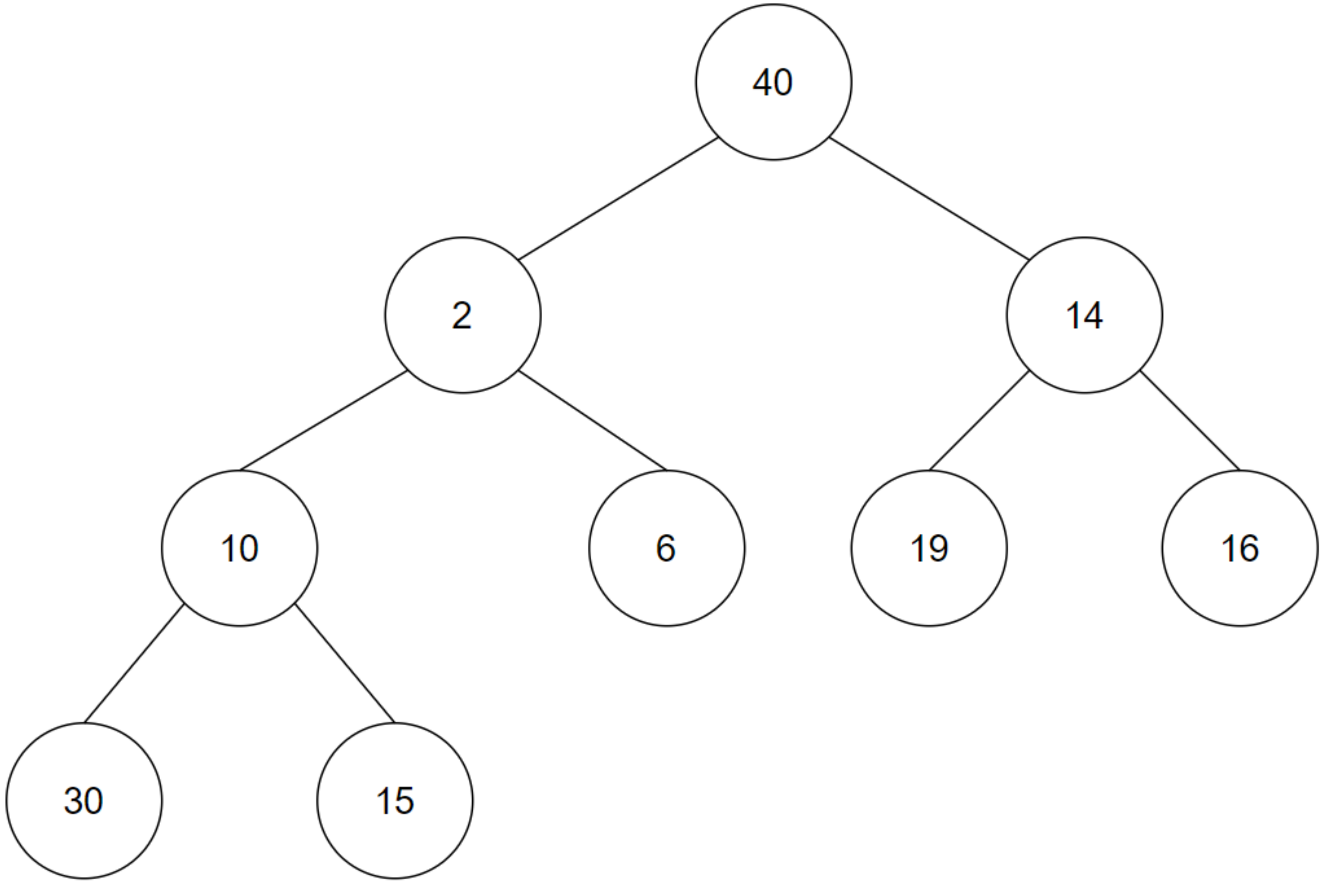
Deleting from a Min Heap

Do we have a problem?

We have lost our heap property again.

The tree is still complete but 40 is too big to be the root of this min heap.





0	1	2	3	4	5	6	7	8	9
40	2	14	10	6	19	16	30	15	

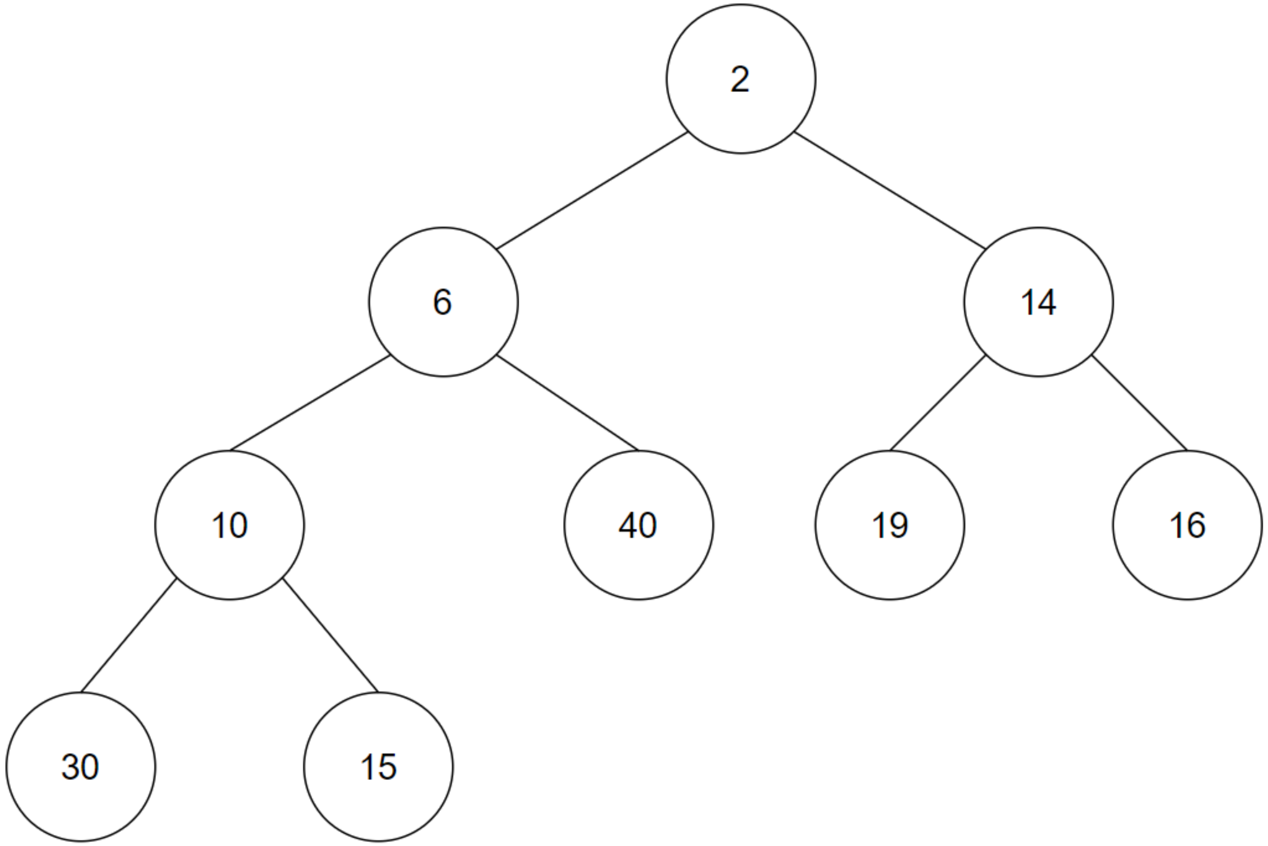
Deleting from a Min Heap

We are back to a min heap

Complete tree

Parents smaller than children

Root is smallest value.



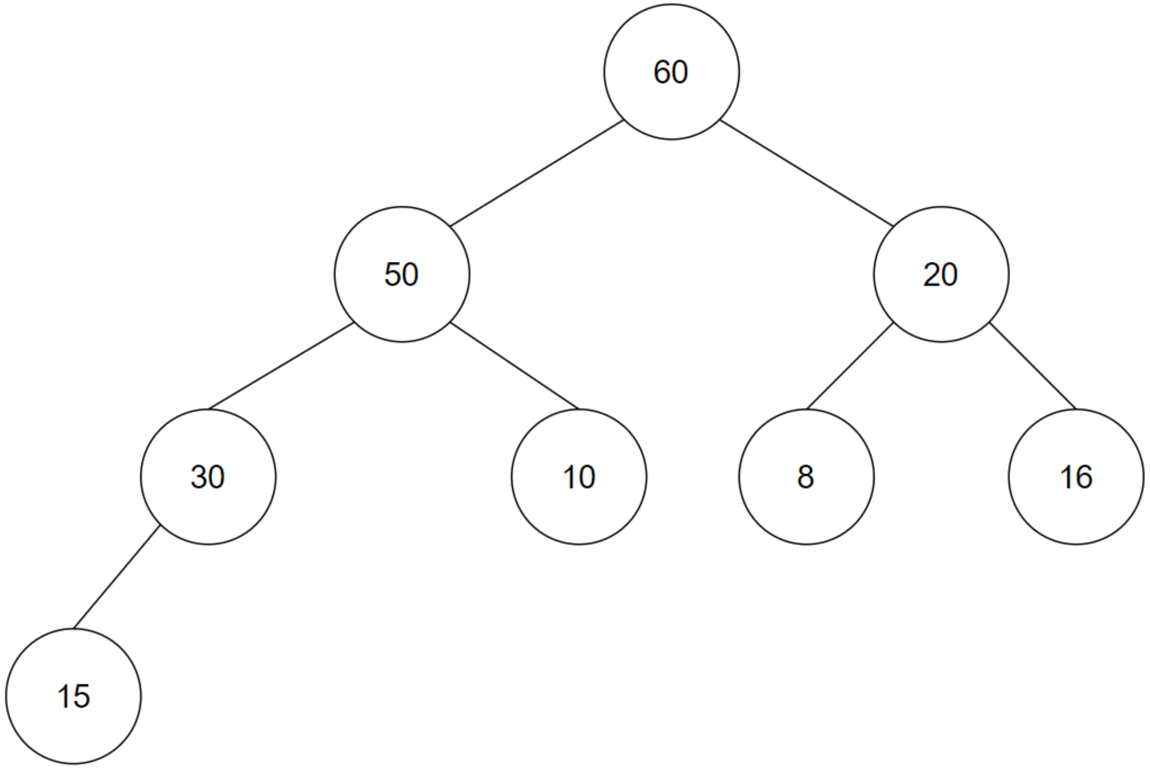
0	1	2	3	4	5	6	7	8	9
2	6	14	10	40	19	16	30	15	

Deleting from a Max Heap

Let's try the same technique with a max heap.

We want remove 60.

We take 60 out and swap it with the last value of 15.



0	1	2	3	4	5	6	7
60	50	20	30	10	8	16	15

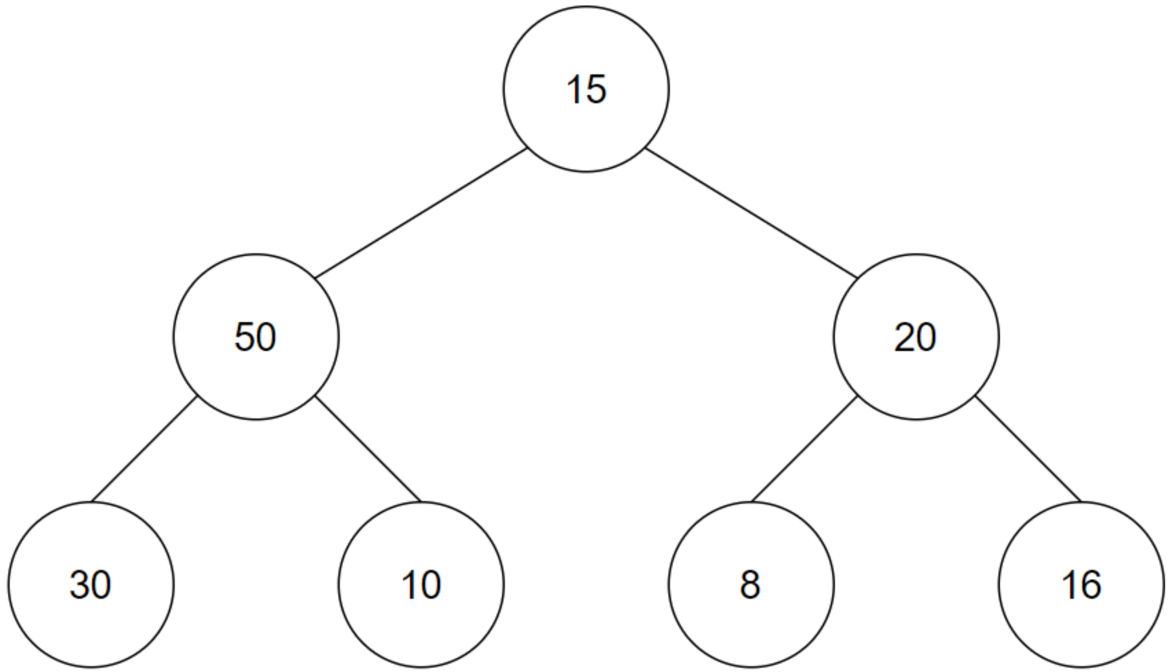
Deleting from a Max Heap

Now let's fix our heap.

Swap 15 and 50

Swap 15 and 30

Are we OK now?

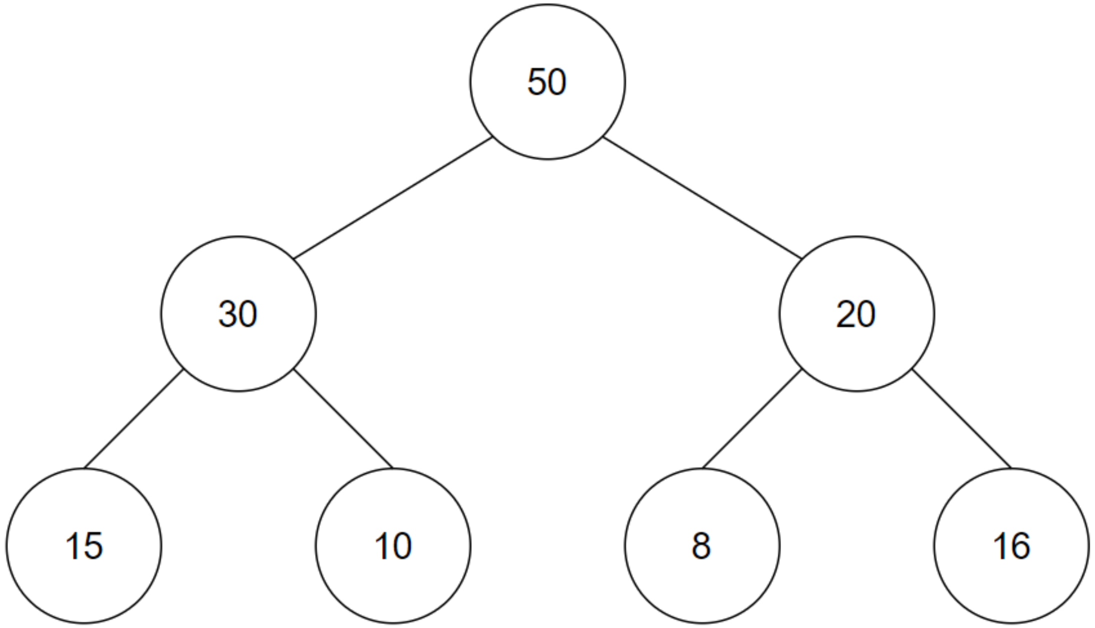


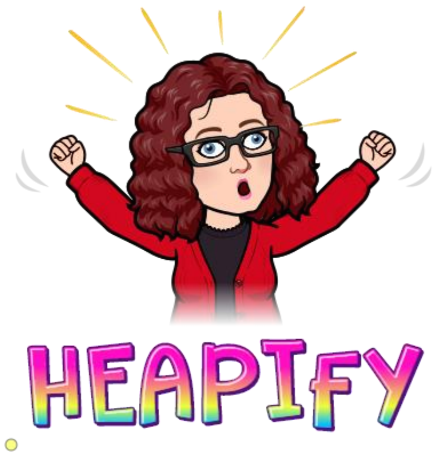
0	1	2	3	4	5	6	7
15	50	20	30	10	8	16	

Deleting from a Max Heap

Yes!

Our tree is complete and we have maintained our heap property.



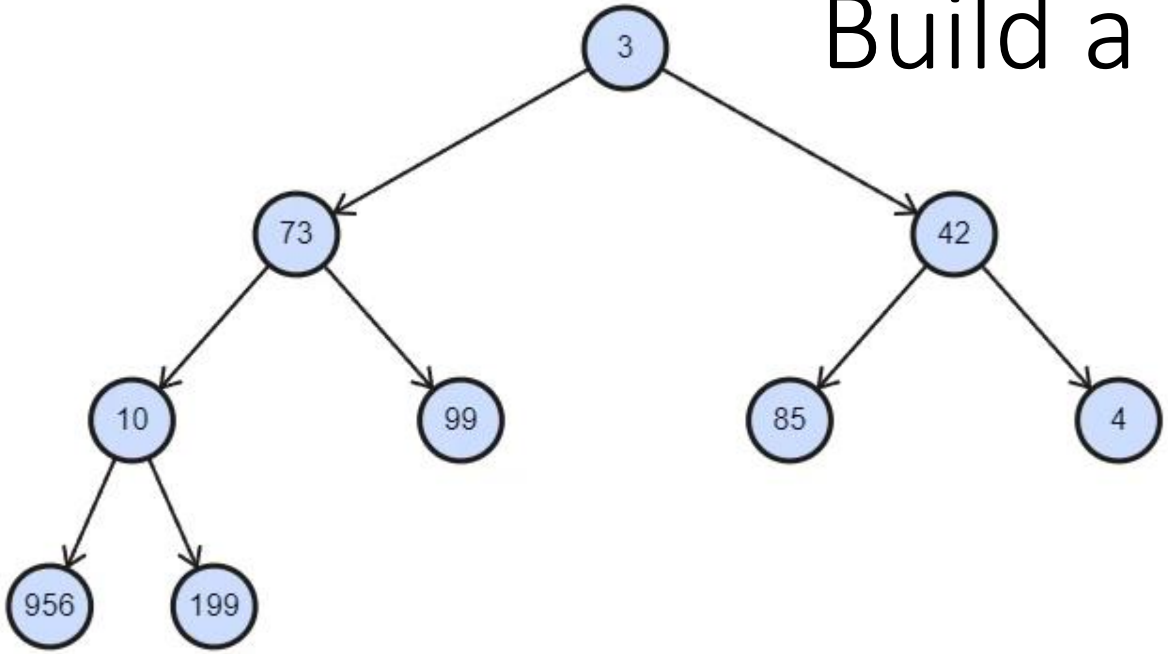


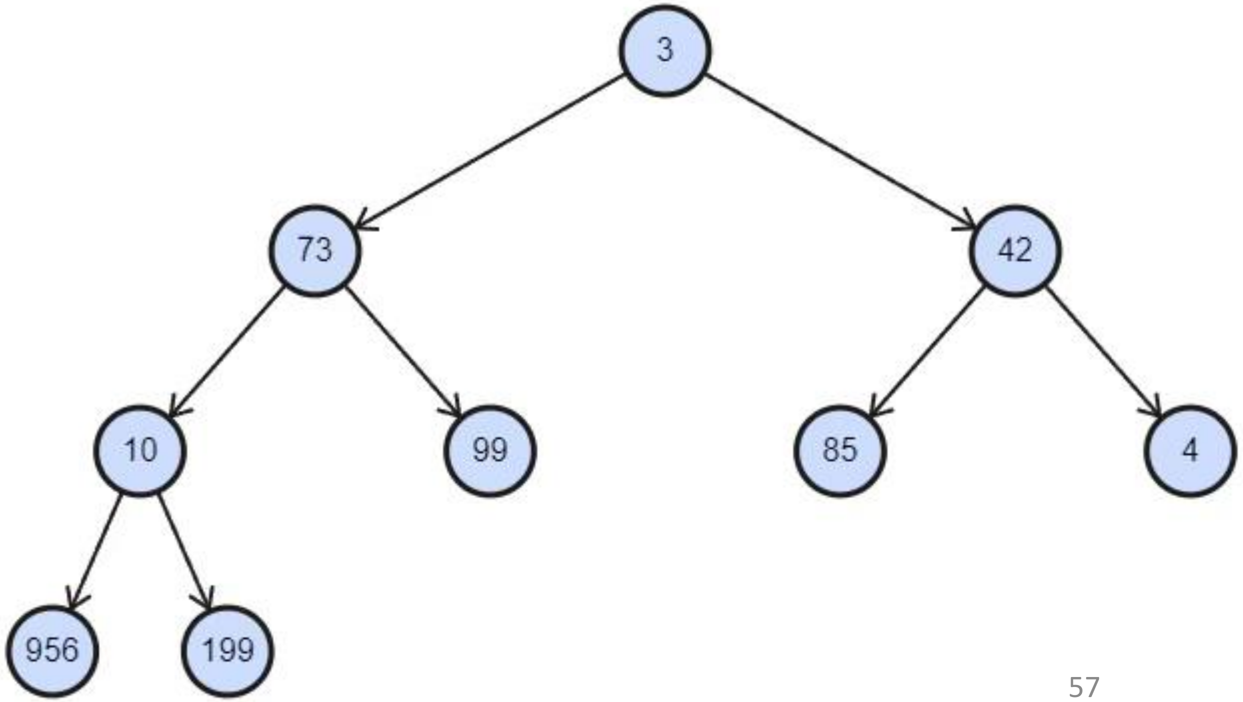
0	1	2	3	4	5	6	7
50	30	20	15	10	8	16	

c

3	73	42	10	99	85	4	956	199

Build a Max Heap

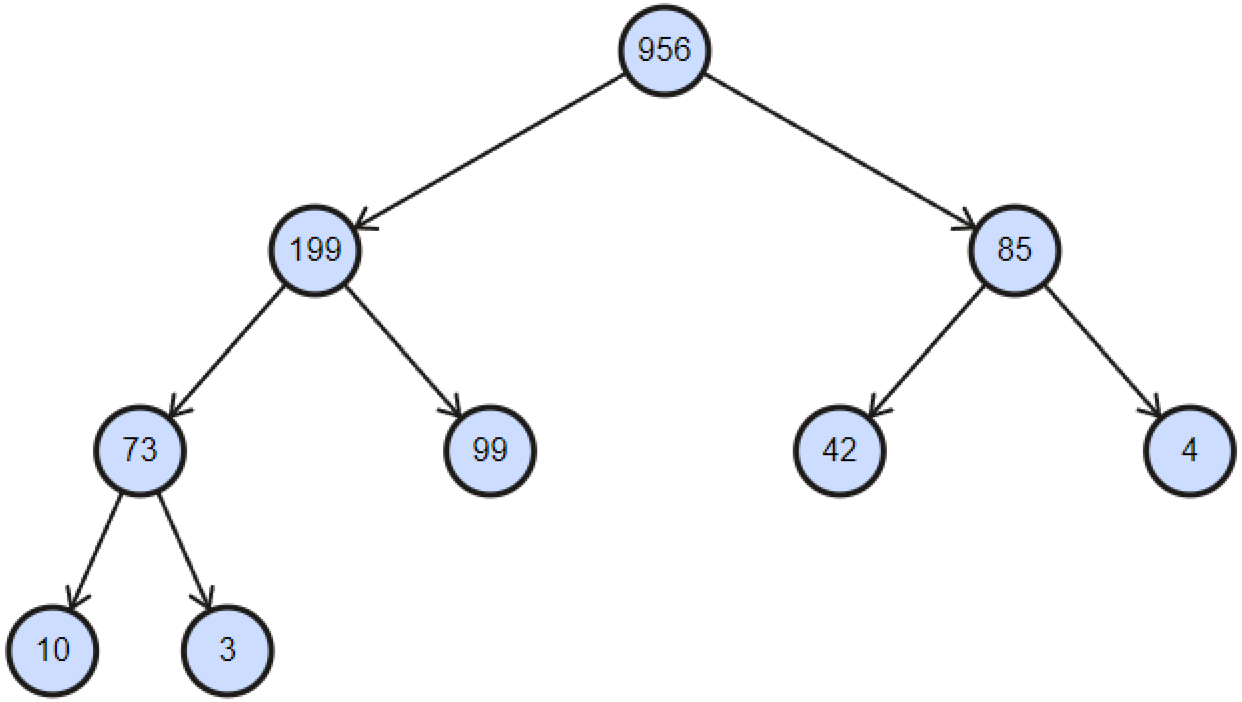




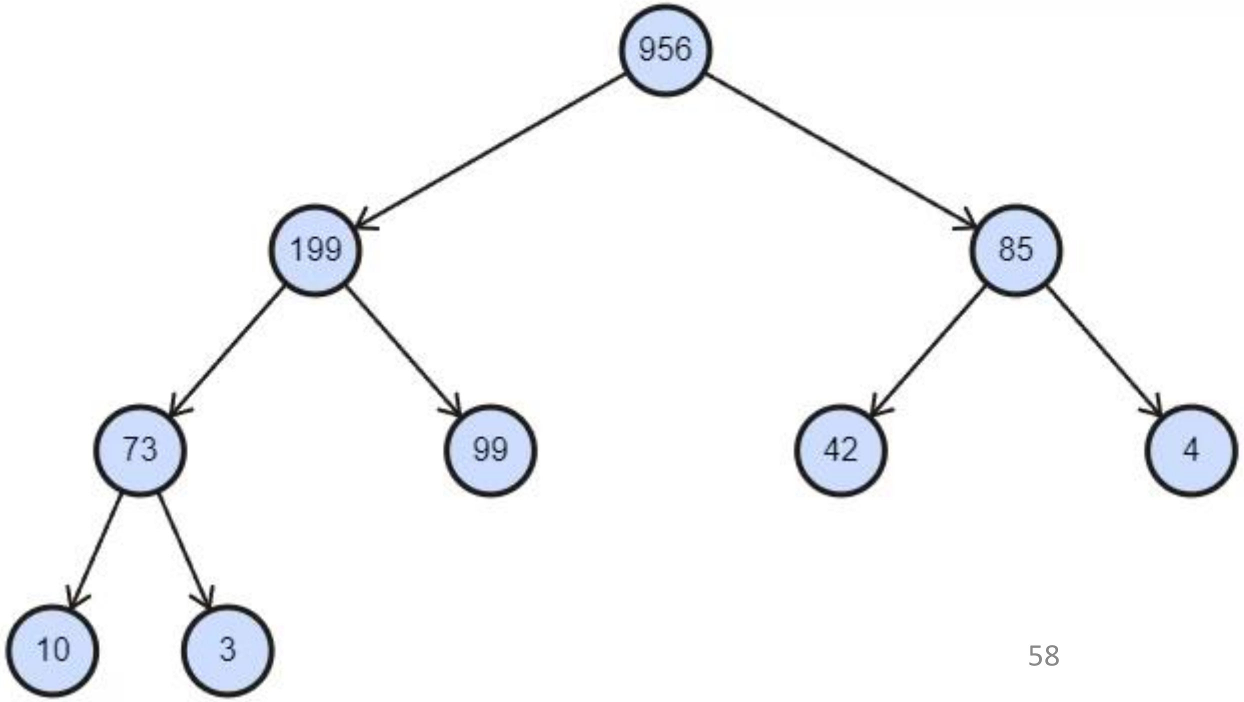
D

956	199	85	73	99	42	4	10	3

Insert a New Value in a Max Heap

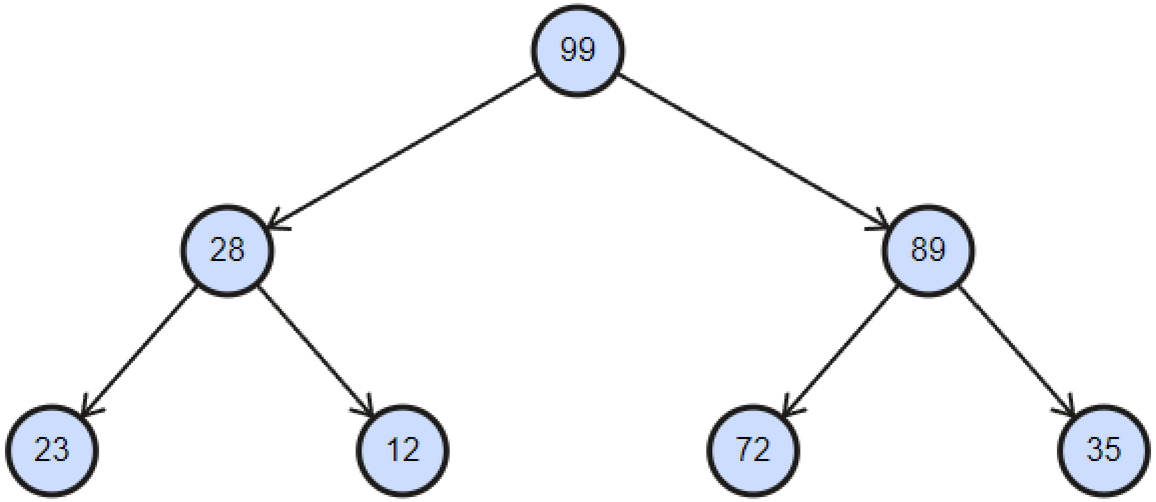


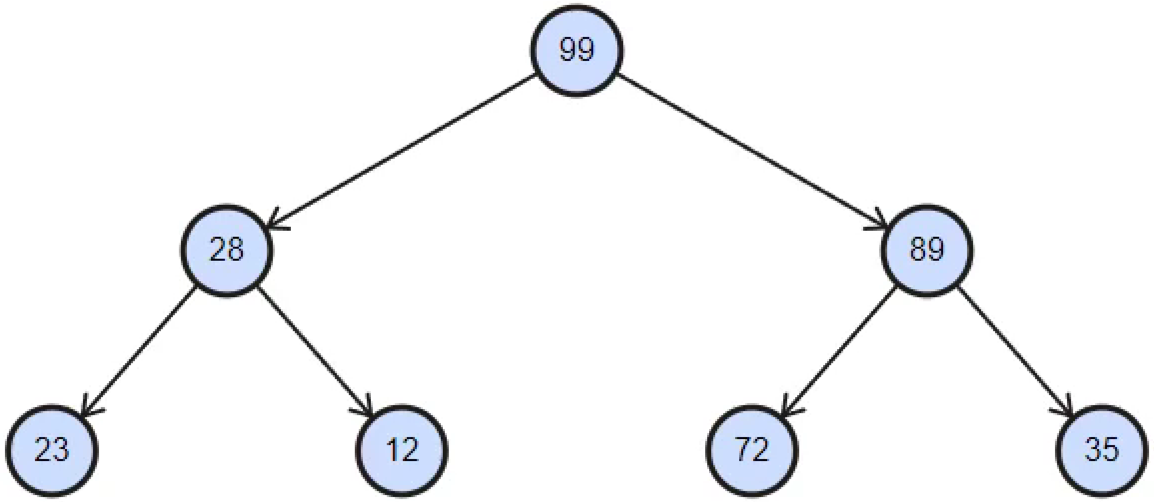
Insert 200



E

Delete from Max Heap



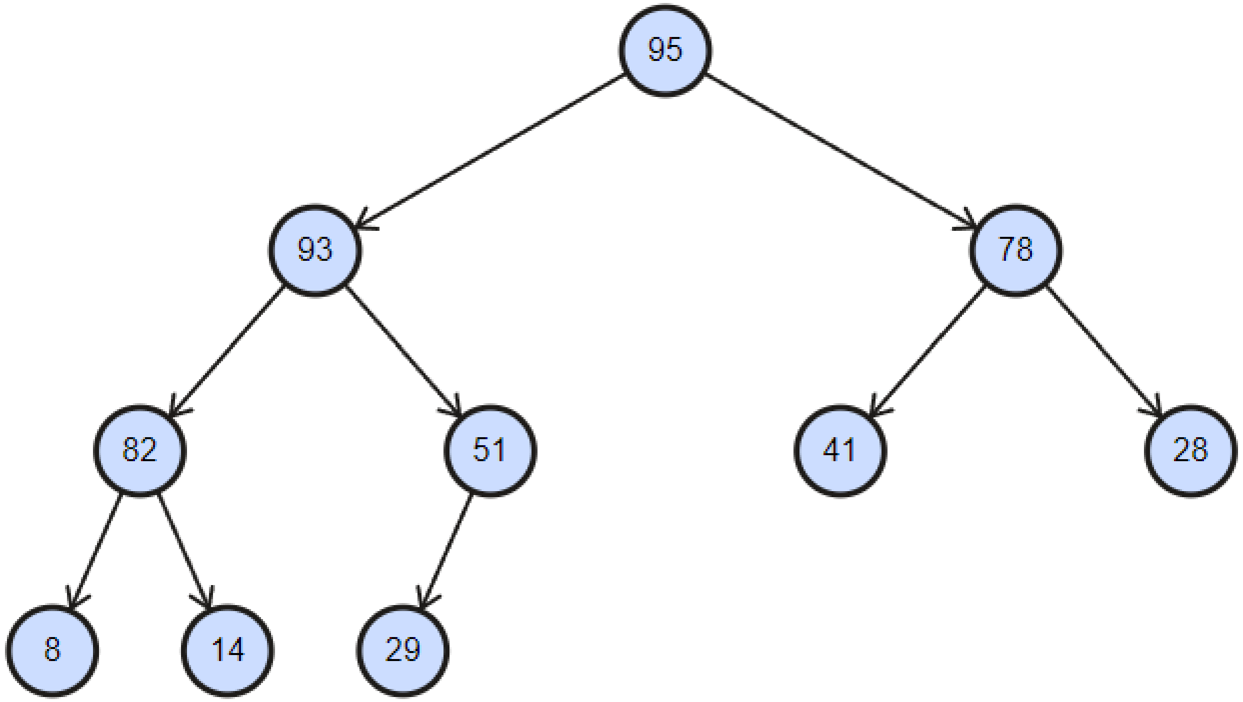


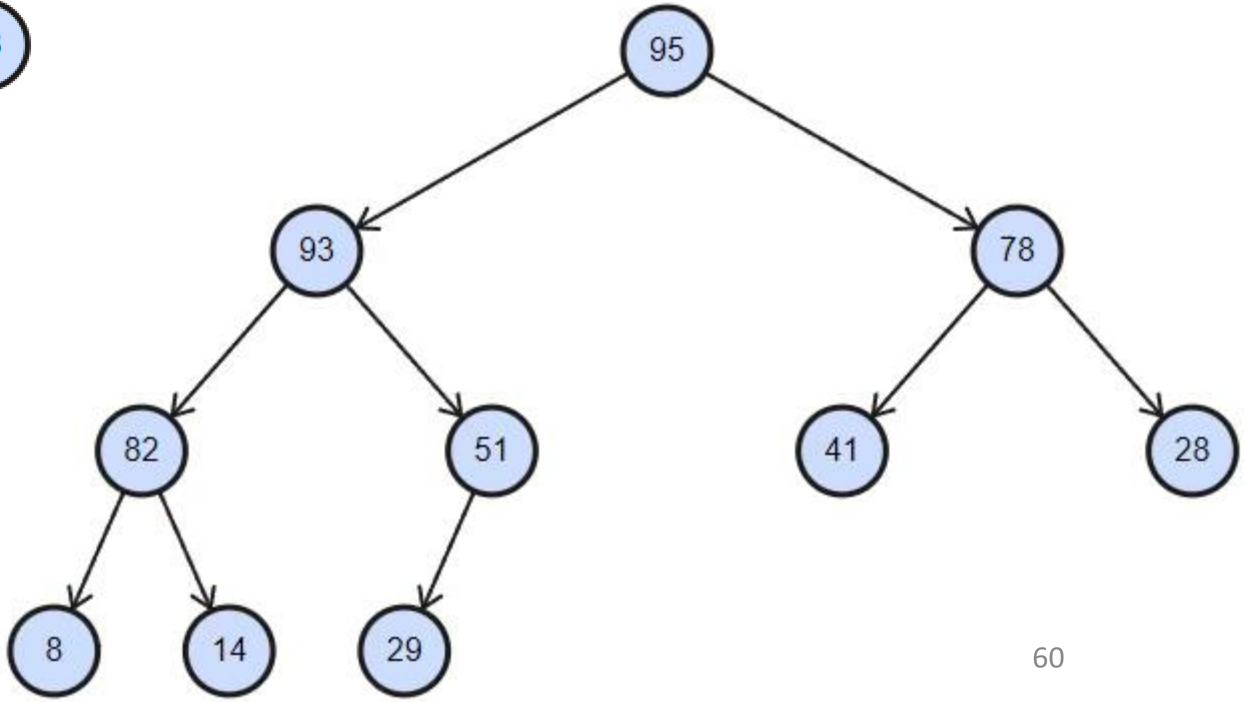
Delete root

F

95	93	78	82	51	41	28	8	14	29

Delete from Max Heap



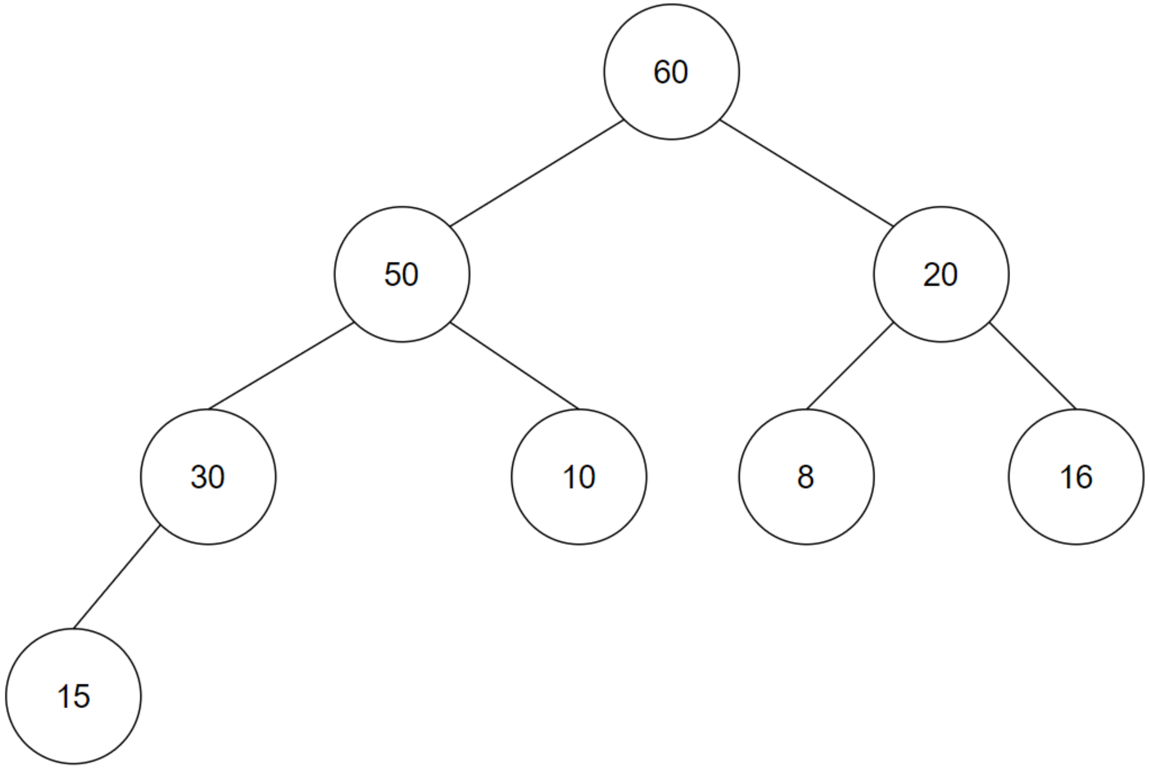


Sorting Using a Max Heap

What happens if we keep deleting the root from the heap?

When we delete the root, we are just taking it out of index 0 – we are not removing array element 0 – doing so just allows us to put the last element of the array in index 0.

What if we stash that 60 in that spot we just emptied?



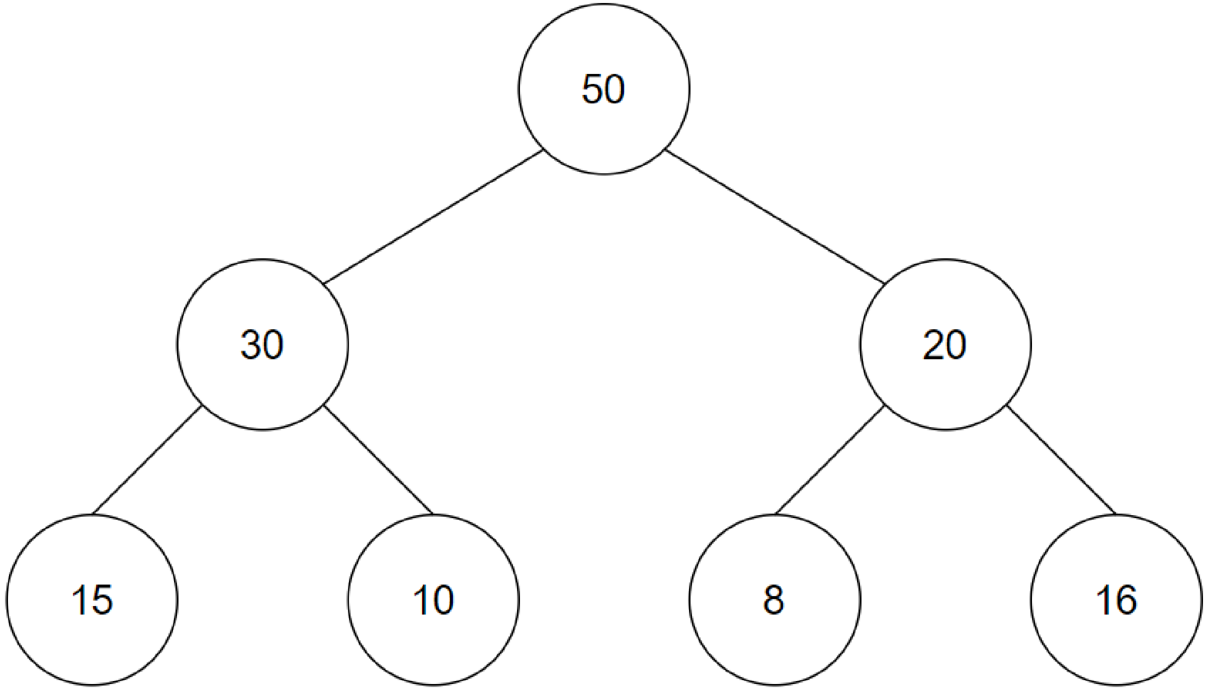
0	1	2	3	4	5	6	7
60	50	20	30	10	8	16	15

Sorting Using a Max Heap

So after heapifying, our heap is back to a good state and we make note that our array ends at element 6.

We are just using the space in element 7 to store that 60.

So what happens if we repeat this process?



0	1	2	3	4	5	6	7
50	30	20	15	10	8	16	60

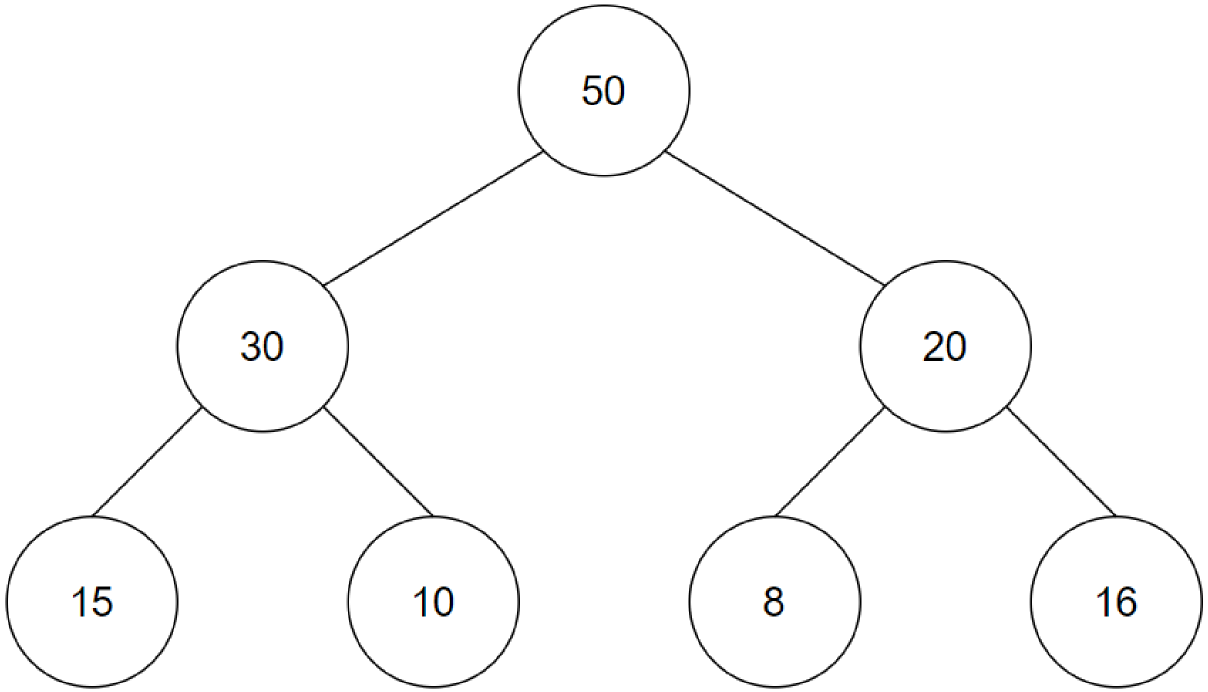
Sorting Using a Max Heap

We swap the 50 and the 16.

Heapify time!!

We make note that the end of our array is now element 5.

This also means that we only have 6 elements in our heap.

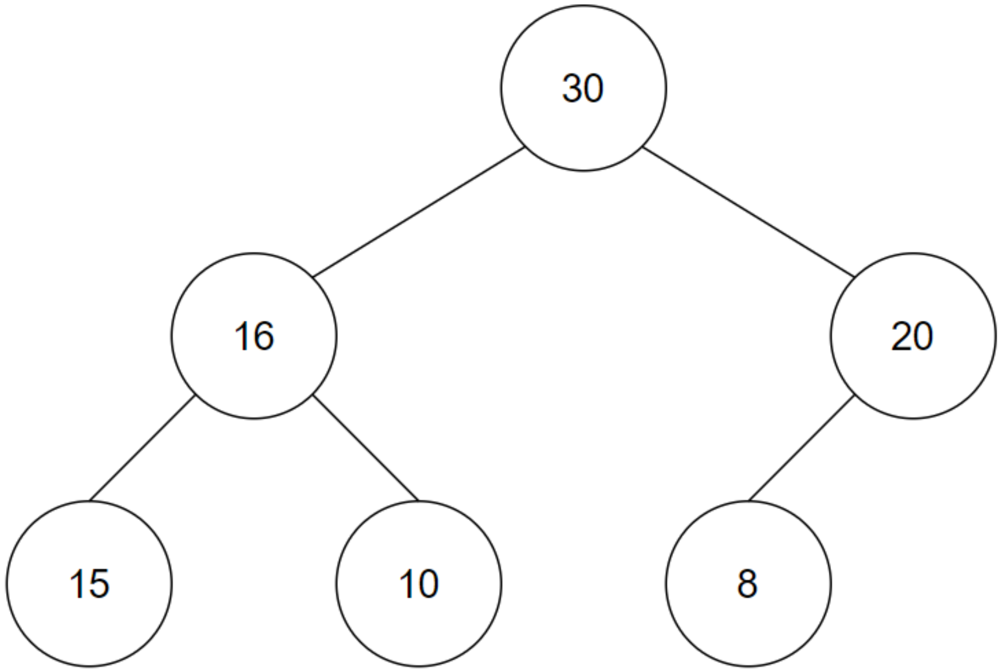


0	1	2	3	4	5	6	7
16	30	20	15	10	8	50	60

Sorting Using a Max Heap

Repeat.

Swap 30 and 8 and heapify.



0	1	2	3	4	5	6	7
30	16	20	15	10	8	50	60

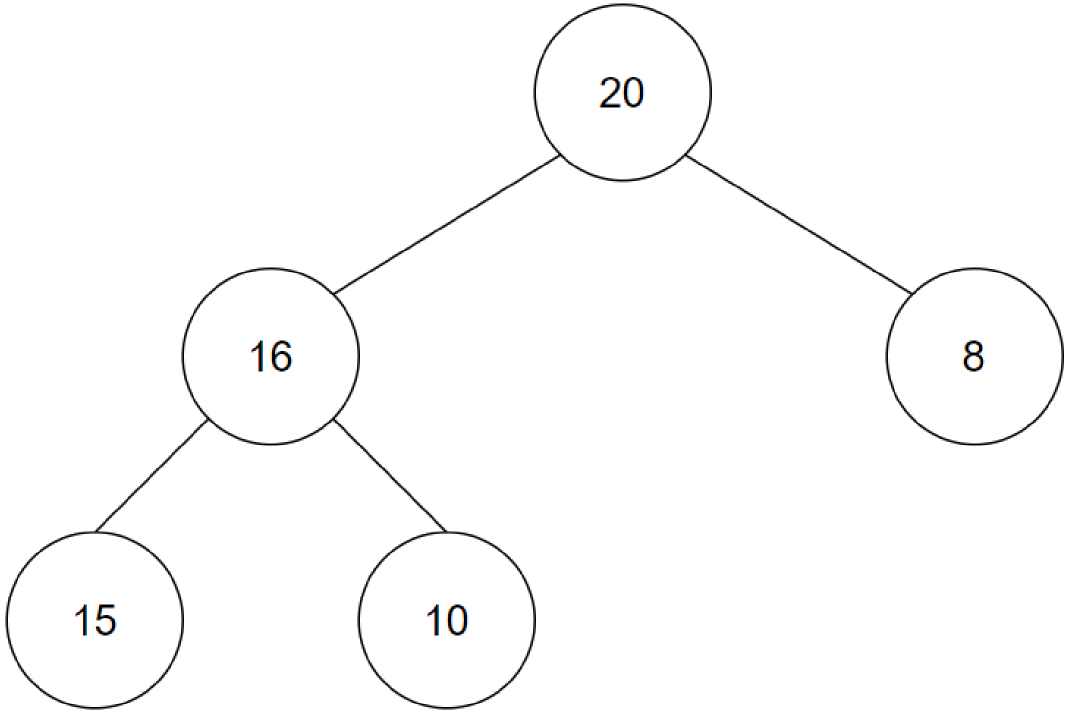
Sorting Using a Max Heap

Repeat.

Swap 20 and 10

Why not swap 20 and 8?

Heapify.



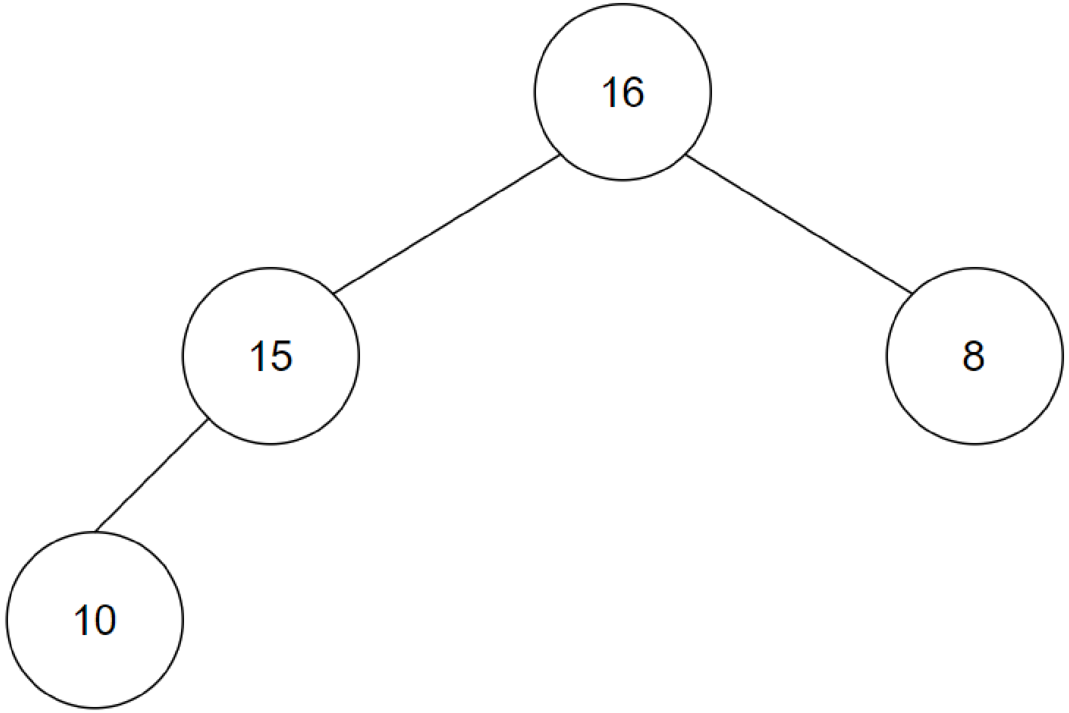
0	1	2	3	4	5	6	7
20	16	8	15	10	30	50	60

Sorting Using a Max Heap

Repeat.

Swap 16 and 10.

Heapify.



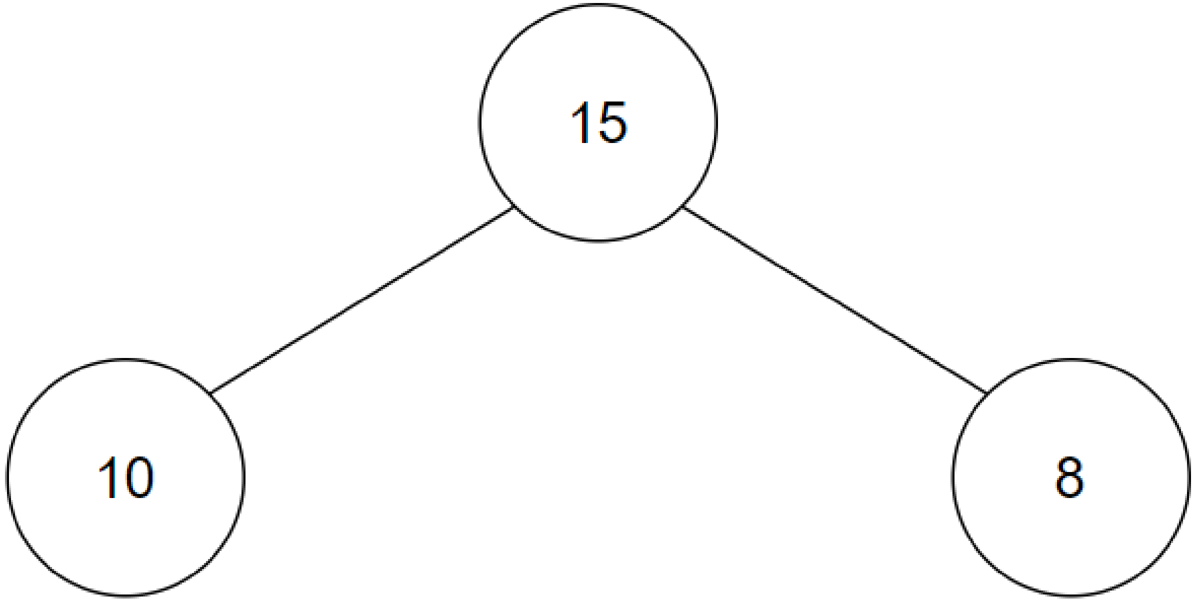
0	1	2	3	4	5	6	7
16	15	8	10	20	30	50	60

Sorting Using a Max Heap

Repeat.

Swap 8 and 15.

Heapify



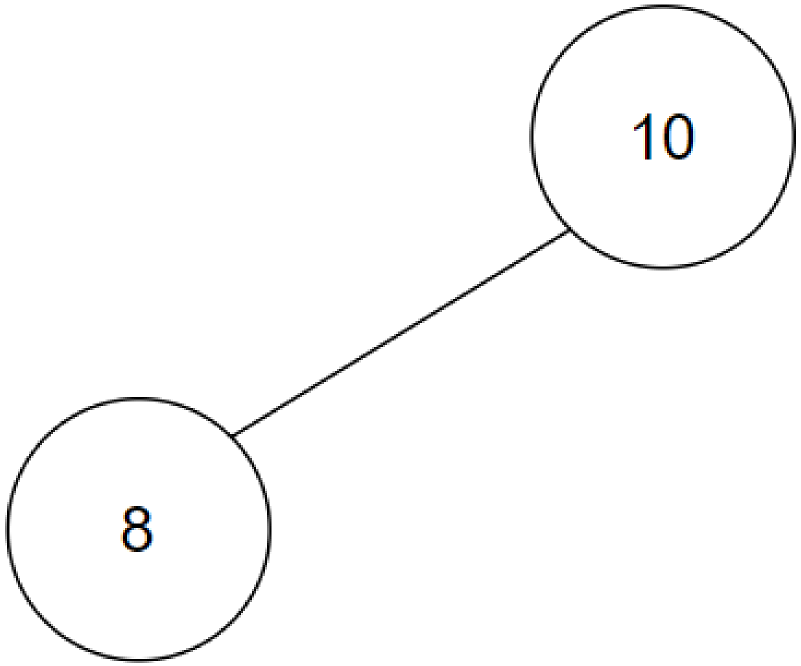
0	1	2	3	4	5	6	7
15	10	8	16	20	30	50	60

Sorting Using a Max Heap

Repeat.

Swap 10 and 8.

Do we need to heapify?



0	1	2	3	4	5	6	7
10	8	15	16	20	30	50	60

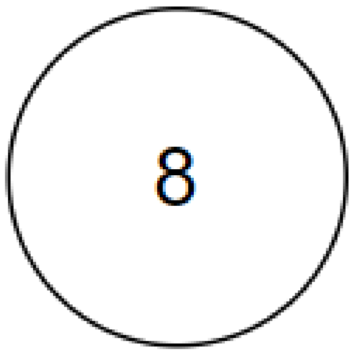
Sorting Using a Max Heap

No.

Final node maintains the heap property.

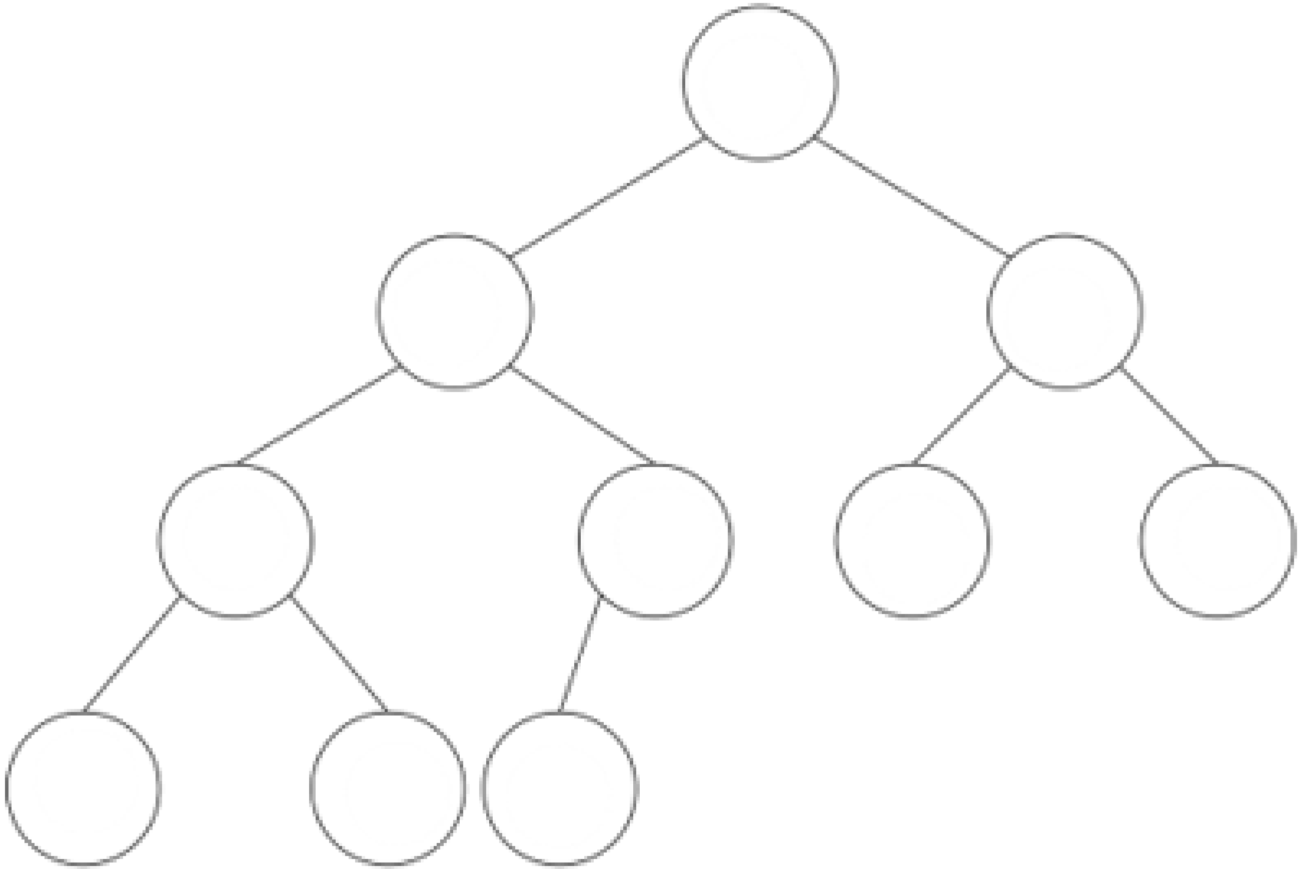
What is in our array now?

It is sorted.



0	1	2	3	4	5	6	7
8	10	15	16	20	30	50	60

1. Create the tree given the array.
2. Sort the array using a min heap delete



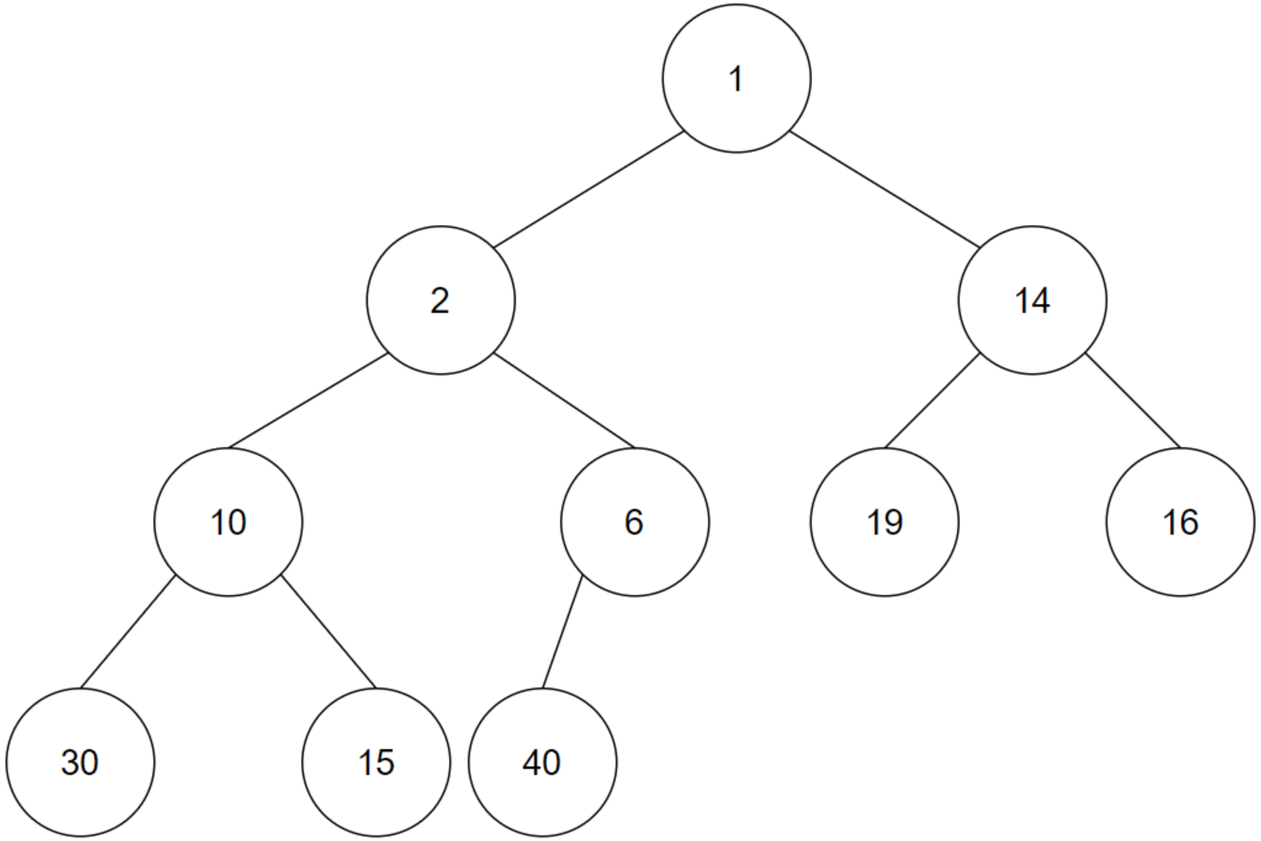
0	1	2	3	4	5	6	7	8	9
1	2	14	10	6	19	16	30	15	40

Sorting Using a Min Heap

Does the process work differently for a Min Heap?

Will the result be different?

Let's try it.

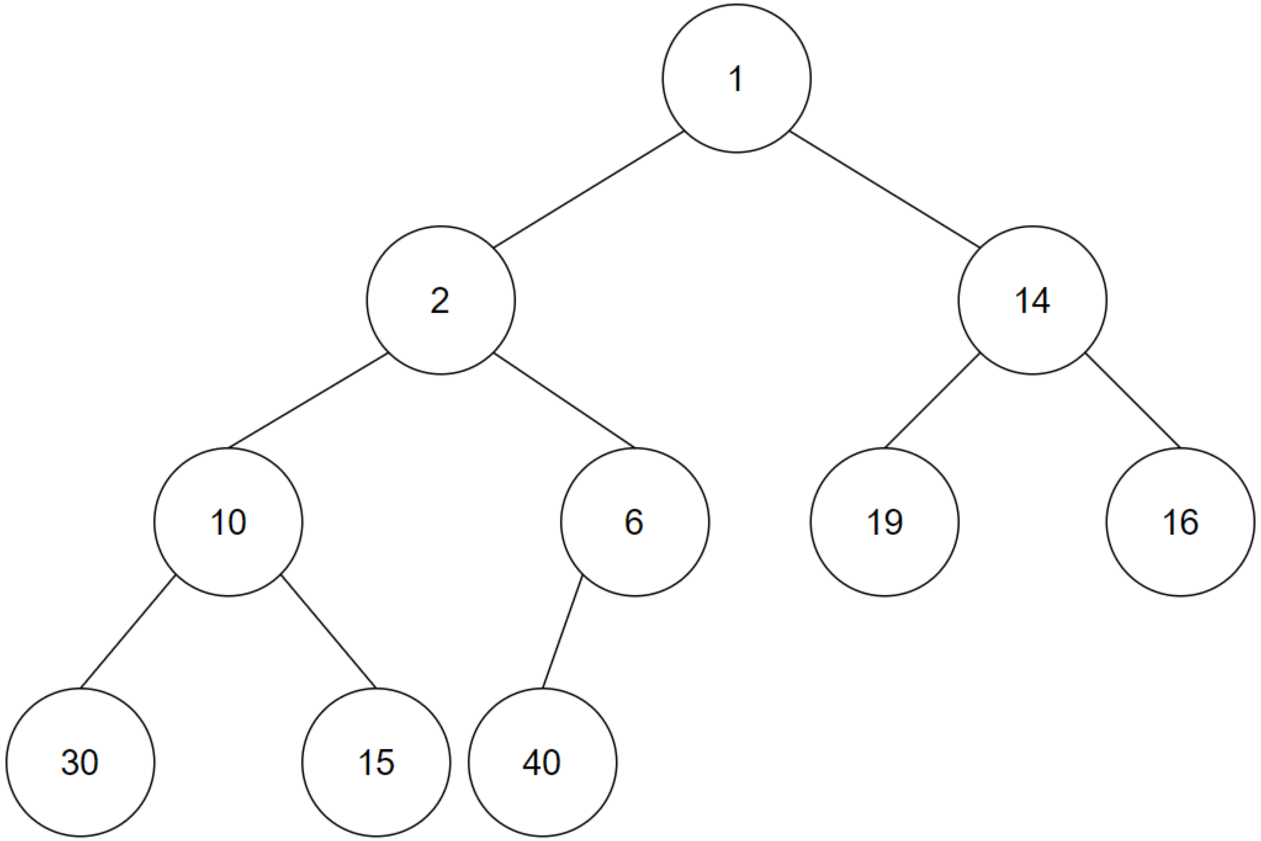


0	1	2	3	4	5	6	7	8	9
1	2	14	10	6	19	16	30	15	40

Sorting Using a Min Heap

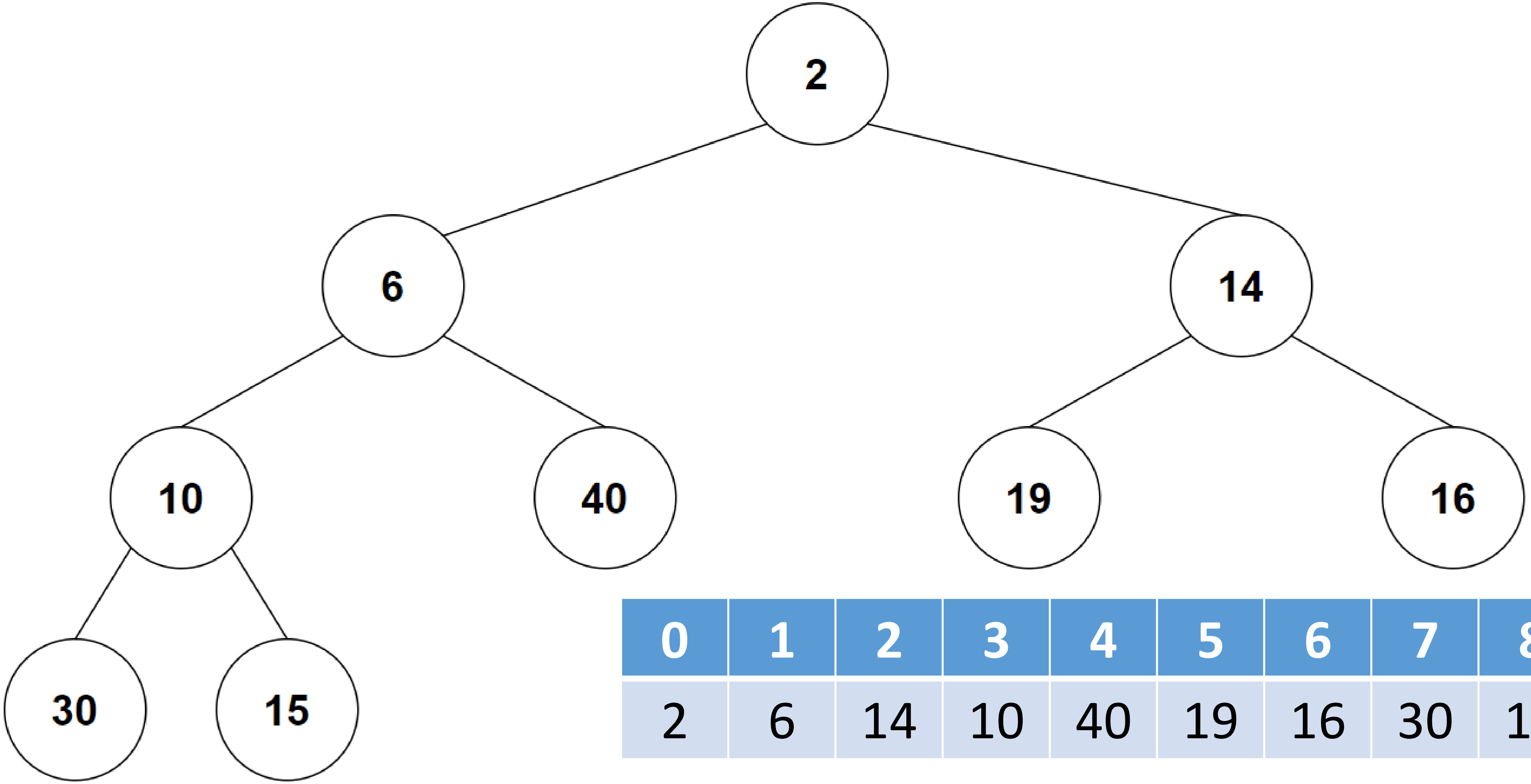
Swap 1 and 40

Heapify!



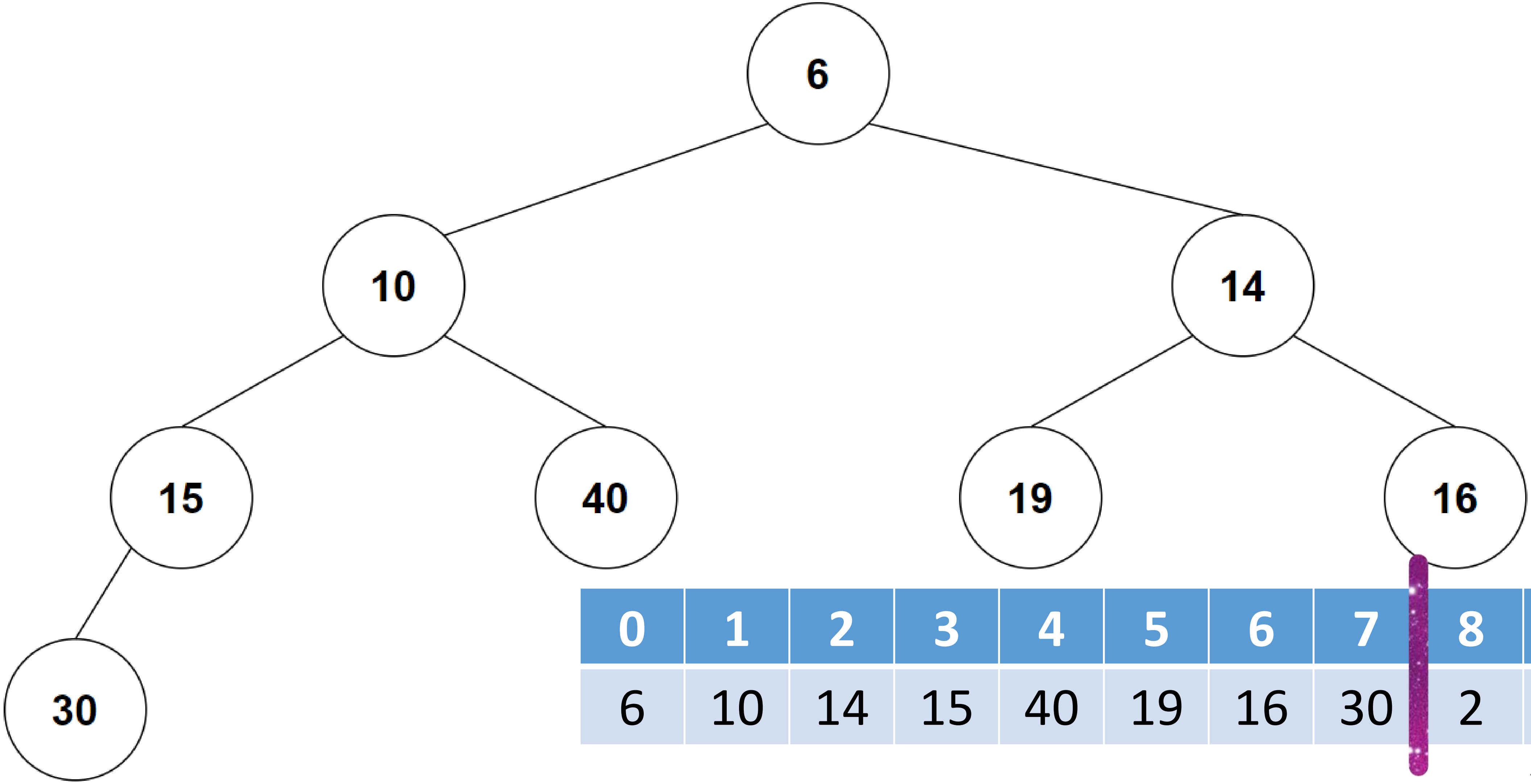
0	1	2	3	4	5	6	7	8	9
40	2	14	10	6	19	16	30	15	1

Sorting Using a Min Heap



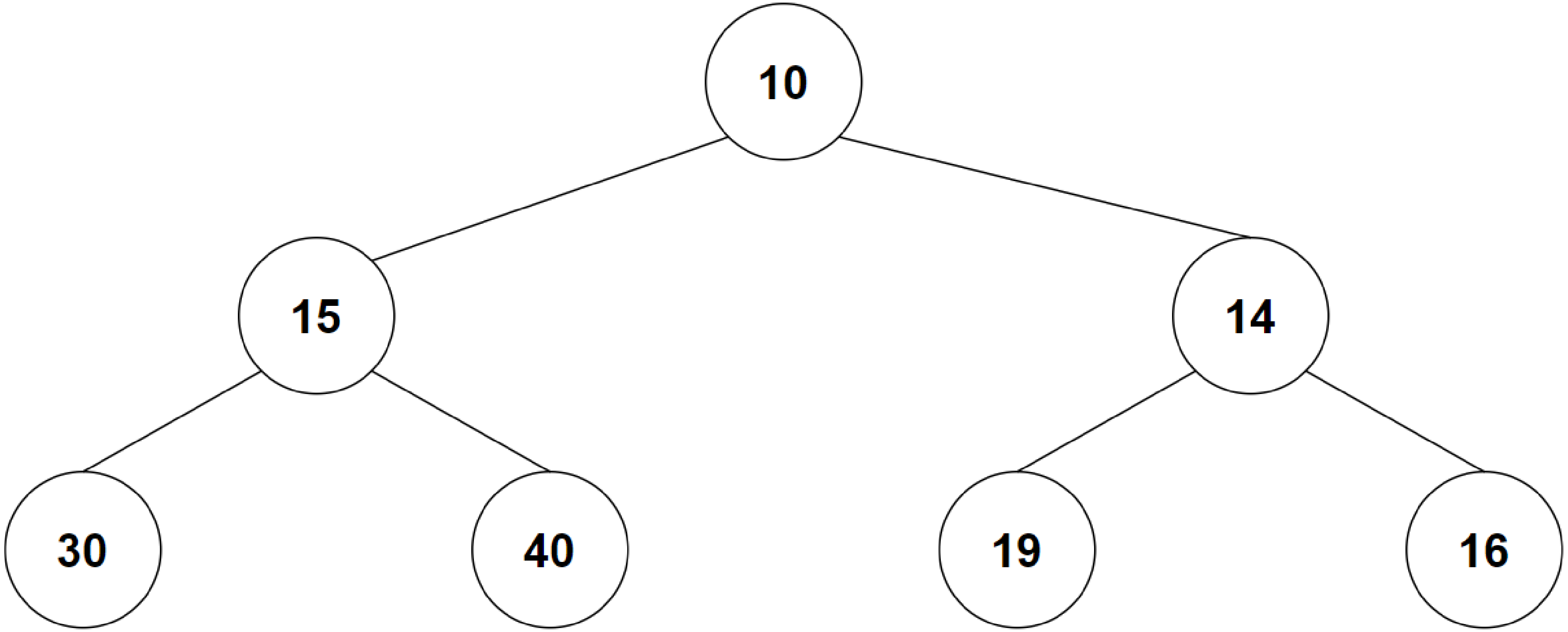
0	1	2	3	4	5	6	7	8	9
2	6	14	10	40	19	16	30	15	1

Sorting Using a Min Heap



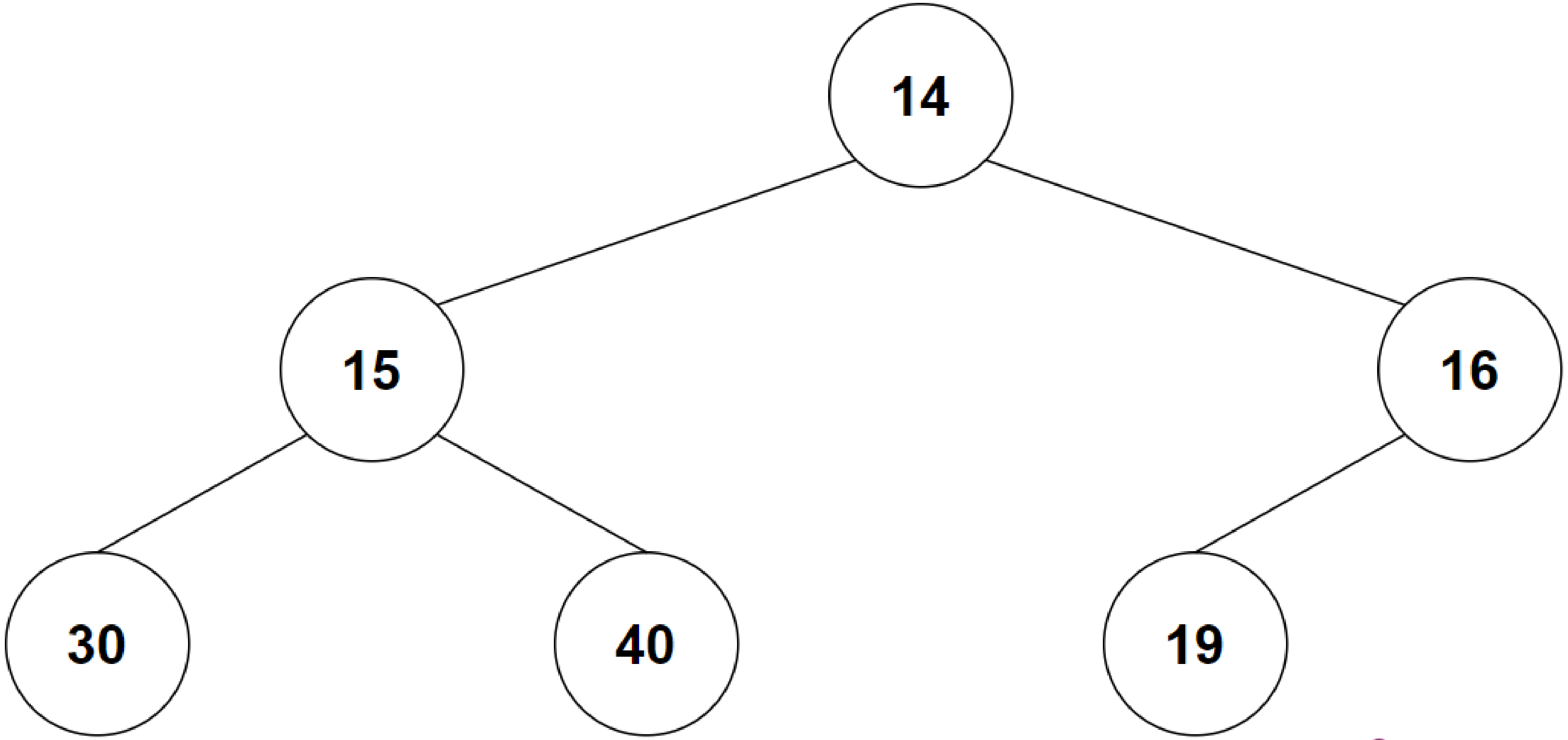
0	1	2	3	4	5	6	7	8	9
6	10	14	15	40	19	16	30	2	1

Sorting Using a Min Heap



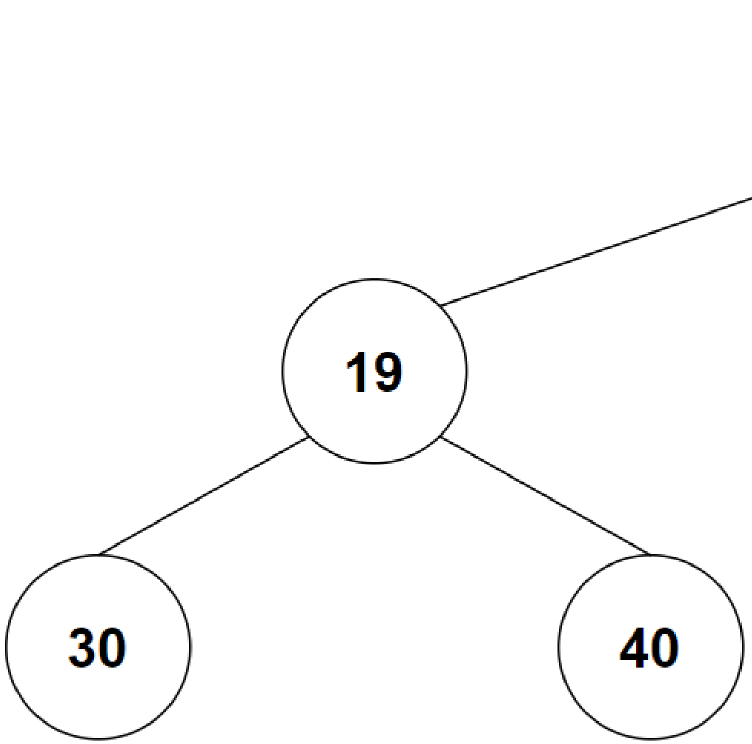
0	1	2	3	4	5	6	7	8	9
10	15	14	30	40	19	16	6	2	1

Sorting Using a Min Heap



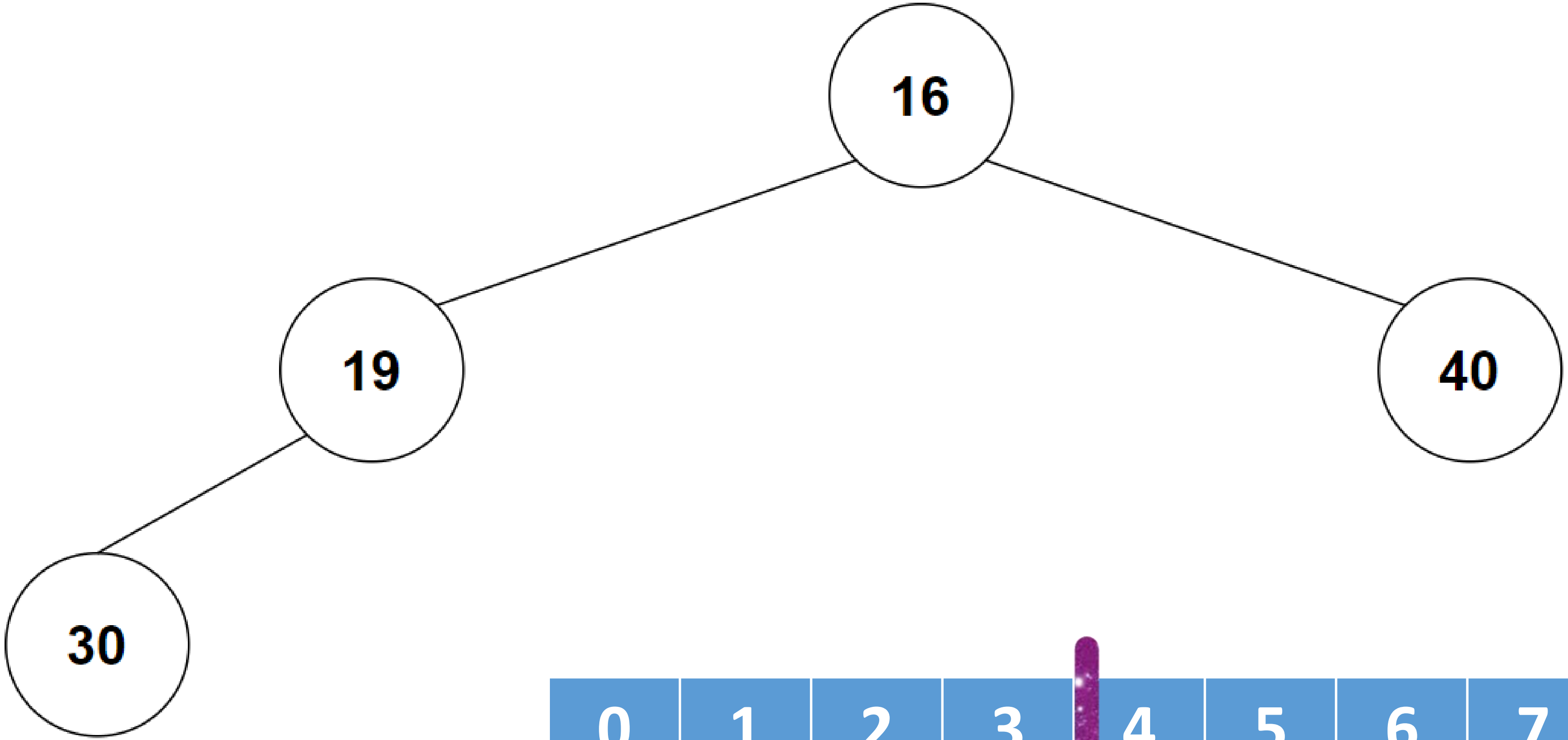
0	1	2	3	4	5	6	7	8	9
14	15	16	30	40	19	10	6	2	1

Sorting Using a Min Heap



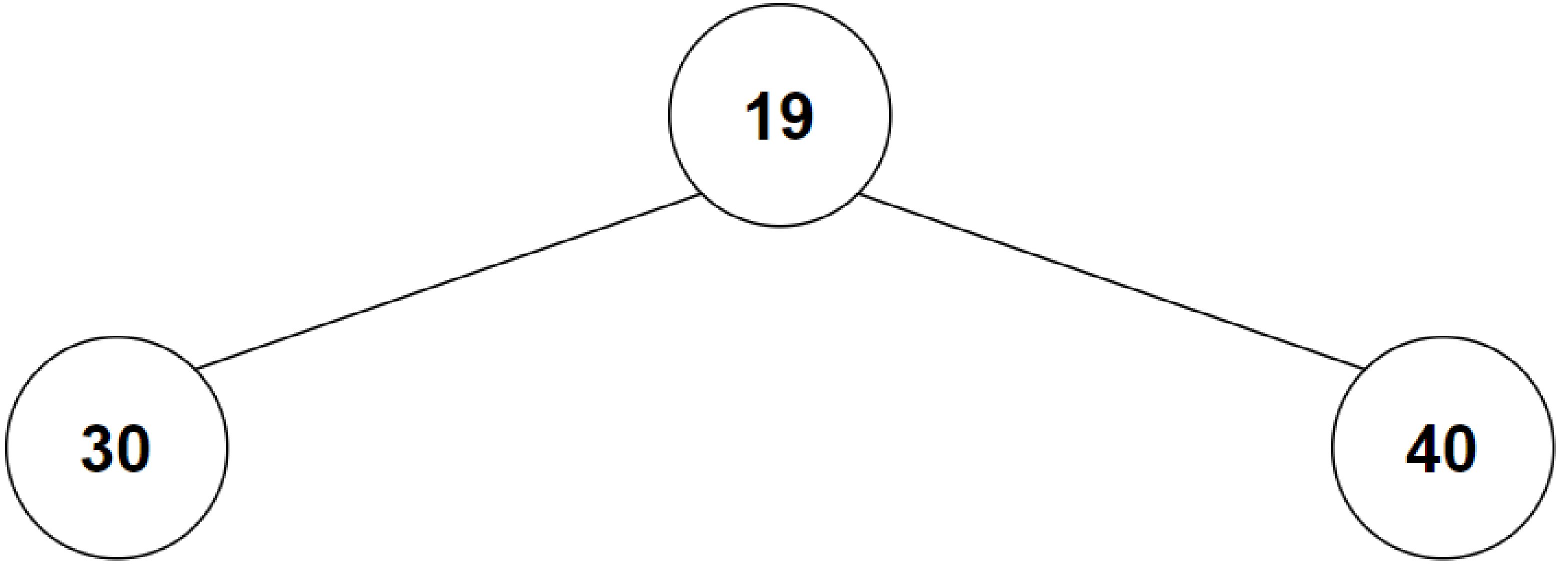
0	1	2	3	4	5	6	7	8	9
15	19	16	30	40	14	10	6	2	1

Sorting Using a Min Heap



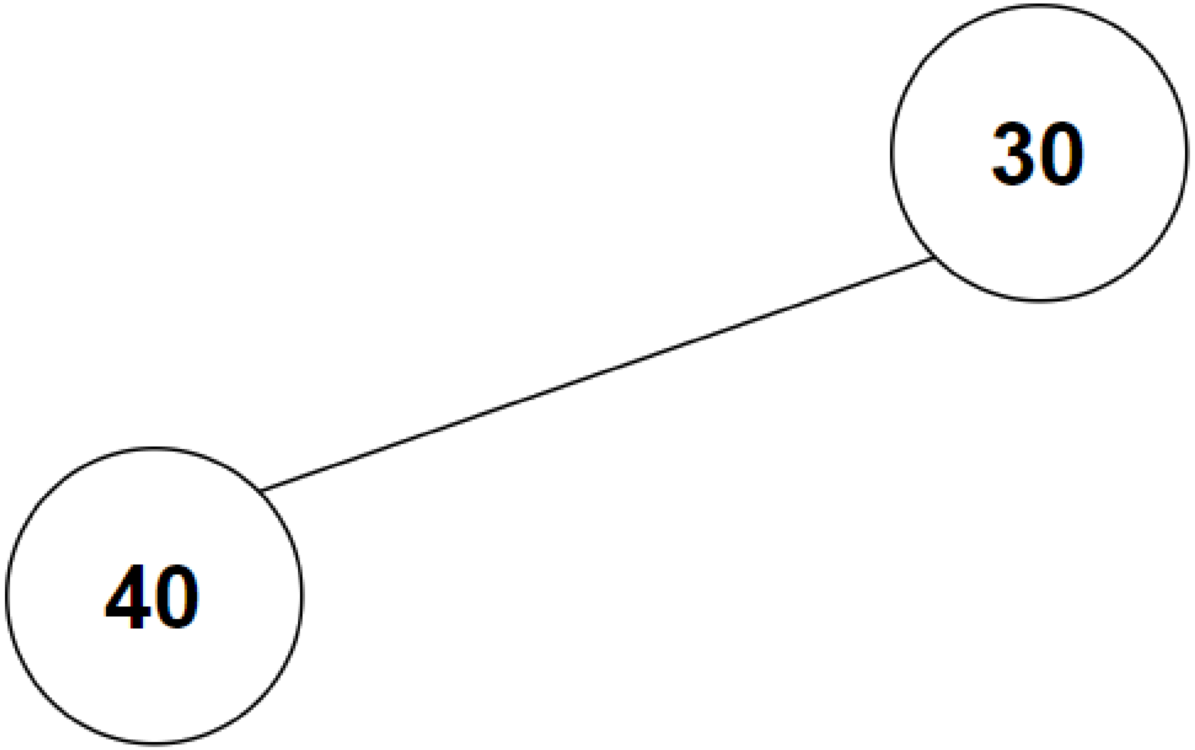
0	1	2	3	4	5	6	7	8	9
16	19	40	30	15	14	10	6	2	1

Sorting Using a Min Heap



0	1	2	3	4	5	6	7	8	9
19	30	40	16	15	14	10	6	2	1

Sorting Using a Min Heap



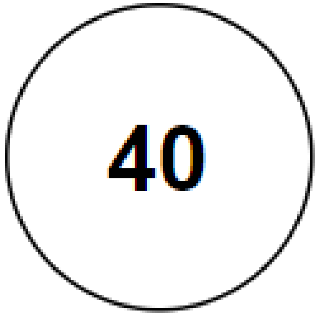
0	1	2	3	4	5	6	7	8	9
30	40	19	16	15	14	10	6	2	1

Sorting Using a Min Heap

Summary

Binary Max Heap sorted the list ascending order.

Binary Min Heap sorted the list in descending order.



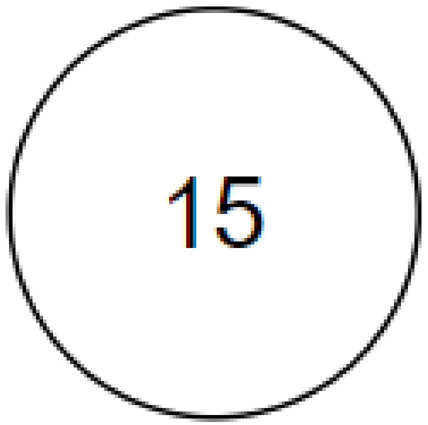
0	1	2	3	4	5	6	7	8	9
40	30	19	16	15	14	10	6	2	1

G

Creating a Max Heap

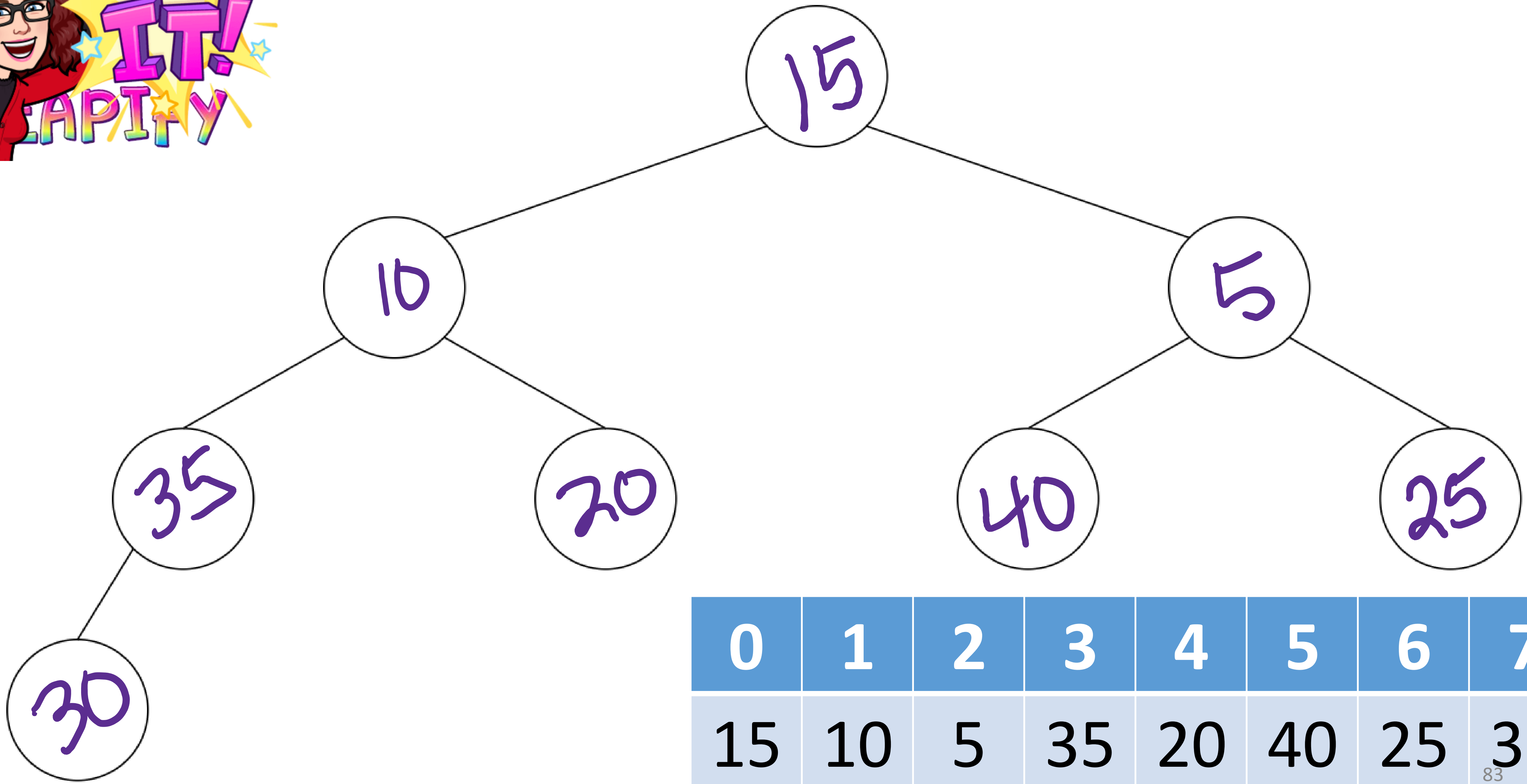
Given this array, create a max heap

0	1	2	3	4	5	6	7
15	10	5	35	20	40	25	30



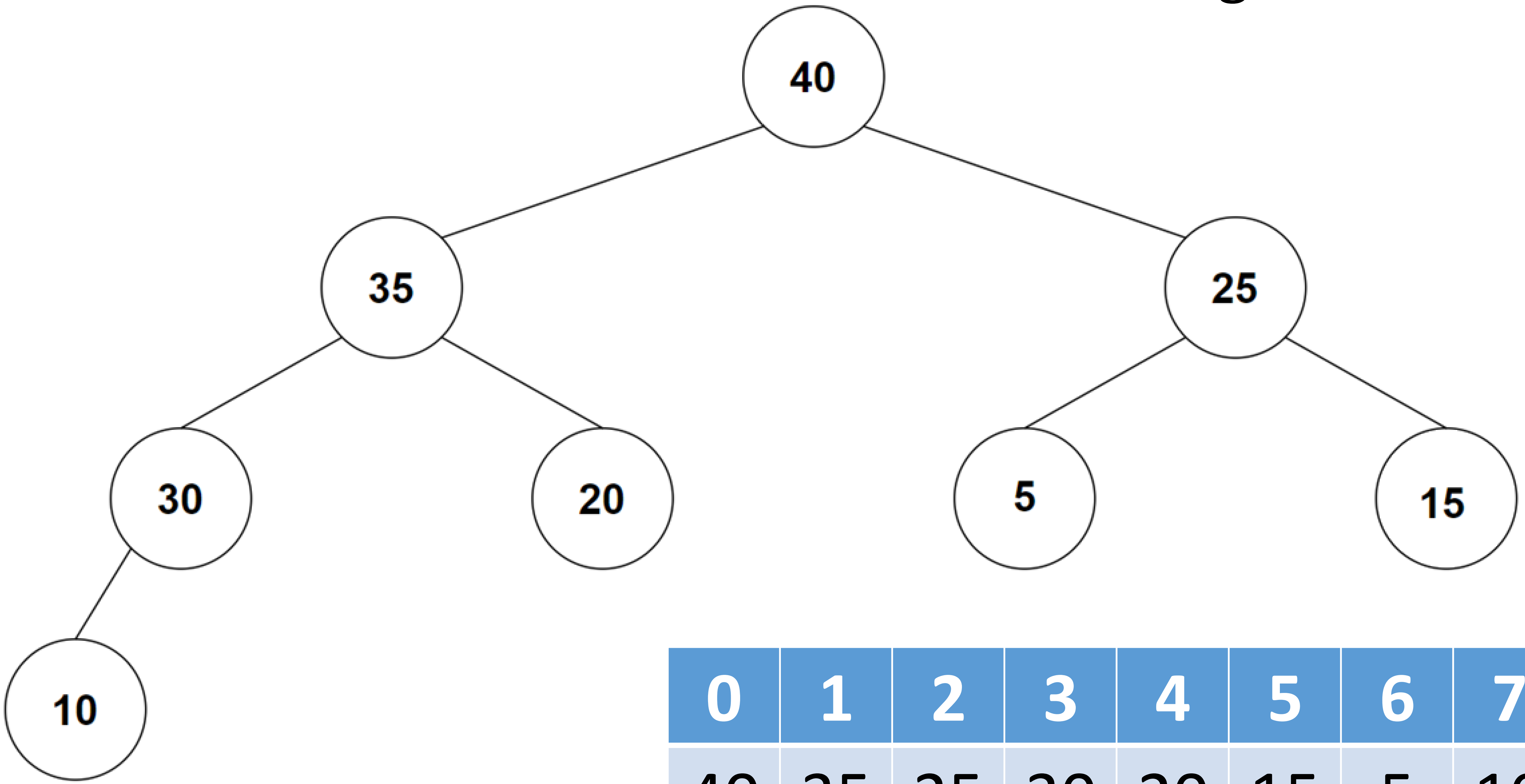
Take the 0th element and make it the root.

Creating a Max Heap



0	1	2	3	4	5	6	7
15	10	5	35	20	40	25	30

Creating a Max Heap



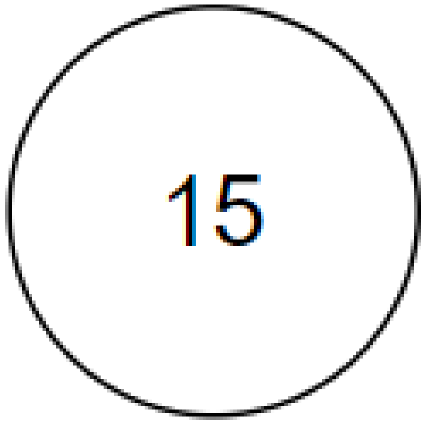
0	1	2	3	4	5	6	7
40	35	25	30	20	15	5	10

H

Creating a Min Heap

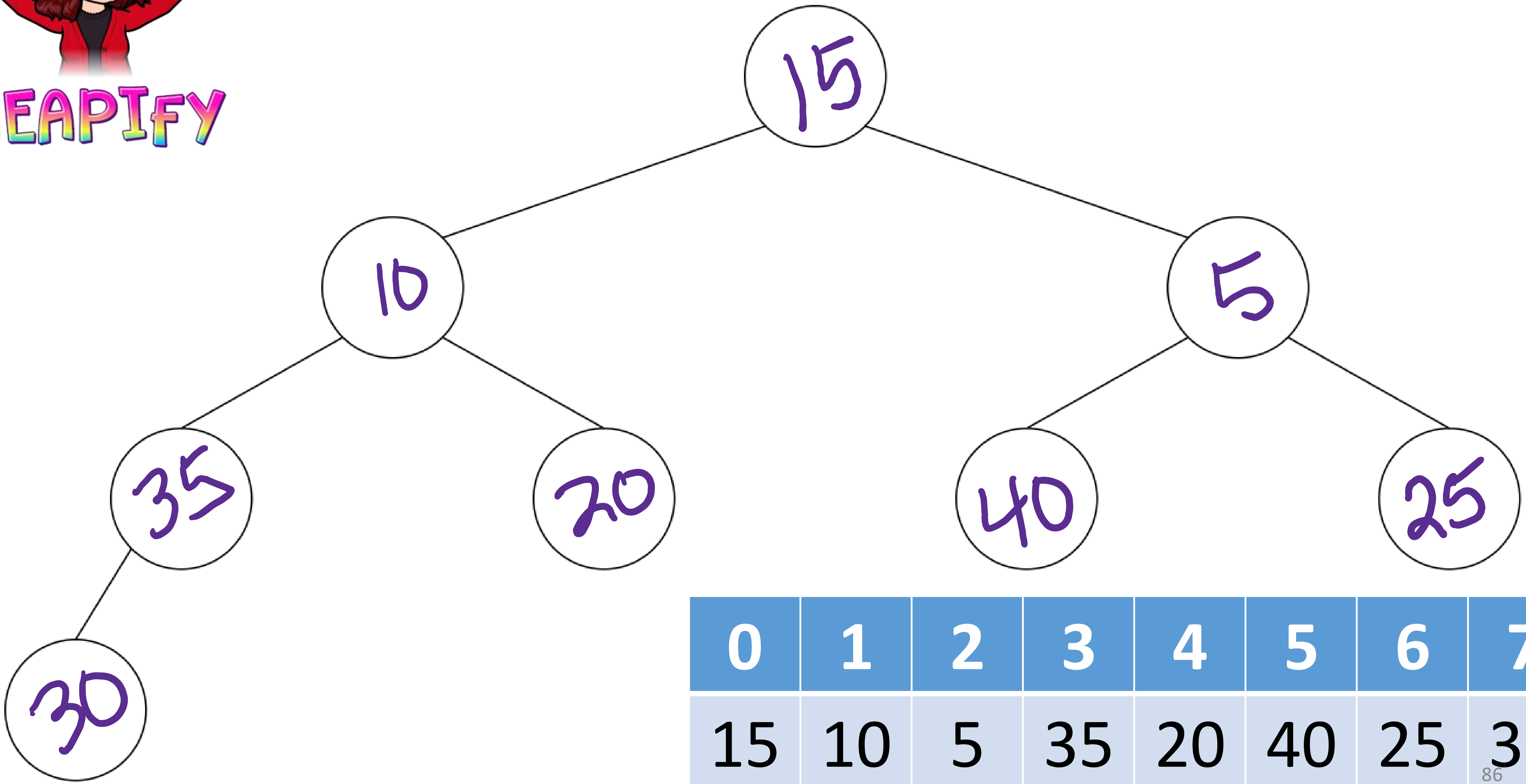
Given this array, create a min heap

0	1	2	3	4	5	6	7
15	10	5	35	20	40	25	30



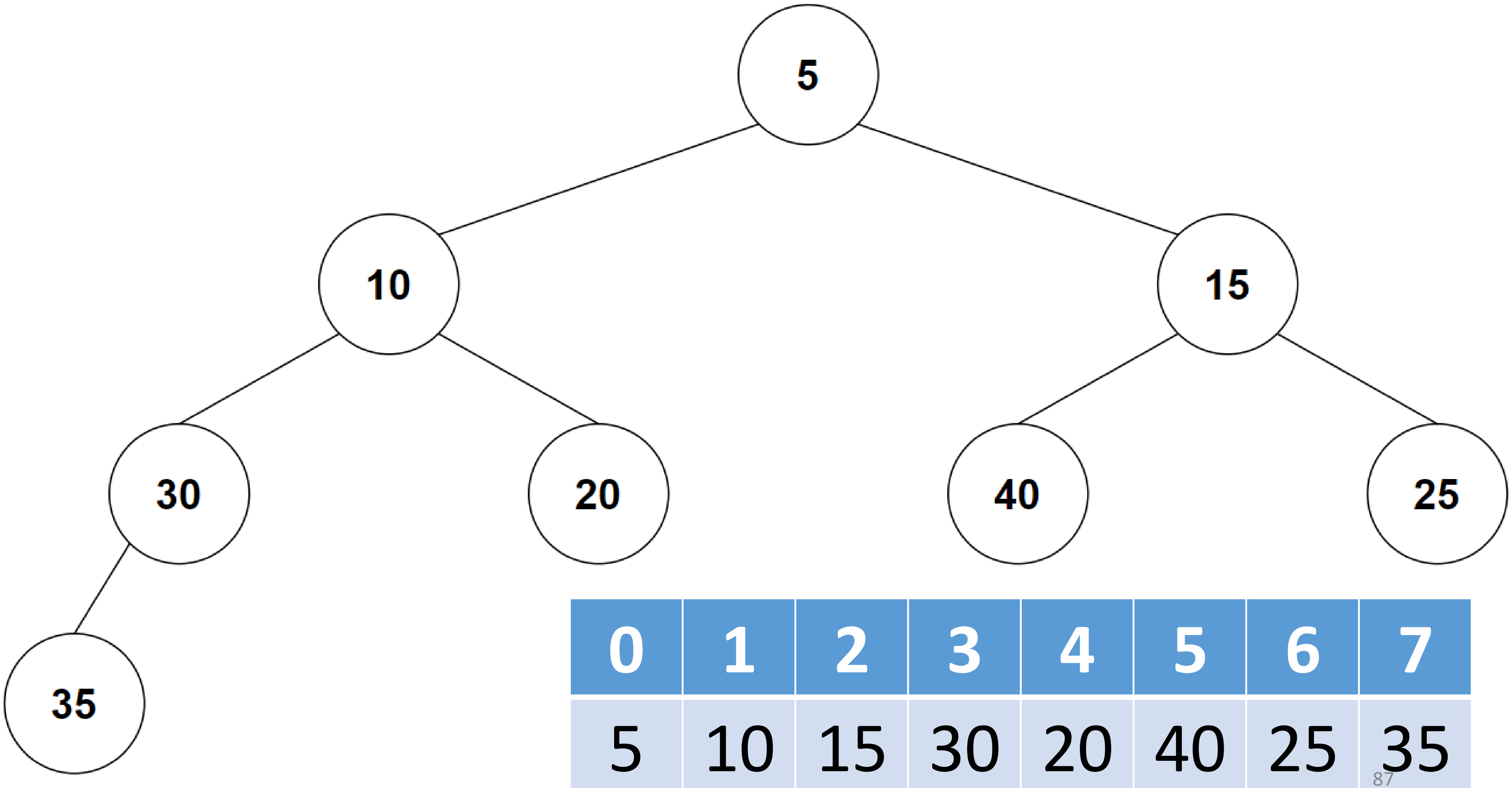
Take the 0th element and make it the root.

Creating a Min Heap



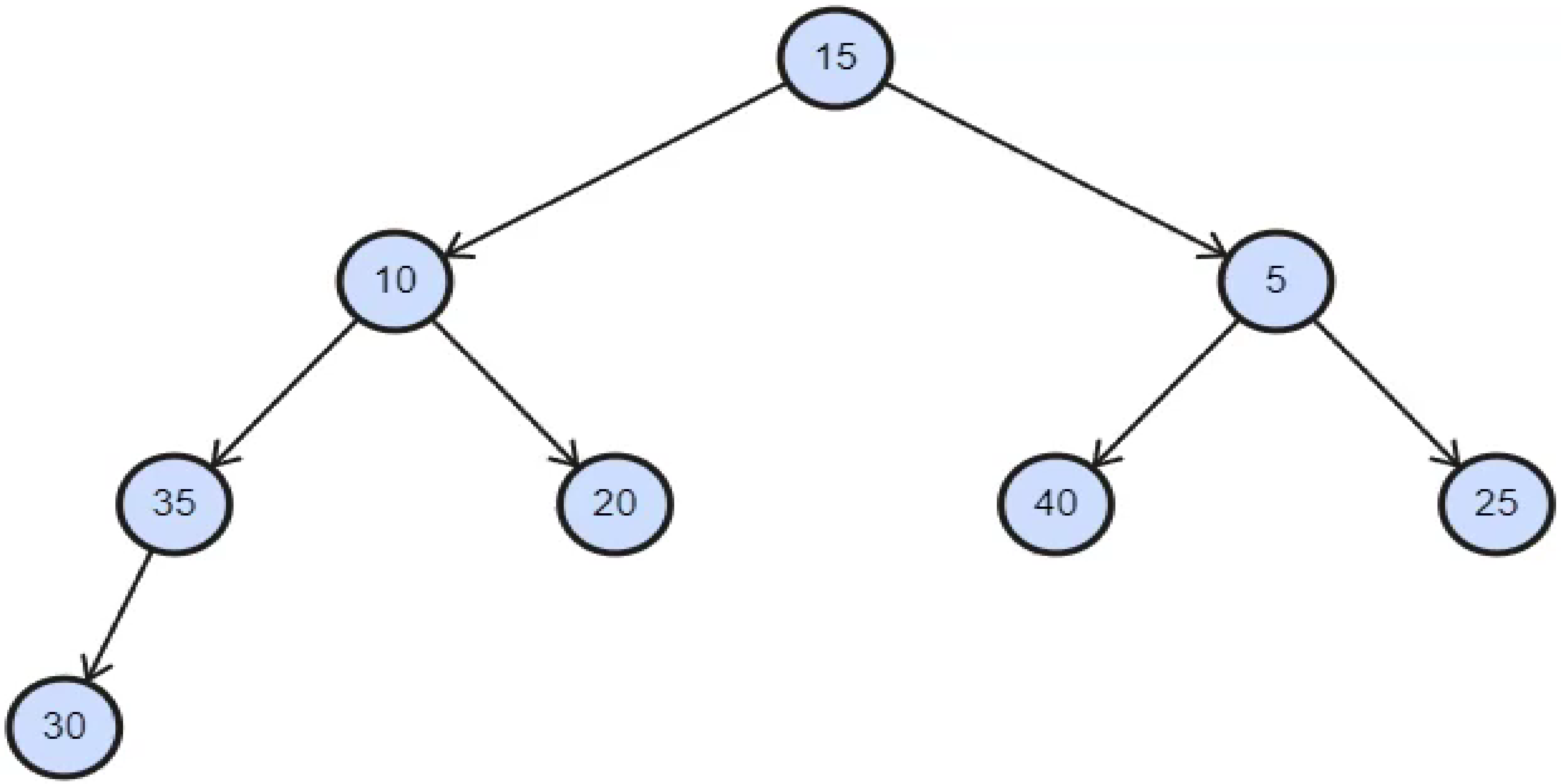
0	1	2	3	4	5	6	7
15	10	5	35	20	40	25	30

Creating a Min Heap



0	1	2	3	4	5	6	7
5	10	15	30	20	40	25	35

Sorting Using a Max Heap



Time Complexity

What is the time complexity of building a binary heap?

How many elements did we insert?

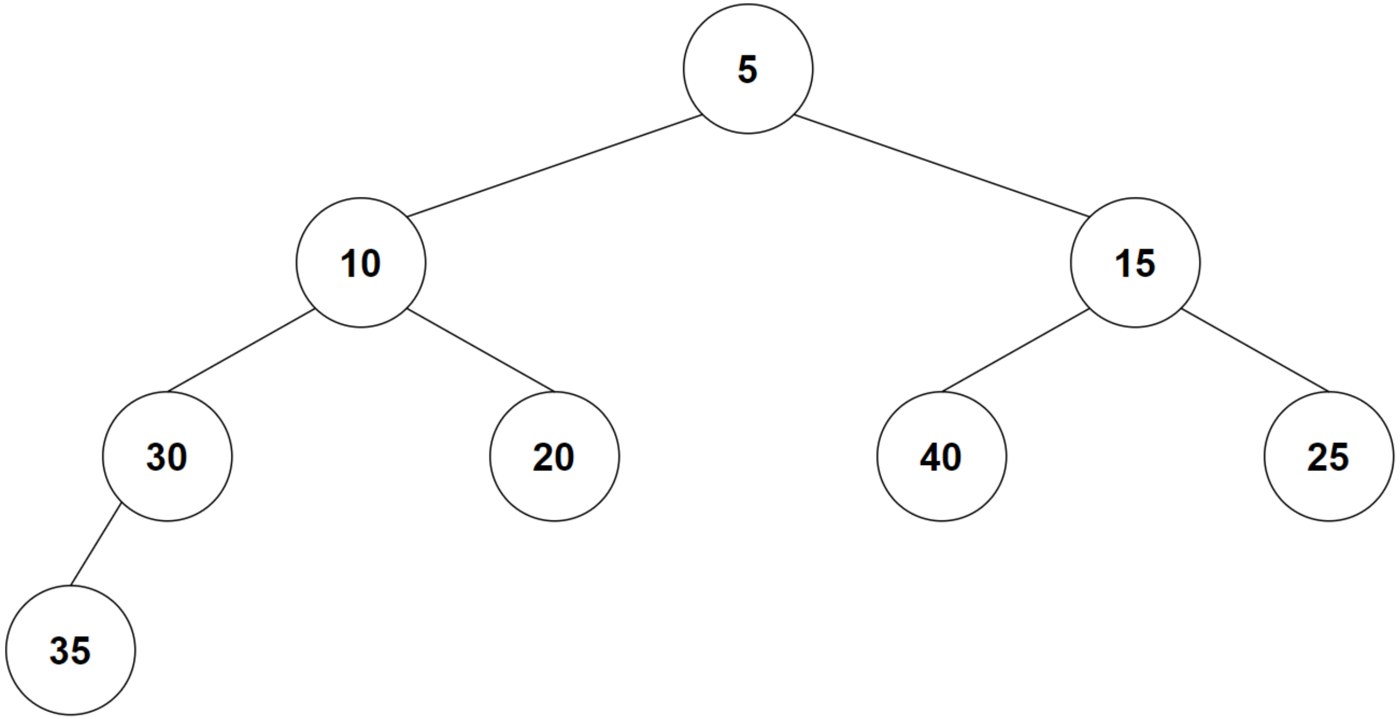
n

The amount of time needed to insert an element into a heap depends on the height of the complete binary tree.

$\log_2 n$

So for worst case, we assume that every element is moved up/down the full height of the tree.

$n\log_2 n$



0	1	2	3	4	5	6	7
5	10	15	30	20	40	25	35

Time Complexity

What is the time complexity of deleting a binary heap?

How many elements did we delete?

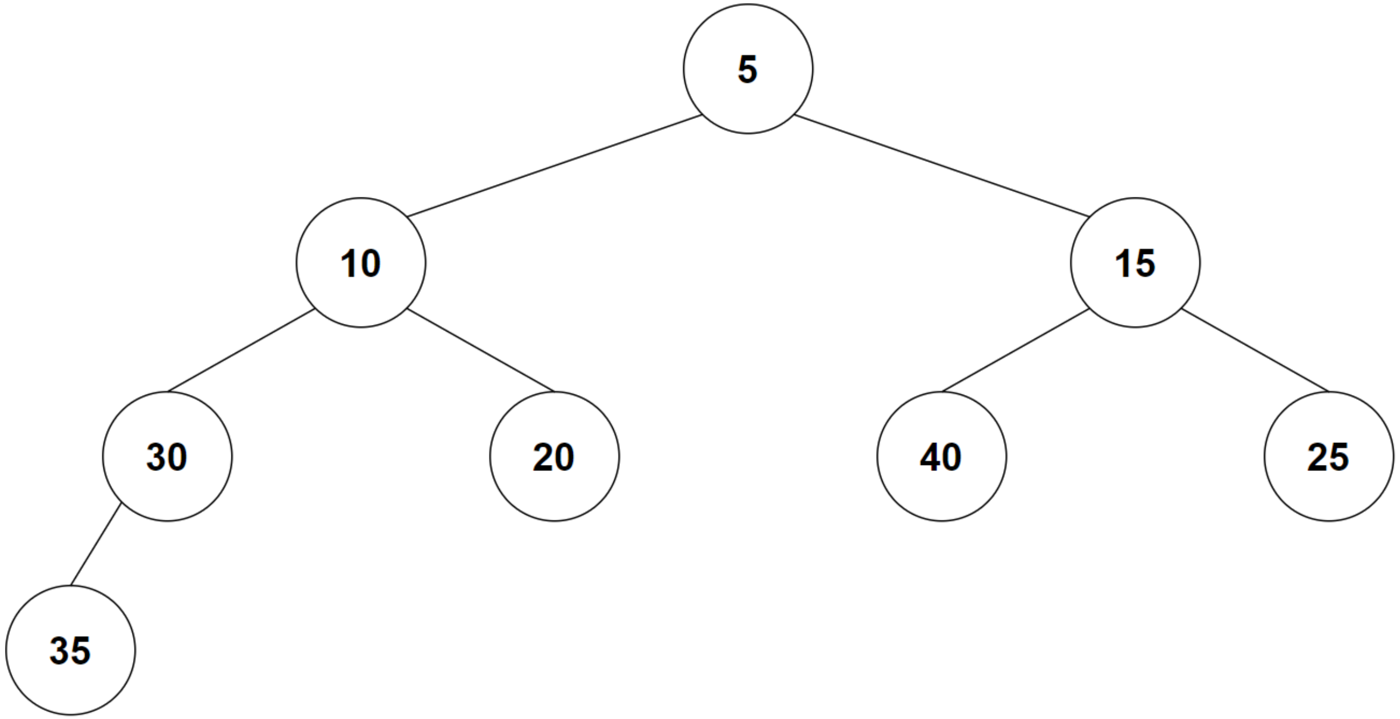
n

The amount of time needed to delete an element into a heap depends on the height of the complete binary tree.

$\log_2 n$

So for worst case, we assume that every element is moved up/down the full height of the tree.

$n\log_2 n$



0	1	2	3	4	5	6	7
5	10	15	30	20	40	25	35

Time Complexity

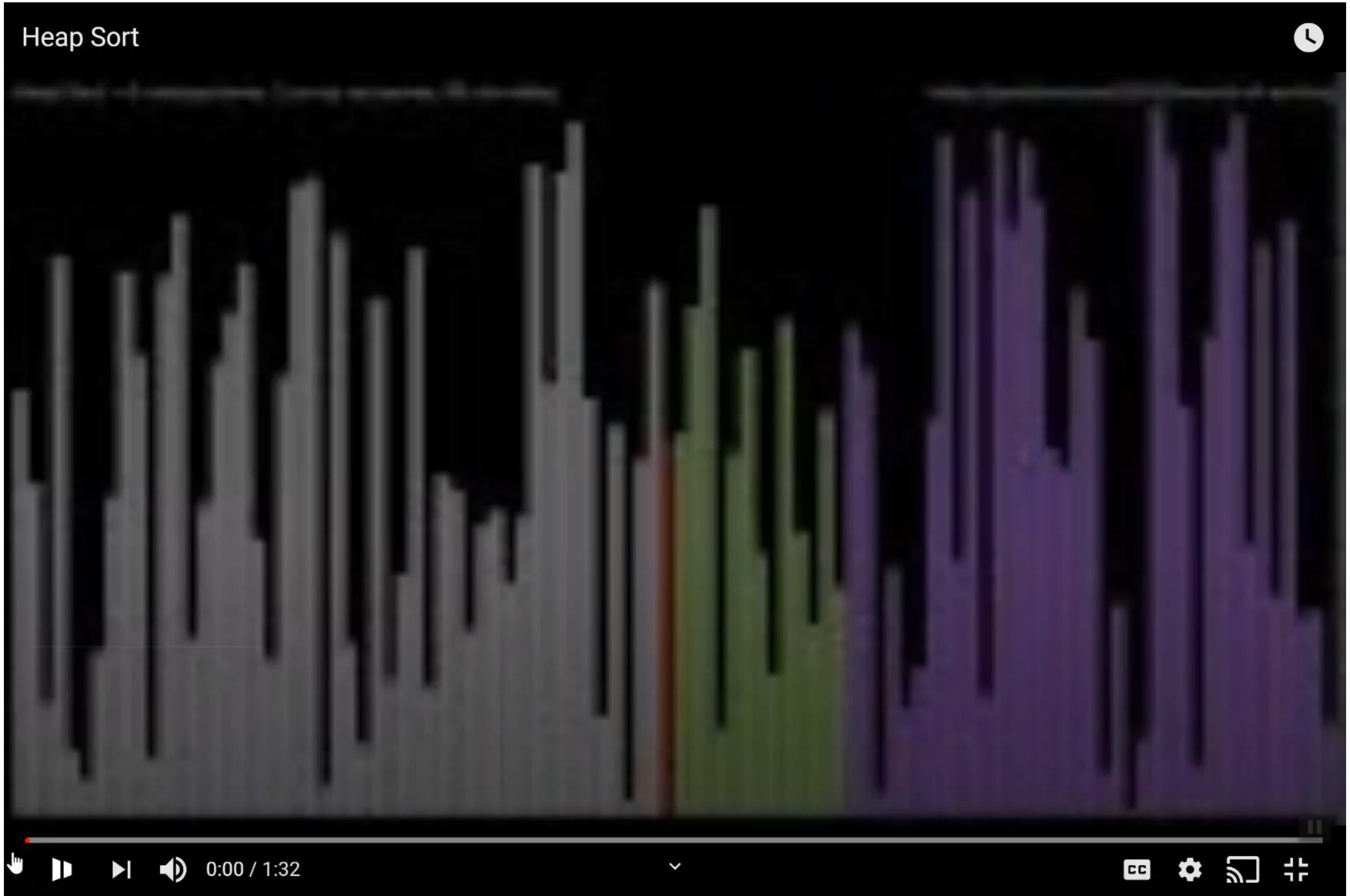
What is the time complexity of sorting using a binary heap?

We sorted by building a binary heap with the data and then deleting it to get the sort.

Building	+	Deleting	=	Sorting
$n\log_2 n$	+	$n\log_2 n$	=	$2*n\log_2 n$

$2n\log_2 n = O(n\log_2 n)$ for Heap Sort

Visual Heap Sort



Priority Queue

Making a priority queue

Using a heap structure is generally preferred to other implementations like linked lists or arrays.

If new values are inserted based on their priority, then the top element will always be the highest priority element.

When the root/highest priority element is removed, then the heap property can be restored in $O(log_2n)$ time.

Change priorities by removing the element and then re-inserting it

Priority Queue

When an element is removed, restoring the heap property takes $O(\log_2 n)$ time.

When an element is added, restoring the heap property takes $O(\log_2 n)$ time.

What happens if we remove the highest priority item AND add a new item at the same time?

Both actions can be done with only time since we only need to heapify once.

Fibonacci Heap

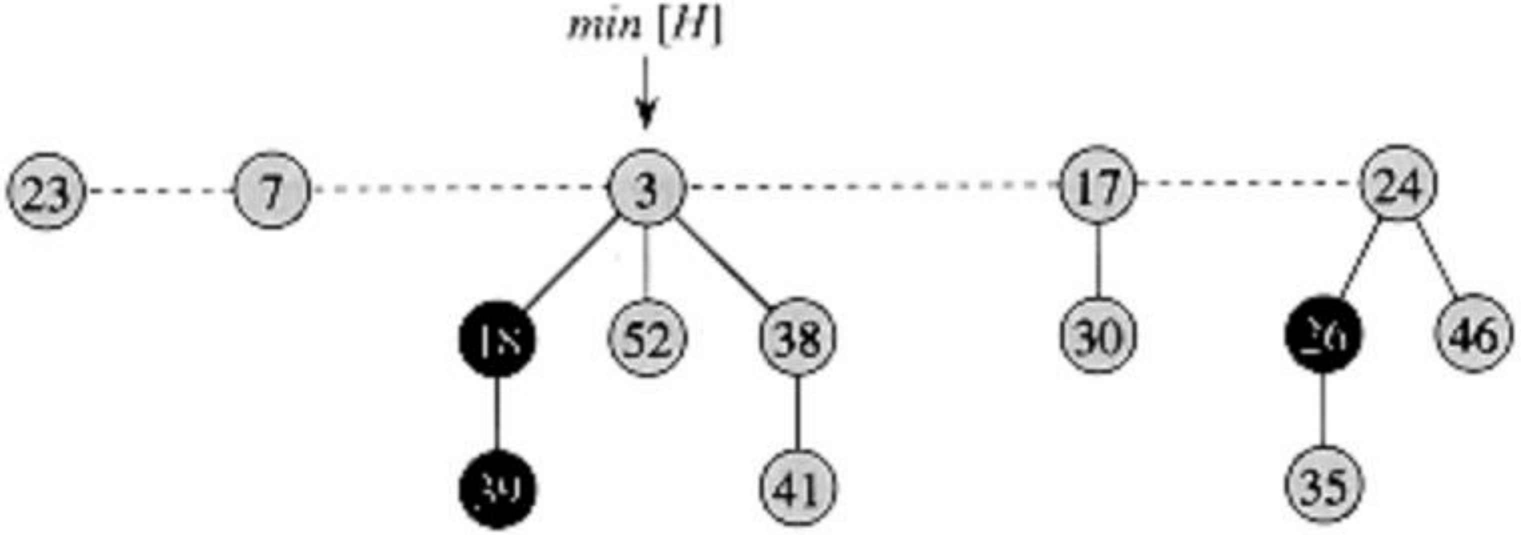
A fibonacci heap is a data structure that consists of a collection of trees which follow min heap or max heap property.

In a fibonacci heap, a node can have more than two children or no children at all.

Fibonacci heaps are linked lists of heap ordered trees.

A pointer to the minimum element is maintained

Fibonacci Heap



Fibonacci Heap

The root nodes of a Fibonacci heap are connected with a circular doubly linked list.

Each layer of the Fibonacci heap are connected with circular doubly linked list.

Parent and child nodes are connected with circular doubly linked list.

Fibonacci Heap

A node can be removed from a circular doubly linked list in a Fibonacci heap in O(1) time.

Two of these types of heaps can be concatenated in O(1) time

An item is added to a Fibonacci Heap by creating a new heap and connecting to the circular doubly linked list of roots.

The pointer to the minimum element is updated if needed when a new item is added.

Nodes are "marked" when they lose a child node and are moved up to the root level when they lose another child node.

Linked List vs Binary Tree

Linked List Node

```
typedef struct node
{
    int node_number;
    struct node *next_ptr;
}
NODE;

NODE *LinkedListHead;
```

Binary Tree Node

```
typedef struct node
{
    int node_number;
    struct node *left_ptr;
    struct node *right_ptr;
}
NODE;

NODE *root;
```

Binary Tree vs Linked List

Linked List

```
NewNode = malloc(sizeof(NODE));
NewNode->node_number = NodeNumber;
NewNode->next_ptr = NULL;
```

Binary Tree

```
NewNode = malloc(sizeof(NODE));
NewNode->node_number = NodeNumber;
NewNode->left_ptr = NULL;
NewNode->right_ptr = NULL;
```

Binary Tree vs Linked List

Add a node to the end of a linked list

```
NewNode = malloc(sizeof(NODE));
NewNode->node_number = NodeNumber;
```

Set the pointer of the last node to the new node

```
TempPtr->next_ptr = NewNode;
```

Add a node to a binary tree

```
NewNode = malloc(sizeof(NODE));
NewNode->node_number = NodeNumber;
NewNode->left_ptr = NULL;
NewNode->right_ptr = NULL;
```

Set the parent node’s left or right ptr to the address of the new child

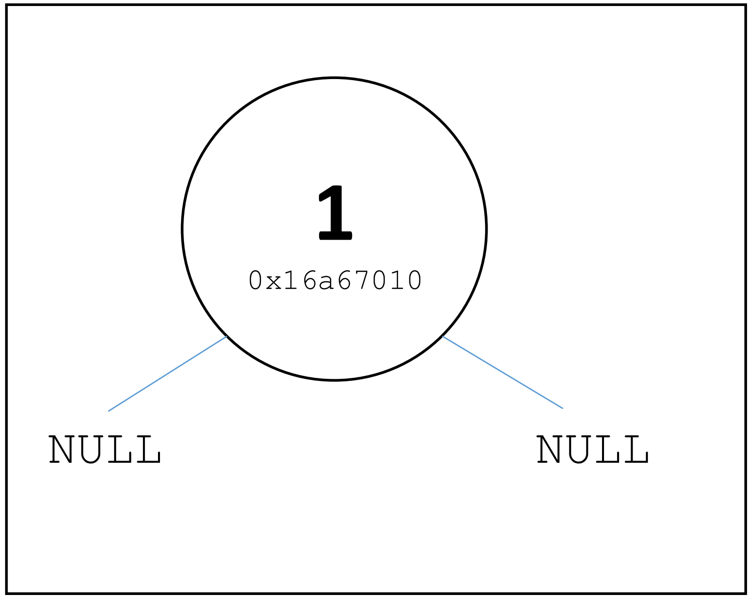
```
/* Allocates memory for a new node with the given data and sets the left and
   right pointers to NULL */
NODE *CreateNewNode(int NodeNumber)
{
      // Allocate memory and assign pointers
      NODE *node = malloc(sizeof(NODE));
      node->left_ptr = NULL;
      node->right_ptr = NULL;

      // Assign data to this node
      node->node_number = NodeNumber;

      printf("Node Number %d %p\n", NodeNumber, node);

      return(node);
}
```

```
typedef struct node
{
   int node_number;
   struct node *left_ptr;
   struct node *right_ptr;
}
NODE;
```

```
Node Number 1 0x16a67010
```

Reminder!!
When you typedef a structure that contains a pointer to the structure, you must use
`typedef struct node`
and not just
`typedef struct`

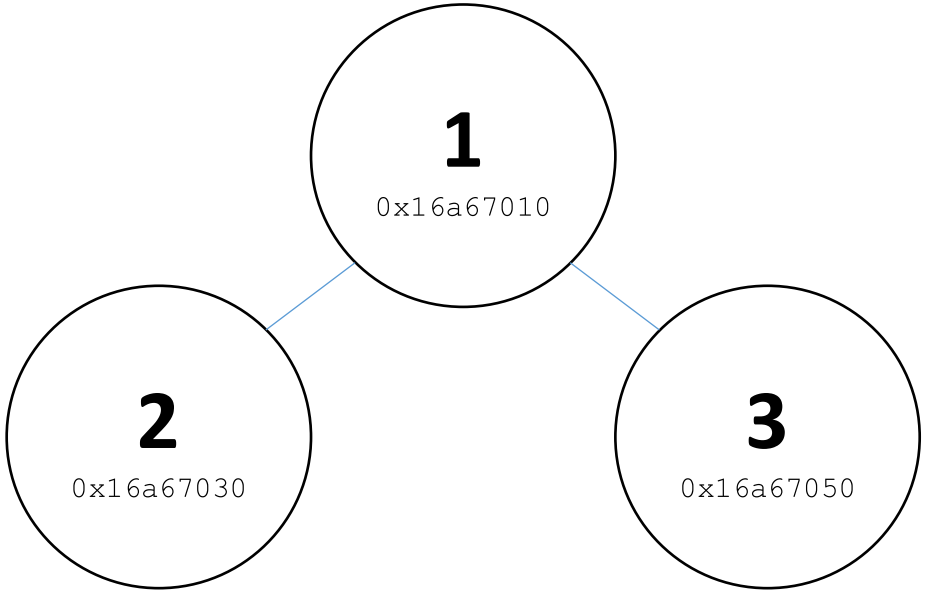
```c
/* declare root of tree */
NODE *root;

/* create root and label with "1" */
root = CreateNewNode(1);


// Print pointer values
printf("\nleft_ptr(1) %p\tright_ptr(1) %p\n",
        root->left_ptr, root->right_ptr);
```




```
left_ptr(1) (nil)        right_ptr(1) (nil)
```


BinaryTreeDemo.c

```
root->left_ptr  = CreateNewNode(2);
root->right_ptr = CreateNewNode(3);
```




```
printf("\nleft_ptr(2) %p\tright_ptr(3) %p\n",root->left_ptr, root->right_ptr);
```

```
Node Number 2 0x16a67030
Node Number 3 0x16a67050

left_ptr(2) 0x16a67030  right_ptr(3) 0x16a67050
```


BinaryTreeDemo.c

Binary Tree vs Binary Search Tree

Binary Tree

Each node can have a maximum of two child nodes and there is no order to how the nodes are organized in the tree.

Binary Search Tree

Each node can have a maximum of two child nodes and there is a relative order to how the nodes are organized in the tree.

Binary Search Tree

A binary search tree (with no duplicate node values) has the characteristic that the values in any left subtree are less than the value in its parent node, and the values in any right subtree are greater than the value in its parent node.

The shape of the binary search tree that corresponds to a set of data can vary, depending on the order in which the values are inserted into the tree.

Binary Search Tree

What makes a binary tree a binary search tree?

- All nodes in the left subtree are less than the root
- All nodes in the right subtree are greater than the root
- Each subtree is itself a binary search tree
- No duplicates allowed*

Binary Search Tree (BST)

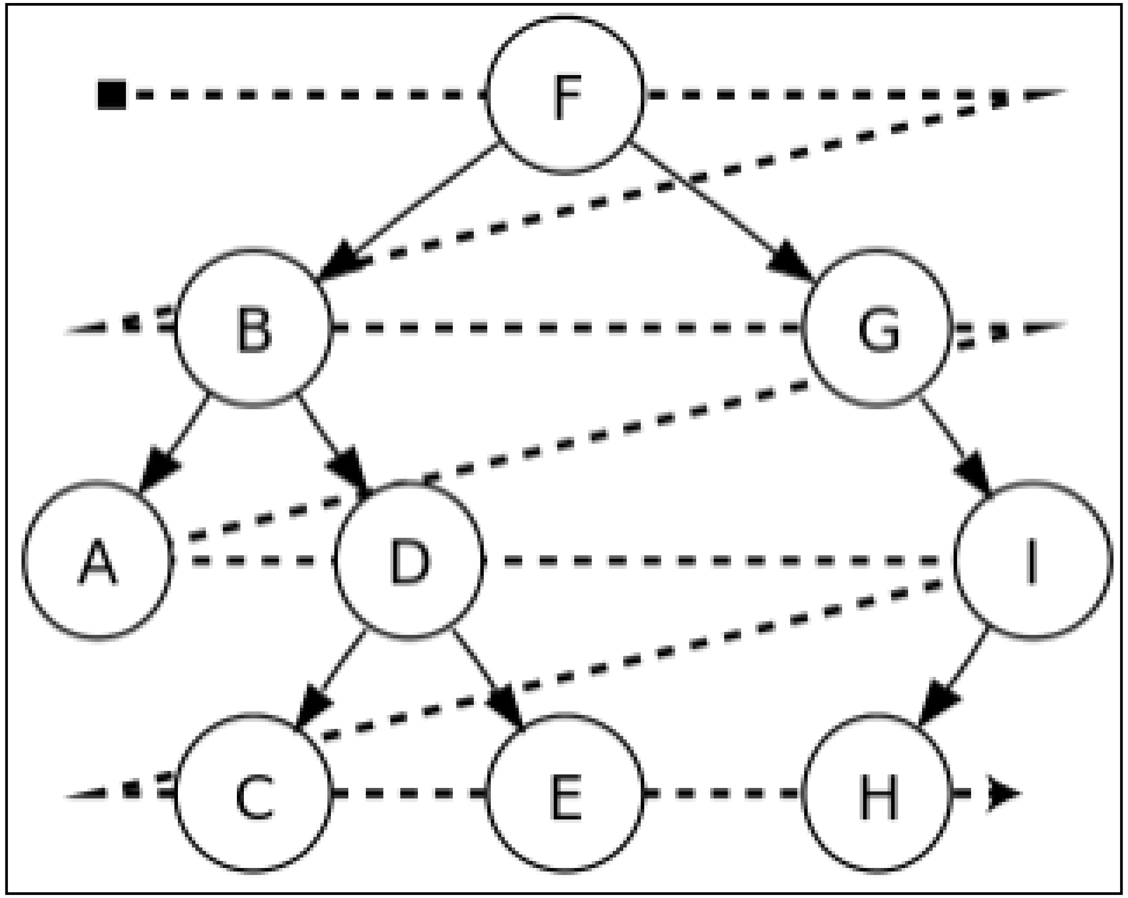
- Linear data structures (linked list, queues and stacks) are traversed in a linear order. Tree structures are traversed in multiple ways - from any given node, there is more than one possible next node in the traversal path

- Tree structures may be traversed in

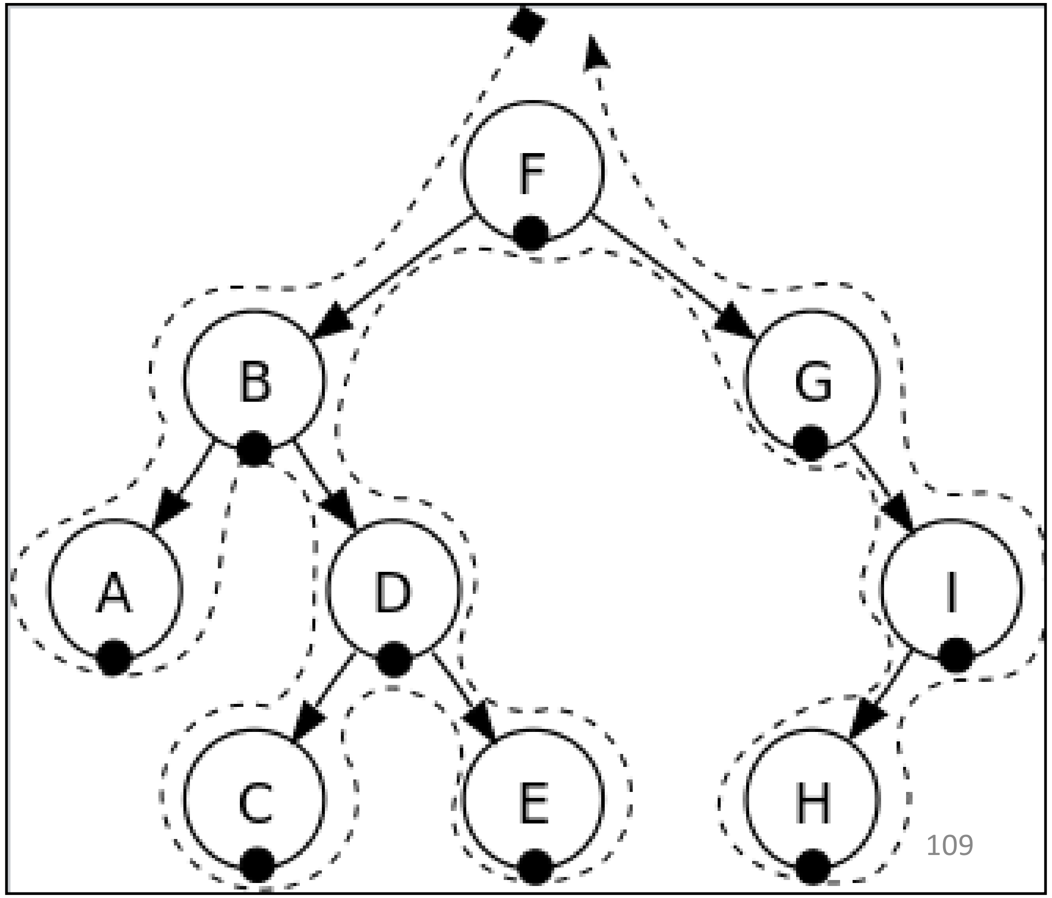
    - Breadth-first Order
    - Depth-first Order

BST Breadth-first vs Depth-first Traversal

Breadth-first



Depth-first



BST Depth-first Traversals

- Inorder Traversal
  - Gives us the nodes in increasing order

- Preorder Traversal
  - Parent nodes are visited before any of its child nodes
  - Used to create a copy of the tree
  - File systems use it to track your movement through directories

- Postorder Traversal
  - Used to delete the tree
  - File systems use it to delete folders and the files under them

BST Depth-first Traversals

Depth First Tree Traversals

Preorder

Root, Left, Right

4 2 1 3 5

Postorder

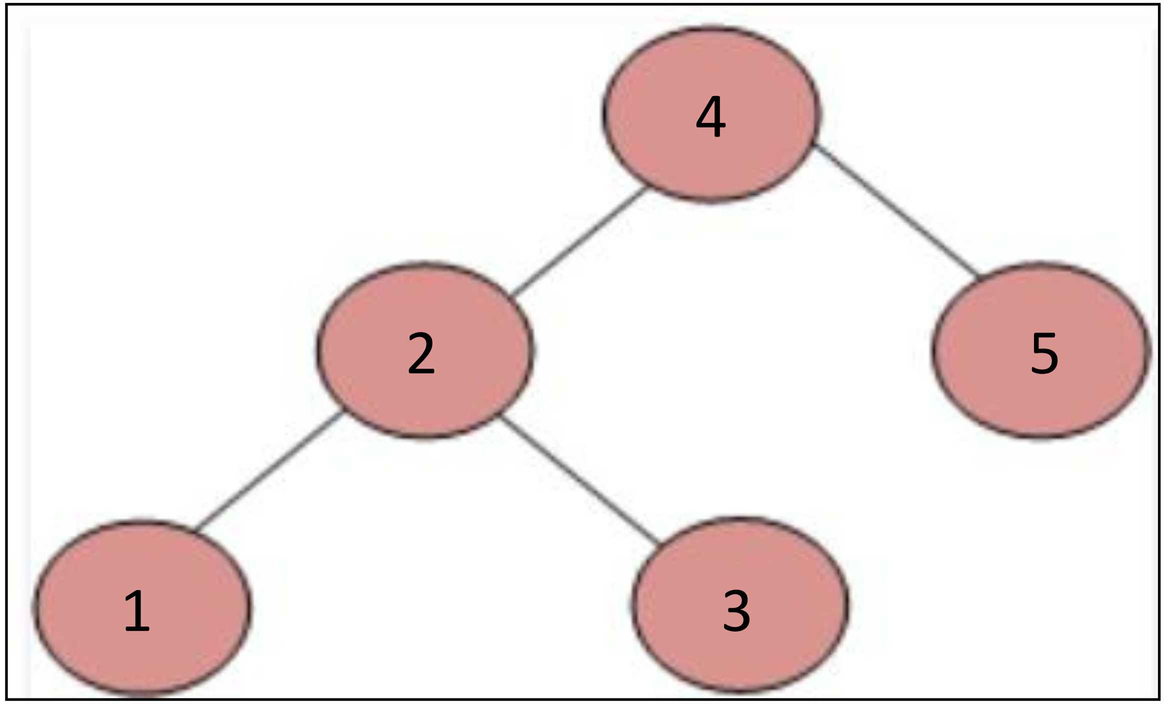
Left, Right, Root

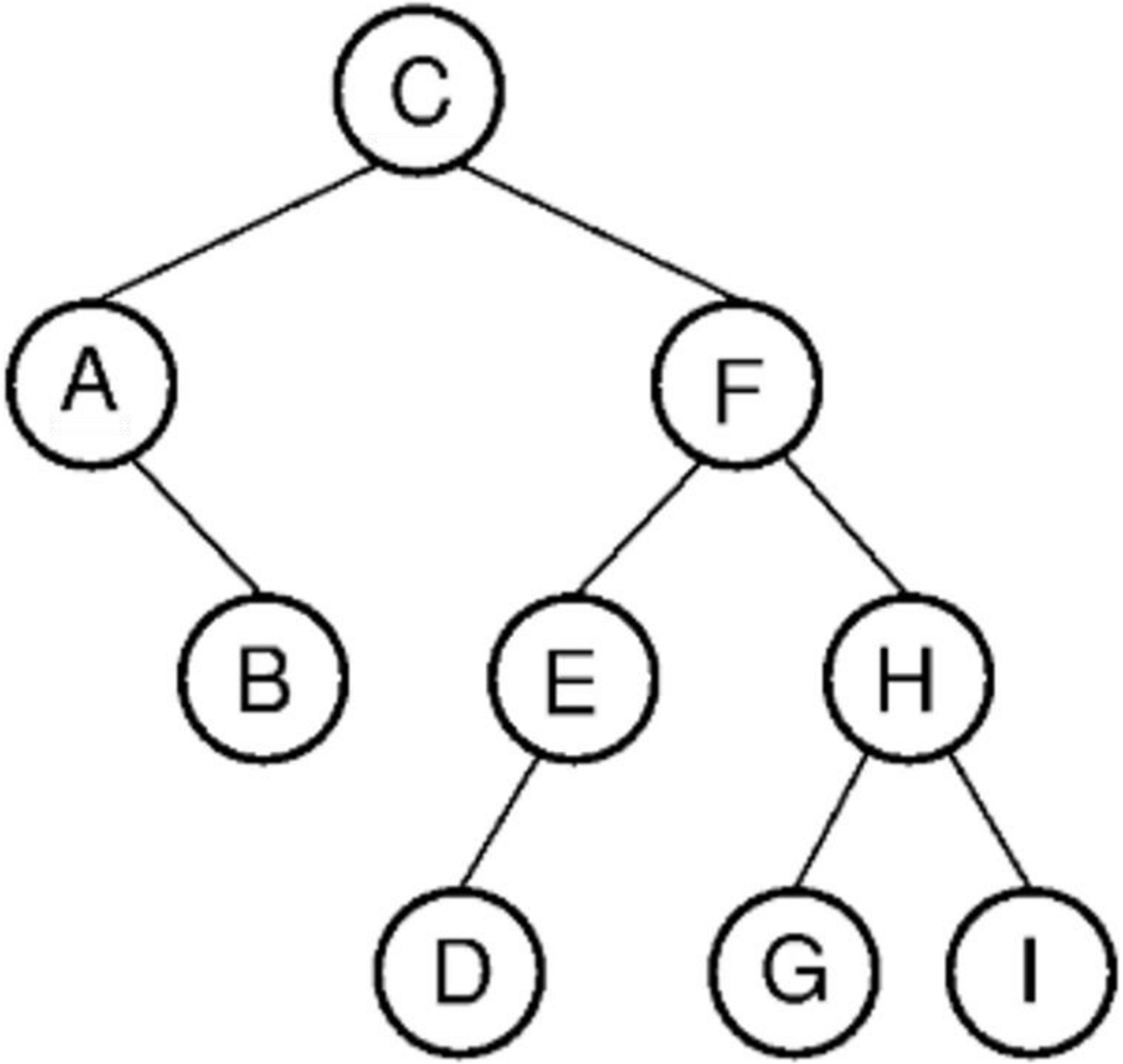
1 3 2 5 4

Inorder

Left, Root, Right

1 2 3 4 5





Depth First Tree Traversals

Preorder

Root, Left, Right

C A B F E D H G I

Postorder

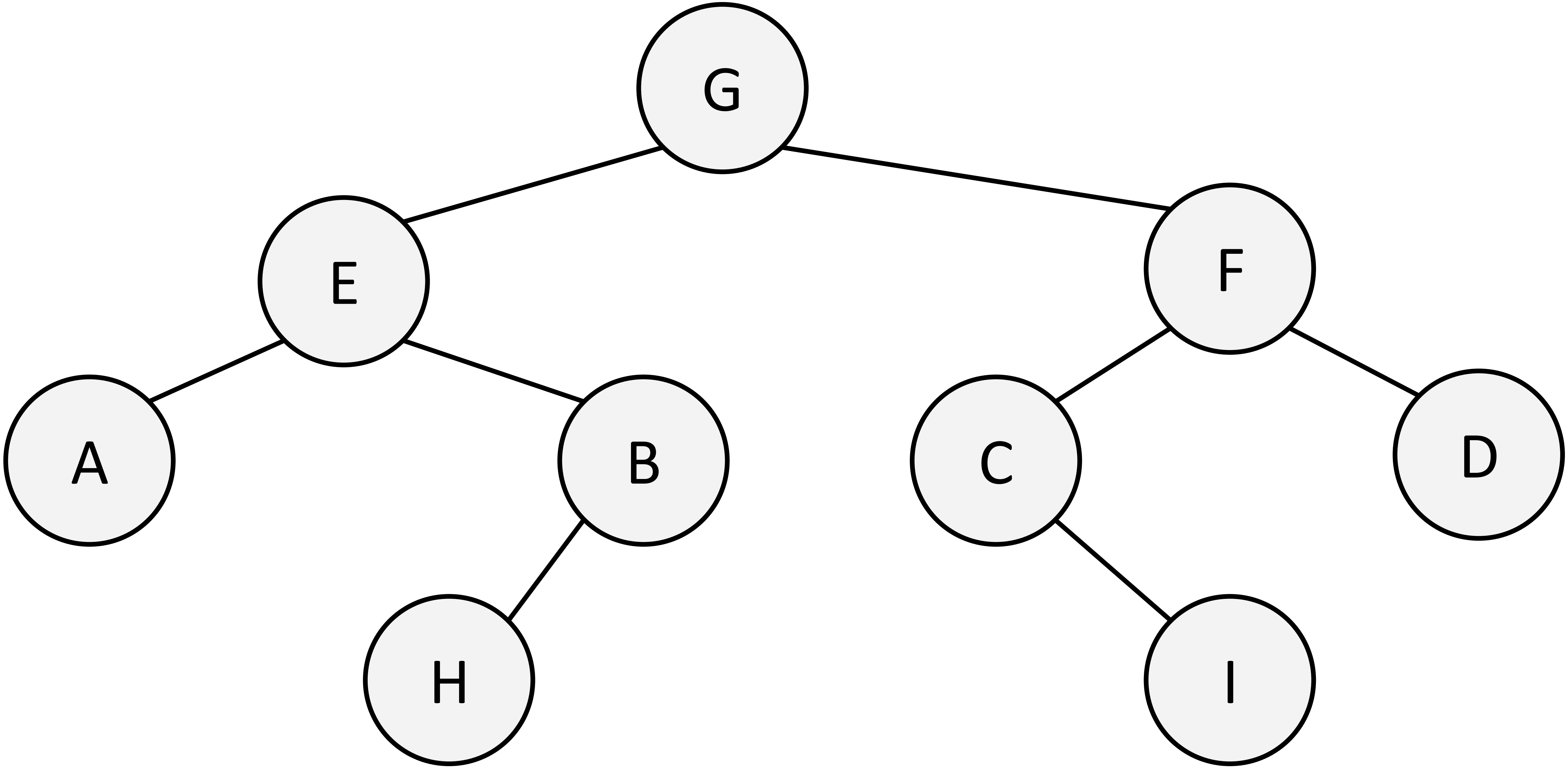
Left, Right, Root

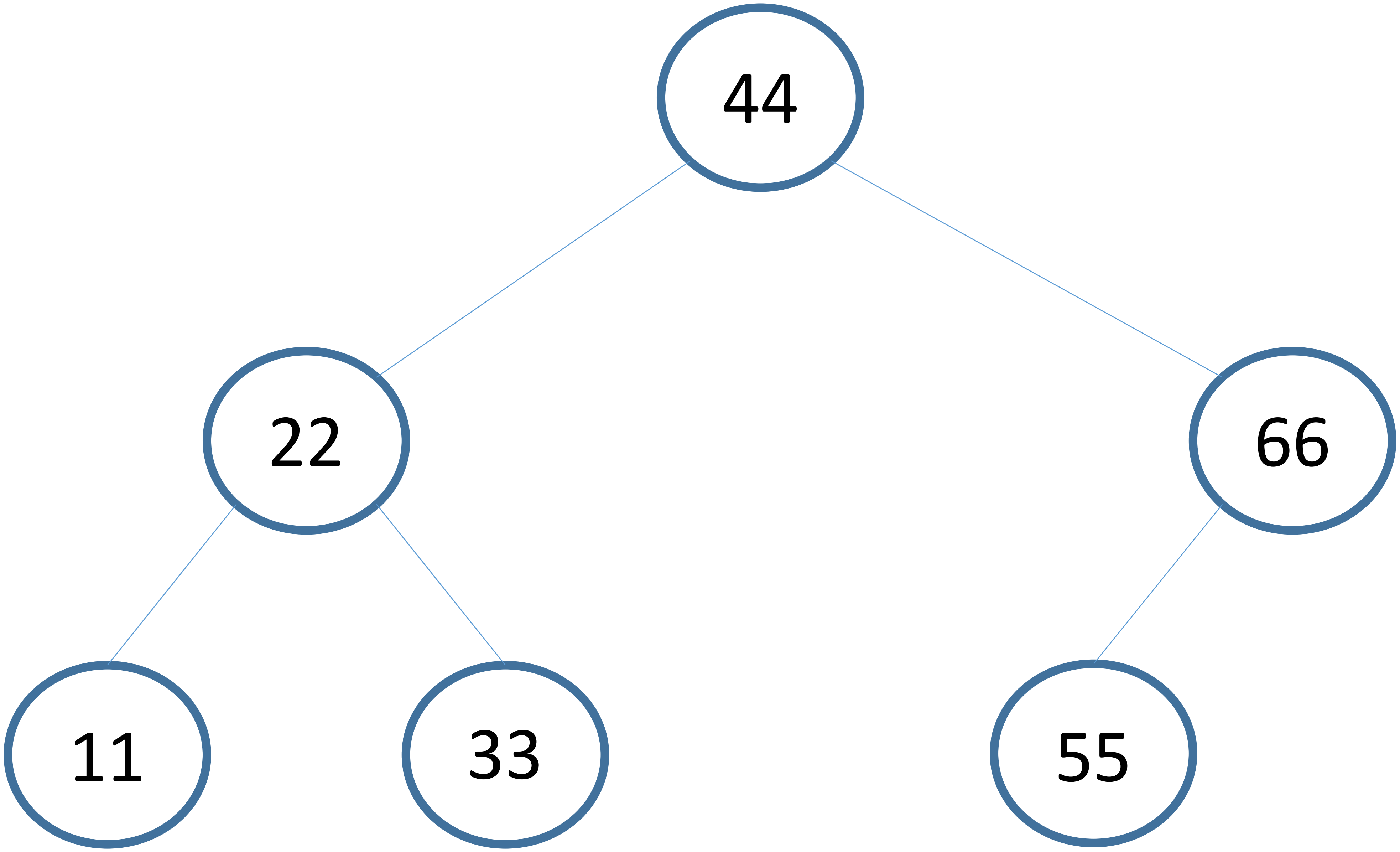
B A D E G I H F C

Inorder

Left, Root, Right

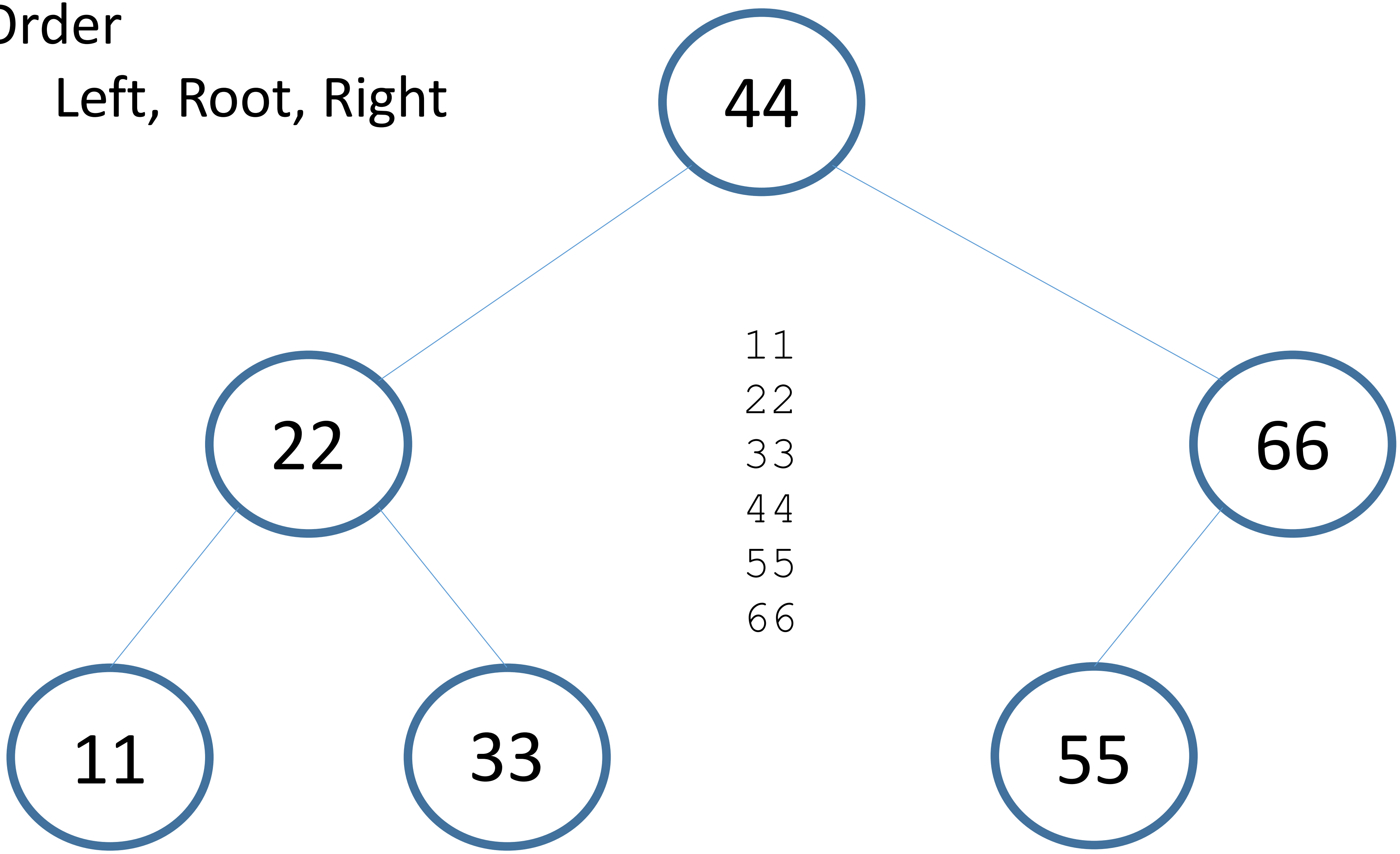
A B C D E F G H I


G
E
F
A
B
C
D
H
I


44
22
66
11
33
55

In Order

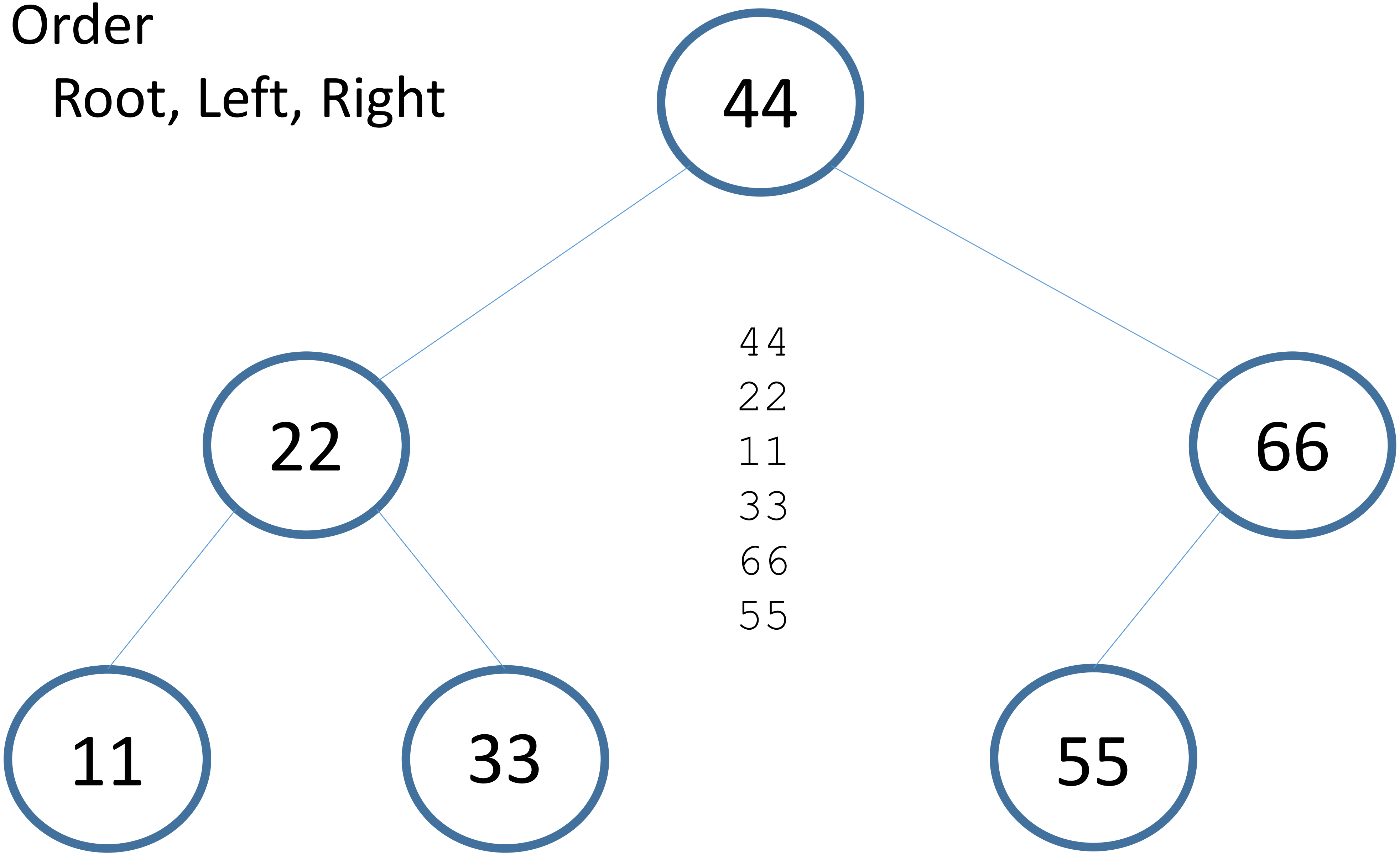
Left, Root, Right



11
22
33
44
55
66

Pre Order

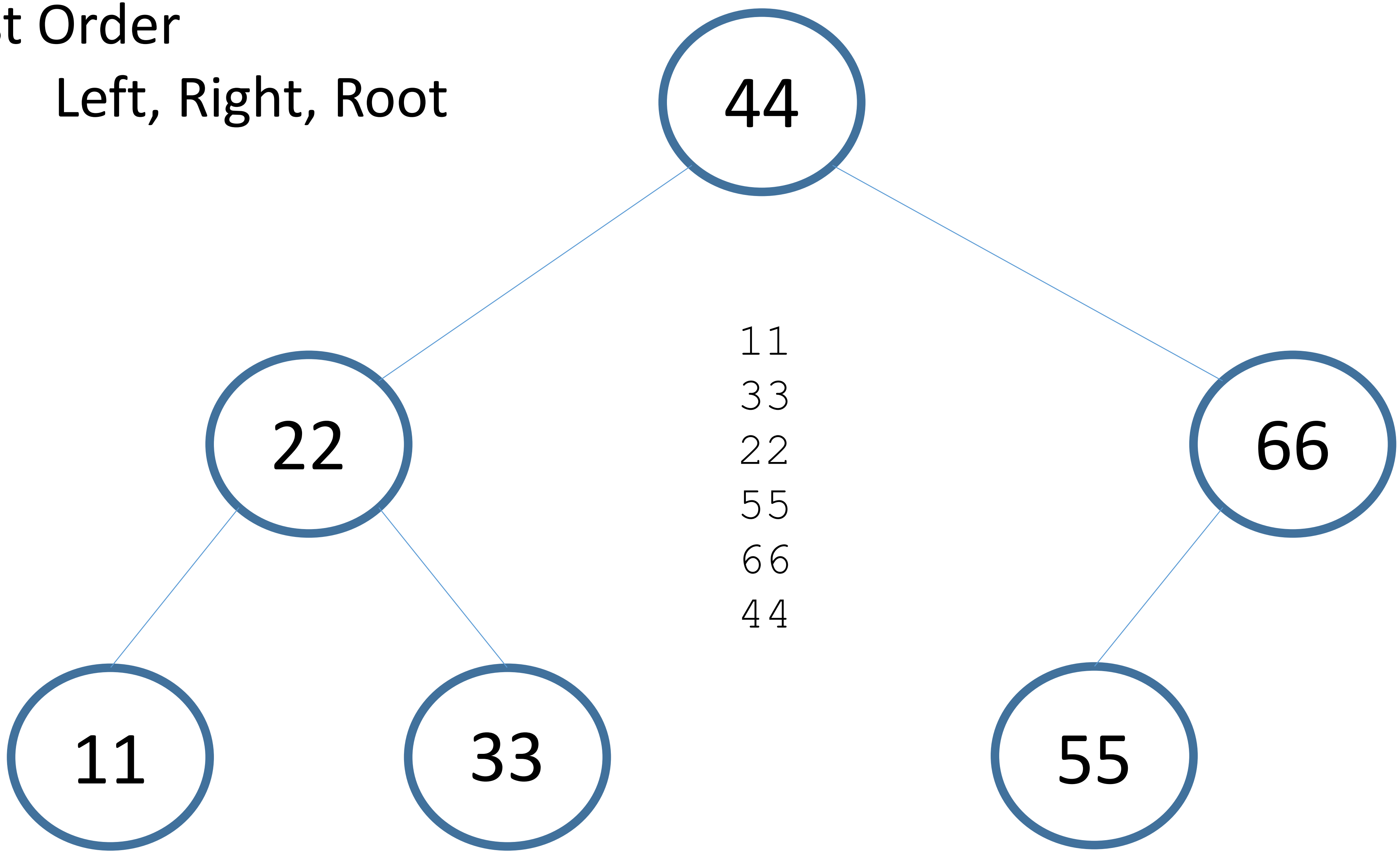
Root, Left, Right



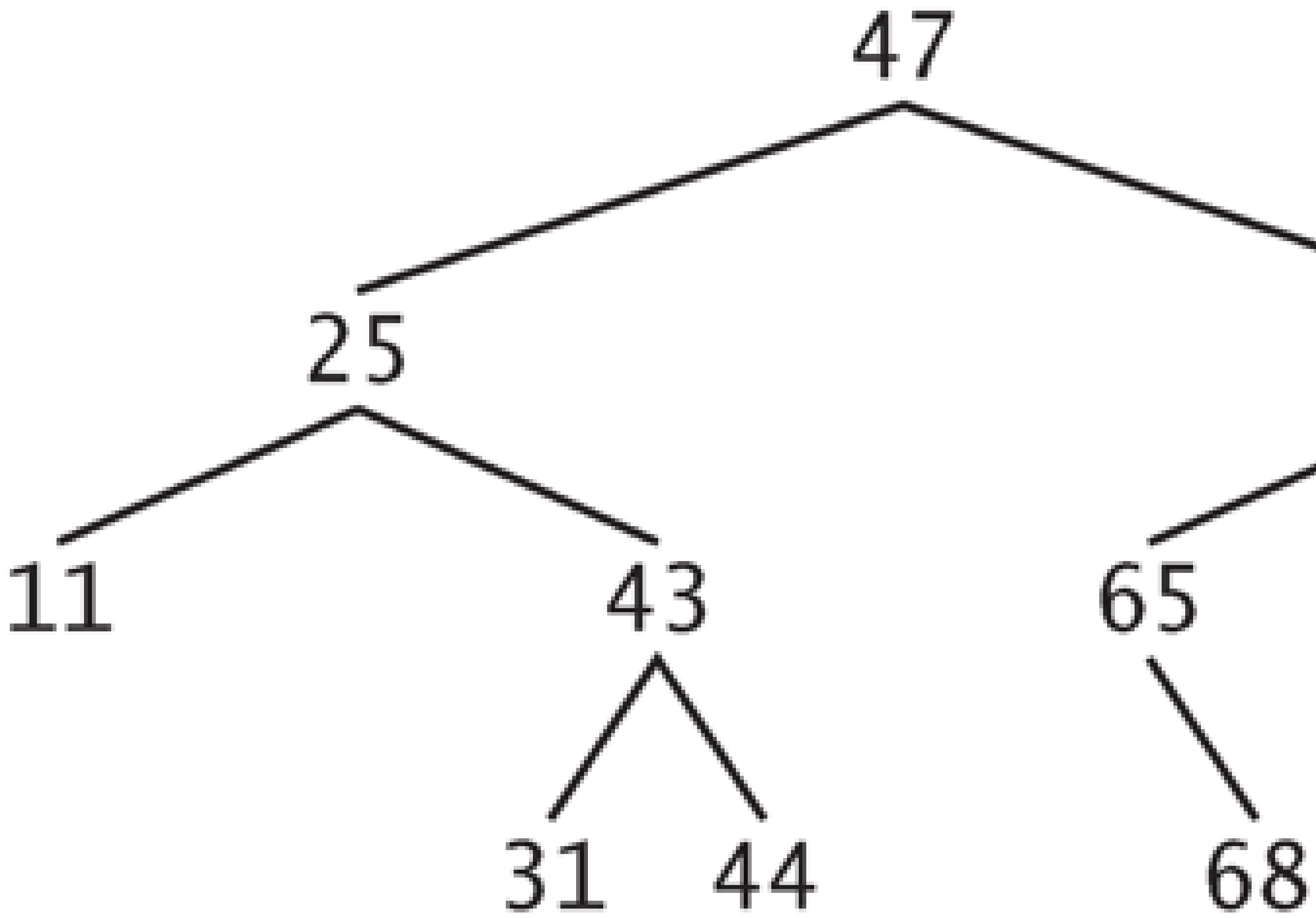
44
22
11
33
66
55

Post Order

Left, Right, Root



11
33
22
55
66
44


47
25
77
11
43
65
31
44
68

```
How many nodes in the tree? 9

Enter data for node 1 : 47

Enter data for node 2 : 25

Enter data for node 3 : 77

Enter data for node 4 : 11

Enter data for node 5 : 43

Enter data for node 6 : 65

Enter data for node 7 : 31

Enter data for node 8 : 44

Enter data for node 9 : 68
```

BST Traversal in Inorder

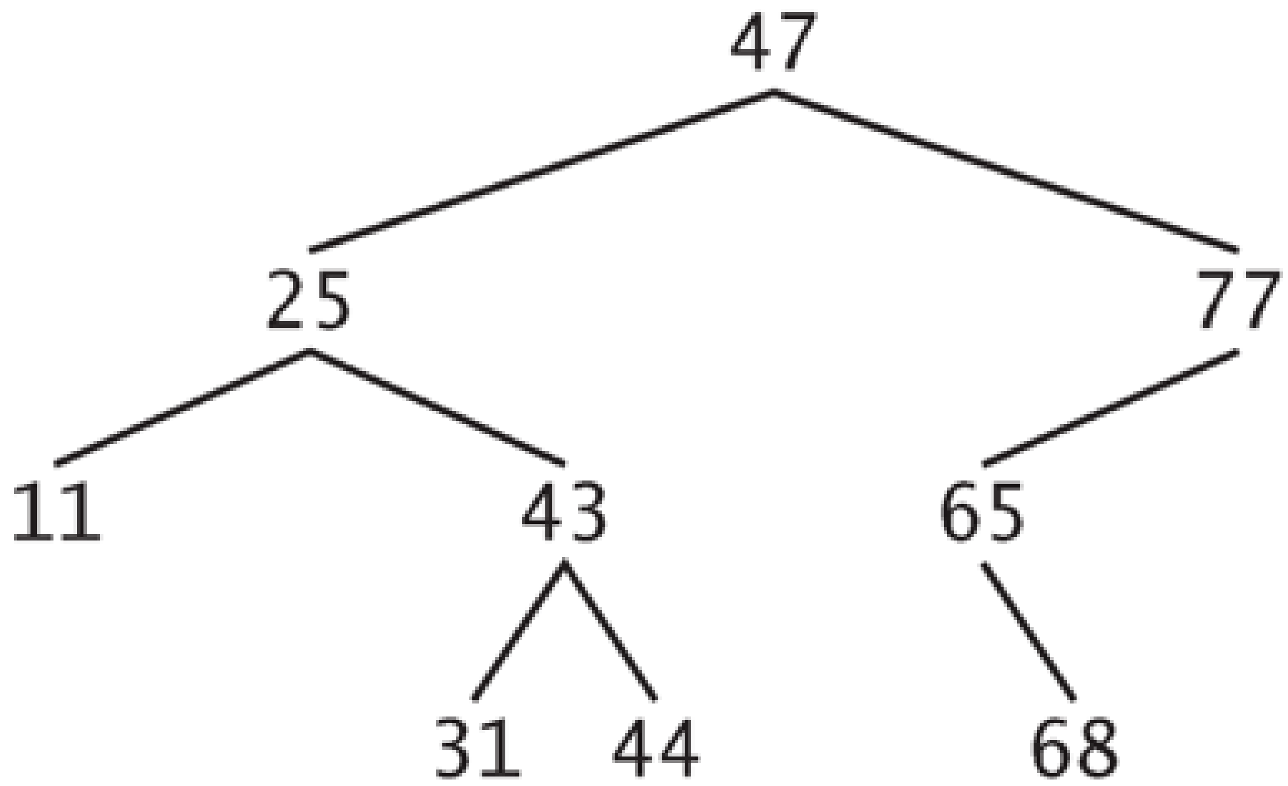
Node4-11 Node2-25 Node7-31
Node5-43 Node8-44 Node1-47
Node6-65 Node9-68 Node3-77

BST Traversal in Preorder

Node1-47 Node2-25 Node4-11
Node5-43 Node7-31 Node8-44
Node3-77 Node6-65 Node9-68

BST Traversal in Postorder

Node4-11 Node7-31 Node8-44
Node5-43 Node2-25 Node9-68
Node6-65 Node3-77 Node1-47



```c
typedef struct node
{
    int node_data;
    struct node *right;
    struct node *left;
}
NODE;


NODE *root = NULL;


AddBSTNode(&root, node_data);
```

```c
void AddBSTNode(NODE **current_node, int add_data)
{
    if (*current_node == NULL)
    {
        *current_node = malloc(sizeof(NODE));
        (*current_node)->left = (*current_node)->right = NULL;
        (*current_node)->node_data = add_data;
    }
    else
    {
        if (add_data < (*current_node)->node_data )
            AddBSTNode(&(*current_node)->left, add_data);

        else if(add_data > (*current_node)->node_data )
            AddBSTNode(&(*current_node)->right, add_data);

        else
            printf(" Duplicate Element !! Not Allowed !!!");
    }
}
```

```c
Inorder(root);

Preorder(root);

Postorder(root);
```

```c
void Inorder(NODE *tree_node)
{
   if(tree_node != NULL)
   {
      Inorder(tree_node->left);
      printf("Node%d",
               tree_node->node_data);
      Inorder(tree_node->right);
   }
}
```

```c
void Preorder(NODE *tree_node)
{
   if(tree_node != NULL)
   {
      printf("Node%d",
               tree_node->node_data);
      Preorder(tree_node->left);
      Preorder(tree_node->right);
   }
}
```

```c
void Postorder(NODE *tree_node)
{
   if(tree_node != NULL)
   {
      Postorder(tree_node->left);
      Postorder(tree_node->right);
      printf("Node%d",
               tree_node->node_data);
   }
}
```

Algorithm Interview Questions & Answers

Question : What is an algorithm? What is the need for an algorithm?

An algorithm is a well-defined computational procedure that takes some values or the set of values, as an input and produces a set of values or some values, as an output.

Algorithms help us measure and analyze the complexity time and space of the problems.

We can compare the performance of the algorithms with respect to other techniques.

Algorithm Interview Questions & Answers

Algorithms provide the basic idea of the problem and an approach to solve it.

Some reasons to use algorithms are...

- improves the efficiency of an existing technique.
- gives a strong description of requirements and goal of the problems to the designer.
- provides a reasonable understanding of the flow of the program.
- measures the performance of the methods in different cases (Best cases, worst cases, average cases).
- identifies the resources (input/output, memory) cycles required by the algorithm.
- reduces the cost of design

Algorithm Interview Questions & Answers

Question : What is the Complexity of an Algorithm?

The complexity of the algorithm is a way to classify how efficient an algorithm is compared to alternative ones. Its focus is on how execution time increases with the data set to be processed. The computational complexity of the algorithm is important in computing.

It is very suitable to classify algorithm based on the relative amount of time or relative amount of space they required and specify the growth of time/ space requirement as a function of input size.

Algorithm Interview Questions & Answers

Time complexity

Time complexity is the running time of a program as a function of the size of the input.

Space complexity

Space complexity analyzes the algorithm, based on how much space an algorithm needs to complete its task. Space complexity analysis was critical in the early days of computing (when storage space on the computer was limited) – not as important today.

Algorithm Interview Questions & Answers

Worst-case: f(n)

It is defined by the maximum number of steps taken on any instance of size n.

Best-case: f(n)

It is defined by the minimum number of steps taken on any instance of size n.

Average-case: f(n)

It is defined by the average number of steps taken on any instance of size n.

A

Algorithm Interview Questions & Answers

Write an algorithm to reverse a string. For example, if my string is "French" then my result will be "hcnerF".

Step1: Create two variables `i` and `j`

Step2: set `i` equal to 0 and set `j` equal to the length of the string - 1

Step3: swap string [i] with string[j]

Step4: increment `i` by 1 and decrement `j` by 1

Step5: repeat steps 3 and 4 while `i` < `j`

Algorithm Interview Questions & Answers

Write an algorithm to insert a node in a linked list.

What are the 2 possibilities?

Linked list is empty

Linked list is not empty

Algorithm Interview Questions & Answers

Linked list is empty

Create a new node

If linked list is empty (linked list head is NULL), then set linked list head to the new node.

Algorithm Interview Questions & Answers

Linked list is not empty

Create a new node

Traverse the linked list to the end.

Set the next pointer of the end node to the new node's address.

Algorithm Interview Questions & Answers

What is Asymptotic Notation?

Asymptotic analysis is used to measure the efficiency of an algorithm that doesn't depend on machine-specific constants and prevents the algorithm from comparing the time taking algorithm.

Asymptotic notation is a mathematical tool that is used to represent the time complexity of algorithms for asymptotic analysis.

What are the 3 asymptotic notations we learned this semester? List each one and draw the graph representation of each.

C.	**D.**	**E.**

Algorithm Interview Questions & Answers

θ Notation

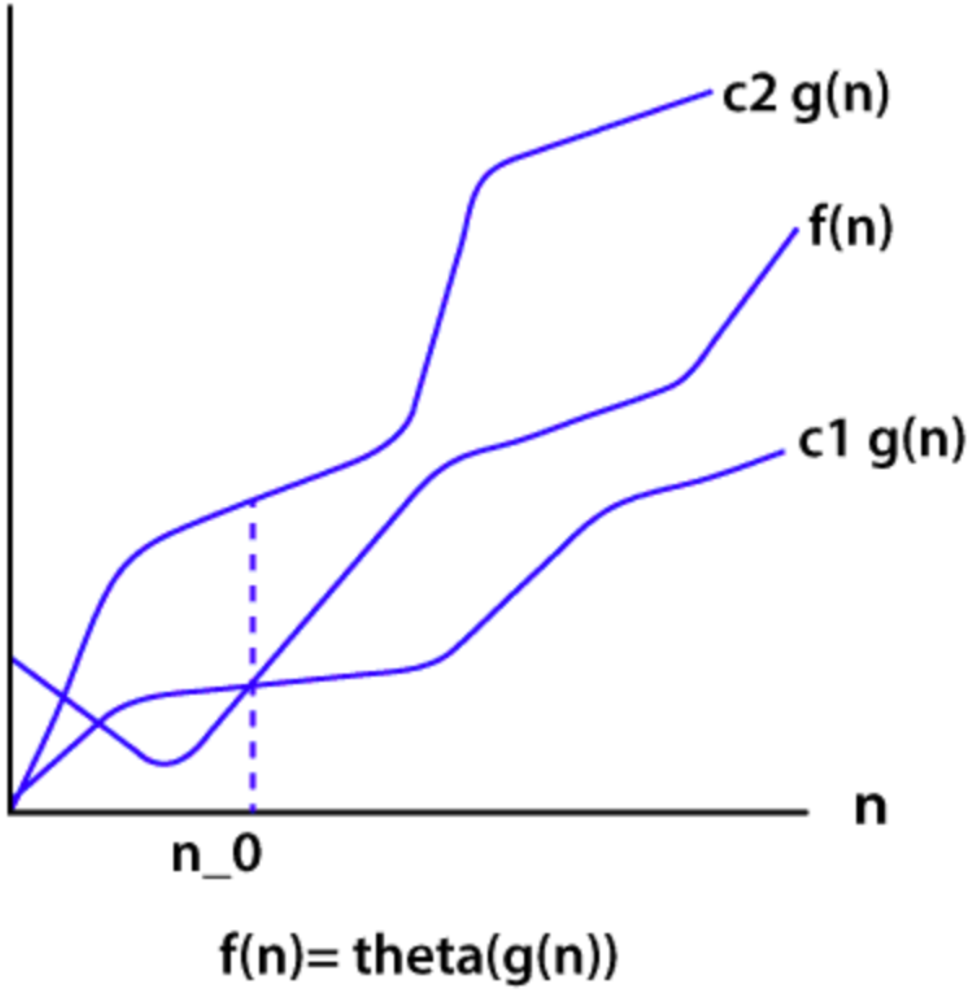
θ Notation defines the exact asymptotic behavior.

To define a behavior, it bounds functions from above and below.

A convenient way to get Theta notation of an expression is to drop low order terms and ignore leading constants.

Algorithm Interview Questions & Answers

θ Notation



Algorithm Interview Questions & Answers

Big O Notation

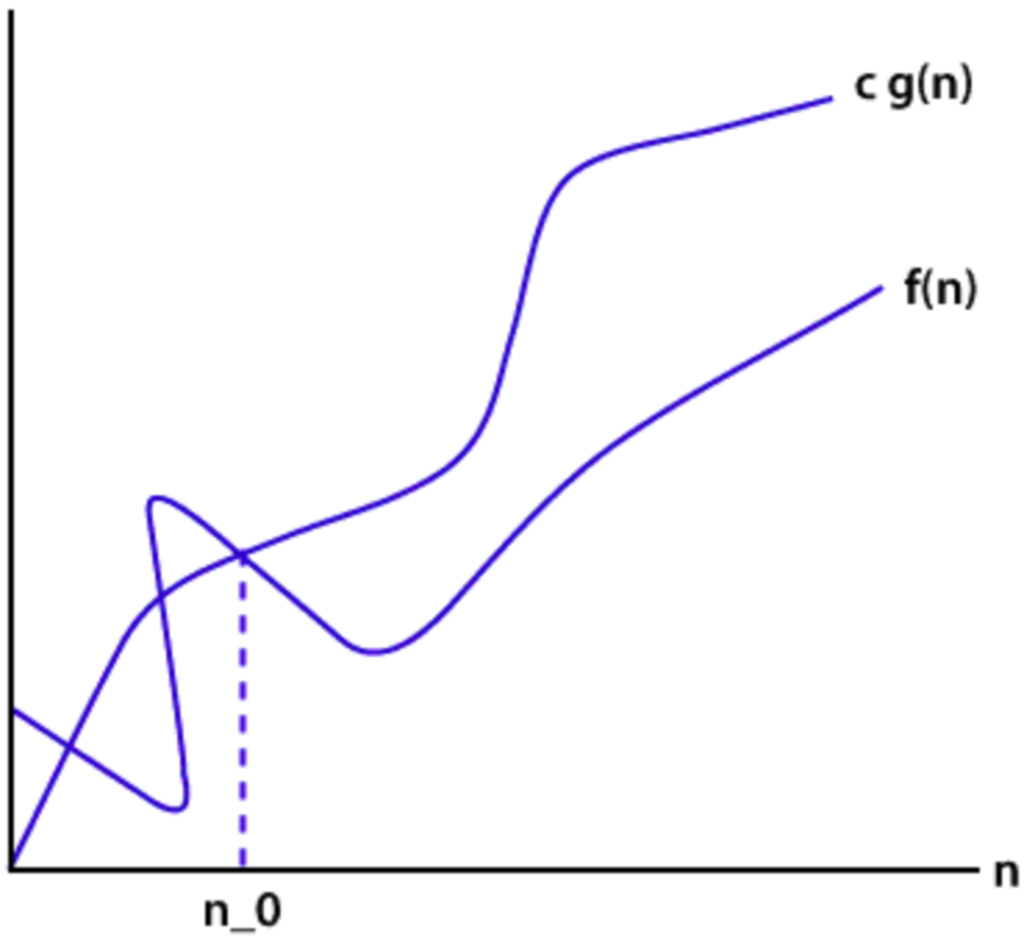
The Big O notation bounds a function from above, it defines an upper bound of an algorithm.

Let's consider the case of insertion sort; it takes linear time in the best case and quadratic time in the worst case.

The time complexity of insertion sort is $O(n^2)$. It is useful when we only have upper bound on time complexity of an algorithm.

Algorithm Interview Questions & Answers

Big O Notation



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

Algorithm Interview Questions & Answers

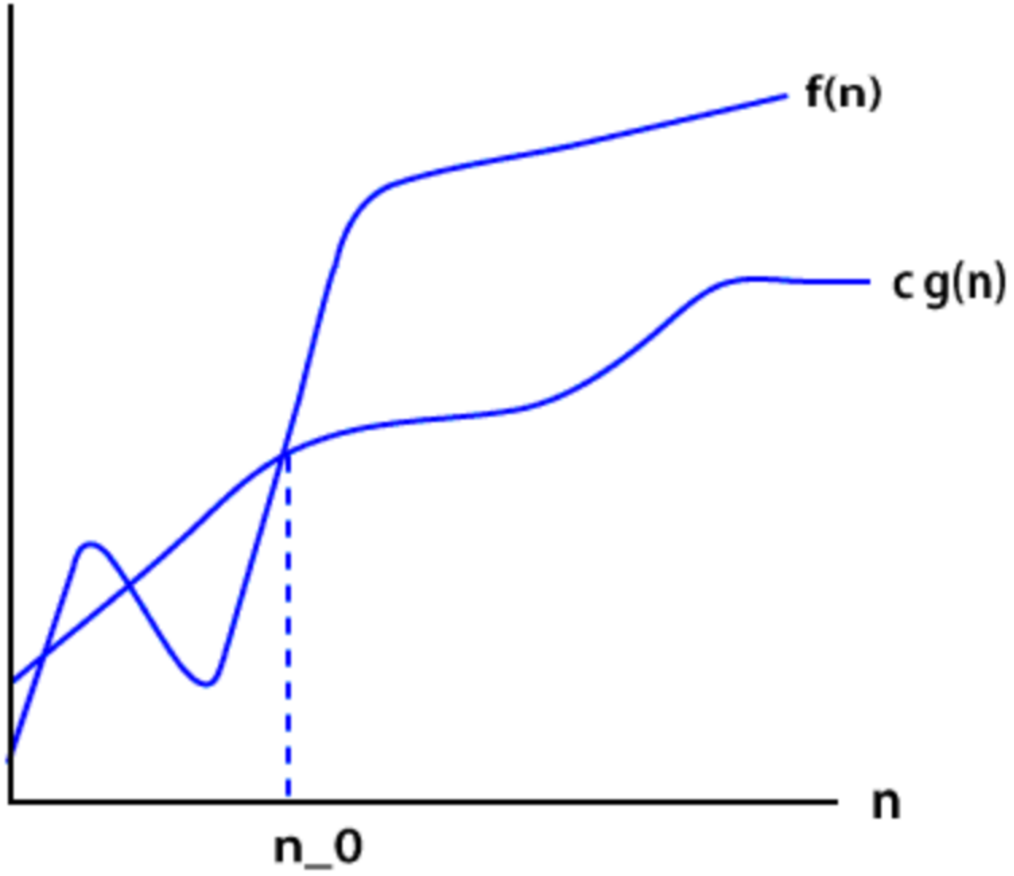
Ω Notation

Just like Big O notation provides an asymptotic upper bound, the **Ω Notation** provides an asymptotic lower bound on a function.

It is useful when we have lower bound on time complexity of an algorithm.

Algorithm Interview Questions & Answers

Ω Notation



f(n)= Omega(g(n))

Algorithm Interview Questions & Answers

Explain the Bubble Sort algorithm

Bubble sort is the simplest sorting algorithm among all sorting algorithm. It repeatedly works by swapping the adjacent elements if they are in the wrong order.

How would you use bubble sort to sort this array?

{7,2,5,3,8}

Algorithm Interview Questions & Answers

Pass 1

(**72**538) -> (27538) swap 7 and 2.

(2**75**38) -> (25738) swap 7 and 5.

(25**73**8) -> (25378) swap 7 and 3.

(253**78**) -> (25378) algorithm does not swap 7 and 8 because 7<8.

Pass 2

(**25**378) -> (25378) algorithm does not swap 2 and 5 because 2<5.

(2**53**78) -> (23578) swap 5 and 3.

(23**57**8) -> (23578) algorithm does not swap 5 and 7 because 5<7.

Pass 3

(**23**578) -> (23578) algorithm does not swap 2 and 3 because 7<8.

(2**35**78) -> (23578) algorithm does not swap 3 and 5 because 7<8.

Pass 4

(**23**578) -> (23578) algorithm does not swap 2 and 3 because 7<8.

Algorithm Interview Questions & Answers

```
void BubbleSort(int arr[], int n)
{
    for (int i = 0; i < n-1; i++)
    {
        for (int j = 0; j < n-i-1; j++)
        {
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
        }
    }
}
```

Algorithm Interview Questions & Answers

n = 5

i	j
0	0,1,2,3
1	0,1,2
2	0,1
3	0

```
void BubbleSort(int arr[], int n)
{
    for (int i = 0; i < n-1; i++)
    {
        for (int j = 0; j < n-i-1; j++)
        {
        }
    }
}
```

What is the time complexity of Bubble Sort?

Algorithm Interview Questions & Answers

How to swap two integers without swapping the temporary variable?

Suppose we have two integers `i` and `j`, the value of `i=7` and `j=8` then how will you swap them without using a third variable?

```
i = i + j
j = i - j
i = i - j
```

You try it (Box G)

Looks good but…

The integer will overflow if the addition is more than the maximum value of `int` as defined by `INT_MAX` and if subtraction is less than minimum value, `INT_MIN`.

Algorithm Interview Questions & Answers

The integer will overflow if the addition is more than the maximum value of `int` as defined by `INT_MAX` and if subtraction is less than minimum value, `INT_MIN`.

```
i = i + j
j = i – j
i = i - j
```

```
/* Minimum and maximum values a `signed int' can hold.  */
#  define INT_MIN (-INT_MAX - 1)
#  define INT_MAX 2147483647

Swap 2147483647 and 1

i = 2147483647 + 1 = -2147483648
j = -2147483648 – 1 = 2147483647
i = -2147483648 – 2147483649 = 1
```

This works in C but would not work in a language like Java that does not handle integer overflow.

7 in binary is 00000111 and 8 in binary is 00001000

Algorithm Interview Questions & Answers

Want to really impress?

Use the XOR method…

```
i = i ^ j;
j = i ^ j;
i = i ^ j;
```

You try it with i = 7 and j = 8 (Box H)

Set `i = 7` and `j = 8`

```
i = i ^ j
i = 7 ^ 8
i = 00000111 ^ 00001000 = 00001111
i = 15

j = i ^ j
j = 15 ^ 8
j = 00001111 ^ 00001000 = 00000111
j = 7

i = i ^ j
i = 15 ^ 7
i = 00001111 ^ 00000111 = 00001000
i = 8
```

Algorithm Interview Questions & Answers

What are Divide and Conquer algorithms?

Divide and Conquer is not an algorithm; it's a pattern for an algorithm.

It is a technique that breaks up a large input into smaller pieces and solves the problem for each of the small pieces.

Then, all of the piecewise solutions are merged into a global solution.

This strategy is called divide and conquer.

Algorithm Interview Questions & Answers

Divide and conquer uses the following steps (list the 3 steps in Box I)

Divide: This step divides the original problem into a set of subproblems.

Conquer: This step solves every subproblem individually.

Combine: This step puts together the solutions of the subproblems to get the solution to the whole problem.

Algorithm Interview Questions & Answers

Give three examples in Box J of algorithms that use Divide and Conquer

Merge Sort

Quick Sort

Binary Search

Algorithm Interview Questions & Answers

Explain the BFS algorithm?

BFS (Breadth First Search) is a graph traversal algorithm.

It starts traversing the graph from the root node and explores all the neighboring nodes.

It selects the nearest node and visits all the unexplored nodes.

The algorithm follows the same procedure for each of the closest nodes until it reaches the goal state.

Algorithm Interview Questions & Answers

What is Dijkstra's shortest path algorithm?

Dijkstra's algorithm is an algorithm for finding the shortest path from a starting node to the target node in a weighted graph. The algorithm makes a tree of shortest paths from the starting vertex and source vertex to all other nodes in the graph.

Suppose you want to go from home to school using the shortest possible way. You know some roads are heavily congested and using those routes will take more time (meaning these edges have a larger weight).

Dijkstra's algorithm can be used to find the shortest path.

Algorithm Interview Questions & Answers

What are Greedy algorithms?

A greedy algorithm is an algorithmic strategy which is made for the best optimal choice at each substage with the goal of this, eventually leading to a globally optimum solution. This means that the algorithm chooses the best solution at the moment without regard for consequences.

In other words, an algorithm that always takes the best immediate, or local, solution while finding an answer.

Greedy algorithms find the overall, ideal solution for some idealistic problems, but may discover less-than-ideal solutions for some instances of other problems.

Algorithm Interview Questions & Answers

List 3 Greedy algorithms in Box K

Prim

Kruskal

Dijkstra

Algorithm Interview Questions & Answers

What is a linear search?

Linear search is used on a group of items. It relies on the technique of traversing a list from start to end by visiting properties of all the elements that are found on the way.

Write a brief description of how linear search works in Box L.

Step1: Traverse the array using **for loop**.

Step2: In every iteration, compare the target value with the current value of the array

Step3: If the values match, return the current index of the array

Step4: If the values do not match, shift on to the next array element.

Step5: If no match is found, return -1

Algorithm Interview Questions & Answers

What is a Binary Search Tree?

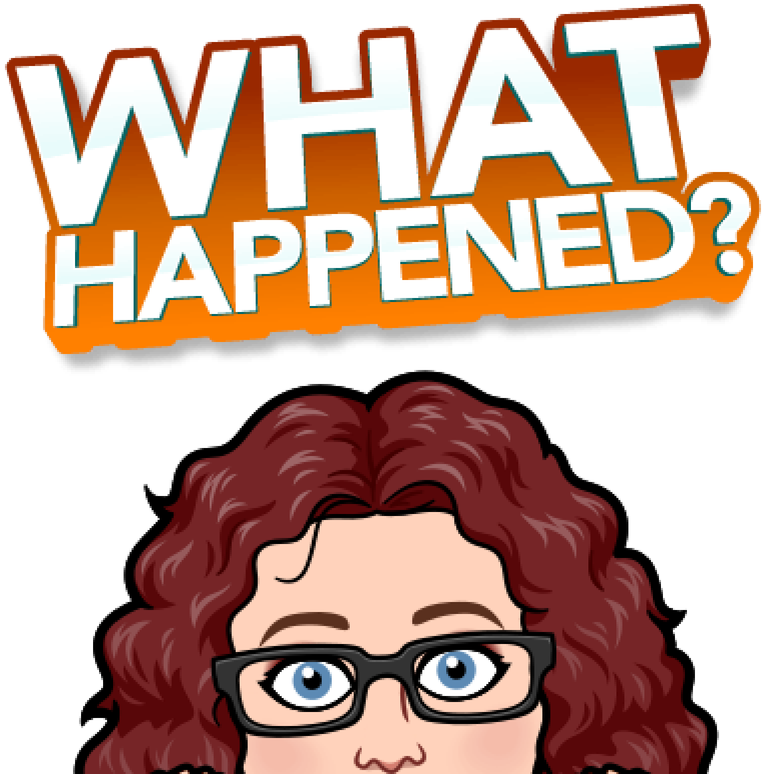
The binary search tree is a special type of data structure which has the following properties. List 4 properties in Box M.

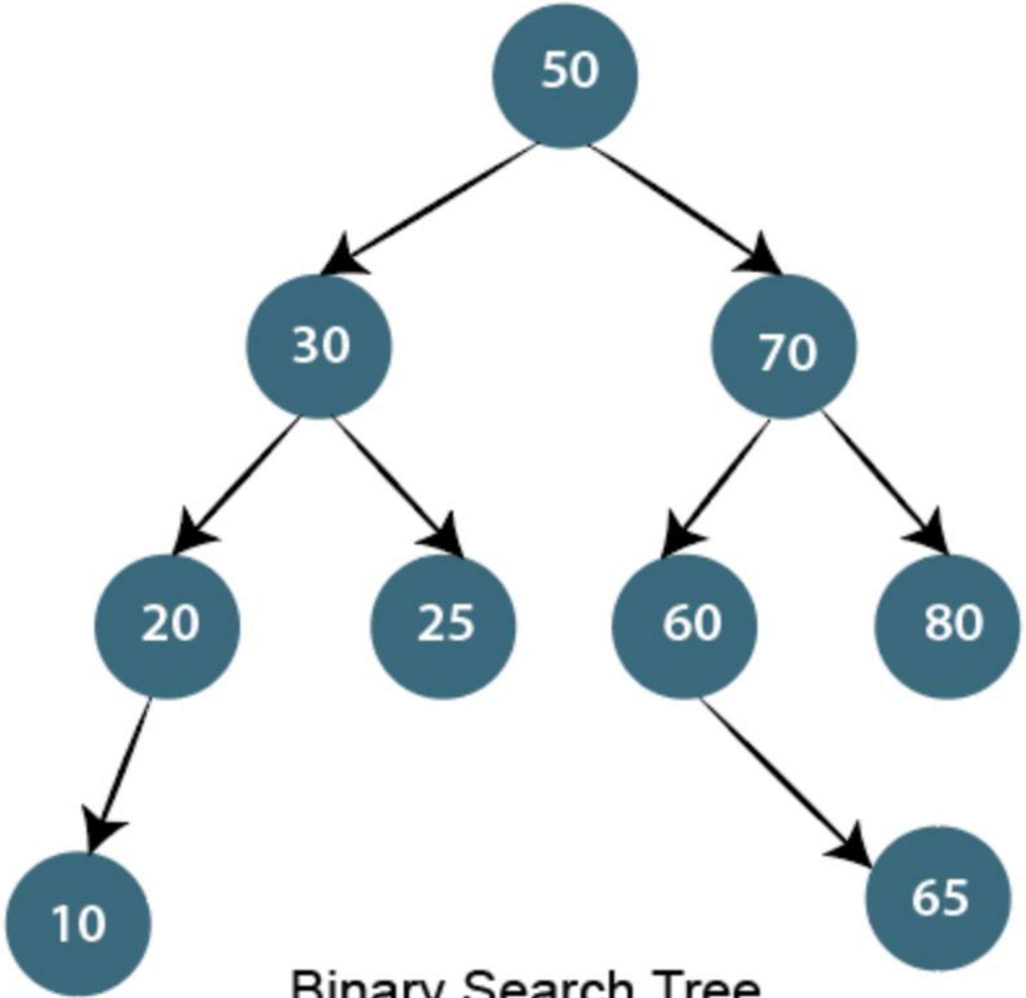
- Nodes which are less than root will be in the left subtree.
- Nodes which are greater than root will be right subtree.
- A binary search tree should not have duplicate nodes.
- Both sides subtree (left and right) also should be a binary search tree.

Algorithm Interview Questions & Answers

50
30
70
20
25
60
80
10
65

Build this binary search tree in Box N





Binary Search Tree

Algorithm Interview Questions & Answers

Write an algorithm to insert a node in the Binary search tree

Compare the node to be inserted with the root node and traverse left (if smaller) or right (if greater) according to the value of the node to be inserted.

Write your algorithm in Box O

Algorithm

Set root node as the current node

If the node to be inserted < root

If it has left child, then traverse left

If it does not have left child, insert node here

If the node to be inserted > root

If it has the right child, traverse right

If it does not have the right child, insert node here.

Algorithm Interview Questions & Answers

Count the leaf nodes of the binary tree

Put your algorithm in Box P

Algorithm

Traverse the tree (inorder, preorder or post order)

If a node is a leaf (both right and left pointers are NULL),
then increment leaf counter

Algorithm Interview Questions & Answers

What is the difference between the Singly Linked List and Doubly Linked List?

Put your answer in Box Q

This is a traditional interview question on the data structure. The major difference between the singly linked list and the doubly linked list is the ability to traverse.

You cannot traverse back in a singly linked list because in it a node only points towards the next node and there is no pointer to the previous node.

On the other hand, the doubly linked list allows you to navigate in both directions in any linked list because it maintains two pointers towards the next and previous node.

Algorithm Interview Questions & Answers

What is a hash algorithm and how is it used?

Put your answer in Box R

You will want to get comfortable answering this question because hash algorithms are popular now due to their use in cryptography.

A hash algorithm refers to a hash function, which takes a string and converts it to a fixed length regardless of how long it was to begin with.

You can use it for a wide range of applications, from cryptocurrency to passwords and a range of other validation tools.

Algorithm Interview Questions & Answers

What are the three laws that govern a recursive algorithm?

Put your answer in Box S

These kinds of algorithm interview questions may come as follow-ups for the "What is a recursive algorithm?" question.

A recursive algorithm needs to follow these laws:

It has to have a base case.

It has to call itself.

It needs to change its state and shift towards the base case.

Algorithm Interview Questions & Answers

What Are the Different Types of Data Structures?

T1. A collection of data values stored sequentially	Arrays
T2. Last-in-first-out (LIFO) data structures where the element placed last is accessed first.	Stacks
T3. A first-in-first-out data structure.	Queues
T4. A collection of data values stored in a linear order and connected to each other	Linked lists
T5. A data structure in which data values are placed in nodes connected by edges	Graphs
T6. Similar to a linked list, but with data values linked in a hierarchical fashion	Trees
T7. A binary tree data structure wherein parent data values can be compared to child data values	Heaps
T8. A table where each value is assigned a key and then stored, making accessing individual values easy.	Hash table

Good Websites for Algorithm Stuff

Desmos

  www.desmos.com/calculator

Know Thy Complexities

  https://www.bigocheatsheet.com/

Dijkstra, Breadth-first, Minimum Spanning Tree

  https://graphonline.ru/en/

Good Websites for Algorithm Stuff

VisuAlgo – adjacency matrix, adjacency list, edge list, directed, undirected, weighted

https://visualgo.net/en/graphds

Tool for drawing graphs

https://app.diagrams.net/

Recursion Problems with Solutions

https://www.techiedelight.com/recursion-practice-problems-with-solutions/

Binary Heaps

http://btv.melezinek.cz/binary-heap.html

Action Items

Tue, Aug 6

Due 11:59pm Homework 7

Wed, Aug 7

Due 11:59pm OLQ10

Due 11:59pm Coding Assignment 6