Linked Lists

- list elements are stored, in memory, in an arbitrary order
- explicit information (called a link) is used to go from one element to the next

Memory Layout

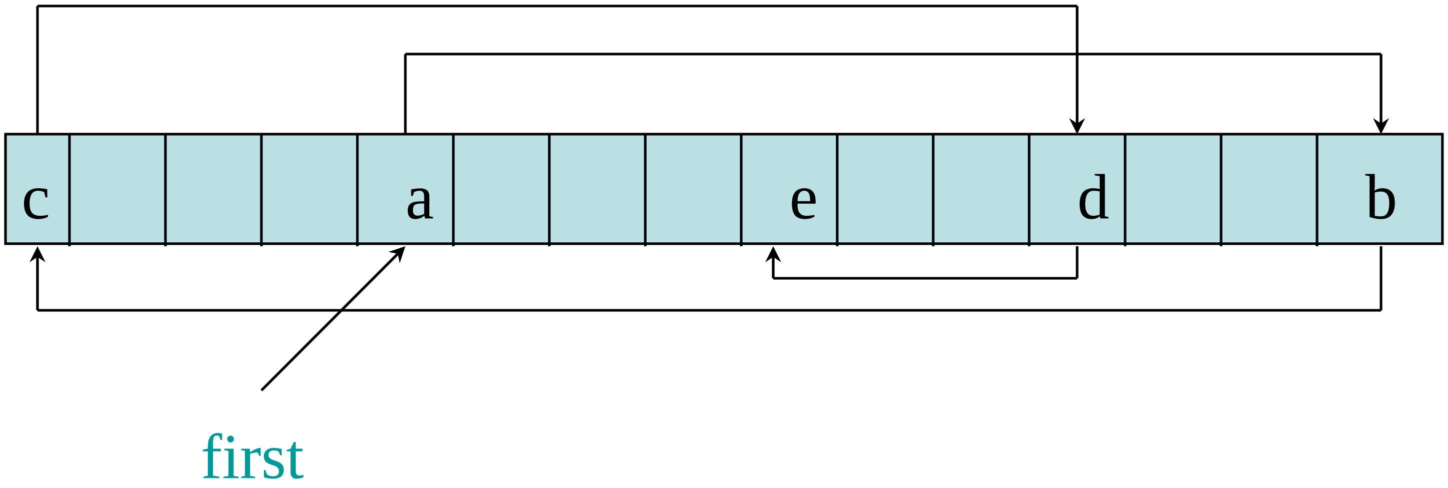
Layout of L = (a,b,c,d,e) using an array representation.

a	b	c	d	e										

A linked representation uses an arbitrary layout.

c				a				e			d			b

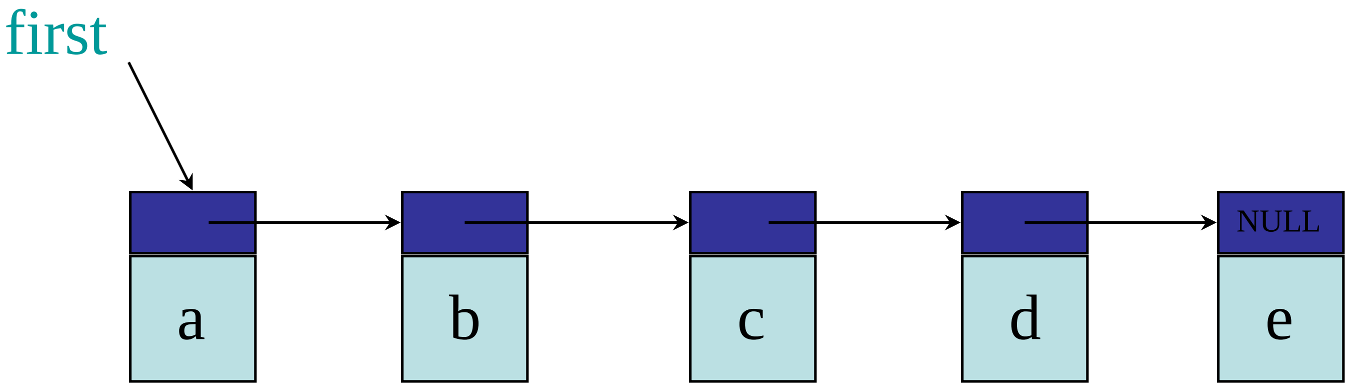
Linked Representation



pointer (or link) in e is NULL

use a variable first to get to the first element a

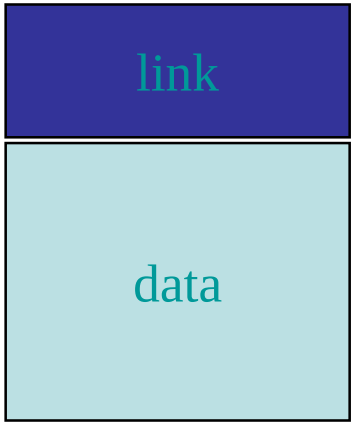
Normal Way To Draw A Linked List



🟦 link or pointer field of node

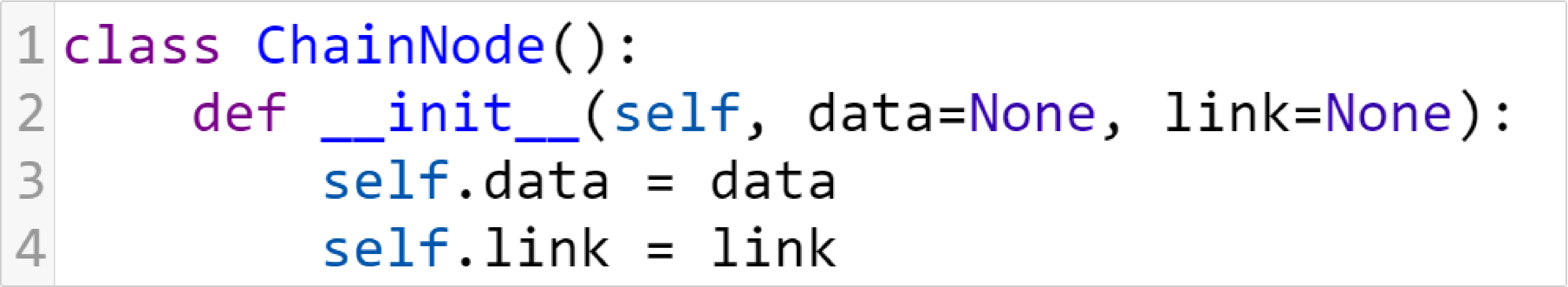
⬜ data field of node

Node Representation



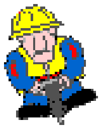
link

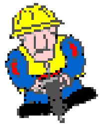
data

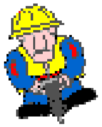


```python
class ChainNode():
    def __init__(self, data=None, link=None):
        self.data = data
        self.link = link
```

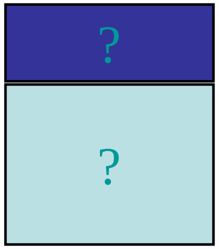
Constructors Of ChainNode



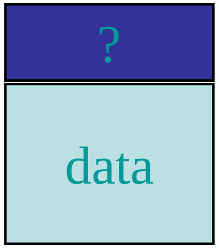




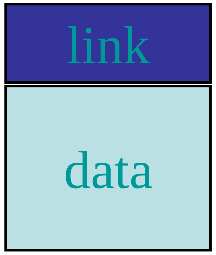
node = ChainNode()



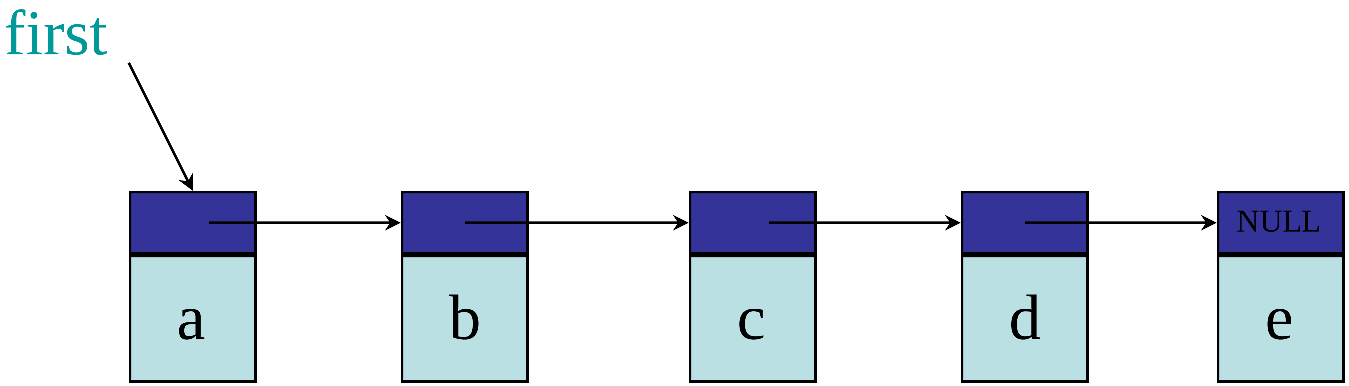
node = ChainNode(data)



node = ChainNode(data, link)

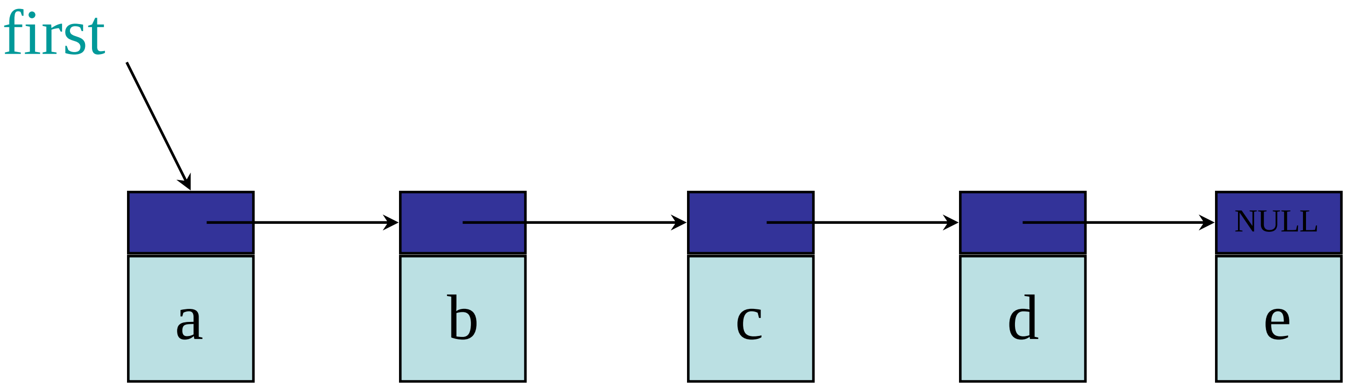


Chain



- A chain is a linked list in which each node represents one element.
- There is a link or pointer from one element to the next.
- The last node has a NULL (or 0) pointer.

The Class Chain



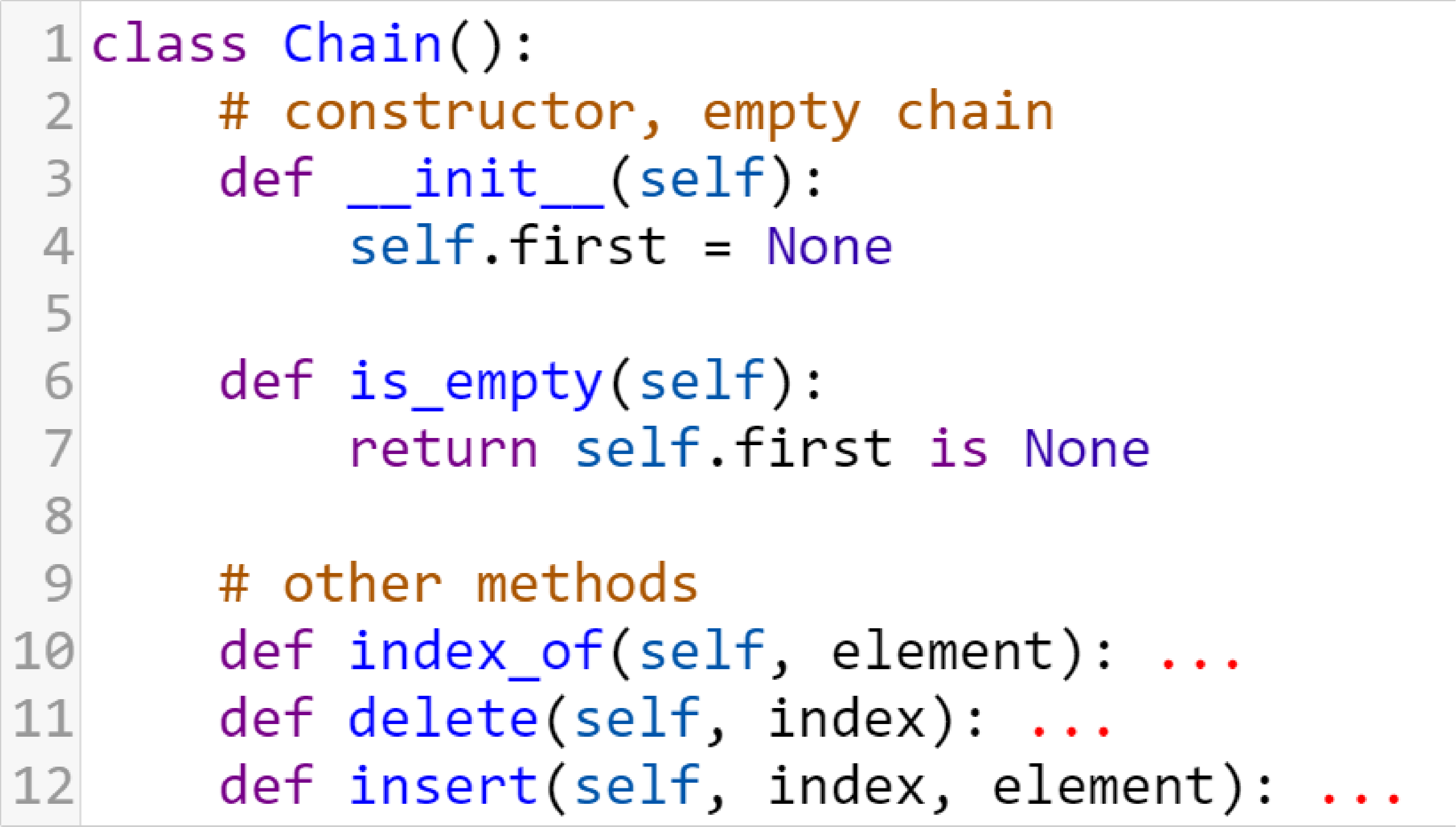
Use ChainNode

🟦 link

⬜ data

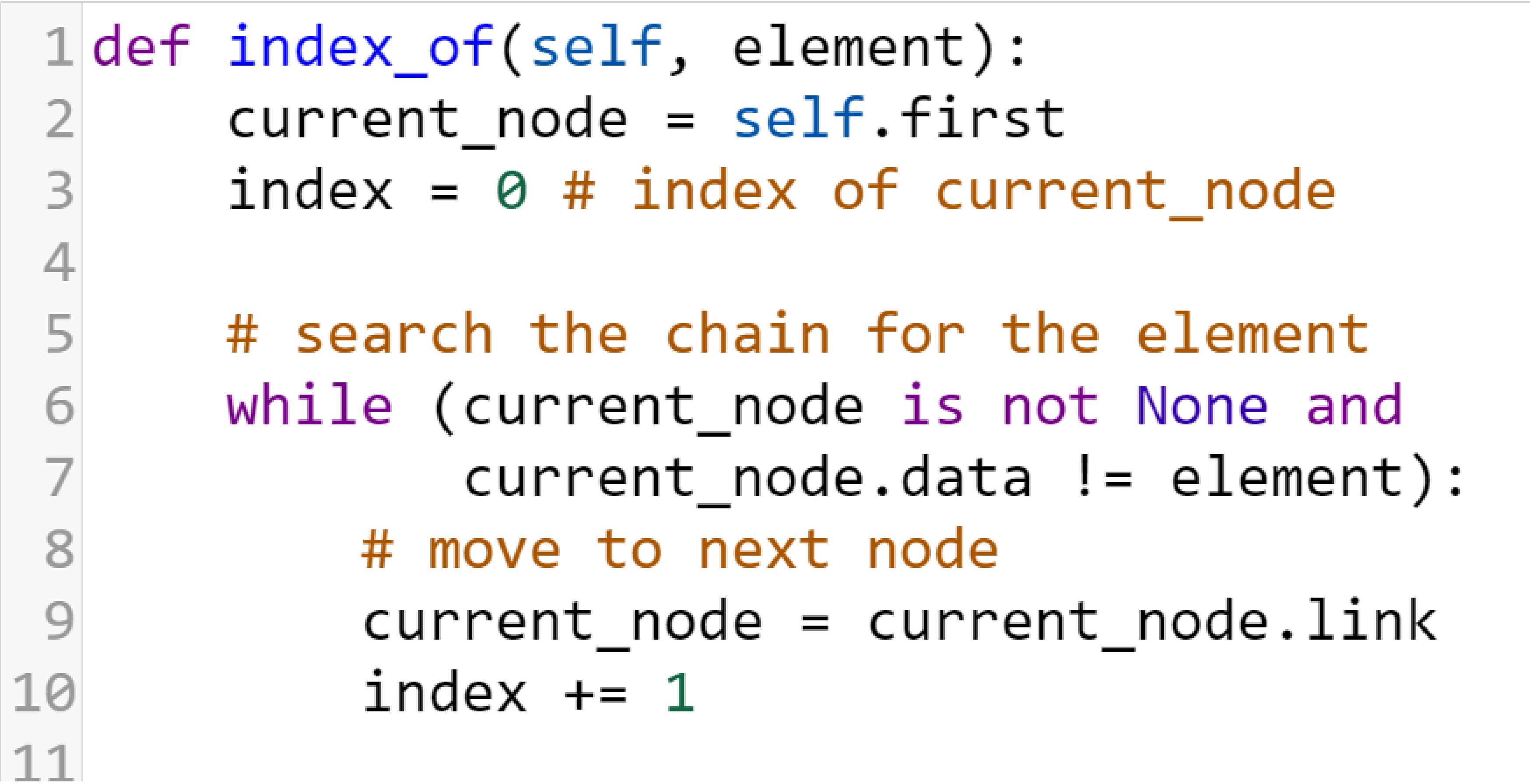
```
class ChainNode():
    def __init__(self, data=None, link=None):
        self.data = data
        self.link = link
```

The Template Class Chain (P 4.6)



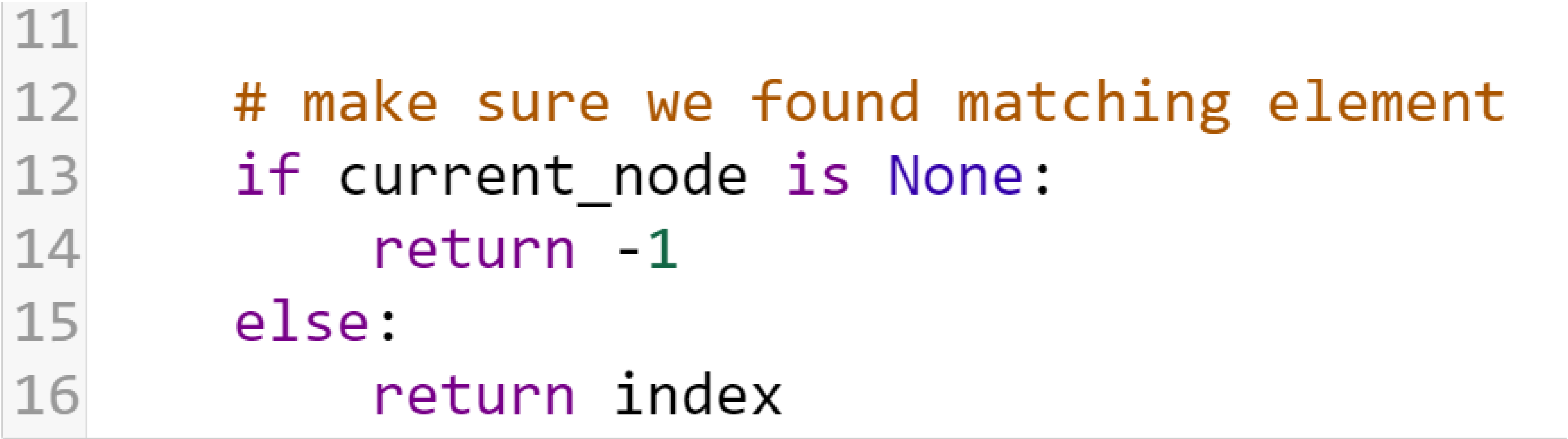
```
class Chain():
    # constructor, empty chain
    def __init__(self):
        self.first = None

    def is_empty(self):
        return self.first is None

    # other methods
    def index_of(self, element): ...
    def delete(self, index): ...
    def insert(self, index, element): ...
```

The Method IndexOf



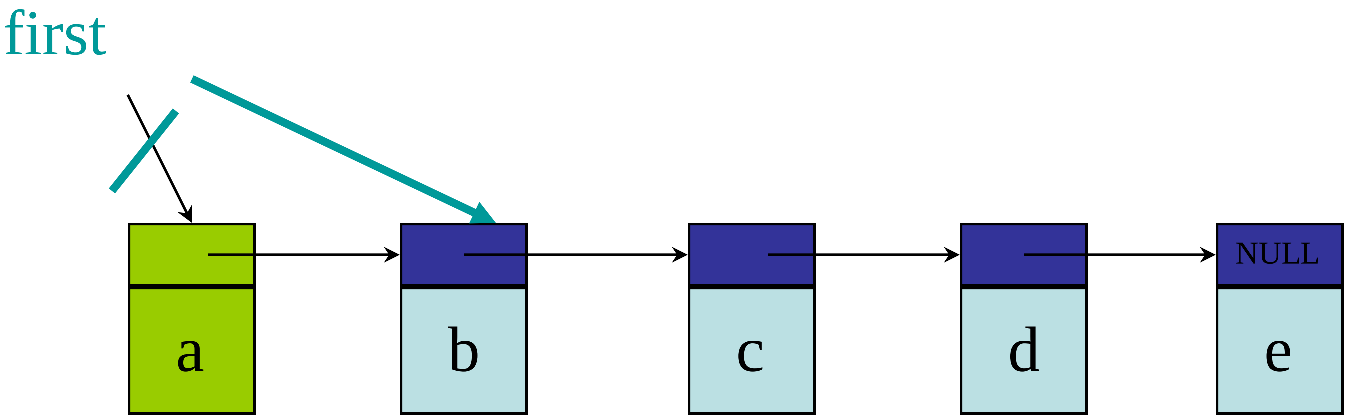
```
def index_of(self, element):
    current_node = self.first
    index = 0 # index of current_node

    # search the chain for the element
    while (current_node is not None and
           current_node.data != element):
        # move to next node
        current_node = current_node.link
        index += 1

```

The Method IndexOf



```

    # make sure we found matching element
    if current_node is None:
        return -1
    else:
        return index
```

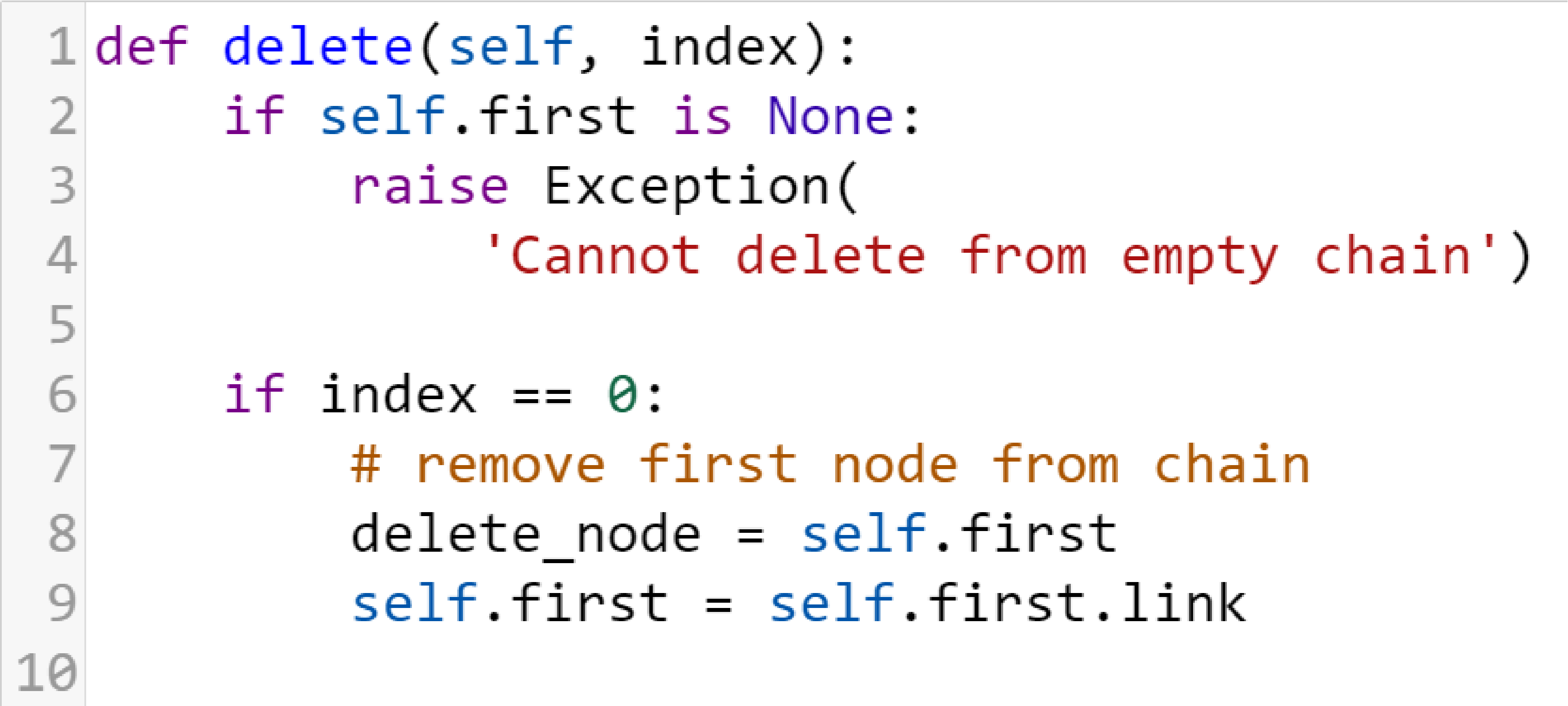
Delete An Element



delete(0)

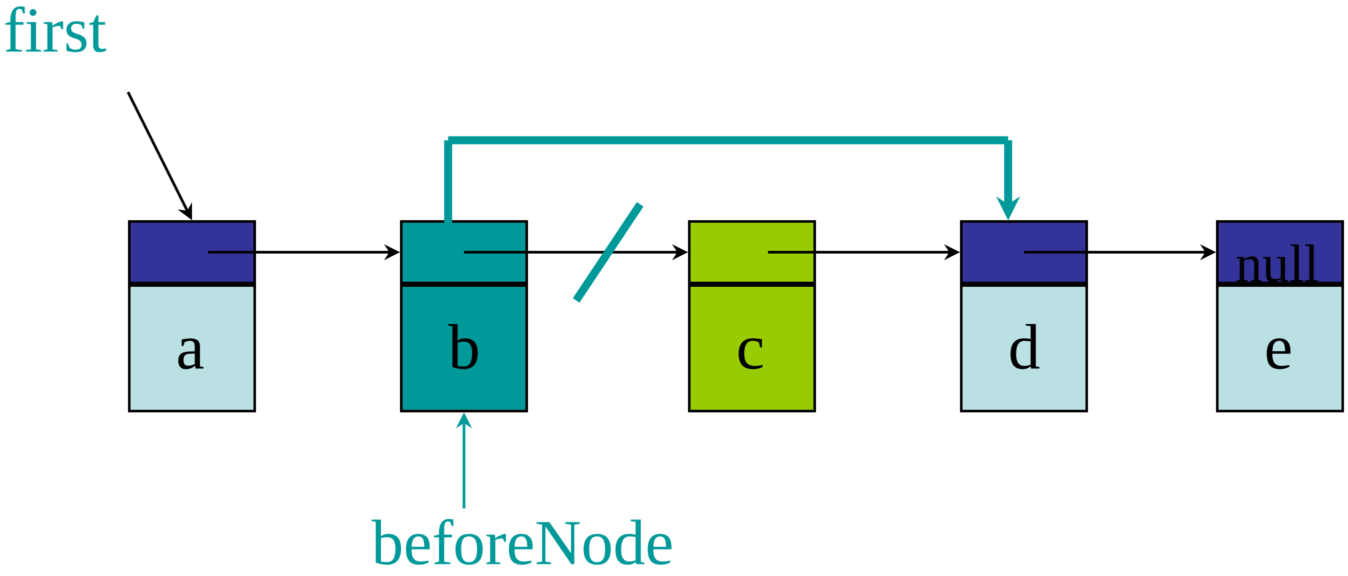
```
deleteNode = first
first = first.link
del deleteNode
```

Delete An Element(0)



```
def delete(self, index):
    if self.first is None:
        raise Exception(
            'Cannot delete from empty chain')

    if index == 0:
        # remove first node from chain
        delete_node = self.first
        self.first = self.first.link

```

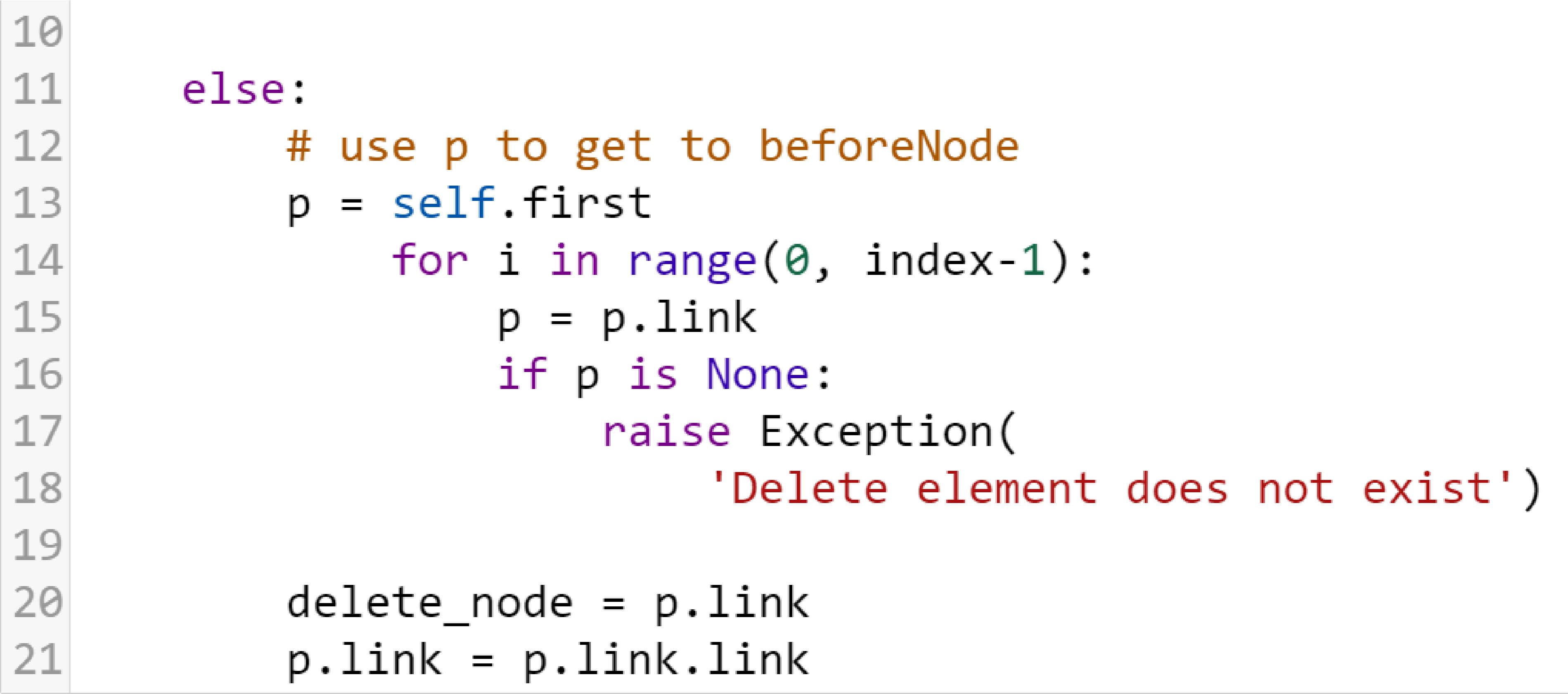
Delete(2)



Find & change pointer in beforeNode

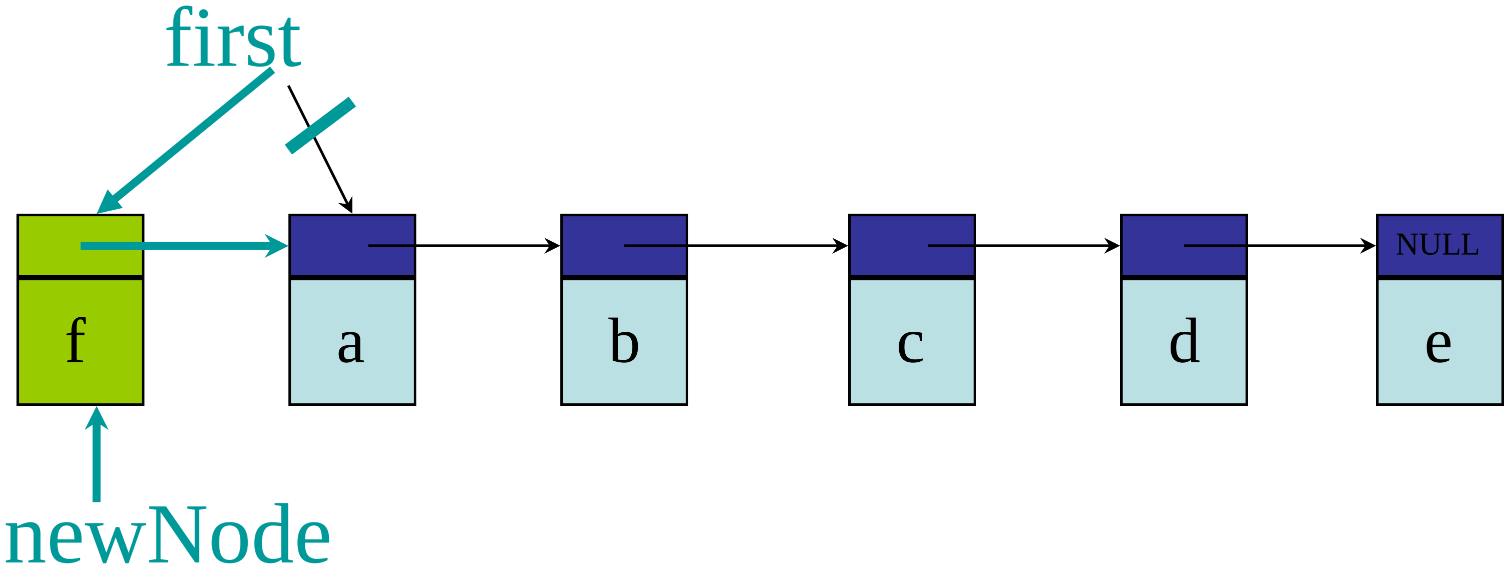
```
beforeNode.link = beforeNode.link.link
del deleteNode
```

Delete An Element



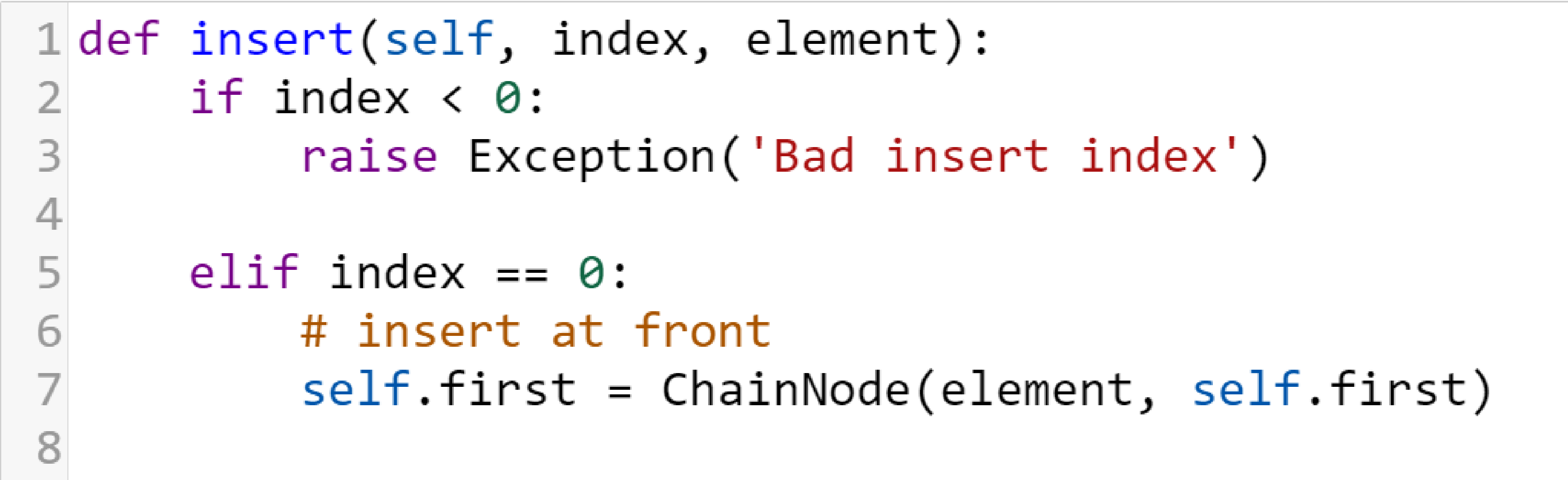
```

    else:
        # use p to get to beforeNode
        p = self.first
            for i in range(0, index-1):
                p = p.link
                if p is None:
                    raise Exception(
                        'Delete element does not exist')

        delete_node = p.link
        p.link = p.link.link
```

One-Step Insert(0,'f')



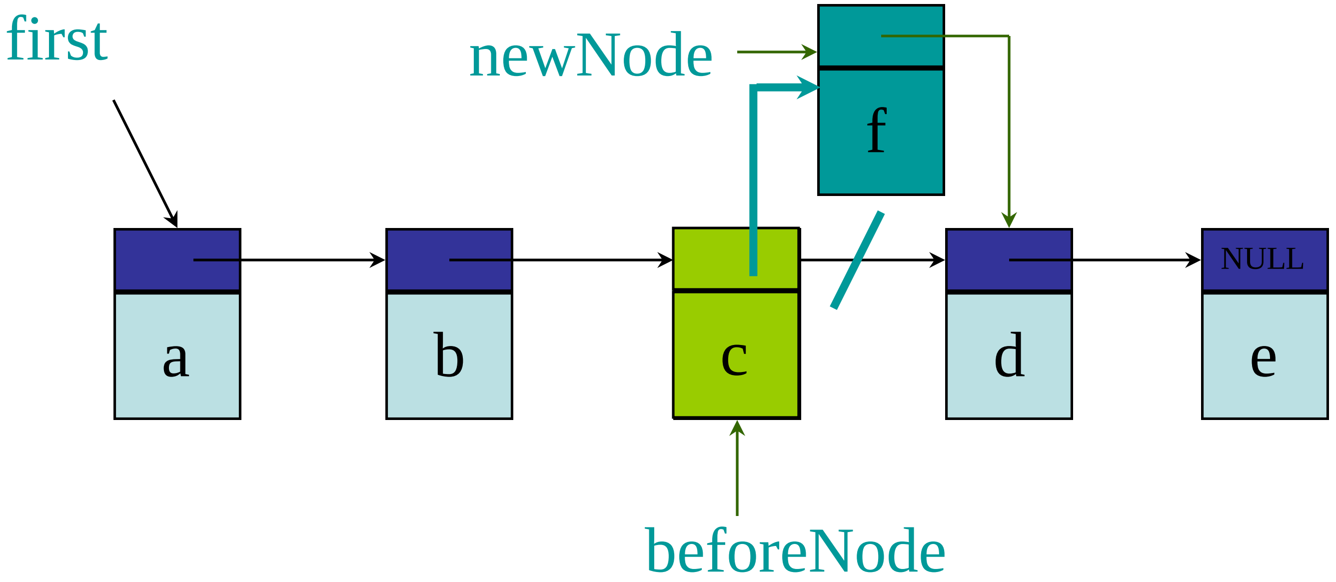
first = ChainNode('f', first)

Insert An Element



```
def insert(self, index, element):
    if index < 0:
        raise Exception('Bad insert index')

    elif index == 0:
        # insert at front
        self.first = ChainNode(element, self.first)

```

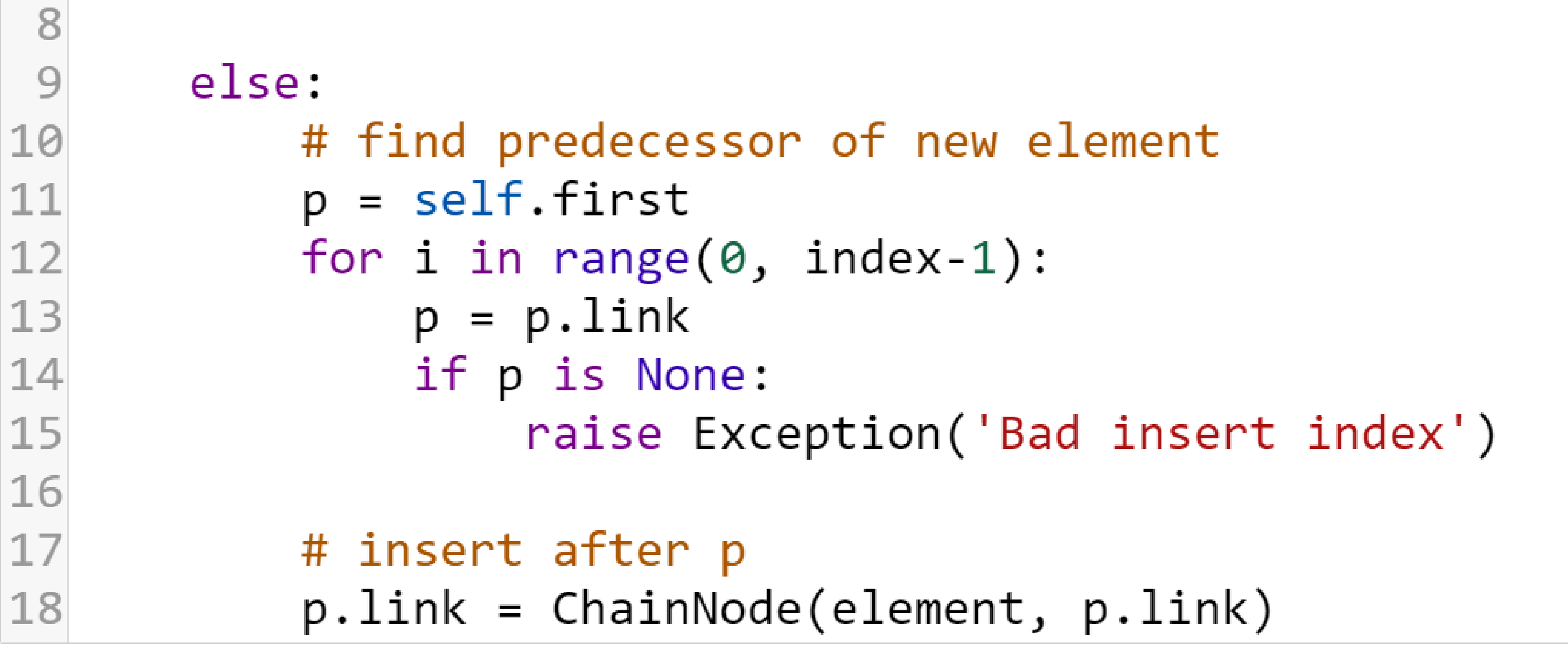
Two-Step Insert(3,’f’)



beforeNode = first.link.link

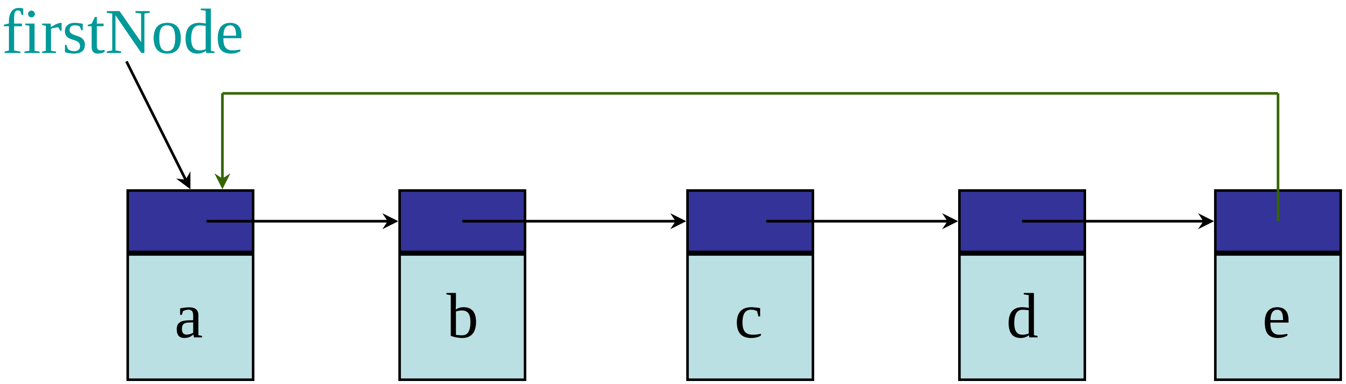
beforeNode.link = ChainNode(‘f’, beforeNode.link)

Inserting An Element

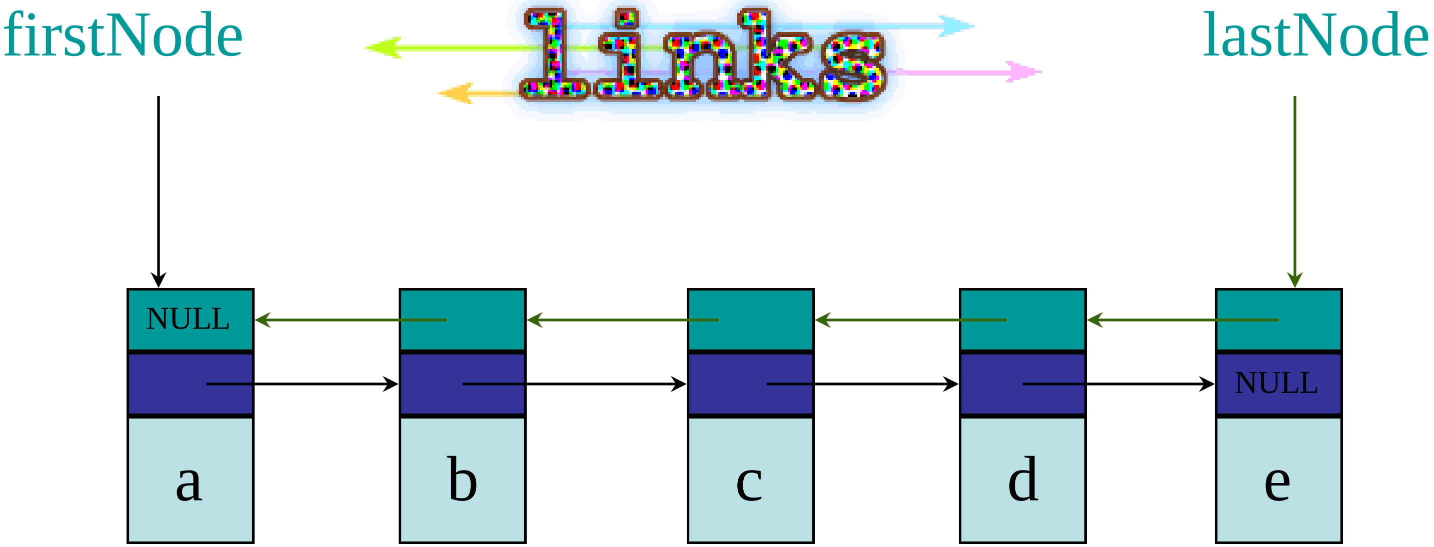


```
    else:
        # find predecessor of new element
        p = self.first
        for i in range(0, index-1):
            p = p.link
            if p is None:
                raise Exception('Bad insert index')

        # insert after p
        p.link = ChainNode(element, p.link)
```

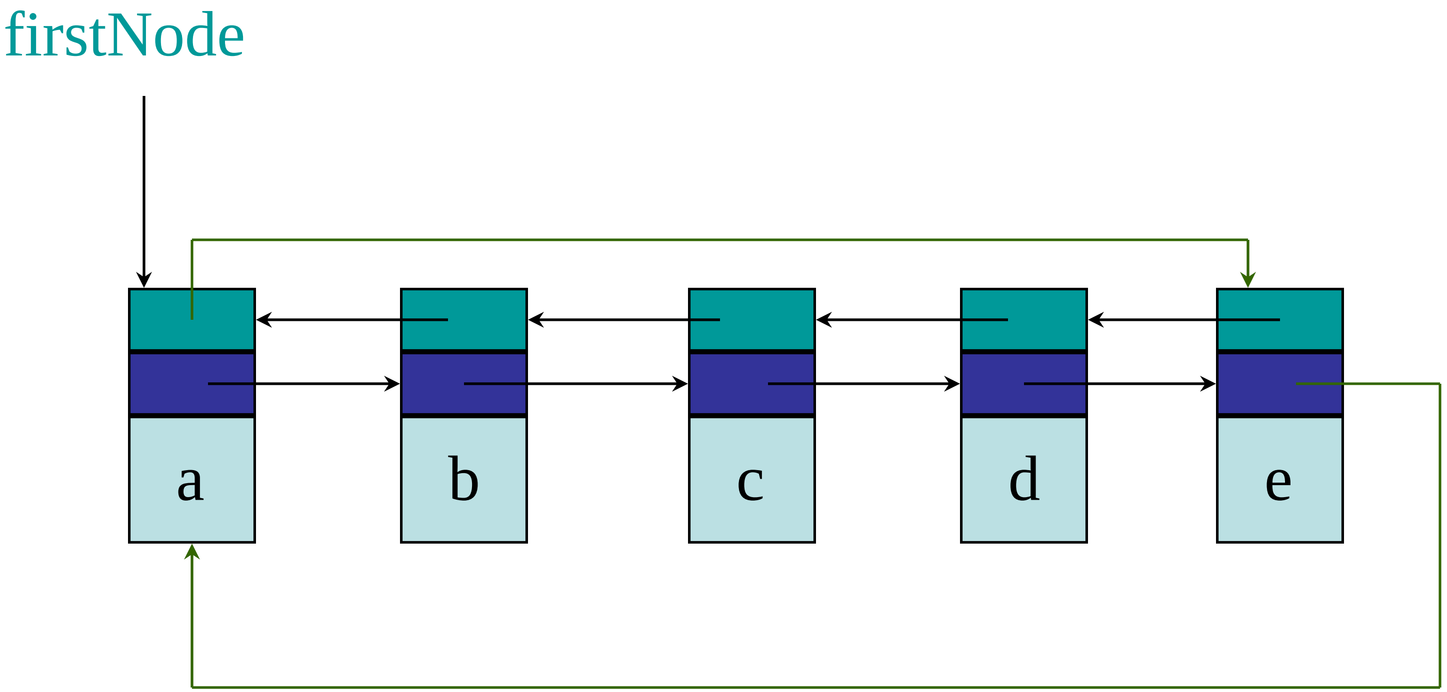
Circular List



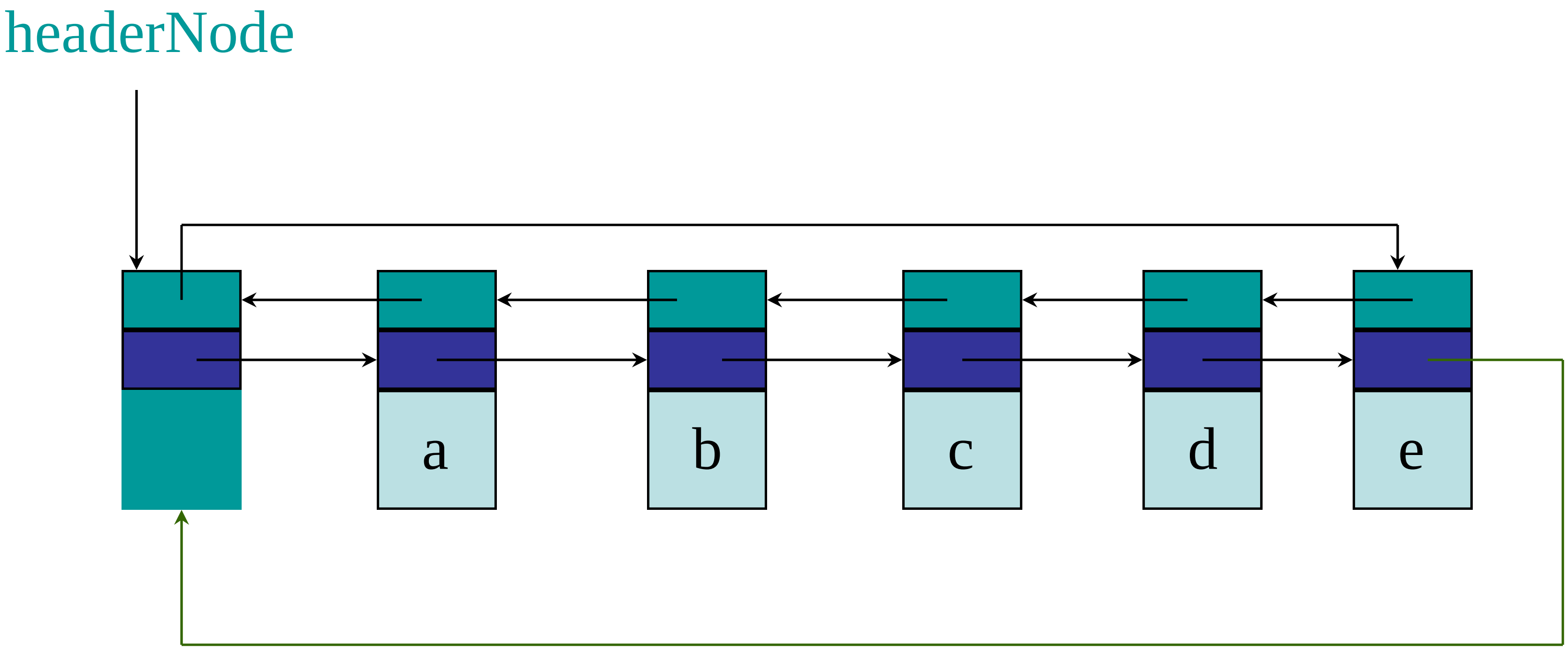
Doubly Linked List



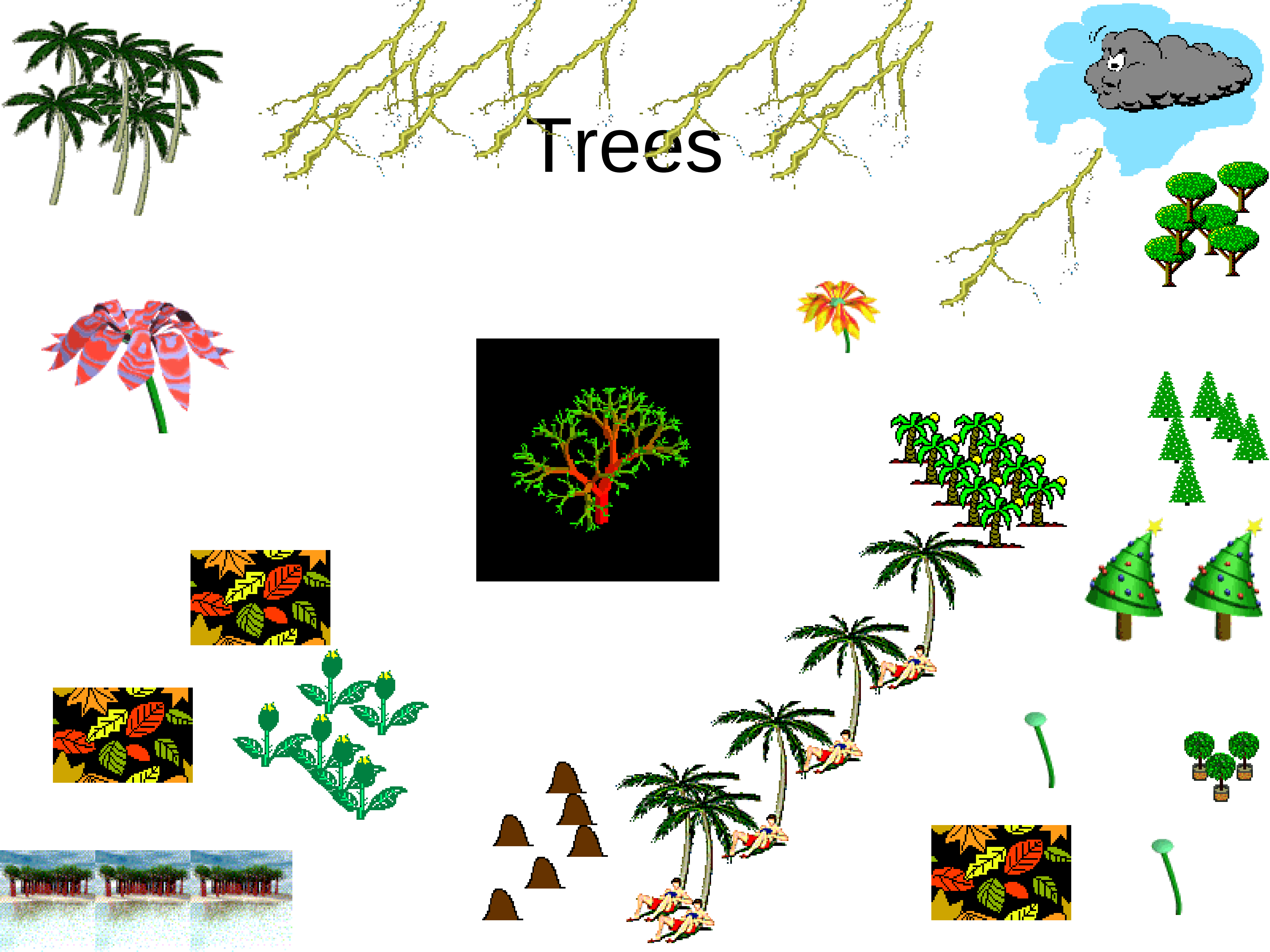
Doubly Linked Circular List



Doubly Linked Circular List With Header Node



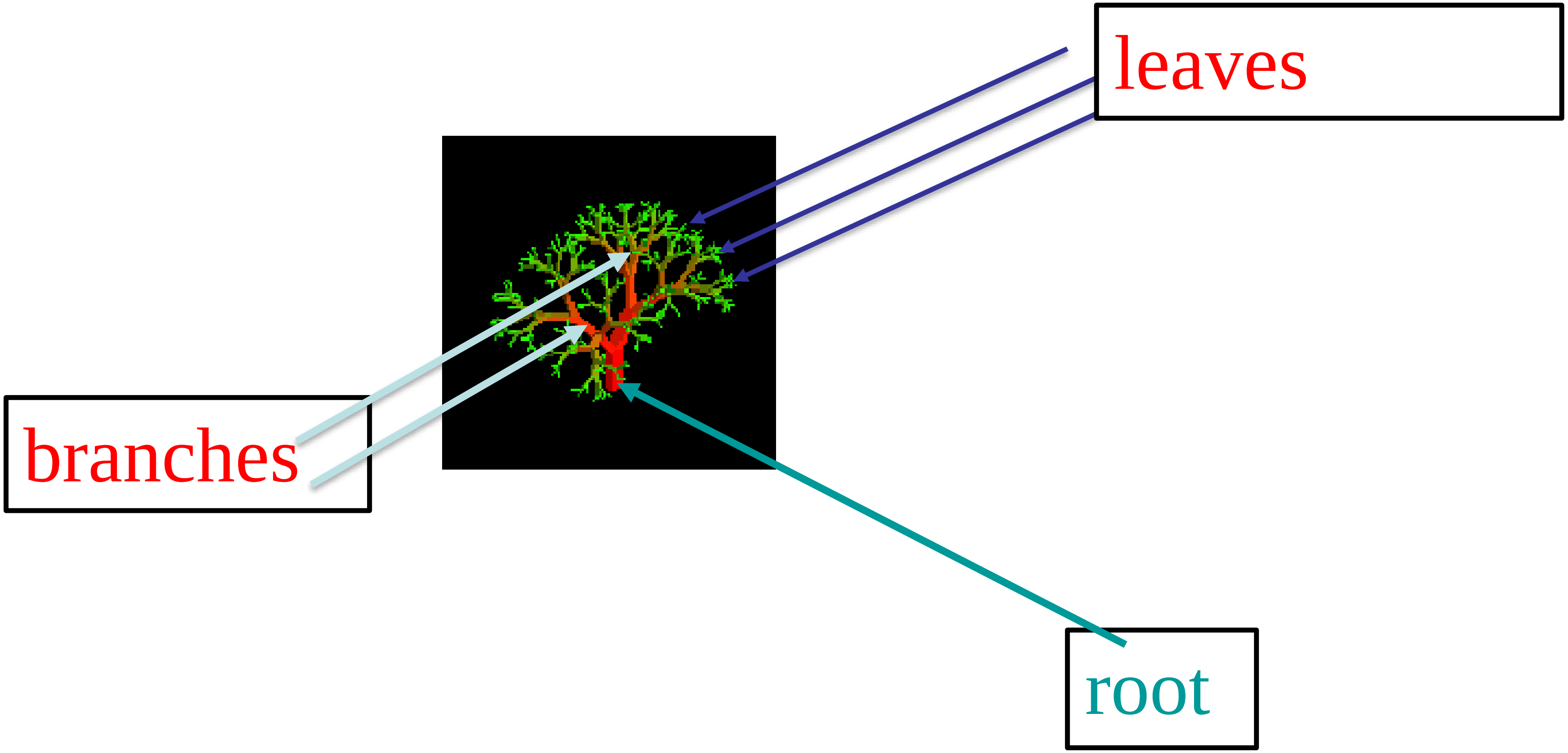
Trees



About Tree

- Definition of Tree
- Tree and Binary Tree
- What it can be used for ? An example
- Postfix, Infix, Prefix
- Full binary Tree and Complete Binary tree
- How to keep the tree data in array or linked list

Nature Lover's View Of A Tree

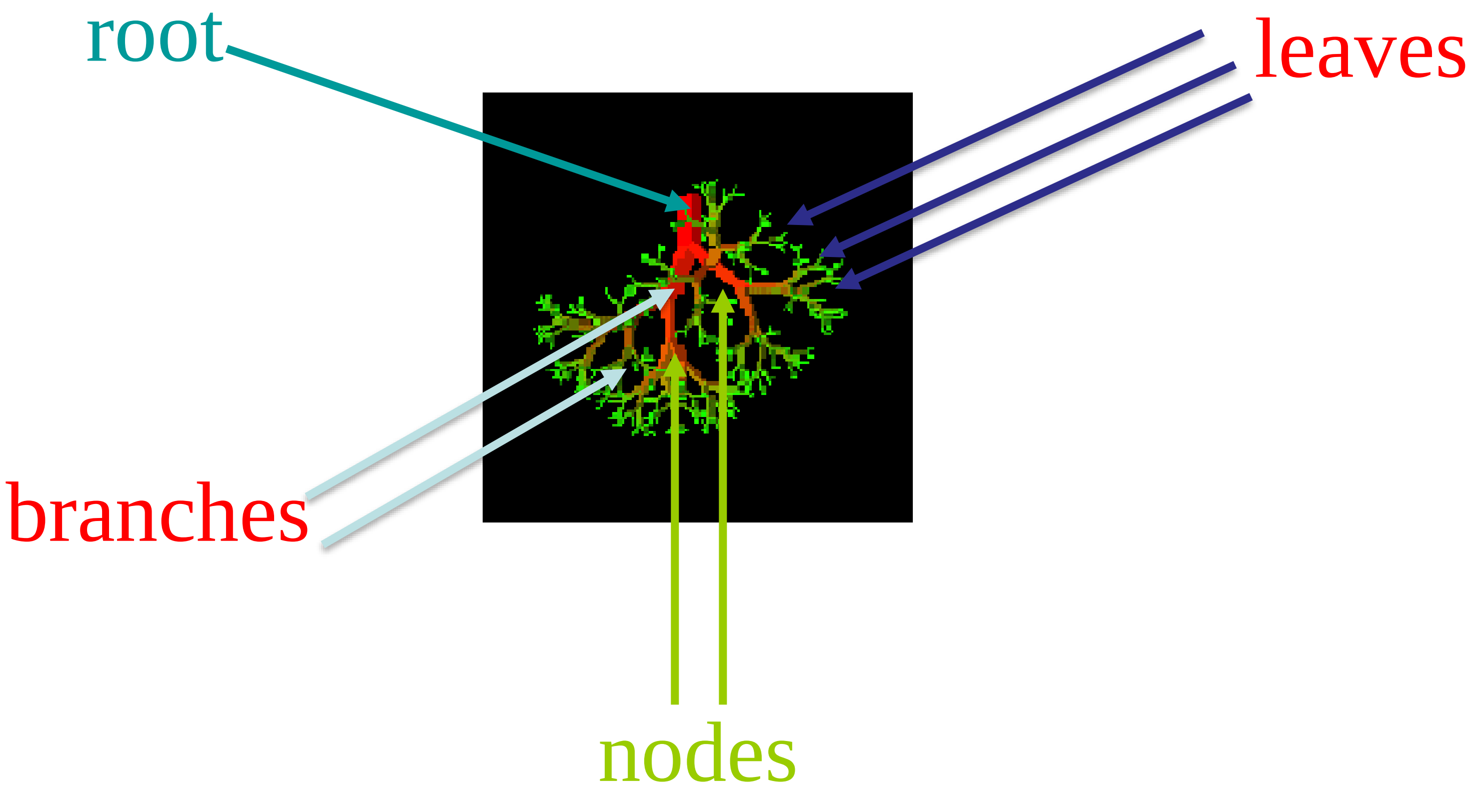


leaves

branches

root

Computer Scientist’s View



root

leaves

branches

nodes

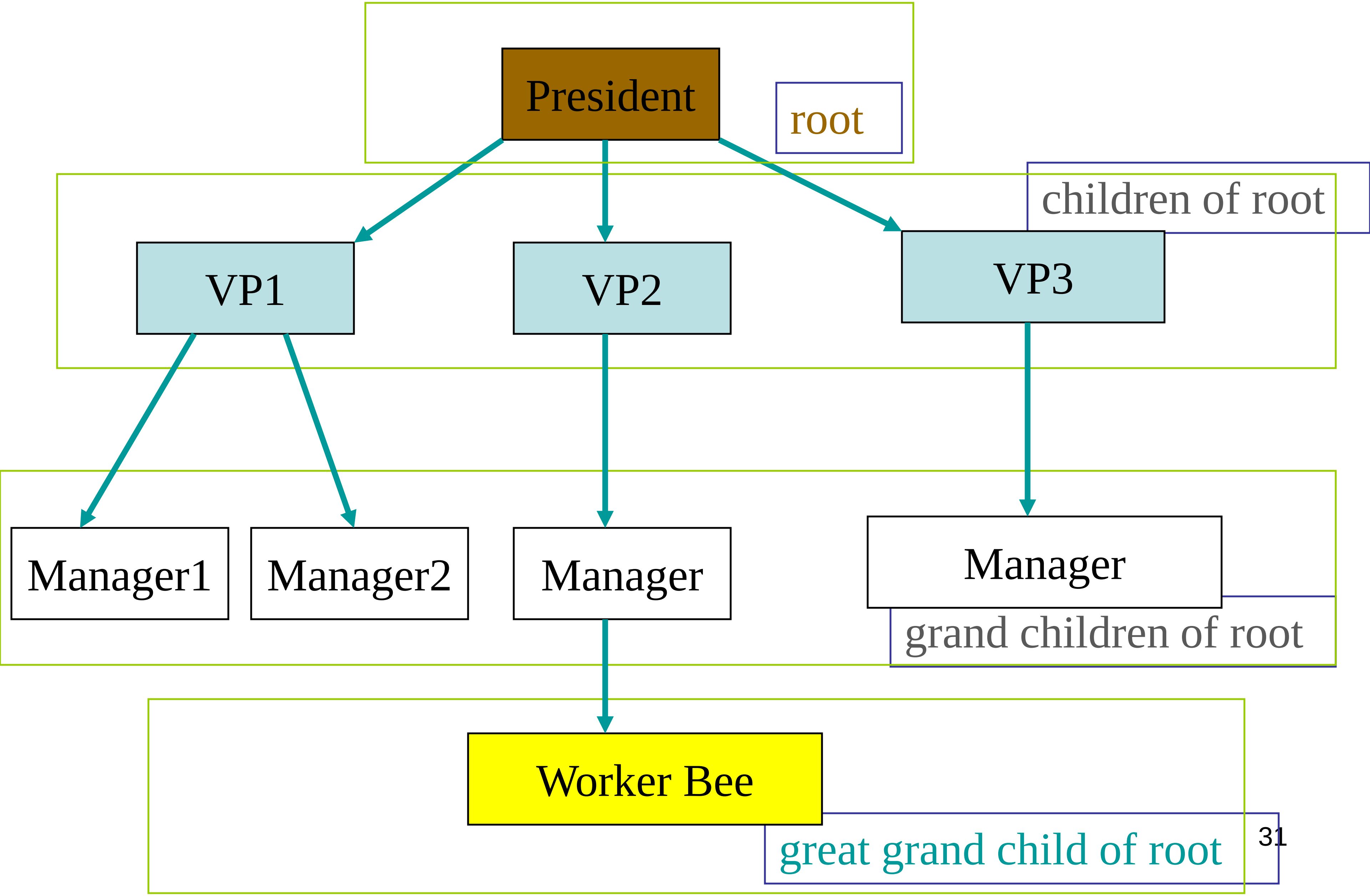
Linear Lists And Trees

- Linear lists are useful for serially ordered data.
  - $(e_0, e_1, e_2, \ldots, e_{n-1})$
  - Days of week.
  - Months in a year.
  - Students in this class.
- Trees are useful for hierarchically ordered data.
  - Employees of a corporation.
    - President, vice presidents, managers, and so on.

Hierarchical Data And Trees

- The element at the top of the hierarchy is the root.
- Elements next in the hierarchy are the children of the root.
- Elements next in the hierarchy are the grandchildren of the root, and so on.
- Elements that have no children are leaves.

Example Tree



President

root

VP1

VP2

VP3

children of root

Manager1

Manager2

Manager

Manager

grand children of root

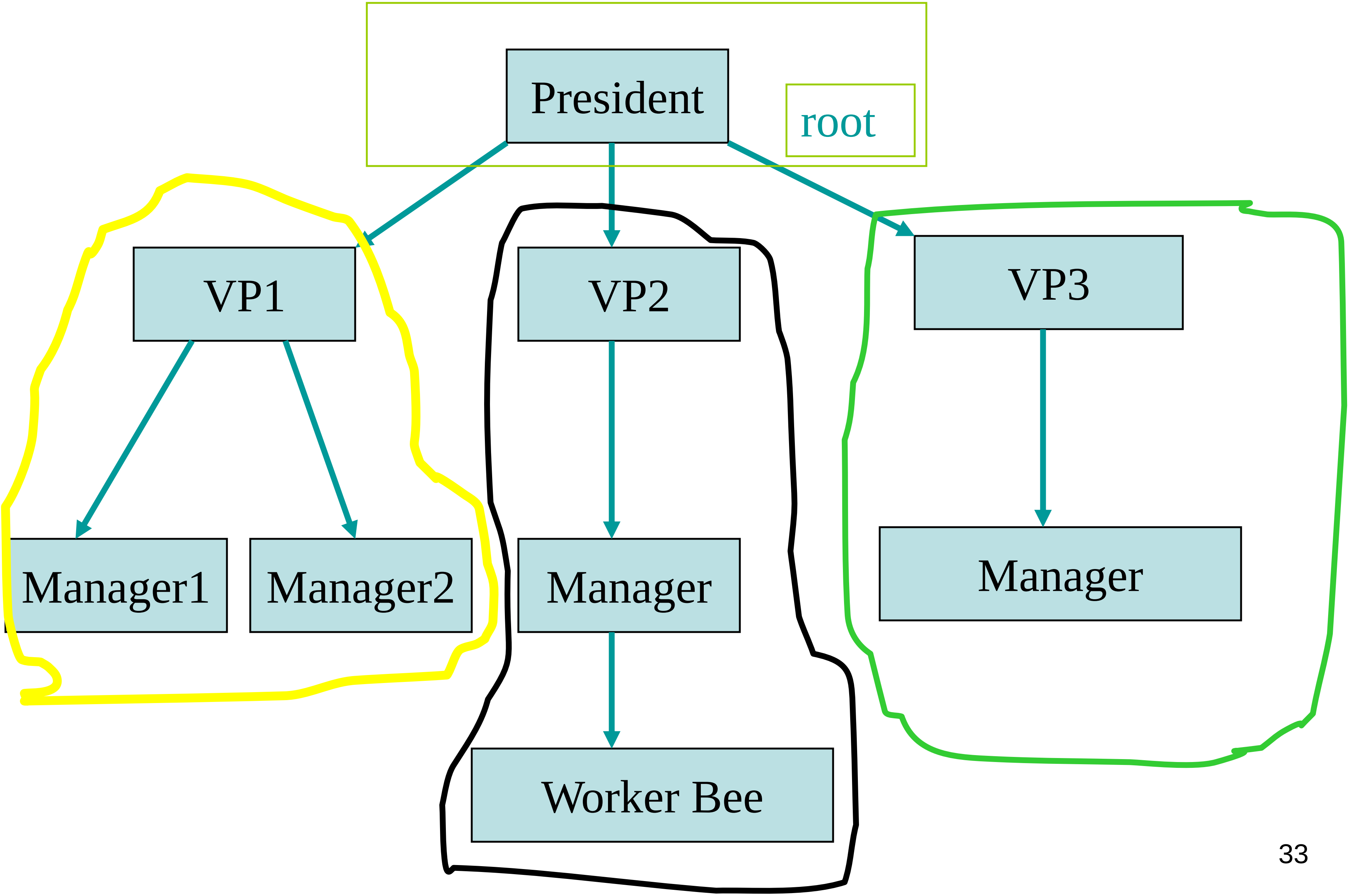
Worker Bee

great grand child of root

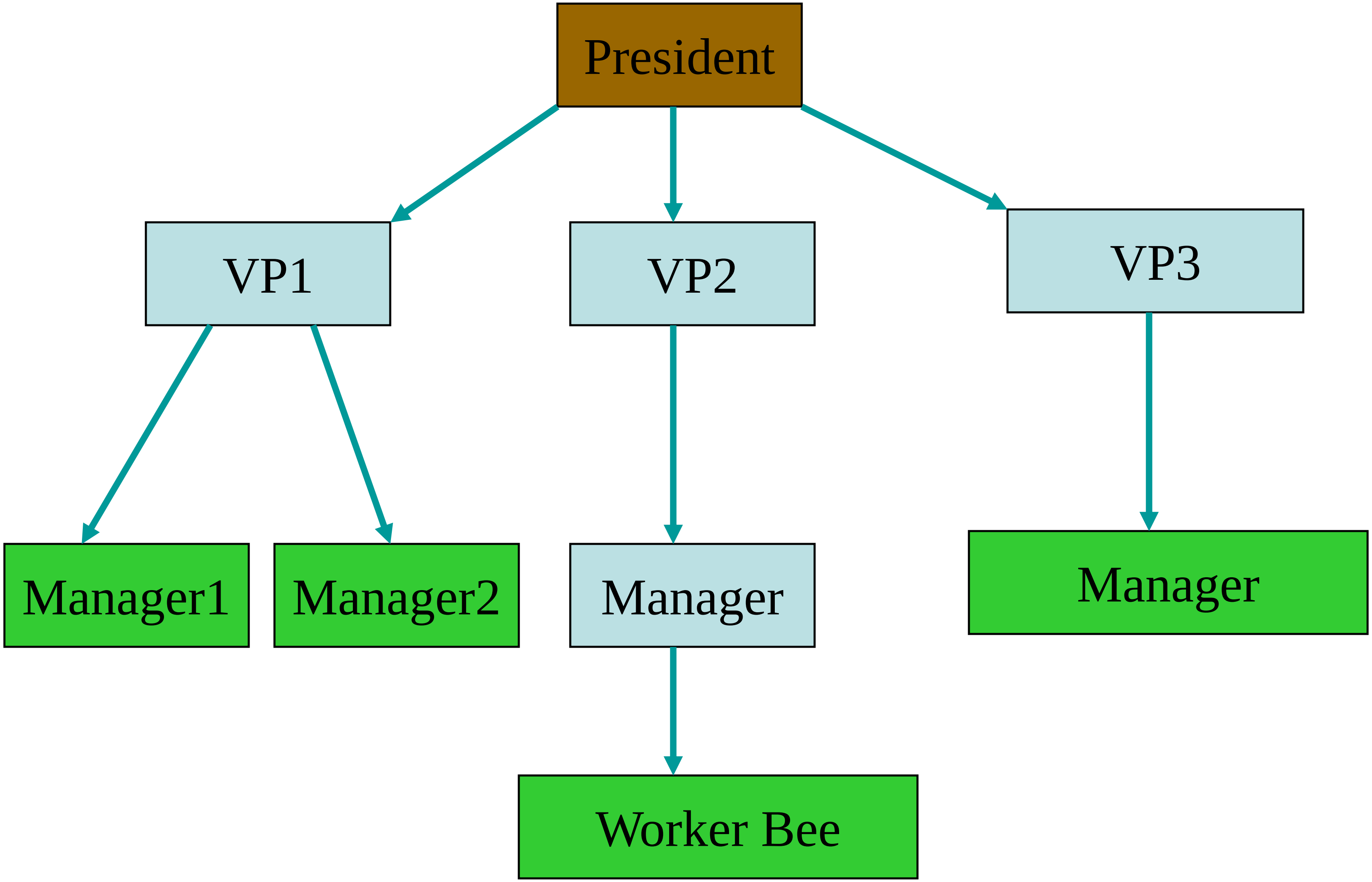
Definition

- A tree t is a finite nonempty set of elements.
- One of these elements is called the root.
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of t.

Subtrees


President
root
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Worker Bee

Leaves



President

VP1

VP2

VP3

Manager1

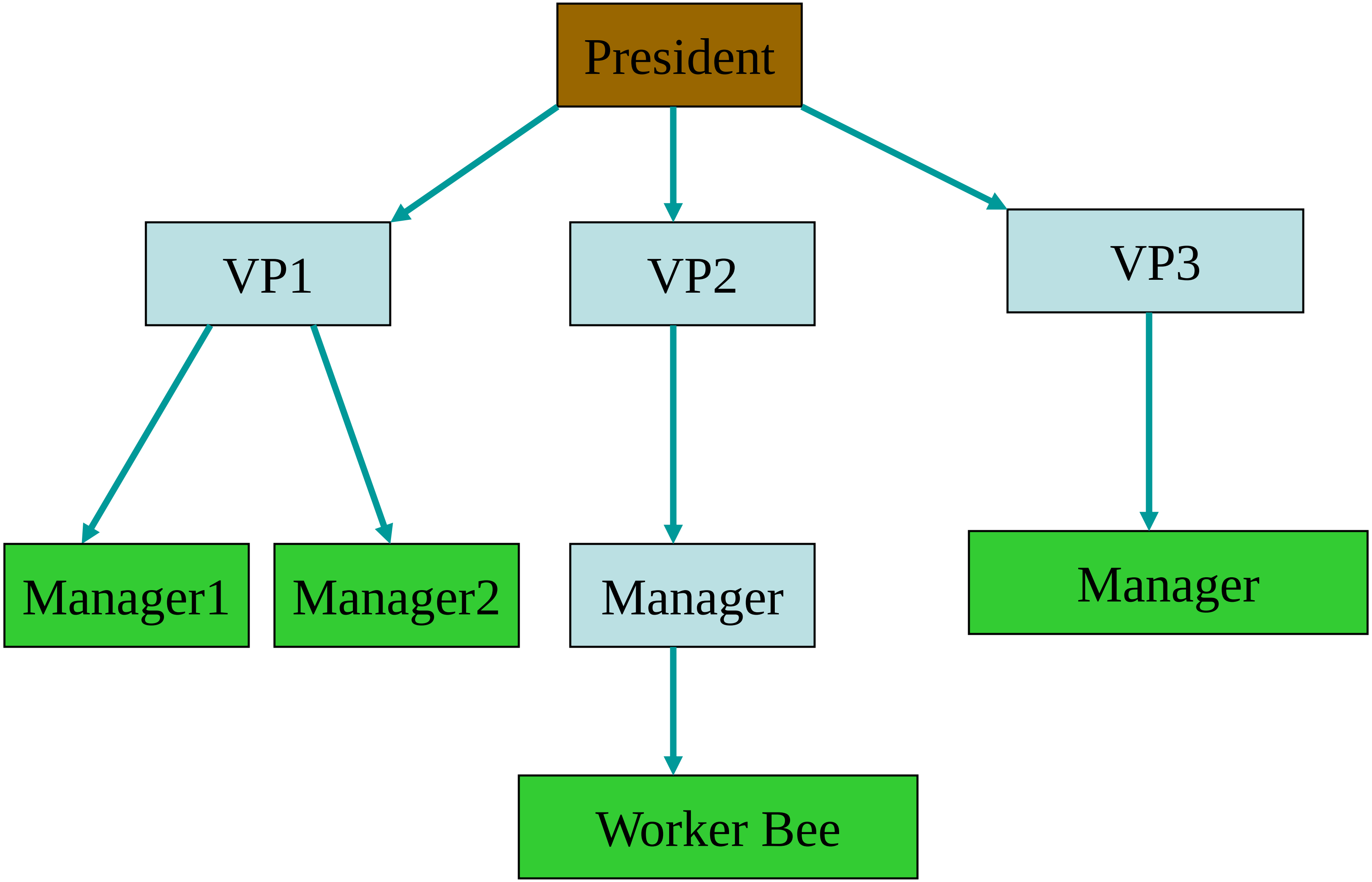
Manager2

Manager

Manager

Worker Bee

Parent, Grandparent, Siblings, Ancestors, Descendants


President

VP1

VP2

VP3

Manager1

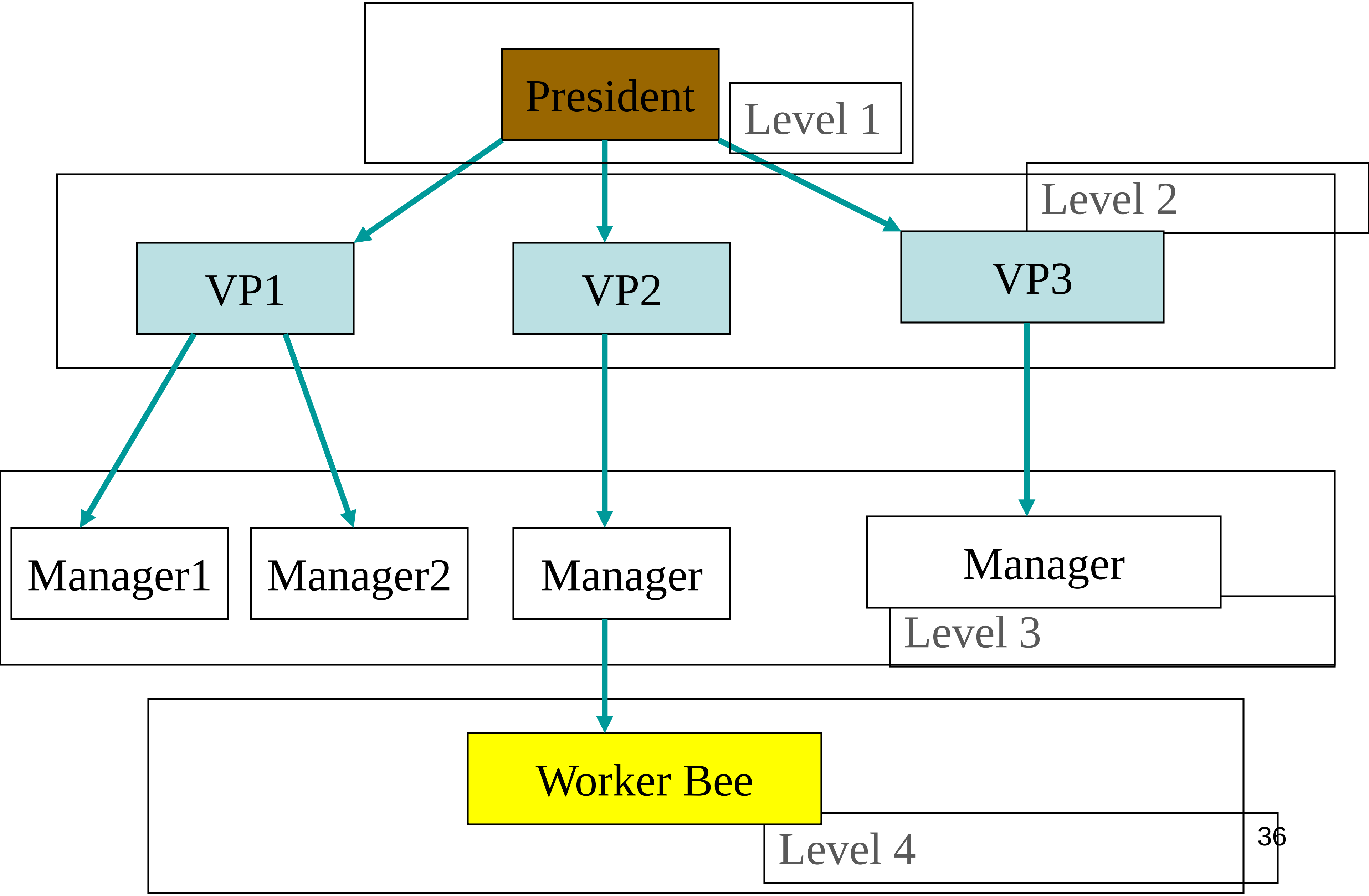
Manager2

Manager

Manager

Worker Bee

Levels



President

Level 1

Level 2

VP1

VP2

VP3

Manager1

Manager2

Manager

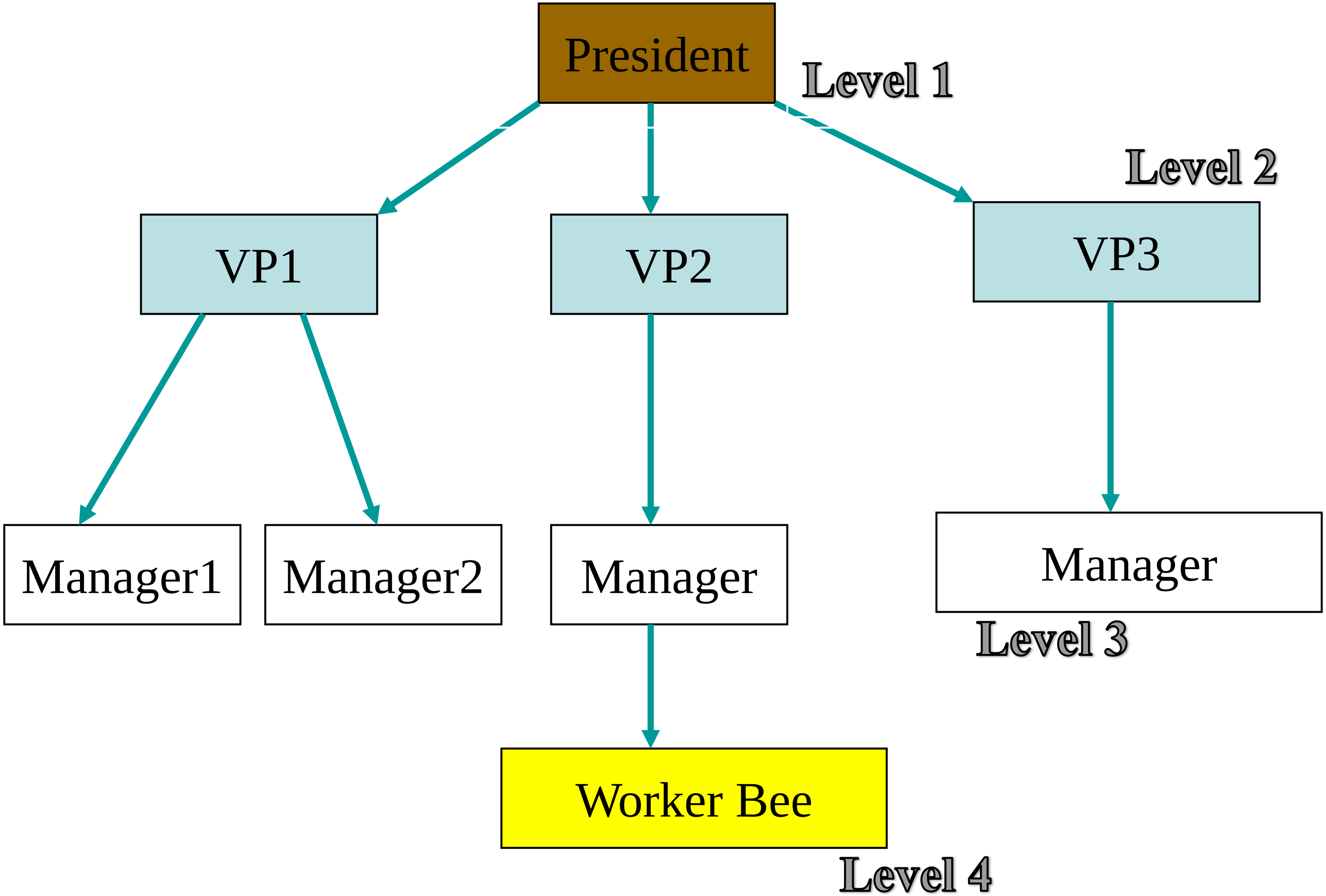
Manager

Level 3

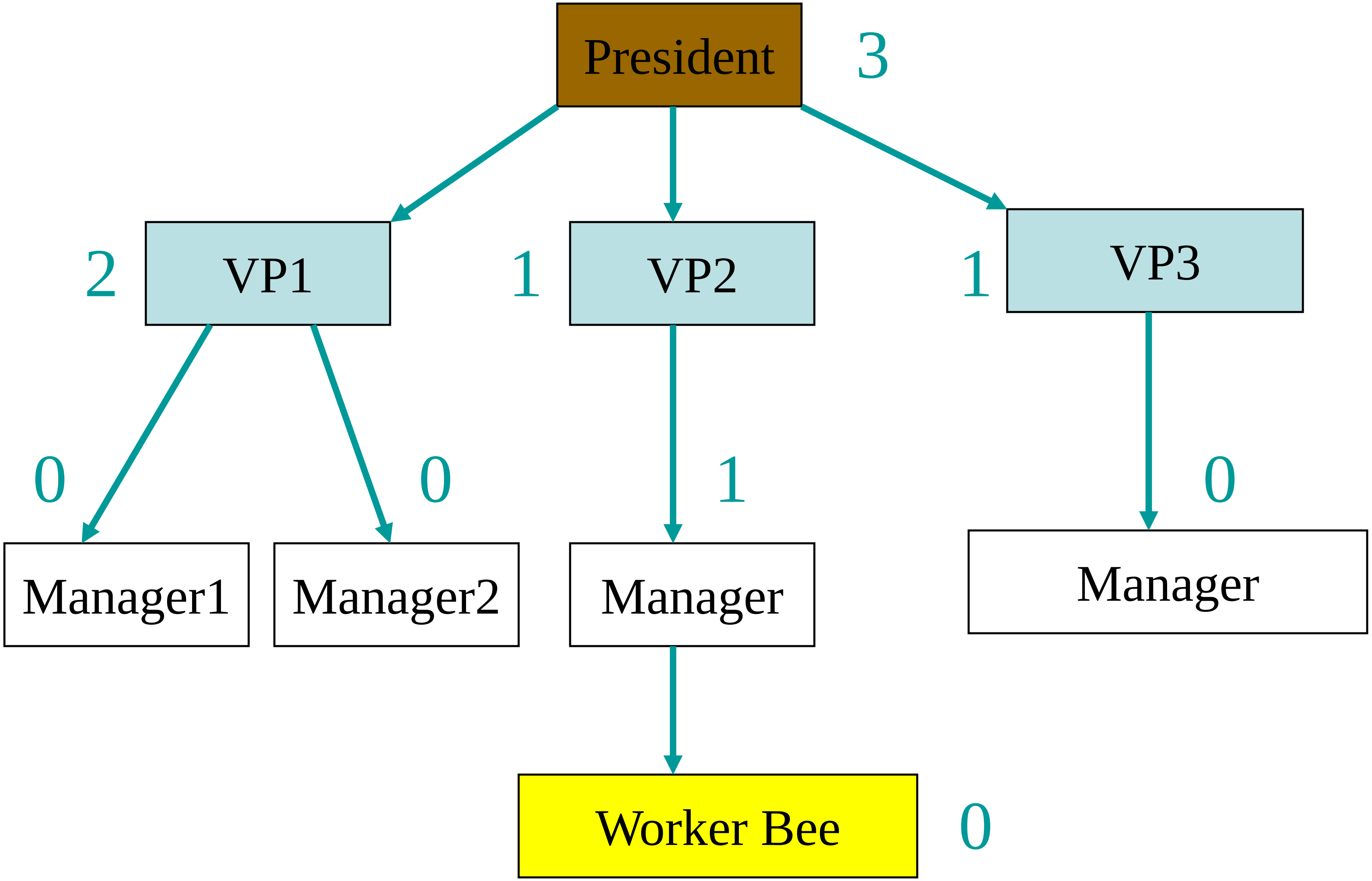
Worker Bee

Level 4

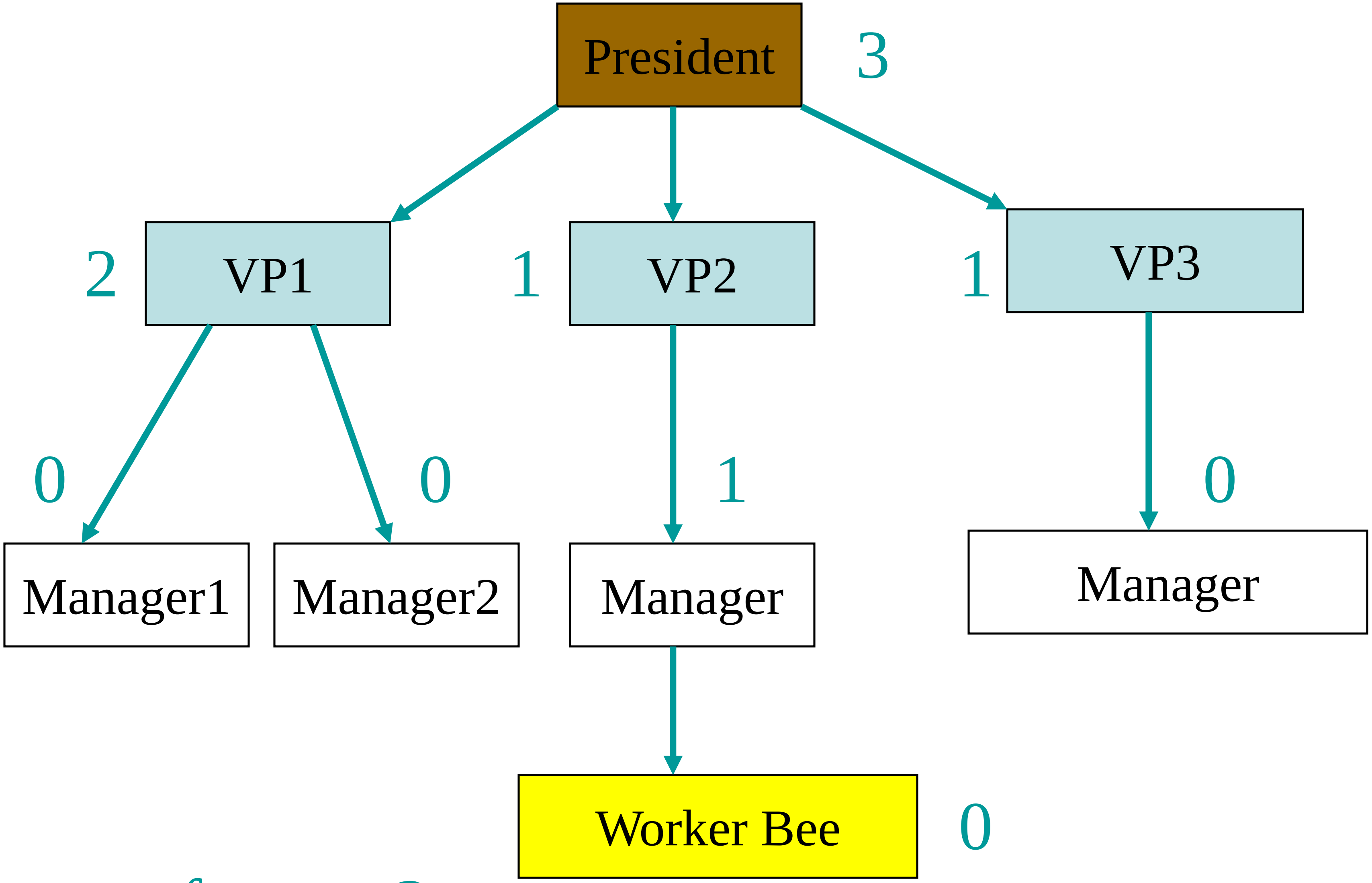
height = depth = number of levels


President
Level 1
Level 2
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Level 3
Worker Bee
Level 4

Node Degree = Number Of Children



Tree Degree = Max Node Degree



Degree of tree = 3.

Binary Tree

- Finite (possibly empty) collection of elements.
- A nonempty binary tree has a root element.
- The remaining elements (if any) are partitioned into two binary trees.
- These are called the left and right subtrees of the binary tree.

Differences Between A Tree & A Binary Tree

- The subtrees of a binary tree are ordered; those of a tree are not ordered.

- Are different when viewed as binary trees.
- Are the same when viewed as trees.

Arithmetic Expressions

- (a + b) * (c + d) + e – f/g*h + 3.25
- Expressions comprise three kinds of entities.
  - Operators (+, -, /, *).
  - Operands (a, b, c, d, e, f, g, h, 3.25, (a + b), (c + d), etc.).
  - Delimiters ((,)).

Operator Degree

- Number of operands that the operator requires.
- Binary operator requires two operands.
  - a + b
  - c / d
  - e - f
- Unary operator requires one operand.
  - + g
  - - h

Infix Form

- Normal way to write an expression.
- Binary operators come in between their left and right operands.

    a * b

    a + b * c

    a * b / c

    (a + b) * (c + d) + e – f/g*h + 3.25

Operator Priorities

- How do you figure out the operands of an operator?

  a + b * c

  a * b + c / d

- This is done by assigning operator priorities.

  priority(*) = priority(/) > priority(+) = priority(-)

- When an operand lies between two operators, the operand associates with the operator that has higher priority.

Tie Breaker

- When an operand lies between two operators that have the same priority, the operand associates with the operator on the left.

  $a + b - c$

  $a * b / c / d$

Delimiters

- Subexpression within delimiters is treated as a single operand, independent from the remainder of the expression.
  - (a + b) * (c – d) / (e – f)

Infix Expression Is Hard To Parse

- Need operator priorities, tie breaker, and delimiters.
- This makes computer evaluation more difficult than is necessary.
- Postfix and prefix expression forms do not rely on operator priorities, a tie breaker, or delimiters.
- So it is easier for a computer to evaluate expressions that are in these forms.

Postfix Form

- The postfix form of a variable or constant is the same as its infix form.
  - a, b, 3.25
- The relative order of operands is the same in infix and postfix forms.
- Operators come immediately after the postfix form of their operands.
  - Infix = a + b
  - Postfix = ab+

Postfix Examples

- Infix = a + b * c

  Postfix = a b c * +

- Infix = a * b + c

  Postfix = a b * c +

- Infix = (a + b) * (c – d) / (e + f)

  Postfix = a b + c d - * e f + /

Unary Operators

- Replace with new symbols.
  - + a => a @
  - + a + b => a @ b +
  - - a => a ?
  - - a-b => a ? b -

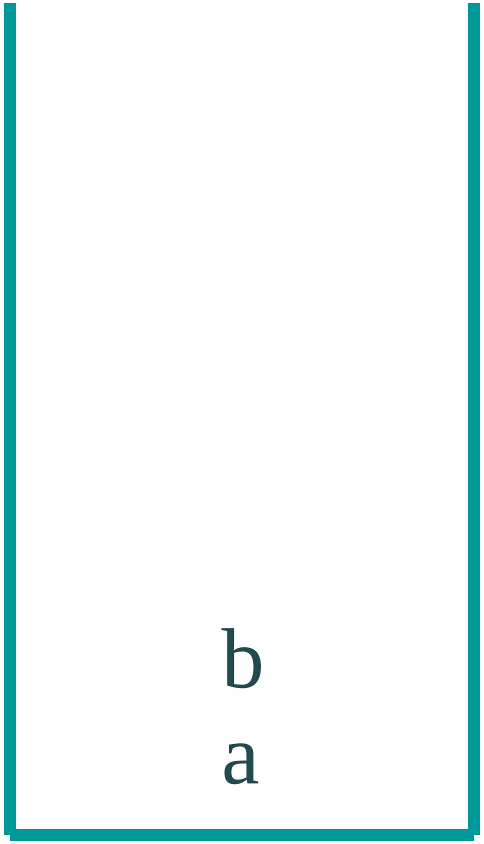
Postfix Evaluation

- Scan postfix expression from left to right pushing operands on to a stack.
- When an operator is encountered, pop as many operands as this operator needs; evaluate the operator; push the result on to the stack.
- This works because, in postfix, operators come immediately after their operands.

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /

    - a b + c d - * e f + /
    - a b + c d - * e f + /



b

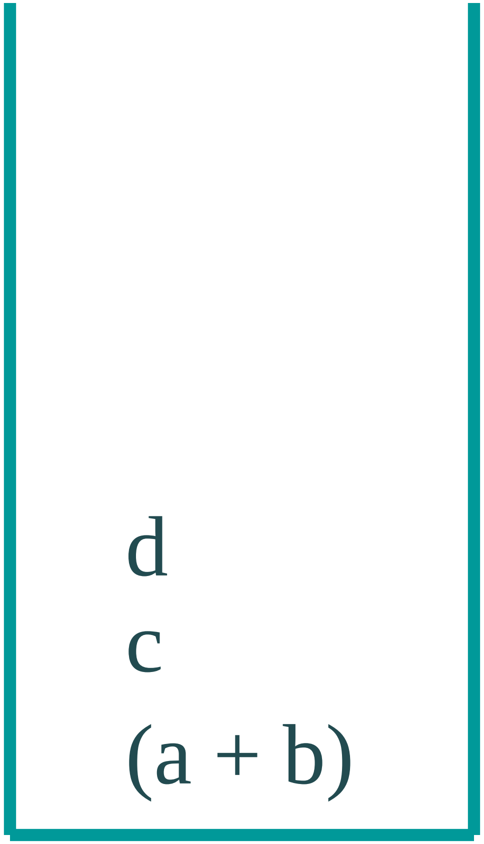
a

stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /

- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /



d

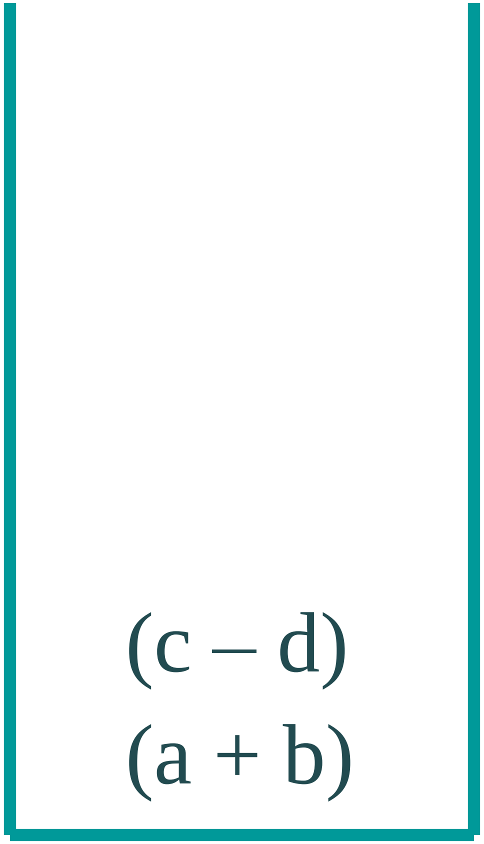
c

(a + b)

stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
  - a b + c d - * e f + /



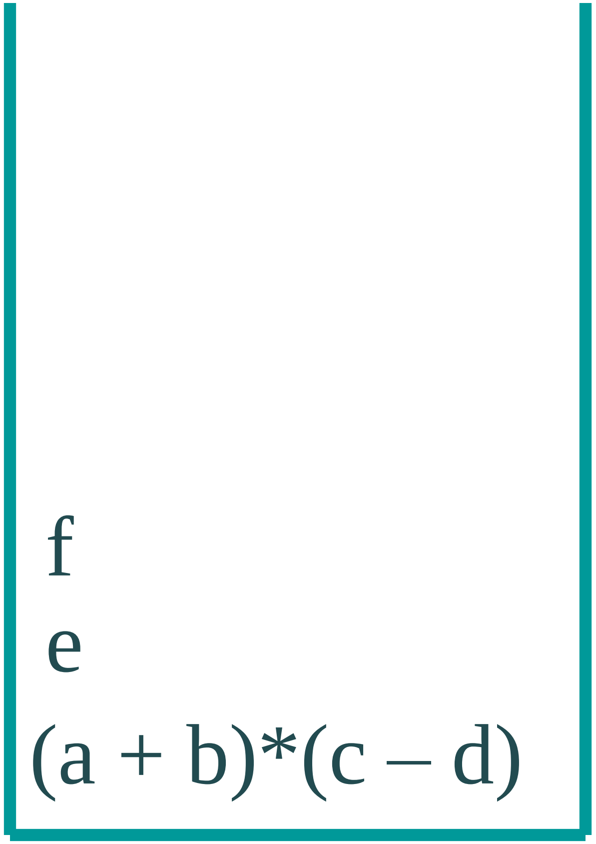
(c – d)

(a + b)

stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
  - a b + c d - * e f + /
  - a b + c d - * e f + /
  - a b + c d - * e f + /
  - a b + c d - * e f + /

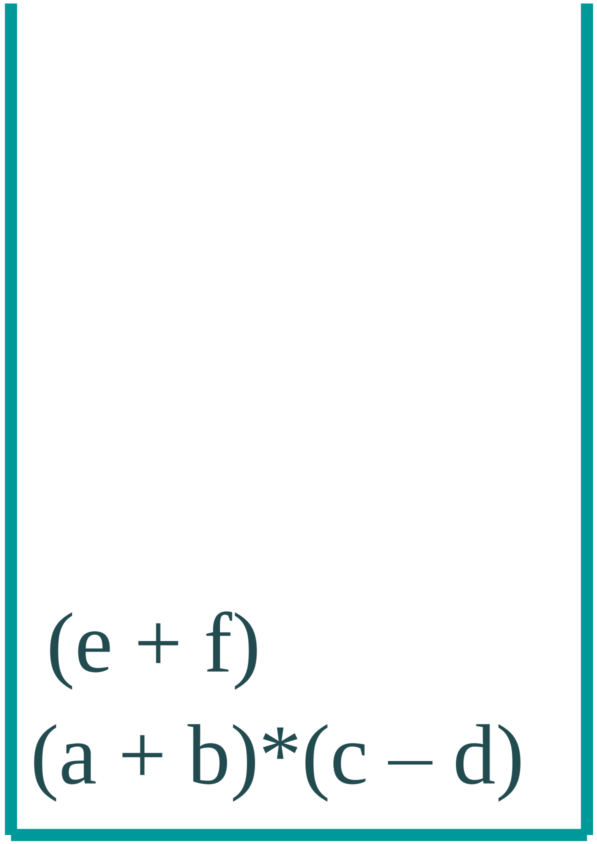


stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /

  - a b + c d - * e f + /
  - a b + c d - * e f + /
  - a b + c d - * e f + /
  - a b + c d - * e f + /
  - a b + c d - * e f + /



(e + f)

(a + b)*(c – d)

stack

Prefix Form

- The prefix form of a variable or constant is the same as its infix form.

  a, b, 3.25

- The relative order of operands is the same in infix and prefix forms.
- Operators come immediately before the prefix form of their operands.

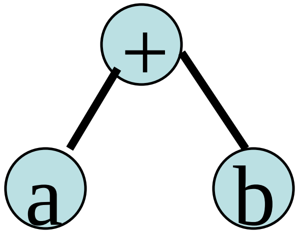
  Infix = a + b

  Postfix = ab+

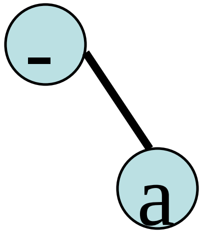
  Prefix = +ab

Binary Tree Form

- a + b

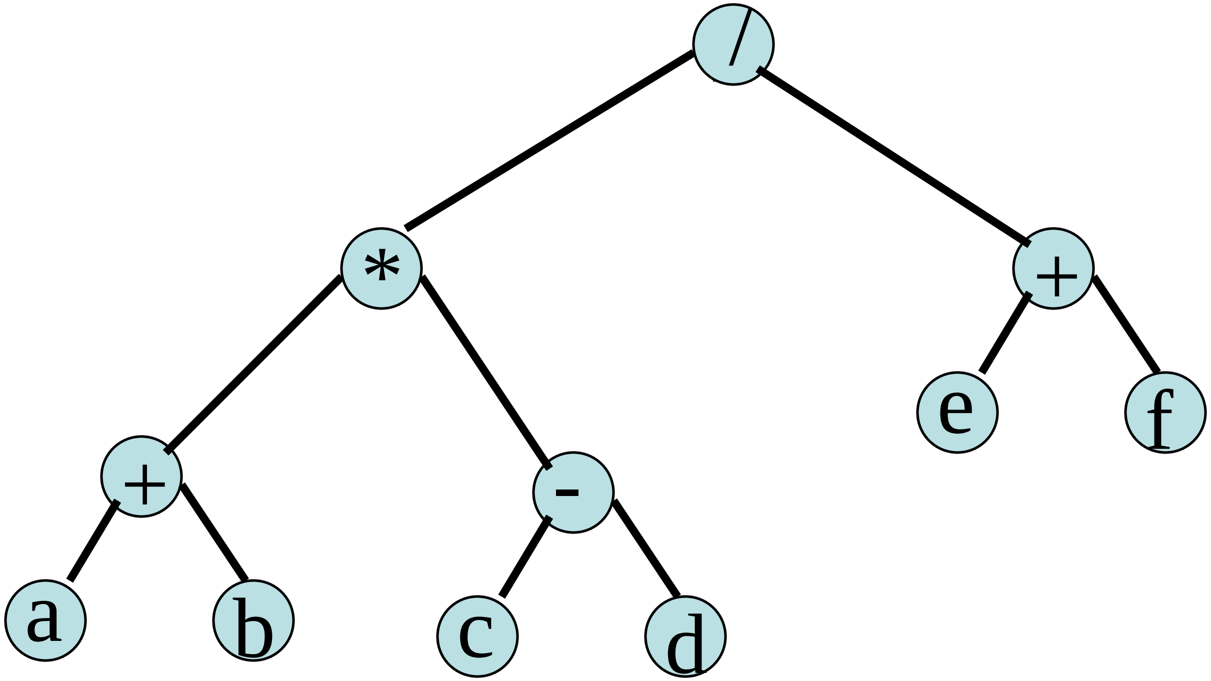


- - a



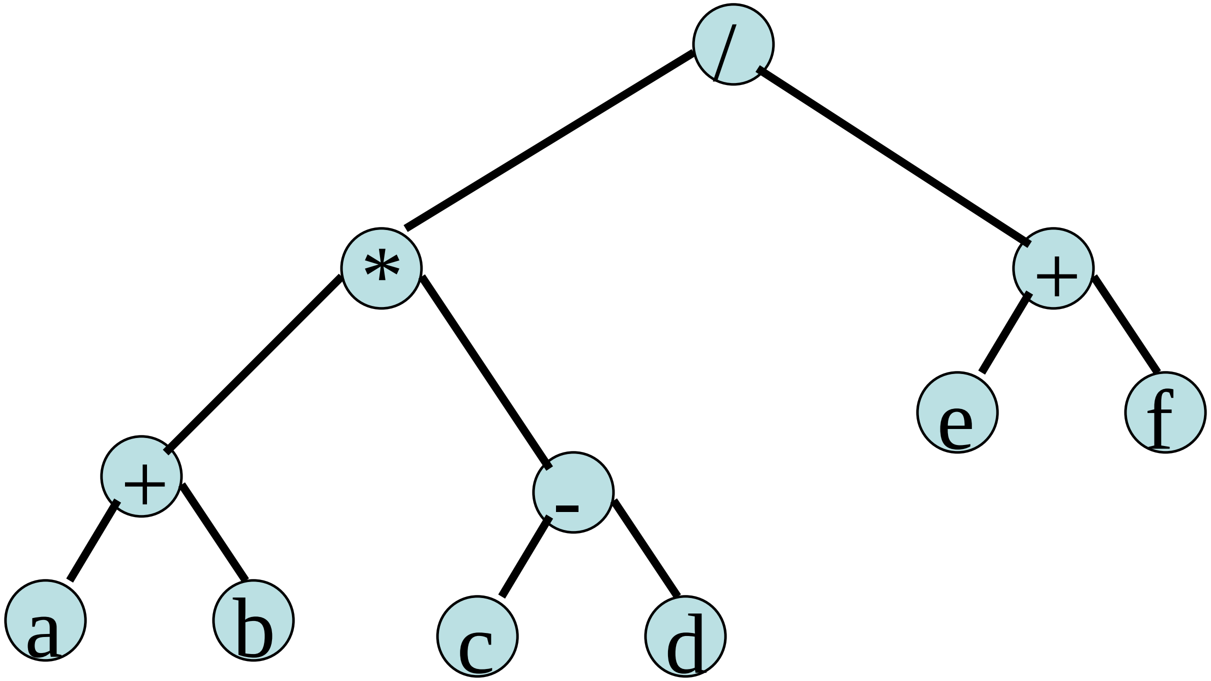
Binary Tree Form

- (a + b) * (c – d) / (e + f)

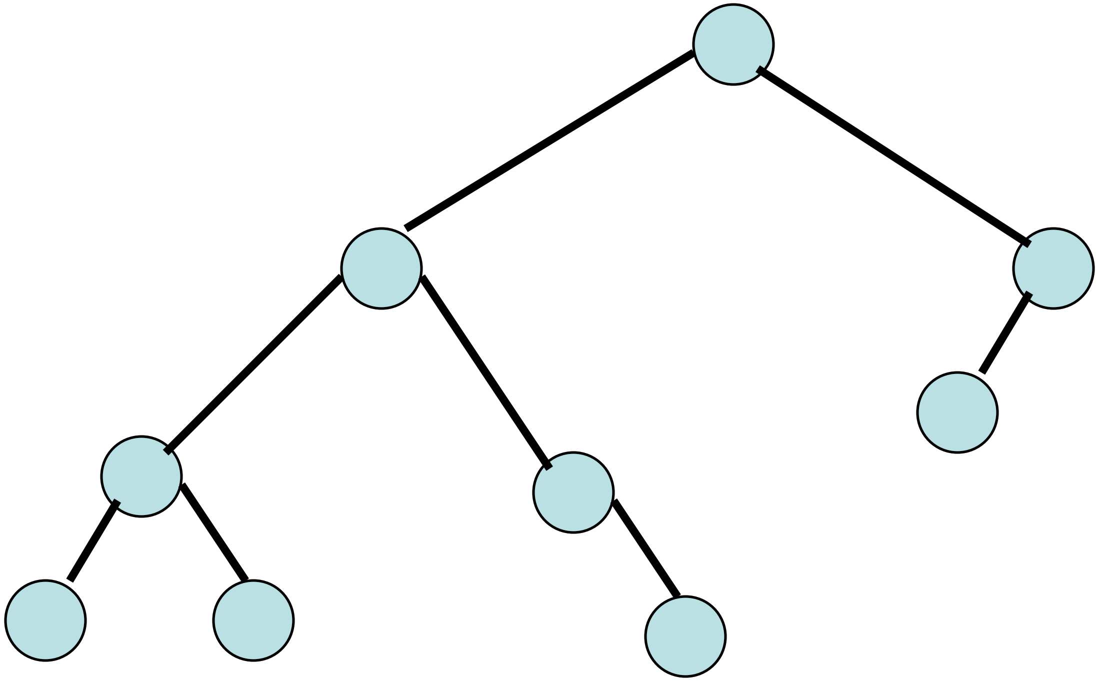


Merits Of Binary Tree Form

- Left and right operands are easy to visualize.
- Code optimization algorithms work with the binary tree form of an expression.
- Simple recursive evaluation of expression.

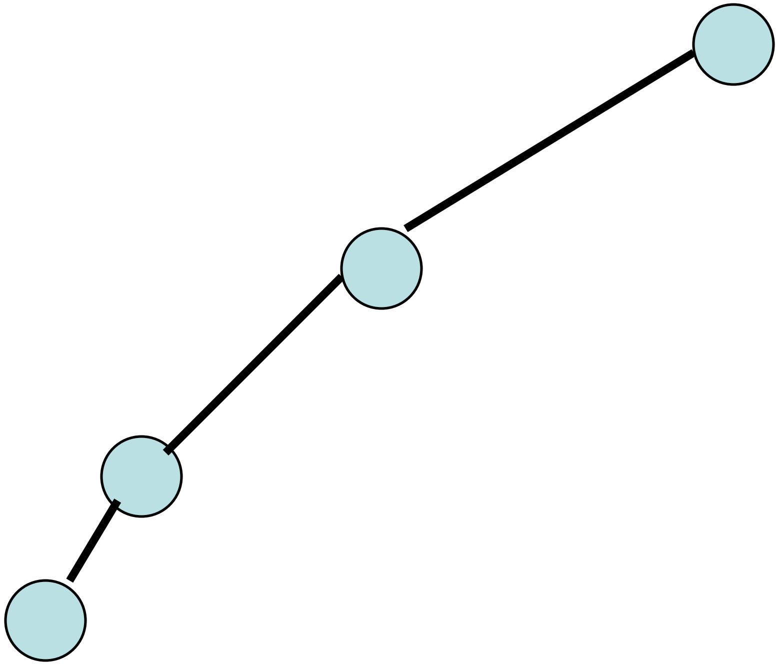


Binary Tree Properties & Representation



Minimum Number Of Nodes

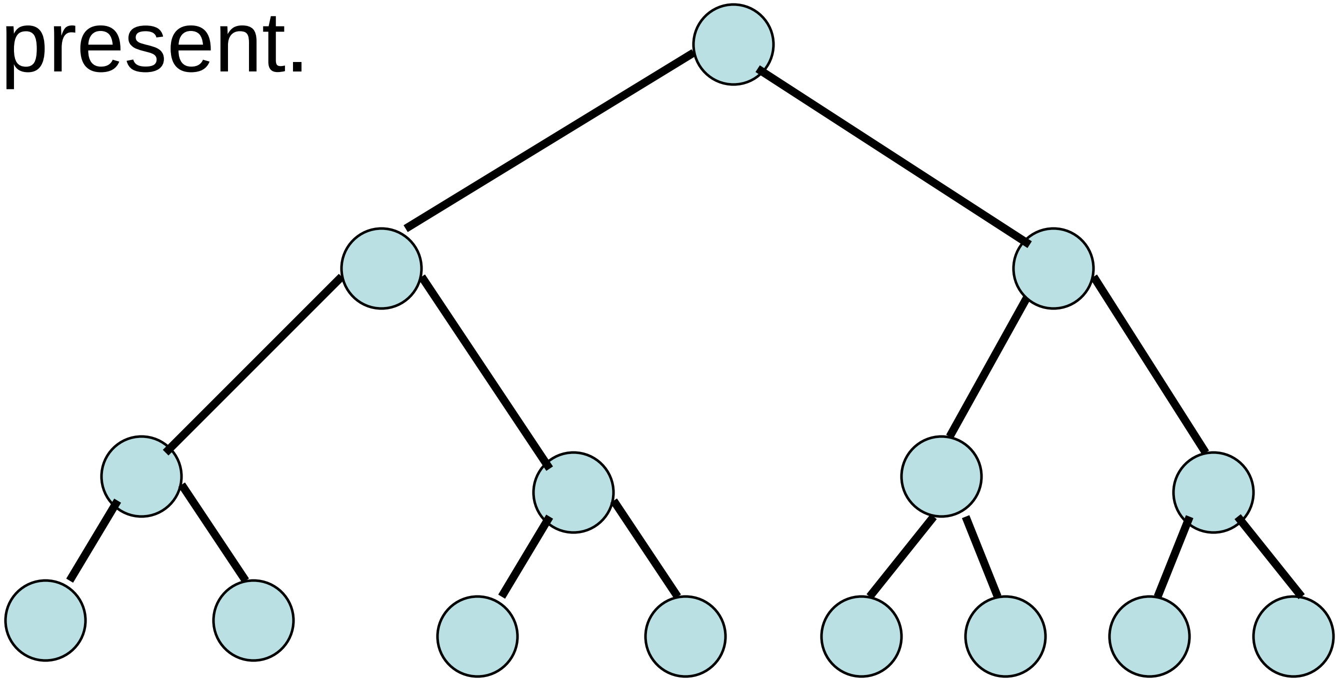
- Minimum number of nodes in a binary tree whose height is h.
- At least one node at each of first h levels.



minimum number of nodes is h

Maximum Number Of Nodes

- All possible nodes at first h levels are present.



Maximum number of nodes

$= 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$

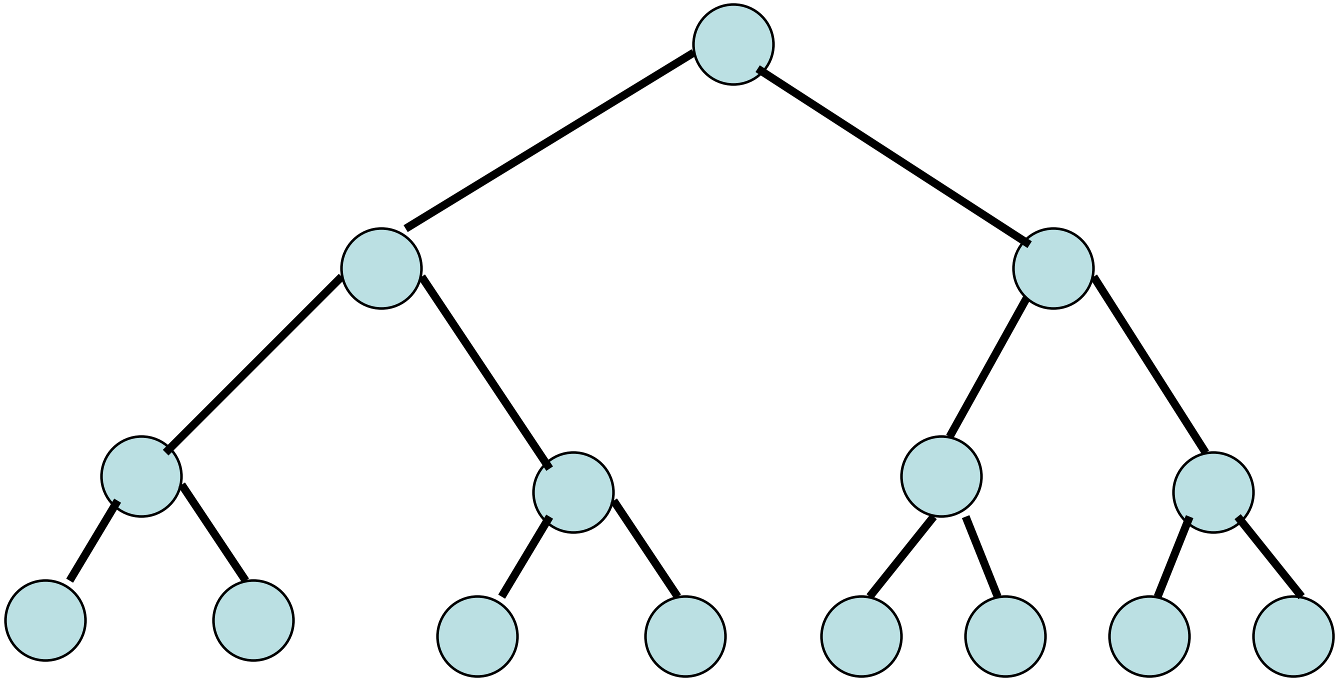
$= 2^h - 1$

Number Of Nodes & Height

- Let n be the number of nodes in a binary tree whose height is h.
- $h <= n <= 2^h – 1$
- $\log_2(n+1) <= h <= n$

Full Binary Tree

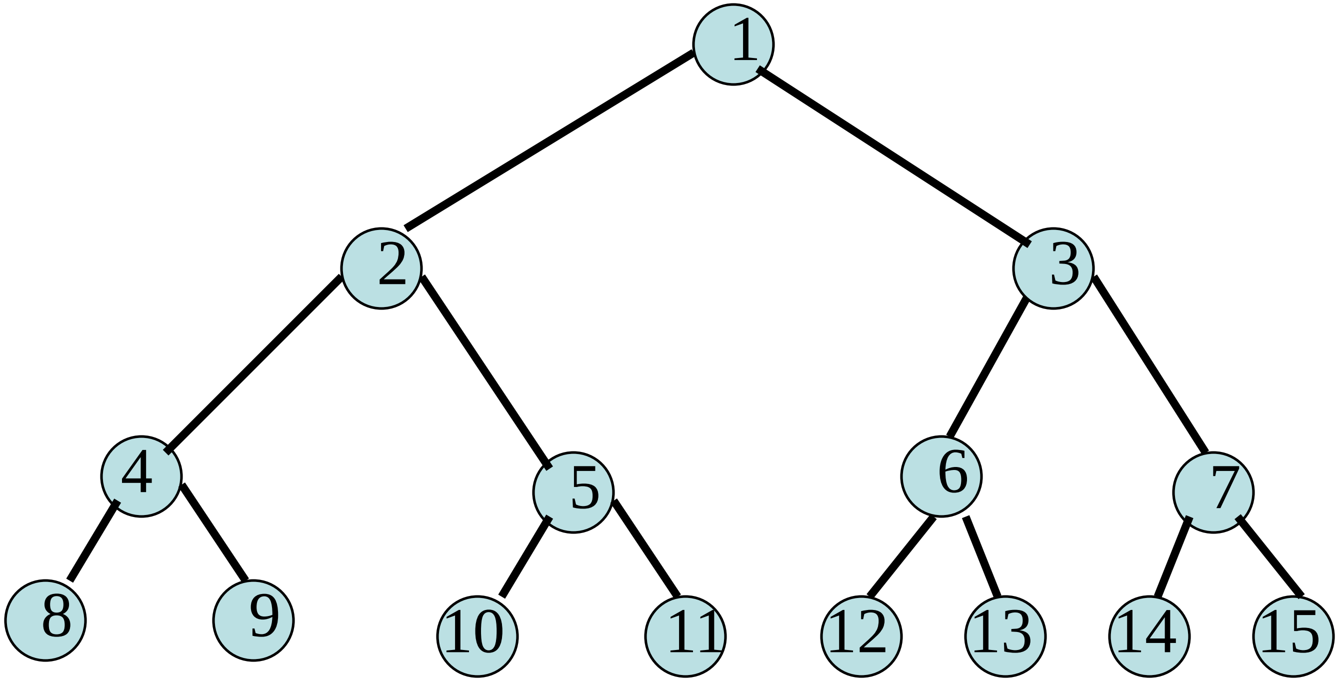
- A full binary tree of a given height h has $2^h - 1$ nodes.



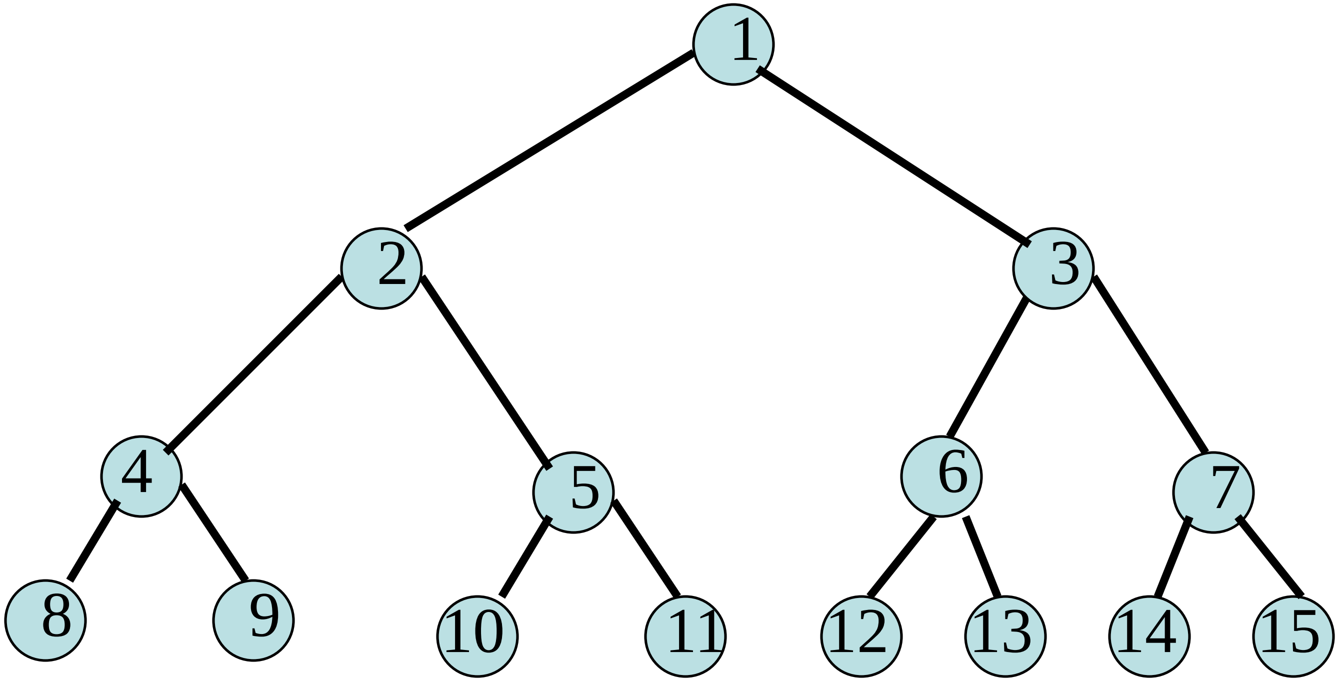
Height 4 full binary tree.

Numbering Nodes In A Full Binary Tree

- Number the nodes 1 through $2^h - 1$.
- Number by levels from top to bottom.
- Within a level number from left to right.

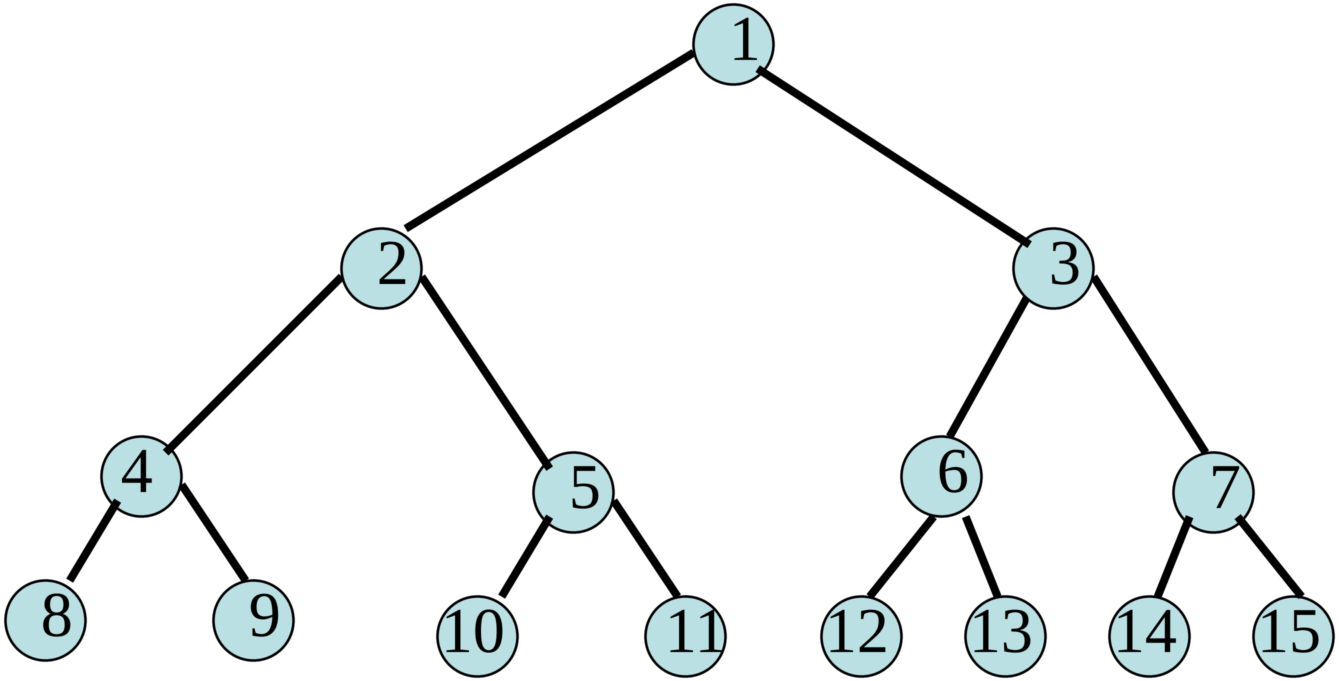


Node Number Properties



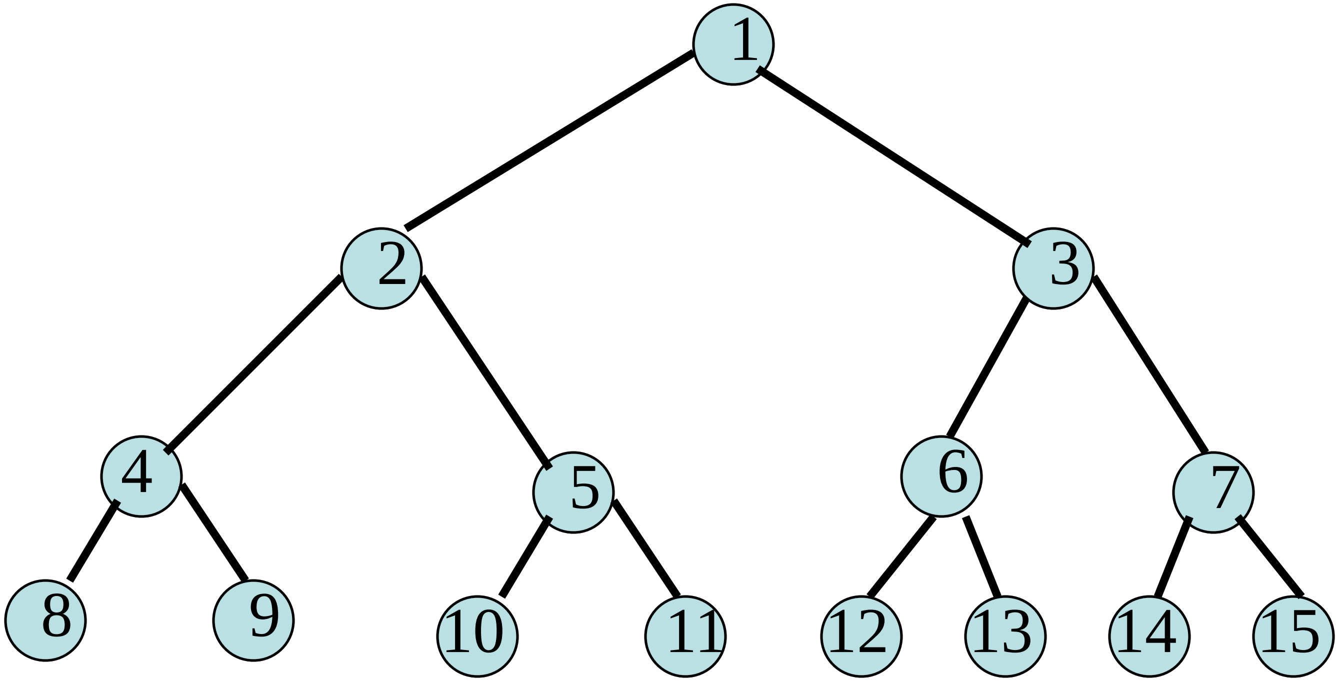
- Parent of node i is node i / 2, unless i = 1.
- Node 1 is the root and has no parent.

Node Number Properties



- Left child of node i is node 2i, unless 2i > n, where n is the number of nodes.
- If 2i > n, node i has no left child.

Node Number Properties

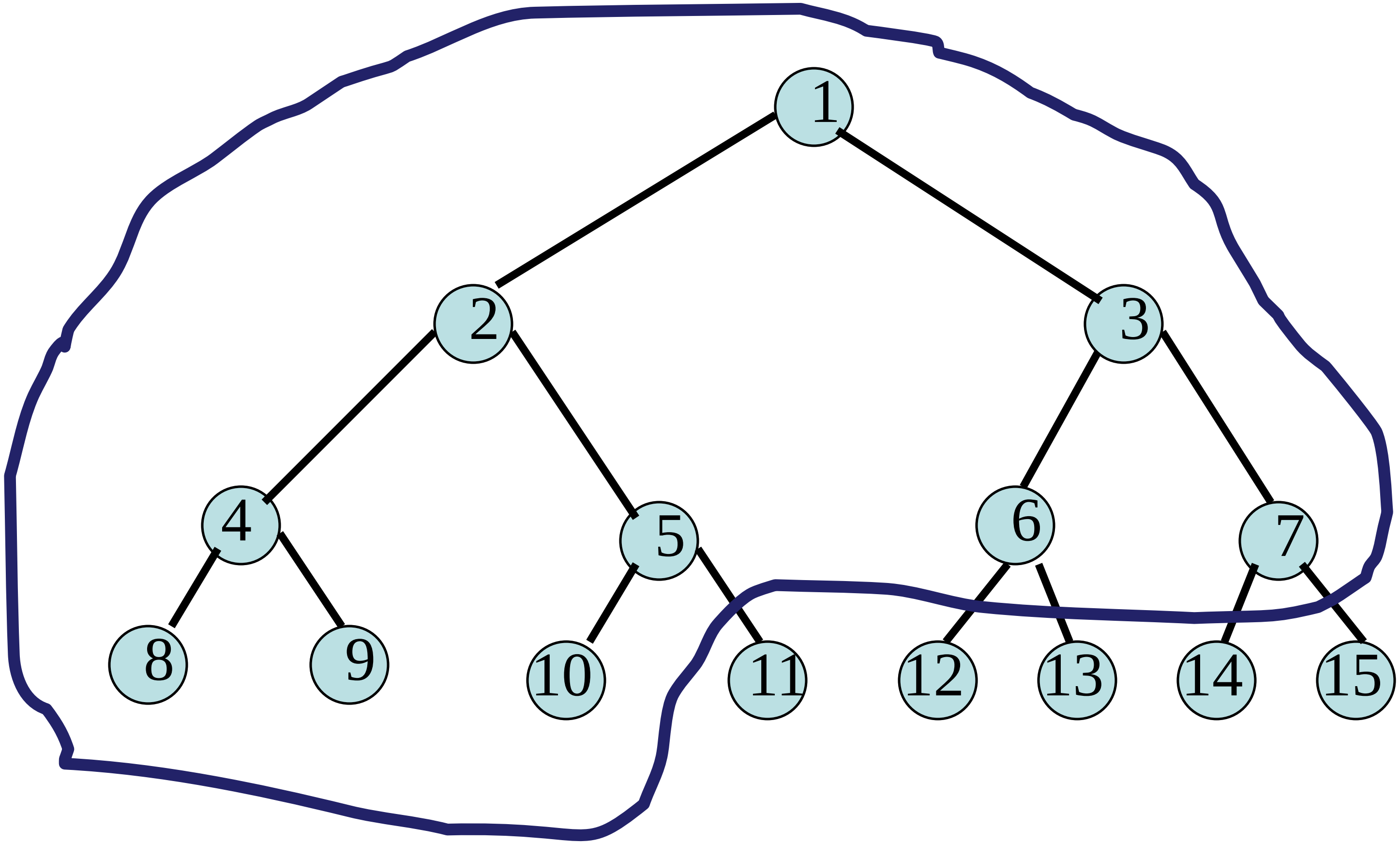


- Right child of node i is node 2i+1, unless 2i+1 > n, where n is the number of nodes.
- If 2i+1 > n, node i has no right child.

Complete Binary Tree With n Nodes

- Start with a full binary tree that has at least n nodes.
- Number the nodes as described earlier.
- The binary tree defined by the nodes numbered 1 through n is the unique n node complete binary tree.

Example



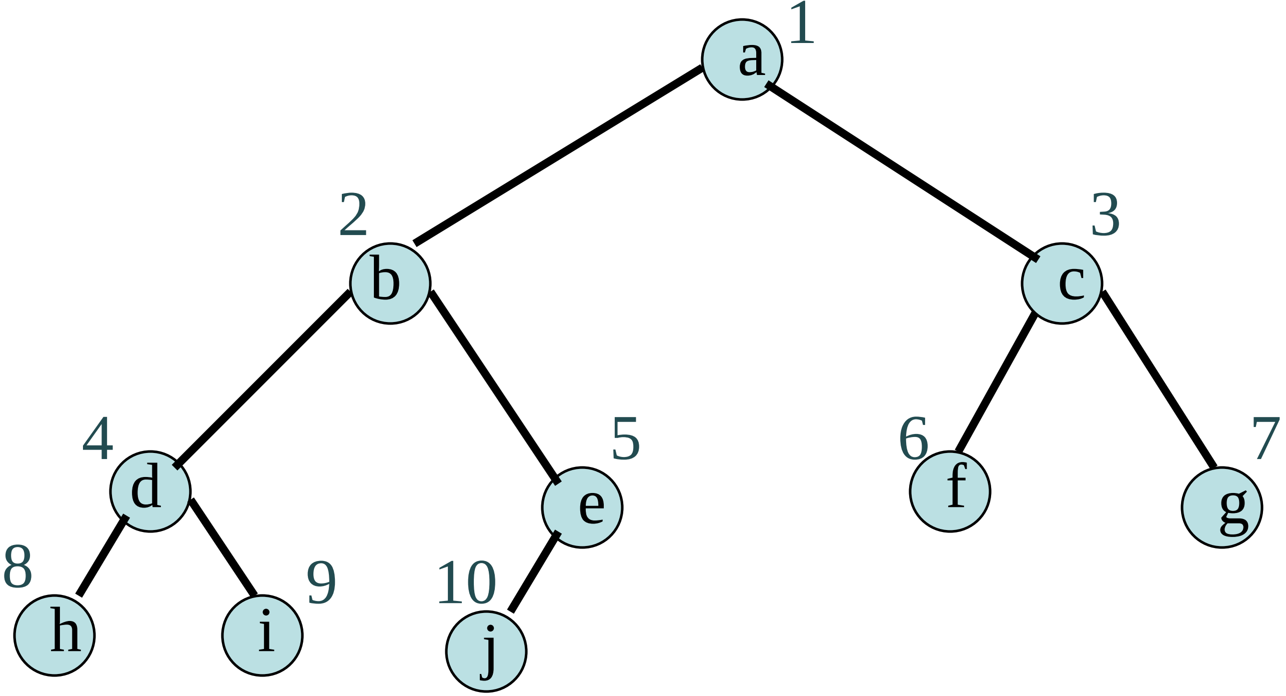
- Complete binary tree with 10 nodes.

Binary Tree Representation

- Array representation.
- Linked representation.

Array Representation

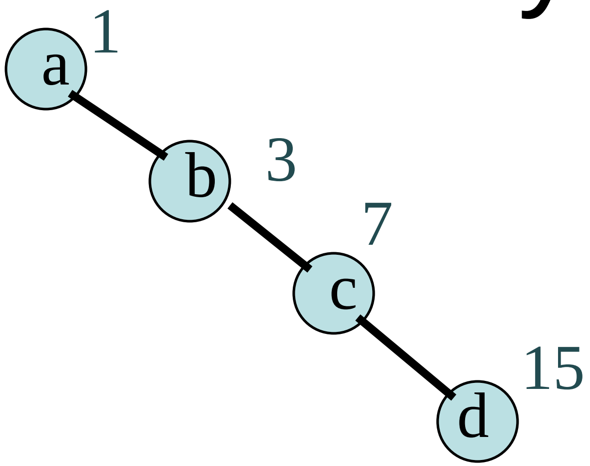
- Number the nodes using the numbering scheme for a full binary tree. The node that is numbered i is stored in tree[i].



tree[]

	a	b	c	d	e	f	g	h	i	j
0					5					10

Right-Skewed Binary Tree



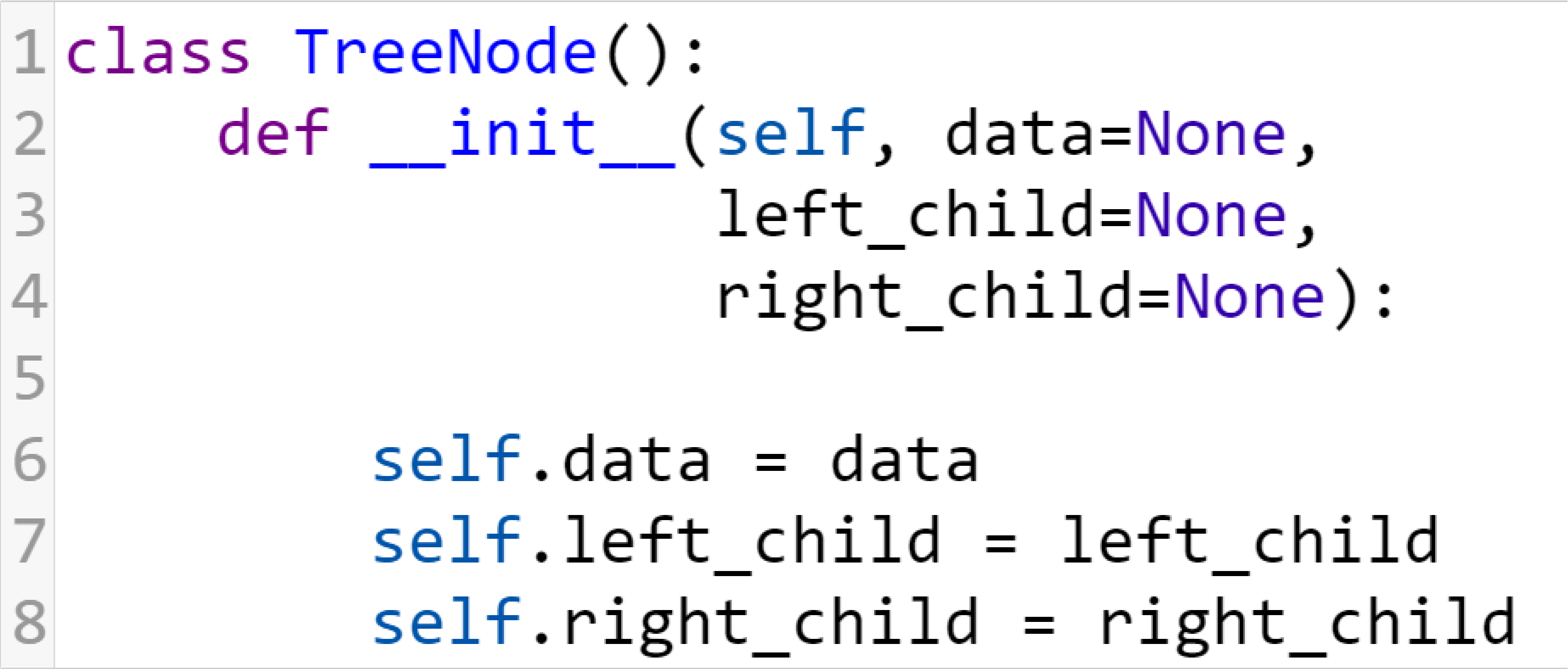
tree[]		a	-	b	-	-	-	c	-	-	-	-	-	-	-	d
	0					5					10					15

- An n node binary tree needs an array whose length is between n+1 and 2^n.

Linked Representation

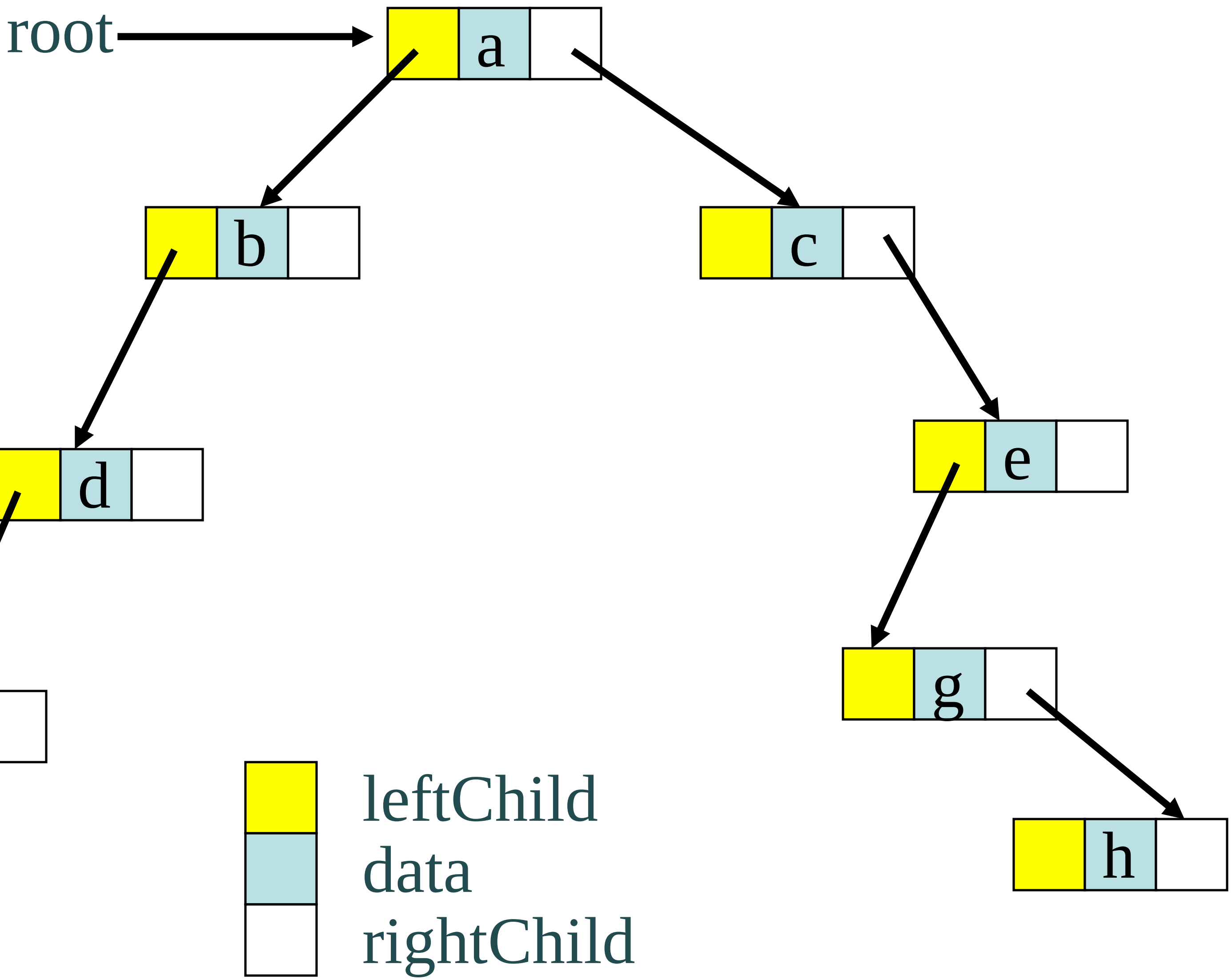
- Each binary tree node is represented as an object whose data type is TreeNode.
- The space required by an n node binary tree is n * (space required by one node).

The Struct binaryTreeNode

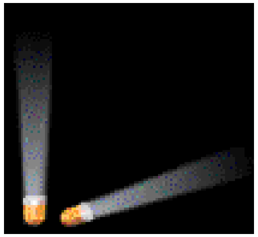


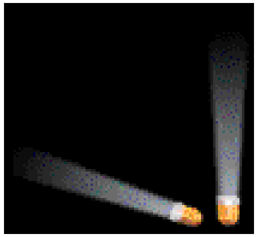
```python
class TreeNode():
    def __init__(self, data=None,
                 left_child=None,
                 right_child=None):

        self.data = data
        self.left_child = left_child
        self.right_child = right_child
```

Linked Representation Example



Binary Tree Traversal Methods



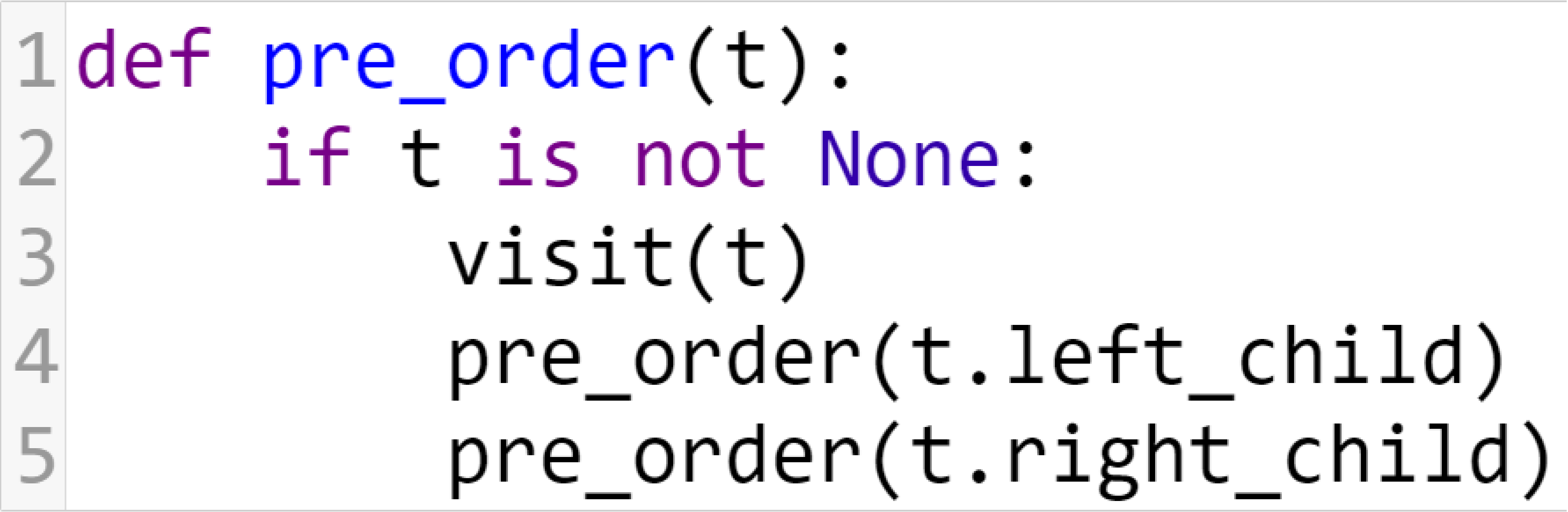


- Many binary tree operations are done by performing a traversal of the binary tree.
- In a traversal of a binary tree, each element of the binary tree is visited exactly once.
- During the visit of an element, all action (make a clone, display, evaluate the operator, etc.) with respect to this element is taken.

Binary Tree Traversal Methods

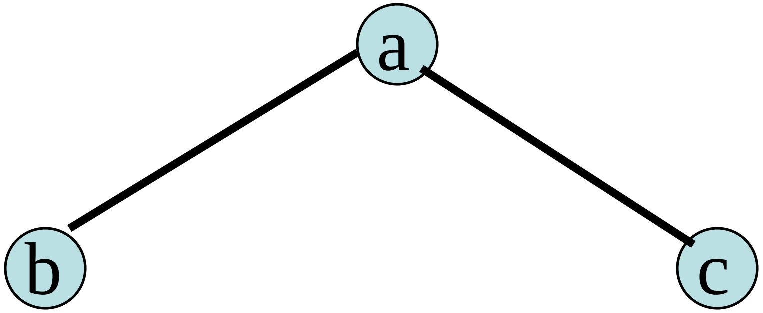
- Preorder
- Inorder
- Postorder
- Level order

Preorder Traversal



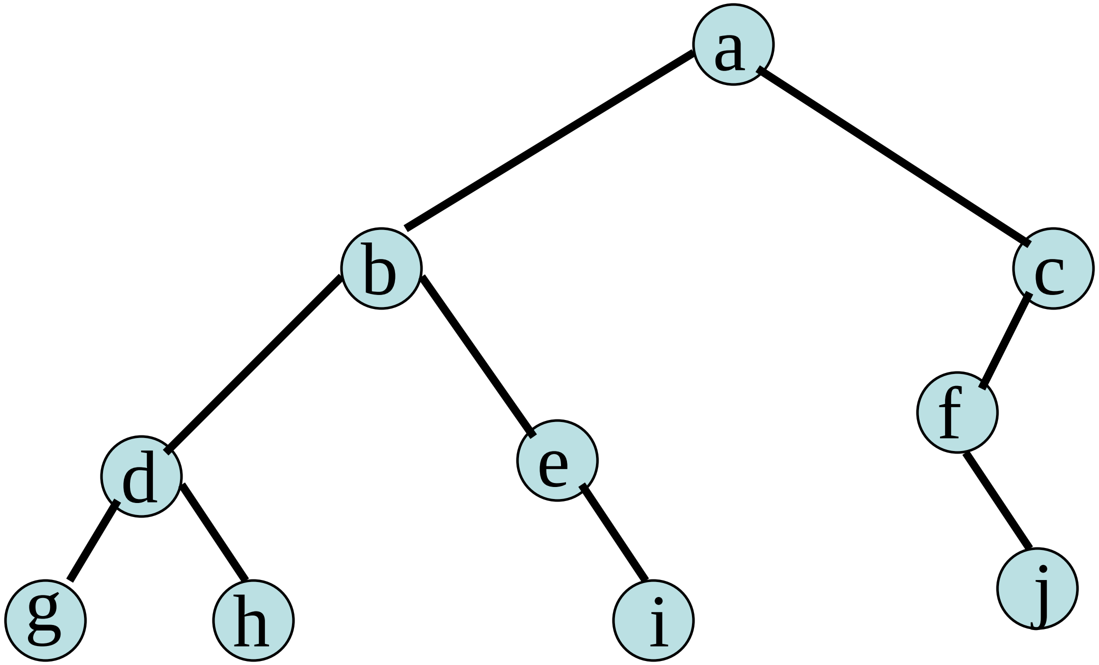
```
def pre_order(t):
    if t is not None:
        visit(t)
        pre_order(t.left_child)
        pre_order(t.right_child)
```

Preorder Example (Visit = print)



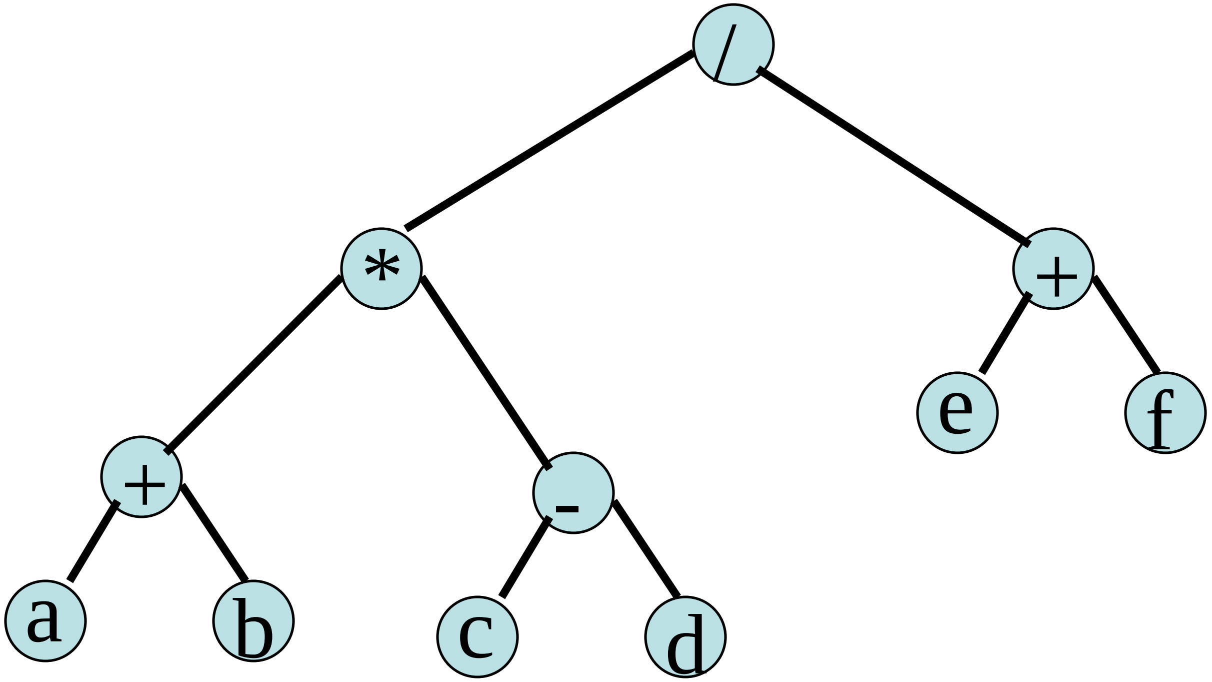
a b c

Preorder Example (Visit = print)



a b d g h e i c f j

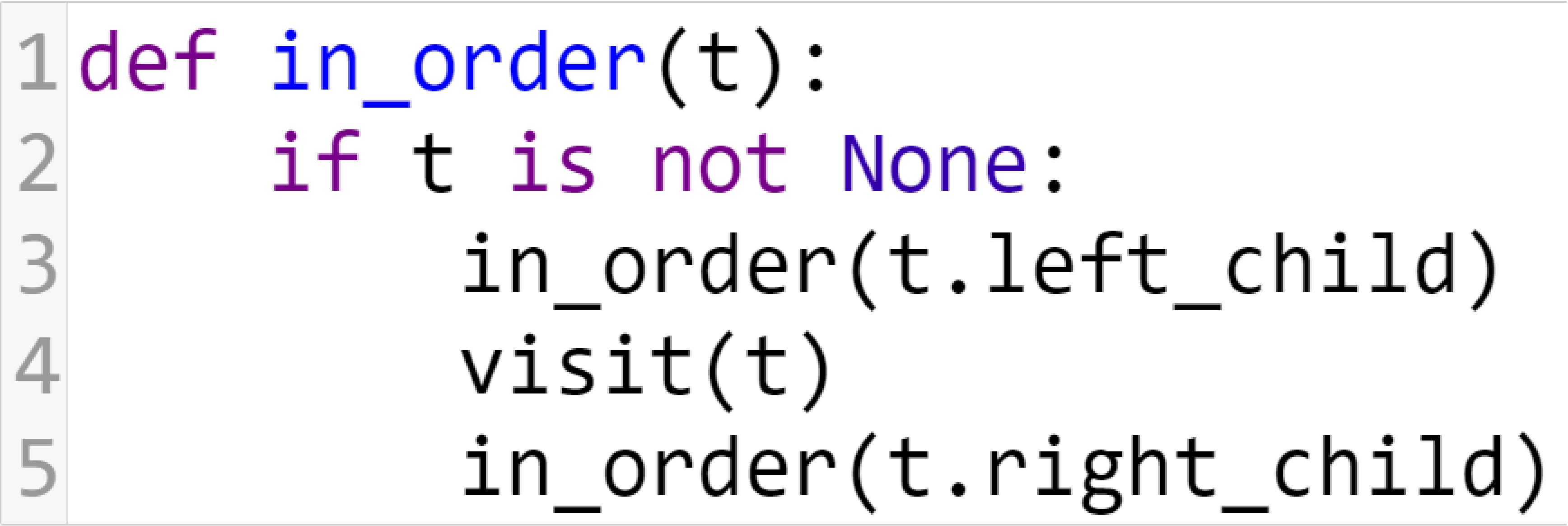
Preorder Of Expression Tree



/ * + a b - c d + e f

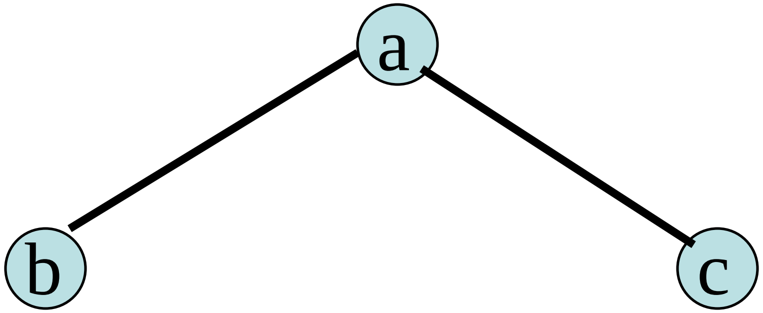
Gives prefix form of expression!

Inorder Traversal



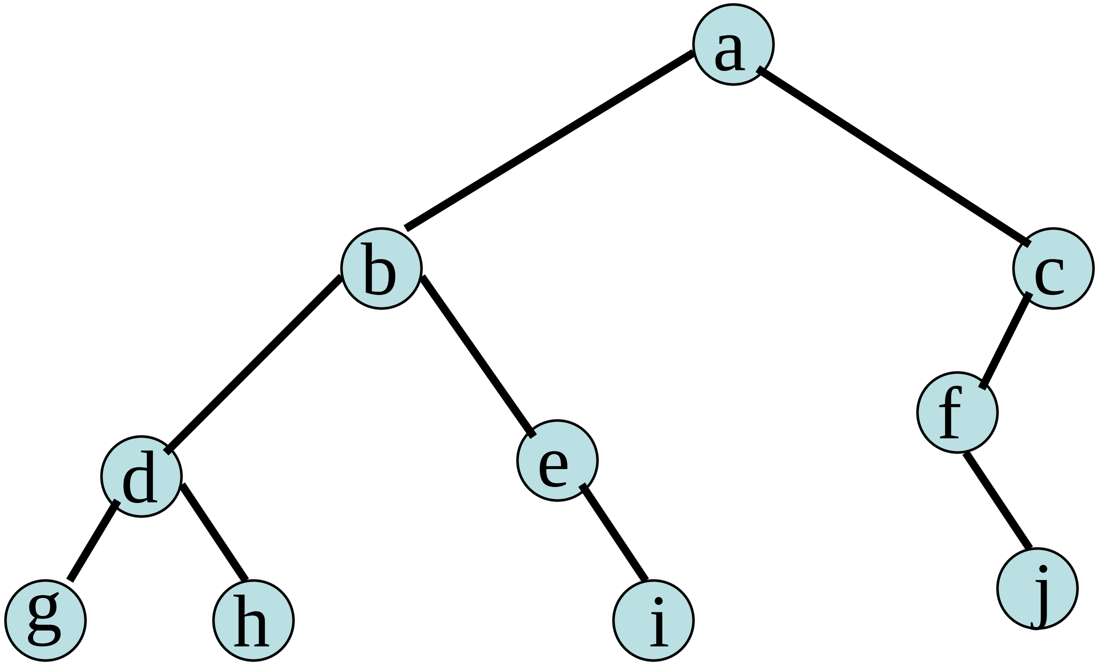
```python
def in_order(t):
    if t is not None:
        in_order(t.left_child)
        visit(t)
        in_order(t.right_child)
```

Inorder Example (Visit = print)



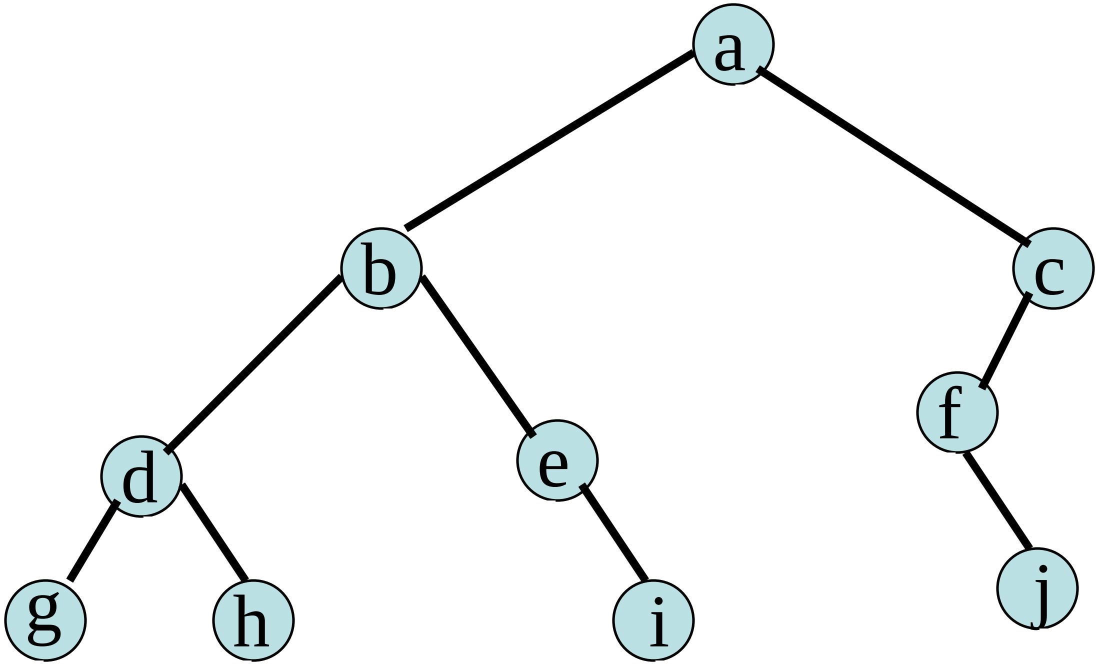
b a c

Inorder Example (Visit = print)



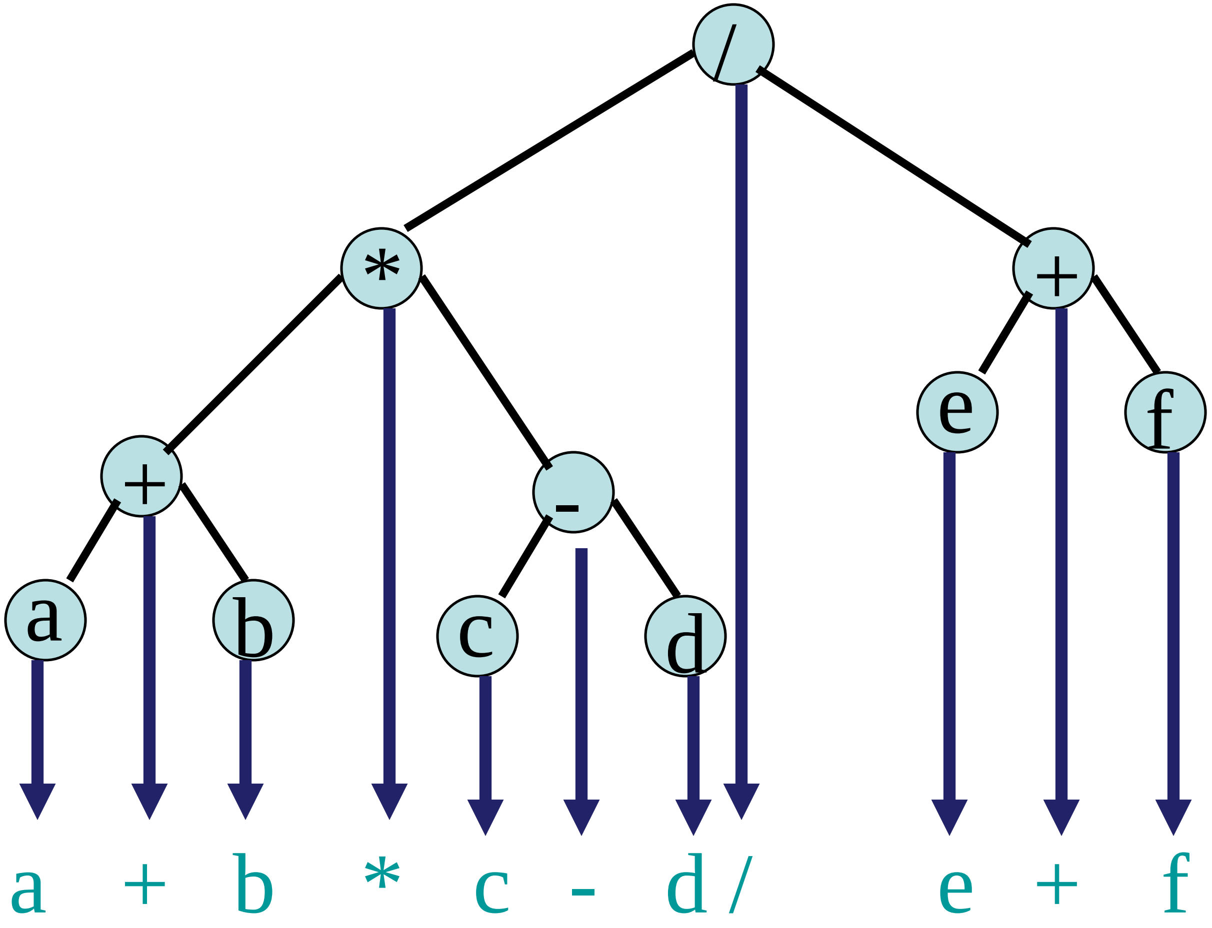
g d h b e i a f j c

Inorder By Projection (Squishing)



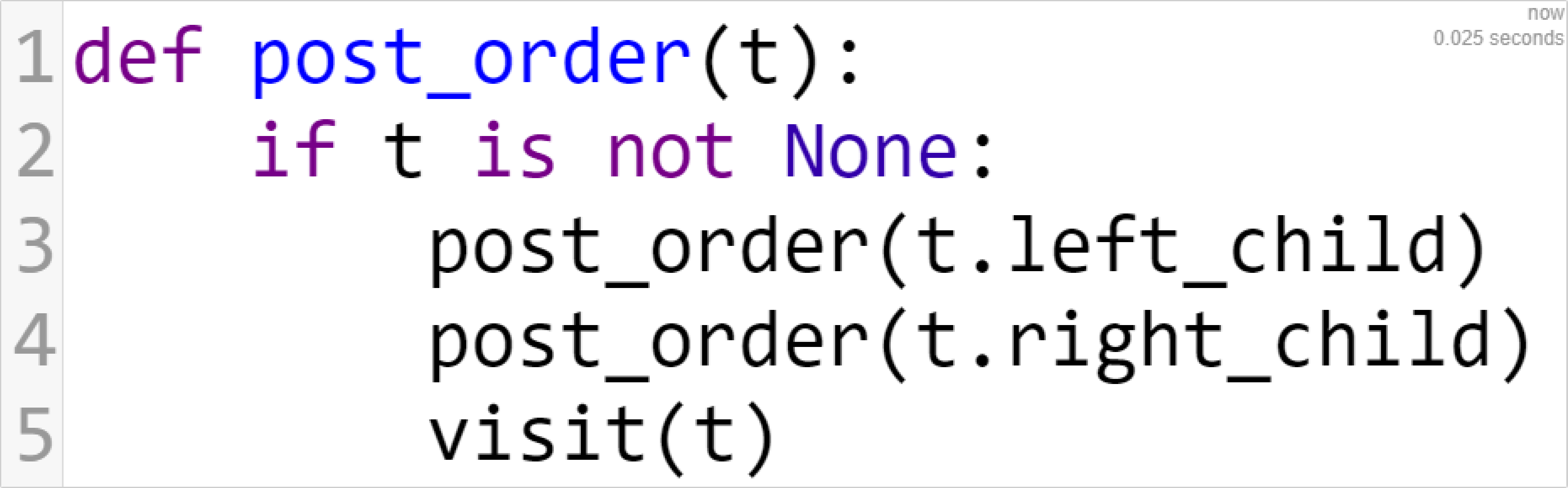
g d h b e i a f j c

Inorder Of Expression Tree



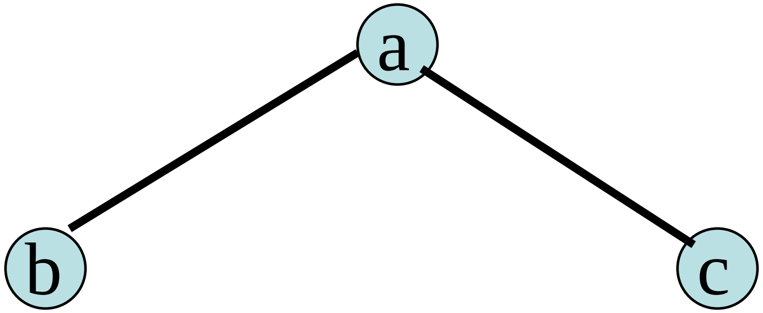
Gives infix form of expression (sans parentheses)!

Postorder Traversal



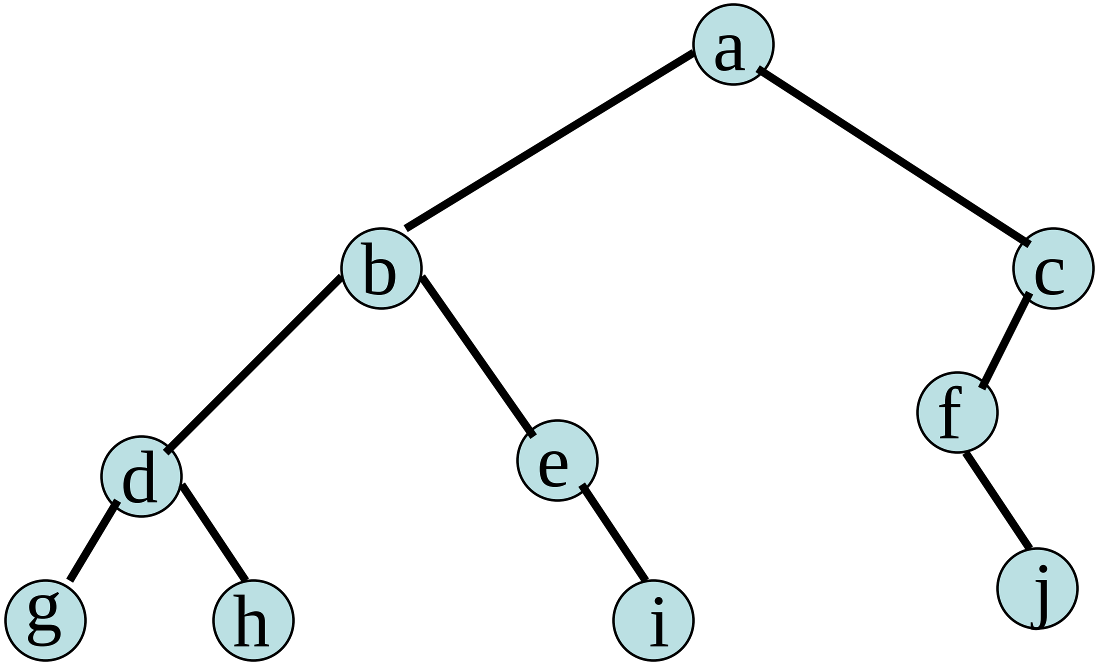
```python
def post_order(t):
    if t is not None:
        post_order(t.left_child)
        post_order(t.right_child)
        visit(t)
```

Postorder Example (Visit = print)



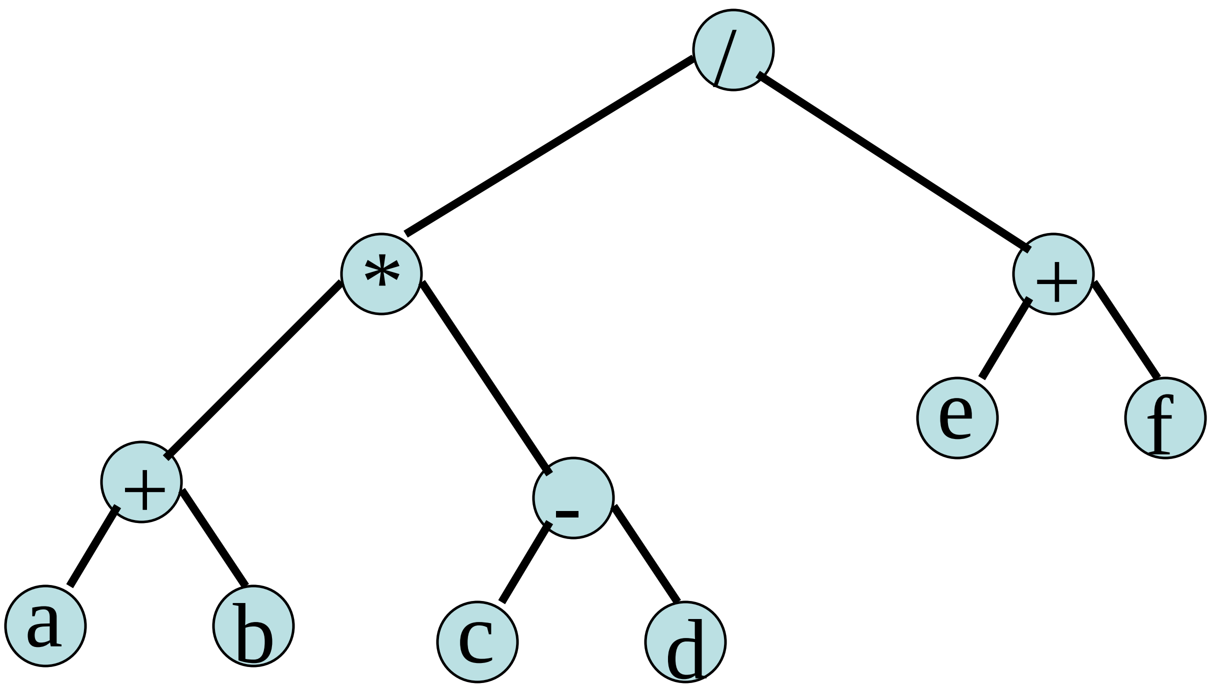
b c a

Postorder Example (Visit = print)



g h d i e b j f c a

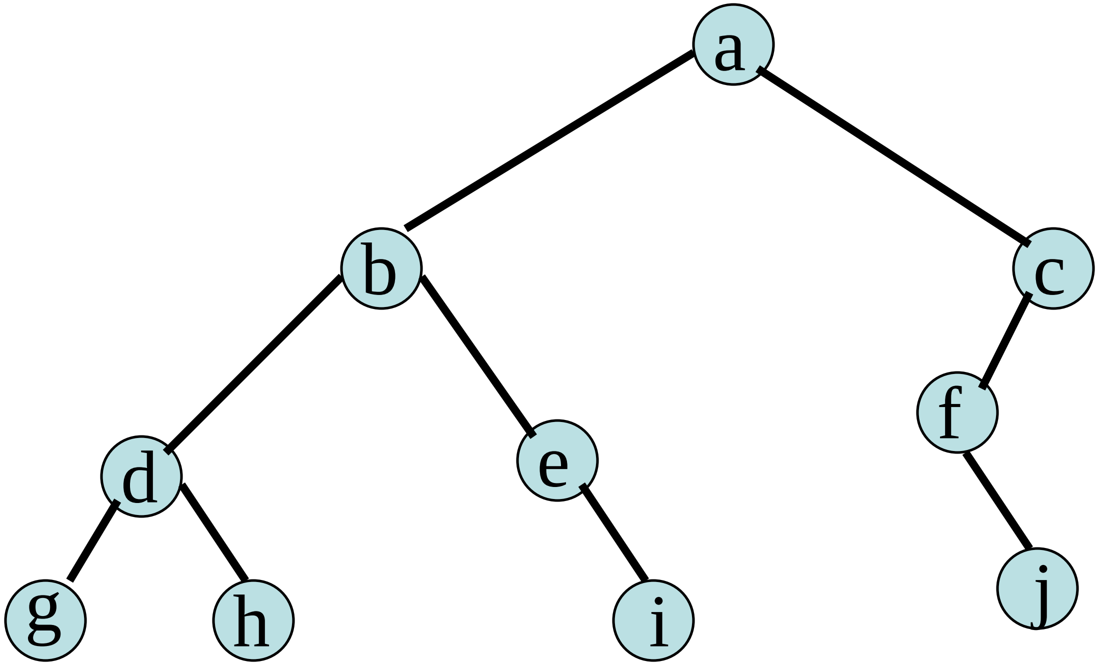
Postorder Of Expression Tree



a b + c d - * e f + /

Gives postfix form of expression!

Traversal Applications

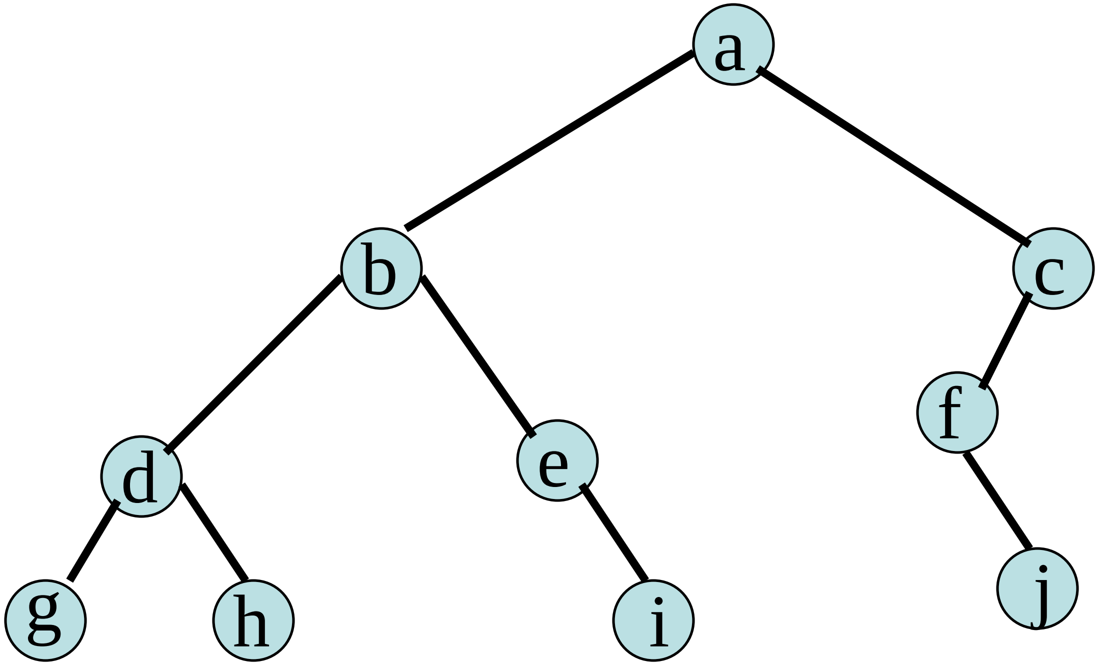


- Make a clone.
- Determine height.
- Determine number of nodes.

Level Order

```
Let t be the tree root.
        (t is not None)
{
    visit t and put its children on a FIFO queue;
    if FIFO queue is empty, set t = None;
    otherwise, pop a node from the FIFO queue and call it t;
}
```

Level-Order Example (Visit = print)



a b c d e f g h i j

Binary Tree Construction

- Suppose that the elements in a binary tree are distinct.
- Can you construct the binary tree from which a given traversal sequence came?
- When a traversal sequence has more than one element, the binary tree is not uniquely defined.
- Therefore, the tree from which the sequence was obtained cannot be reconstructed uniquely.

Some Examples

preorder = ab

inorder = ab

postorder = ab

level order = ab

Binary Tree Construction

- Can you construct the binary tree, given two traversal sequences?
- Depends on which two sequences are given.

Preorder And Postorder

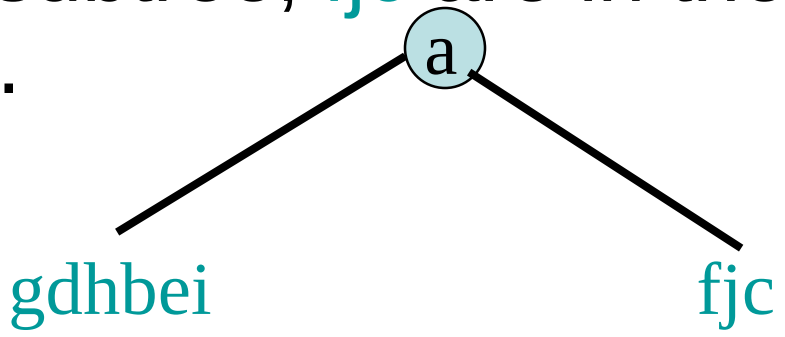
preorder = ab

postorder = ba

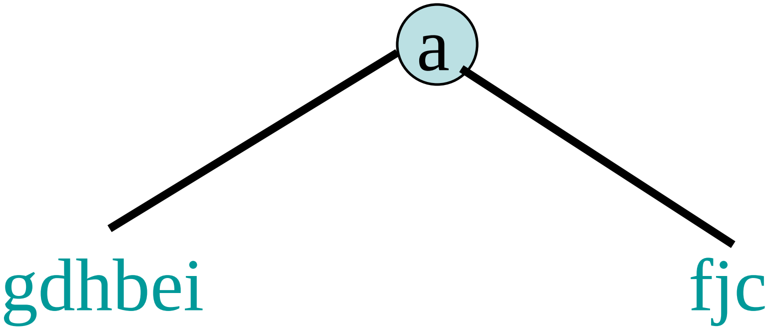
- Preorder and postorder do not uniquely define a binary tree.
- Nor do preorder and level order (same example).
- Nor do postorder and level order (same example).

Inorder And Preorder

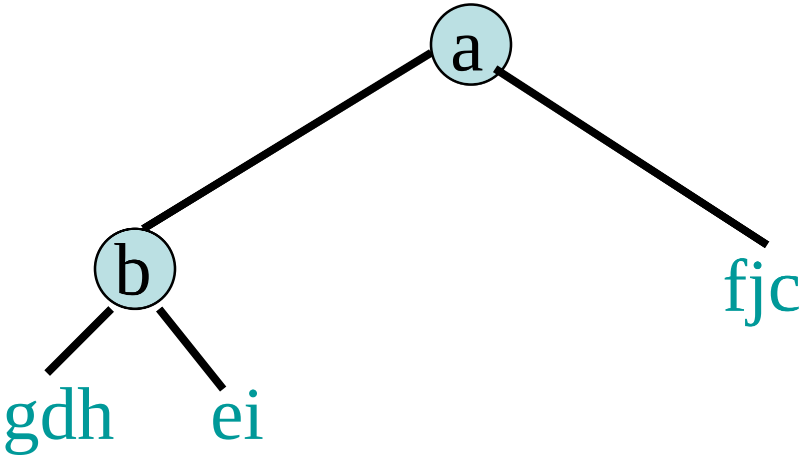
- inorder = g d h b e i a f j c
- preorder = a b d g h e i c f j
- Scan the preorder left to right using the inorder to separate left and right subtrees.
- a is the root of the tree; gdhbei are in the left subtree; fjc are in the right subtree.



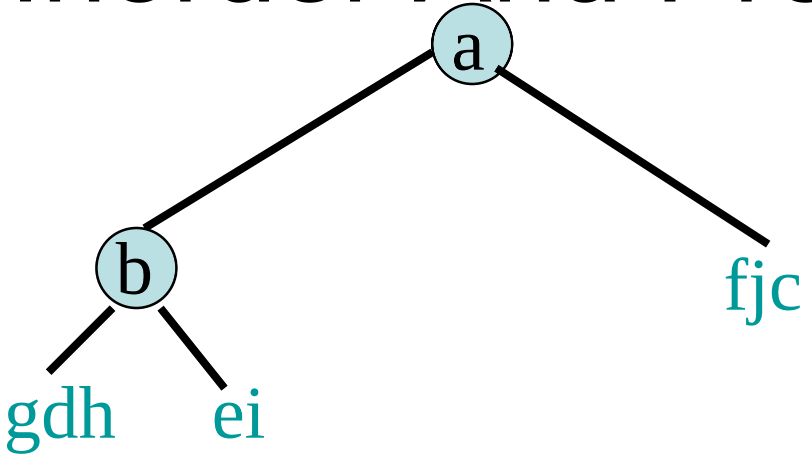
Inorder And Preorder



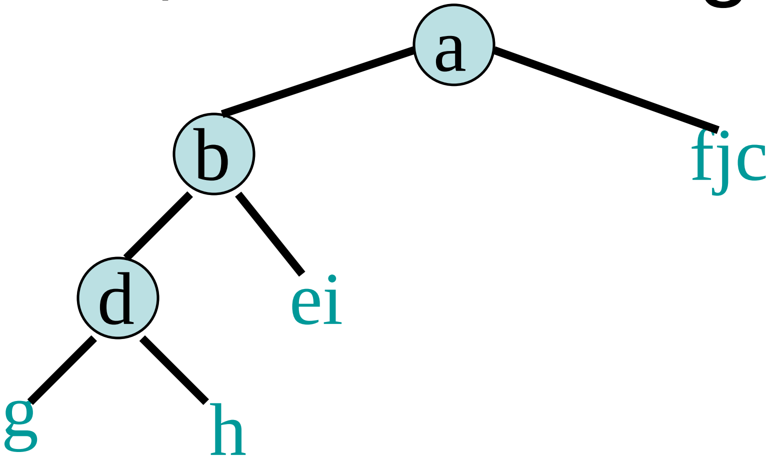
- preorder = a b d g h e i c f j
- b is the next root; gdh are in the left subtree; ei are in the right subtree.



Inorder And Preorder



- preorder = a b d g h e i c f j
- d is the next root; g is in the left subtree; h is in the right subtree.



Inorder And Postorder

- Scan postorder from right to left using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- postorder = g h d i e b j f c a
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

Inorder And Level Order

- Scan level order from left to right using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- level order = a b c d e f g h i j
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

Agenda

- What is Priority Queue
  - Min Priority Queue
  - Max Priority Queue
- What can Priority Queue do?
  - Sorting
  - Machine Schedule
- Heap Tree
- Leftist Tree
  - Extended binary tree
- Binary Search Tree
- Selection Tree

Priority Queues

Two kinds of priority queues:

- Min priority queue.
- Max priority queue.

Min Priority Queue

- Collection of elements.
- Each element has a priority or key.
- Supports following operations:
  - empty
  - size
  - insert an element into the priority queue (push)
  - get element with min priority (top)
  - remove element with min priority (pop)

Max Priority Queue

- Collection of elements.
- Each element has a priority or key.
- Supports following operations:
  - empty
  - size
  - insert an element into the priority queue (push)
  - get element with max priority (top)
  - remove element with max priority (pop)

Complexity Of Operations

Use a heap or a leftist tree (both are defined later).

empty, size, and top => O(1) time

insert (push) and remove (pop) => O(log n) time where n is the size of the priority queue

Applications

Sorting

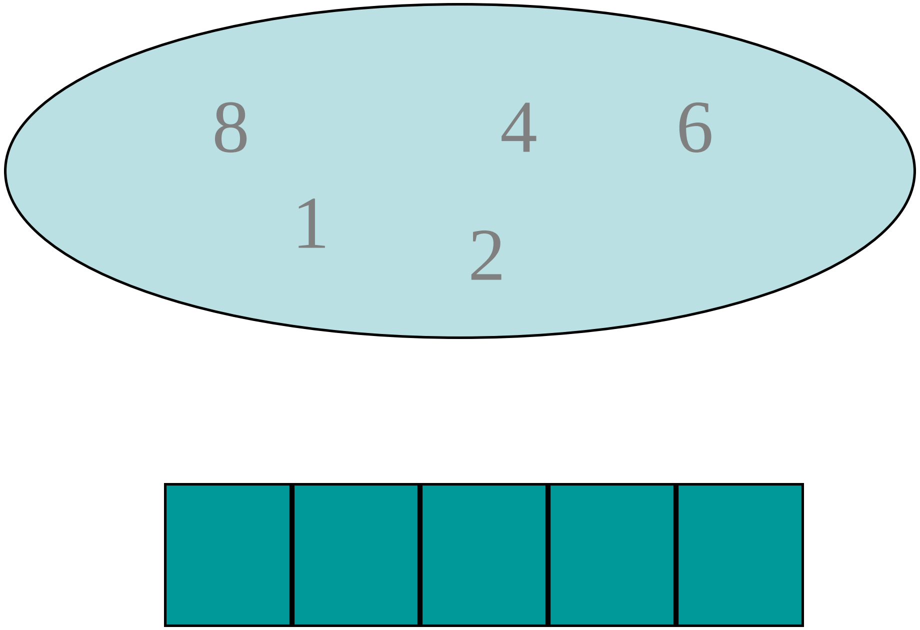
- use element key as priority
- insert elements to be sorted into a priority queue
- remove/pop elements in priority order
  - if a min priority queue is used, elements are extracted in ascending order of priority (or key)
  - if a max priority queue is used, elements are extracted in descending order of priority (or key)

Sorting Example

Sort five elements whose keys are 6, 8, 2, 4, 1 using a max priority queue.

- Insert the five elements into a max priority queue.
- Do five remove max operations placing removed elements into the sorted array from right to left.

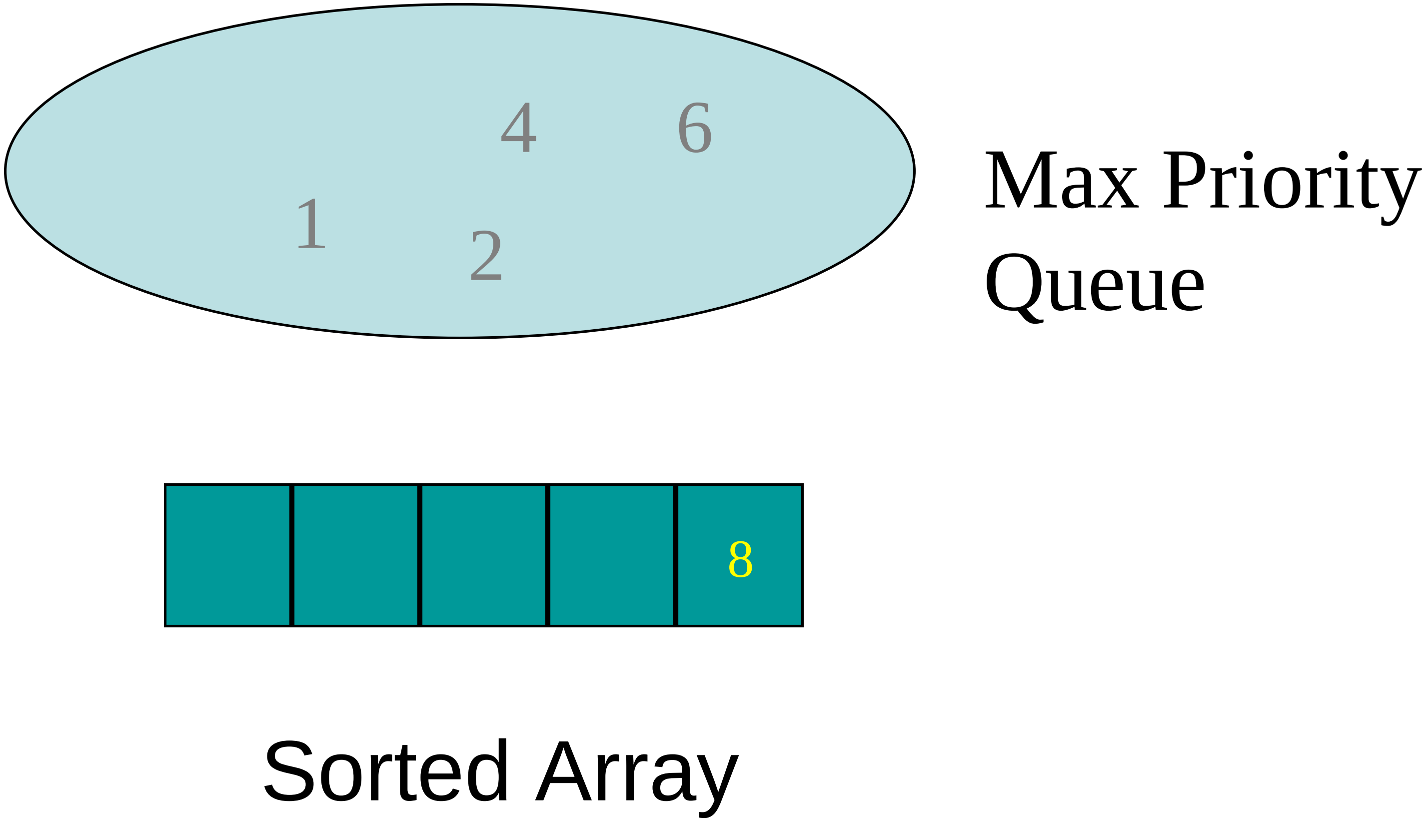
After Inserting Into Max Priority Queue



Max Priority Queue

Sorted Array

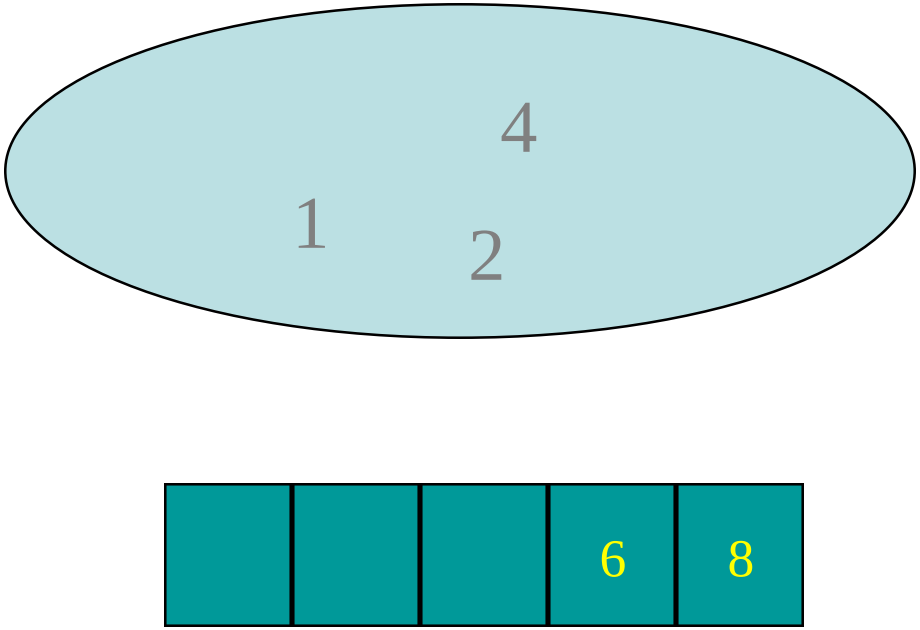
After First Remove Max Operation



Max Priority Queue

Sorted Array

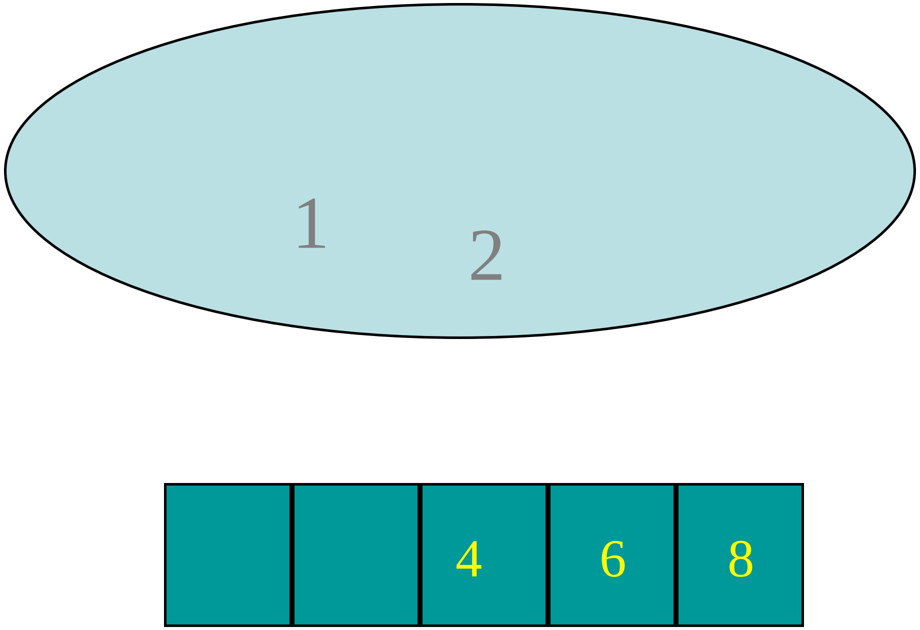
After Second Remove Max Operation



Max Priority Queue

Sorted Array

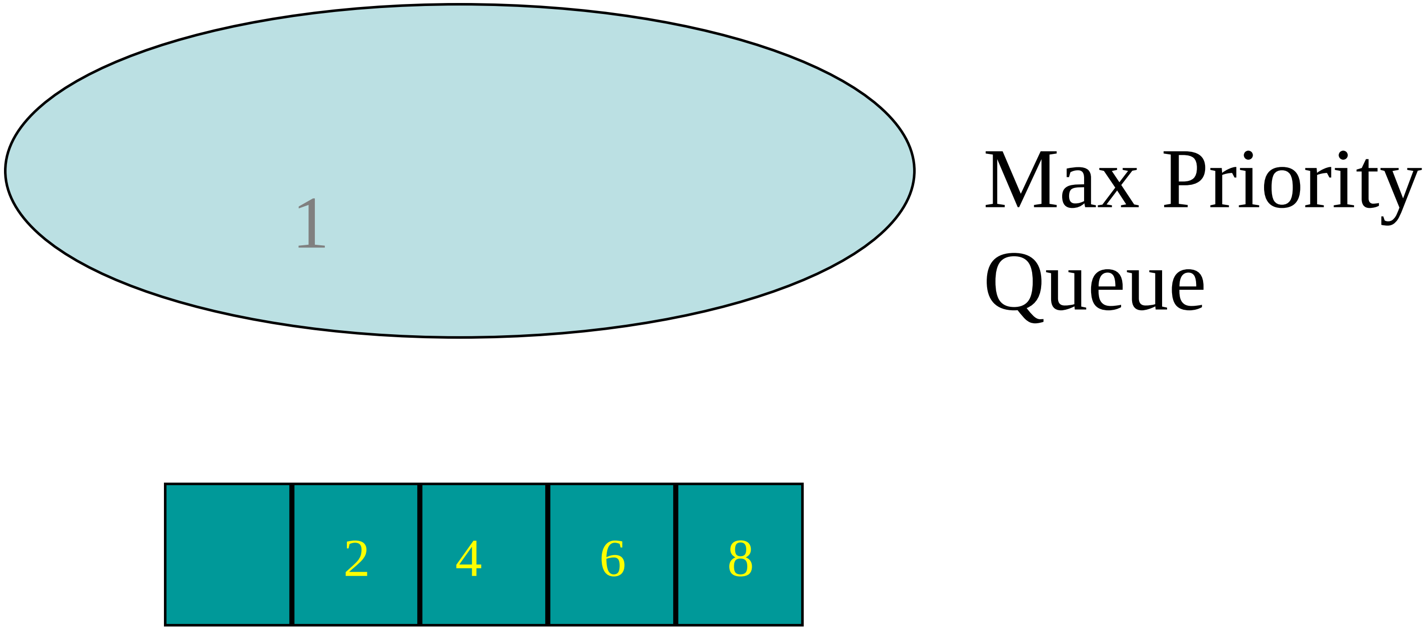
After Third Remove Max Operation



Max Priority Queue

Sorted Array

After Fourth Remove Max Operation



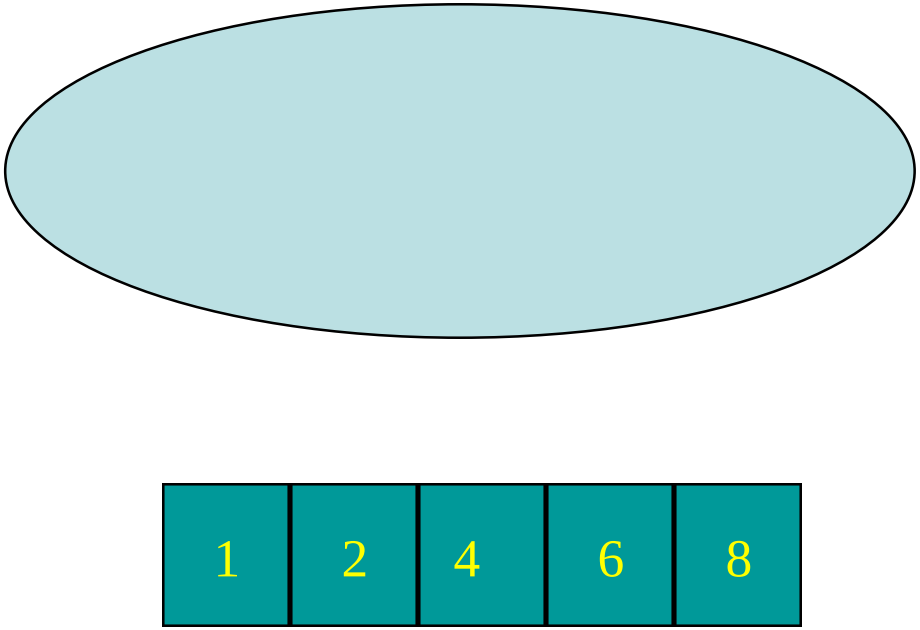
1

Max Priority Queue

	2	4	6	8

Sorted Array

After Fifth Remove Max Operation



Max Priority Queue

1	2	4	6	8

Sorted Array

Heap Sort

Uses a min(max) priority queue that is implemented as a heap.

Initial insert operations are replaced by a heap initialization step that takes O(n) time.

Min Heap Definition

- complete binary tree
- min tree

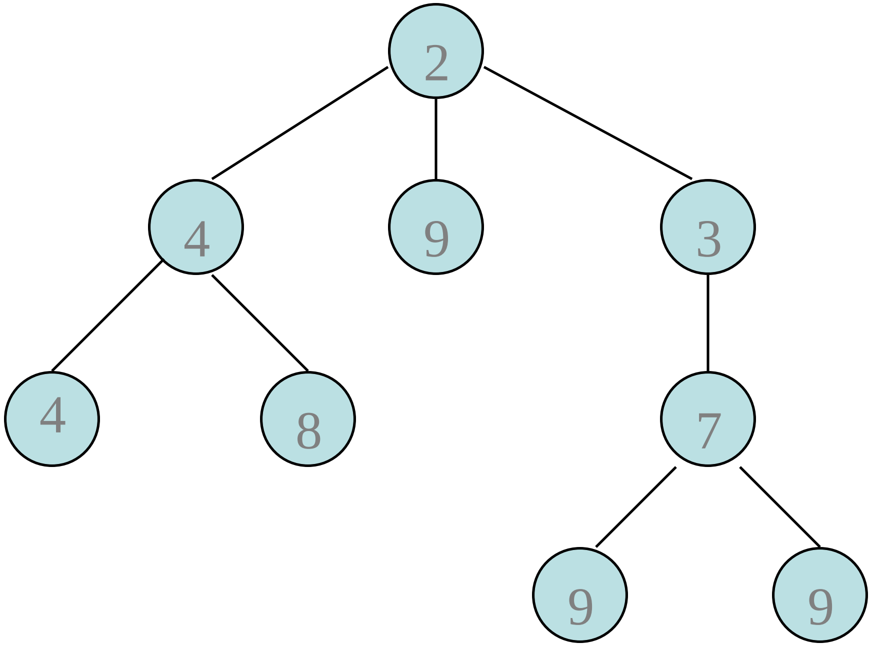
Min Tree Definition

Each tree node has a value.

Value in any node is the minimum value in the subtree for which that node is the root.

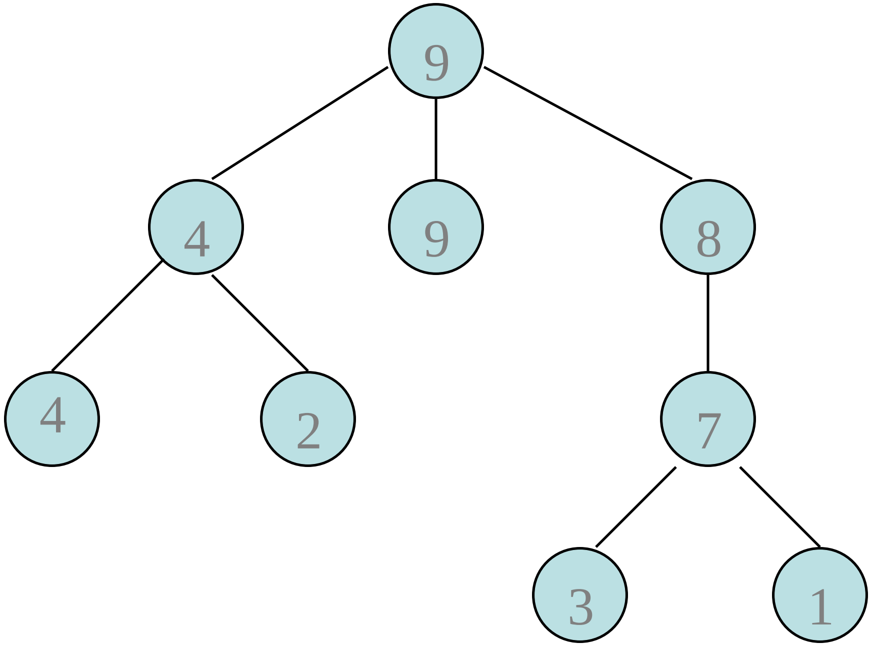
Equivalently, no descendent has a smaller value.

Min Tree Example



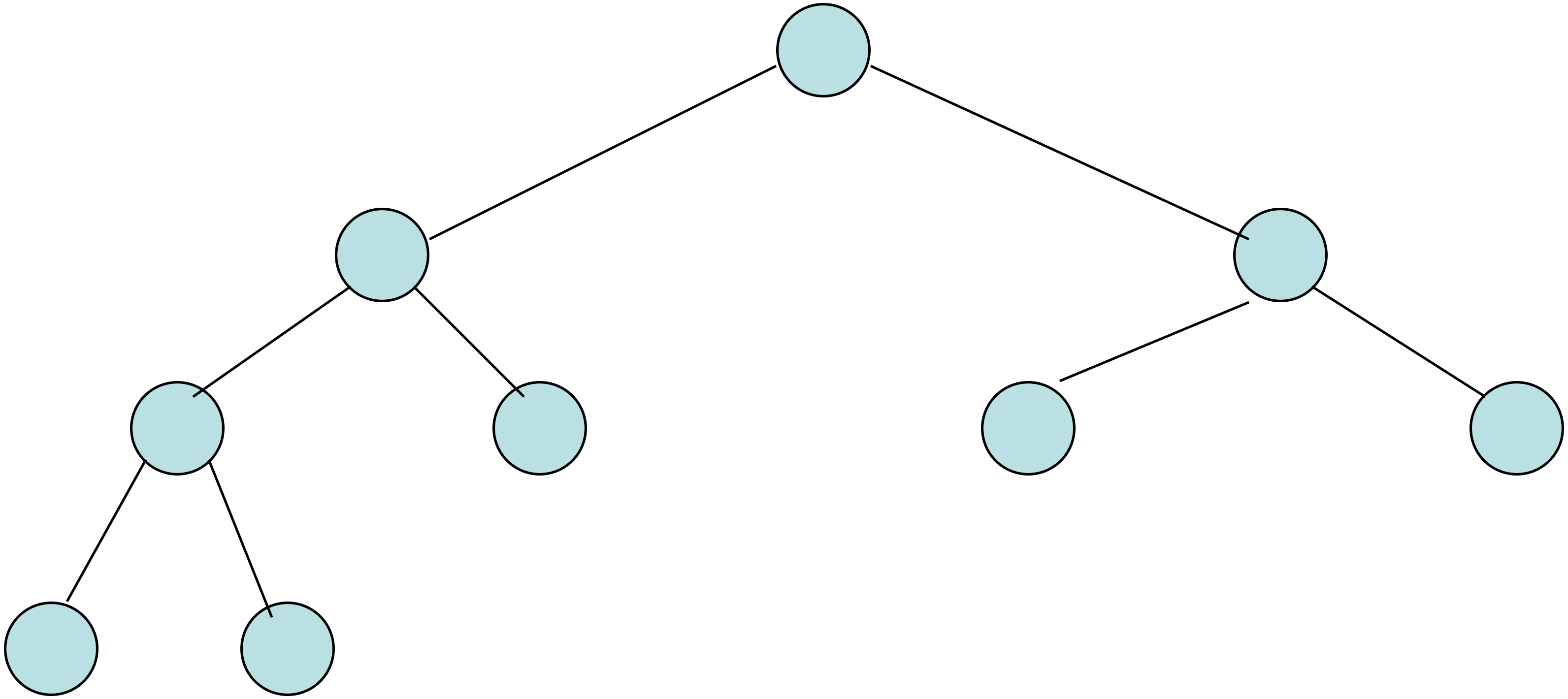
Root has minimum element.

Max Tree Example



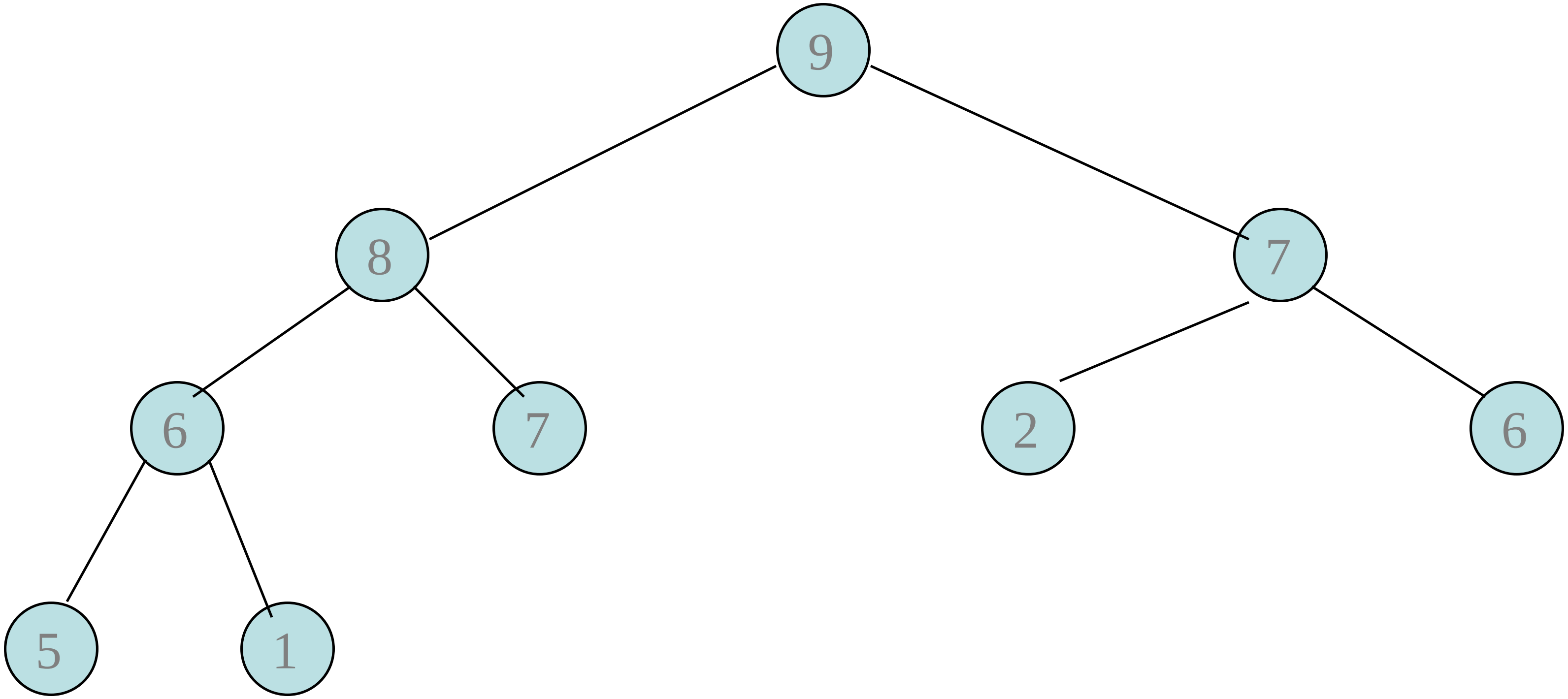
Root has maximum element.

Max Heap With 9 Nodes



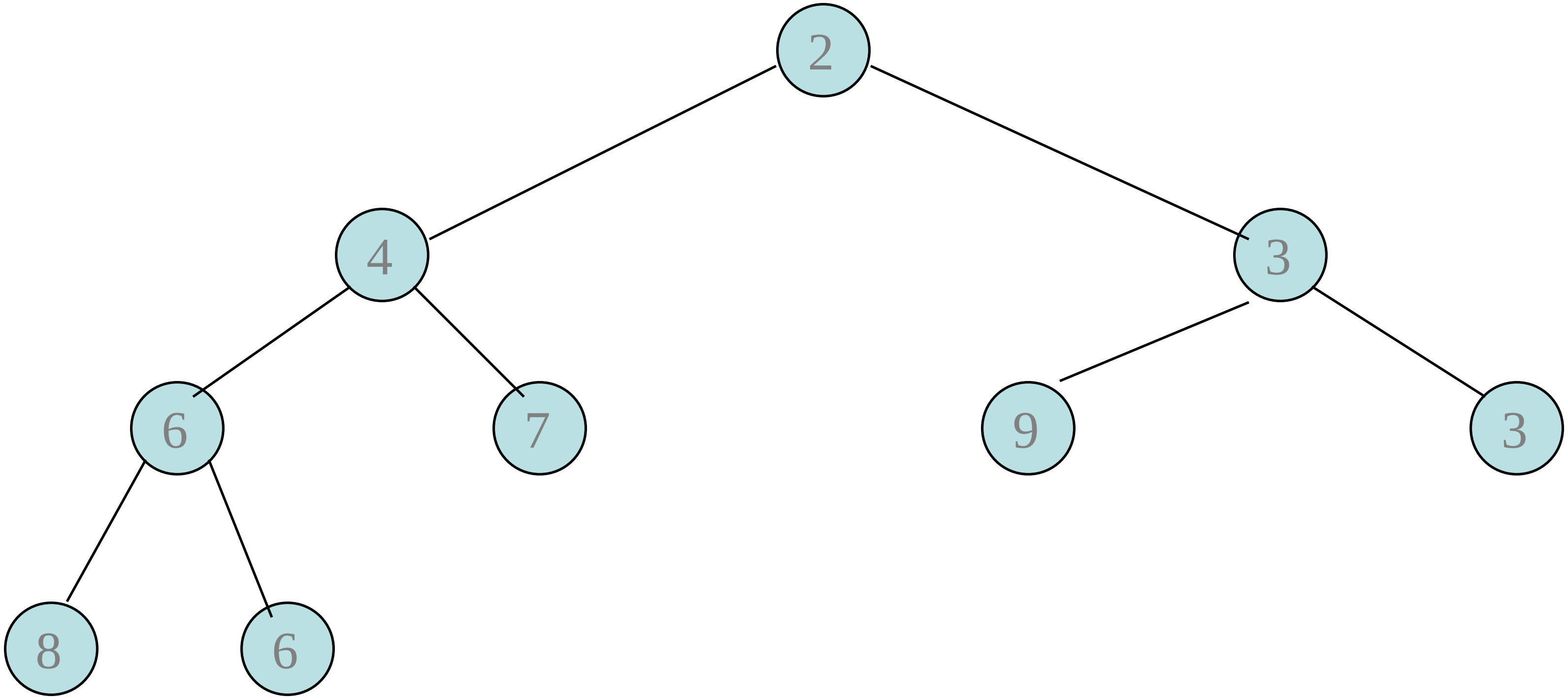
Complete binary tree with 9 nodes.

Max Heap With 9 Nodes



Complete binary tree with 9 nodes that is also a max tree.

Min Heap With 9 Nodes

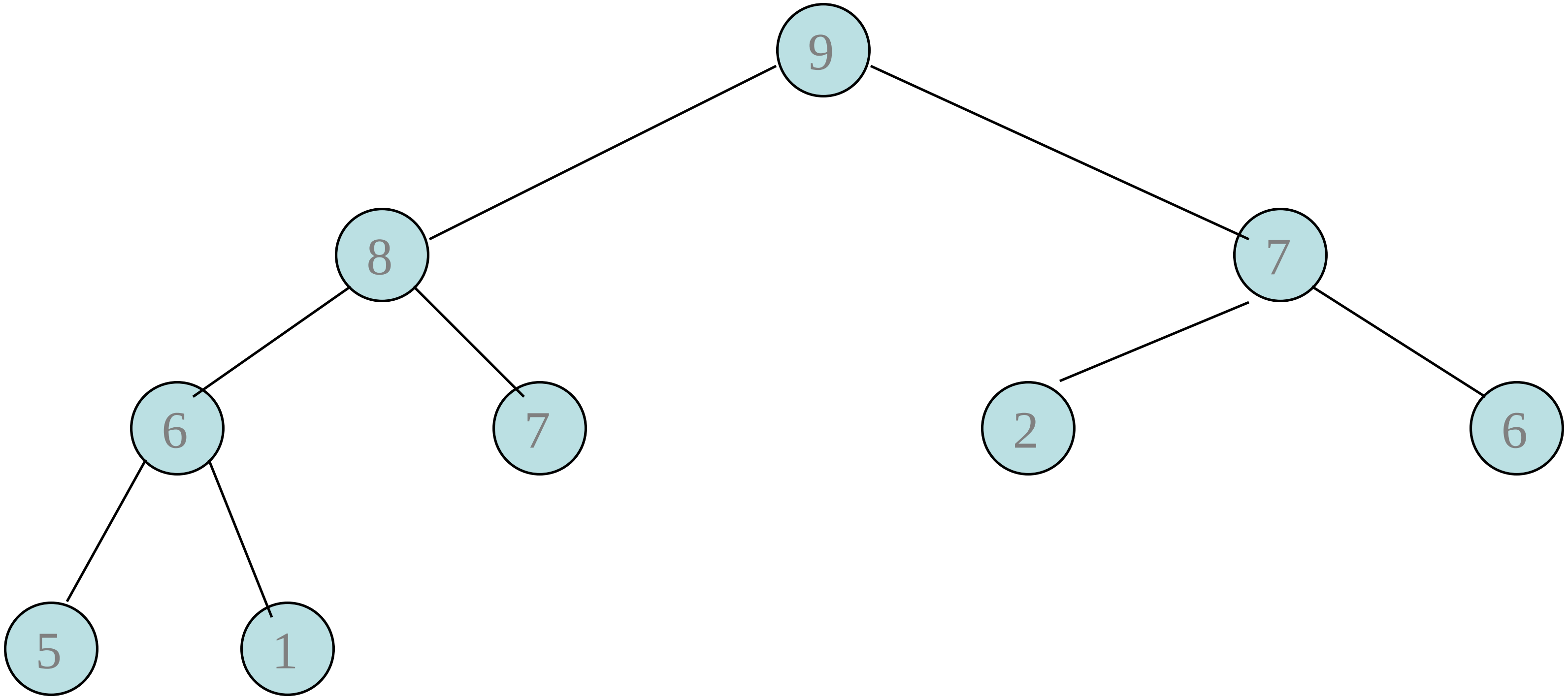


Complete binary tree with 9 nodes that is also a min tree.

Heap Height

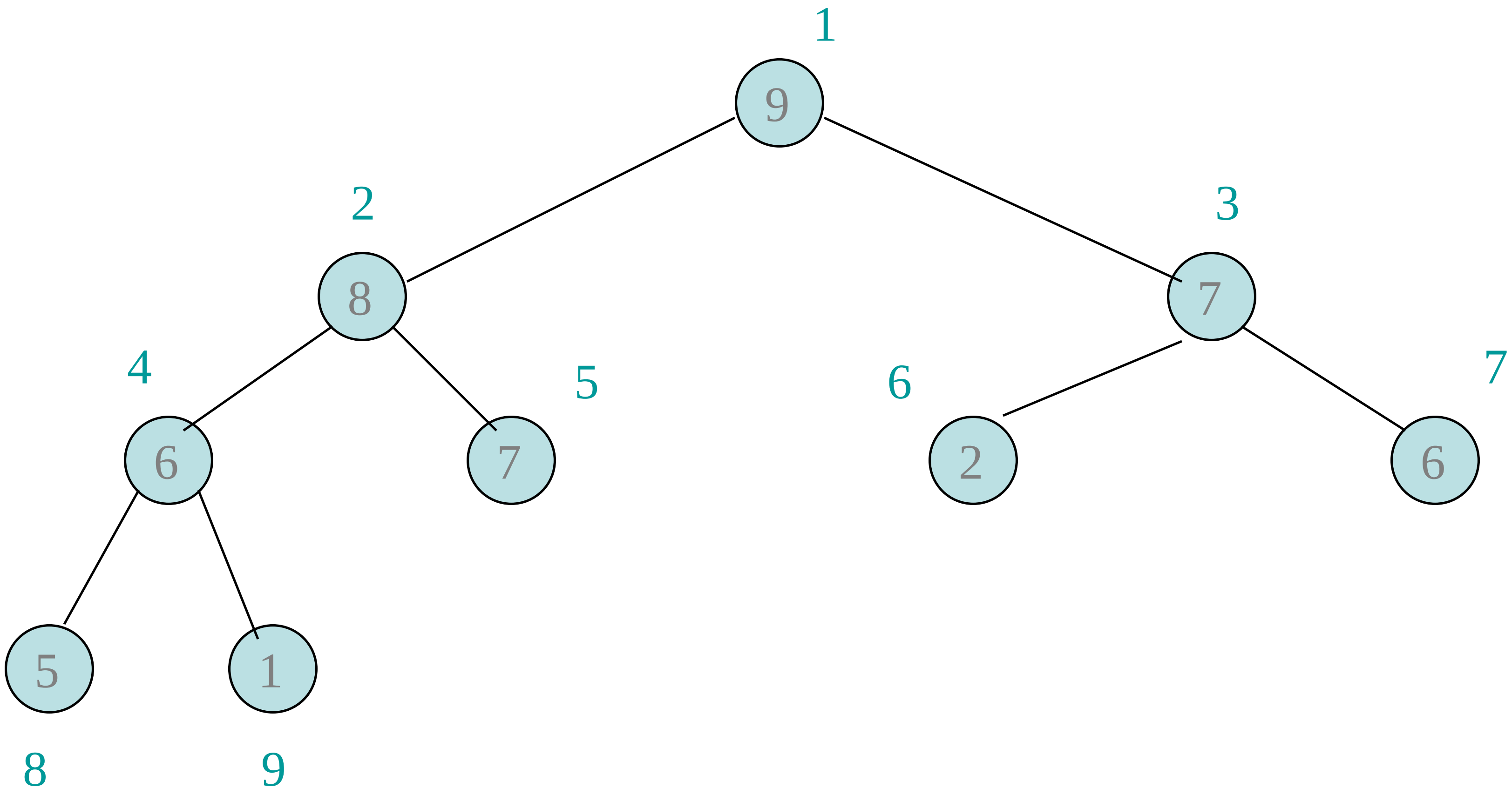
Since a heap is a complete binary tree, the height of an n node heap is upper bound of $\log_2 (n+1)$.

A Heap Is Efficiently Represented As An Array

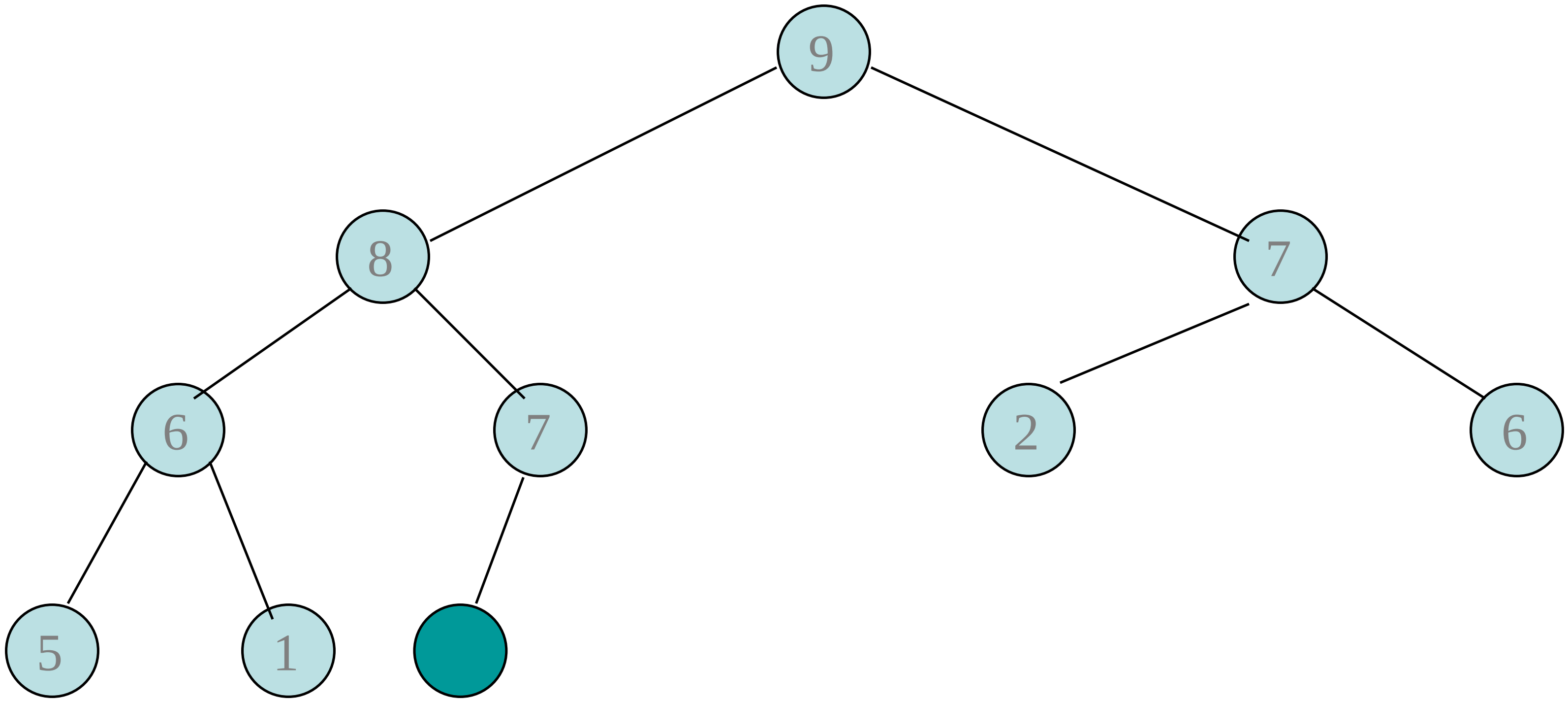


	9	8	7	6	7	2	6	5	1							
0	1	2	3	4	5	6	7	8	9	10						

Moving Up And Down A Heap

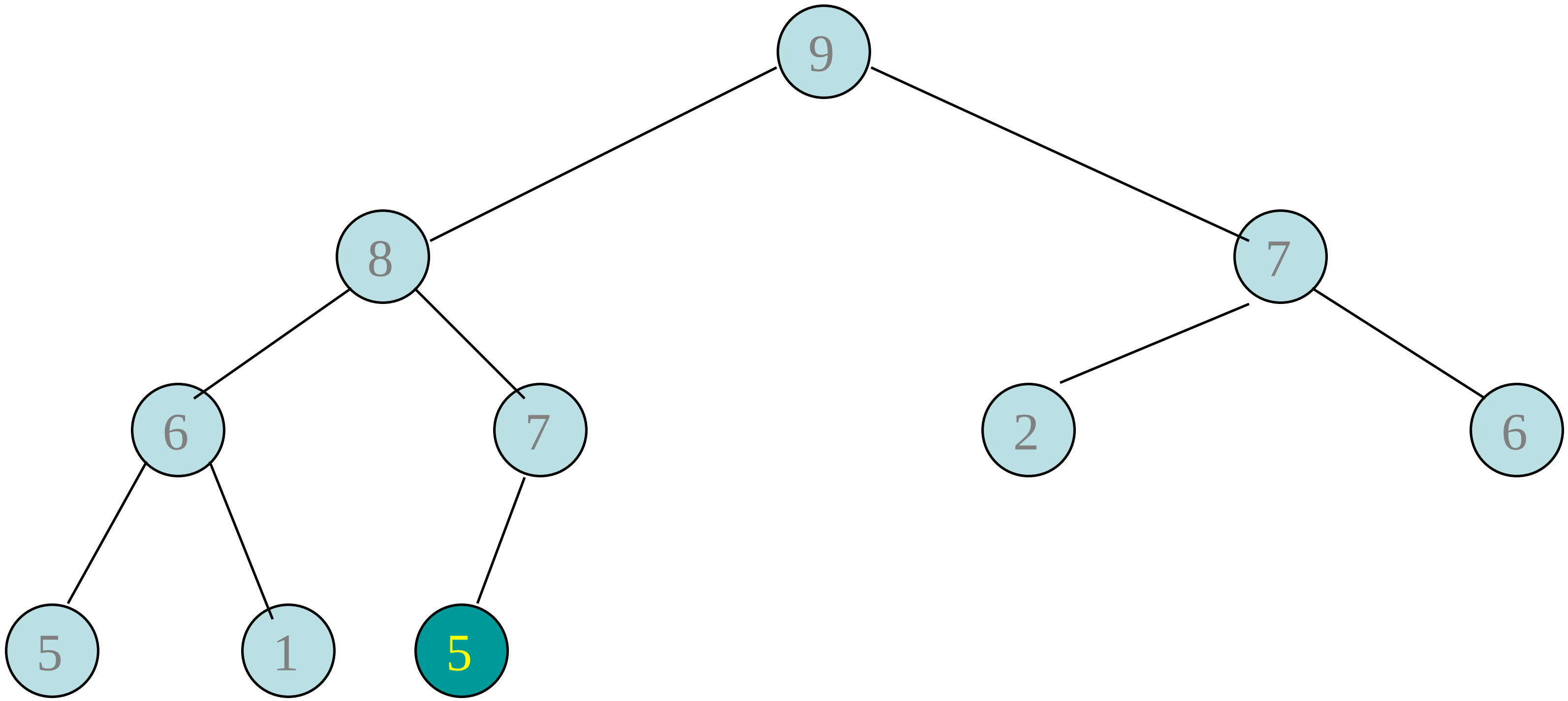


Inserting An Element Into A Max Heap



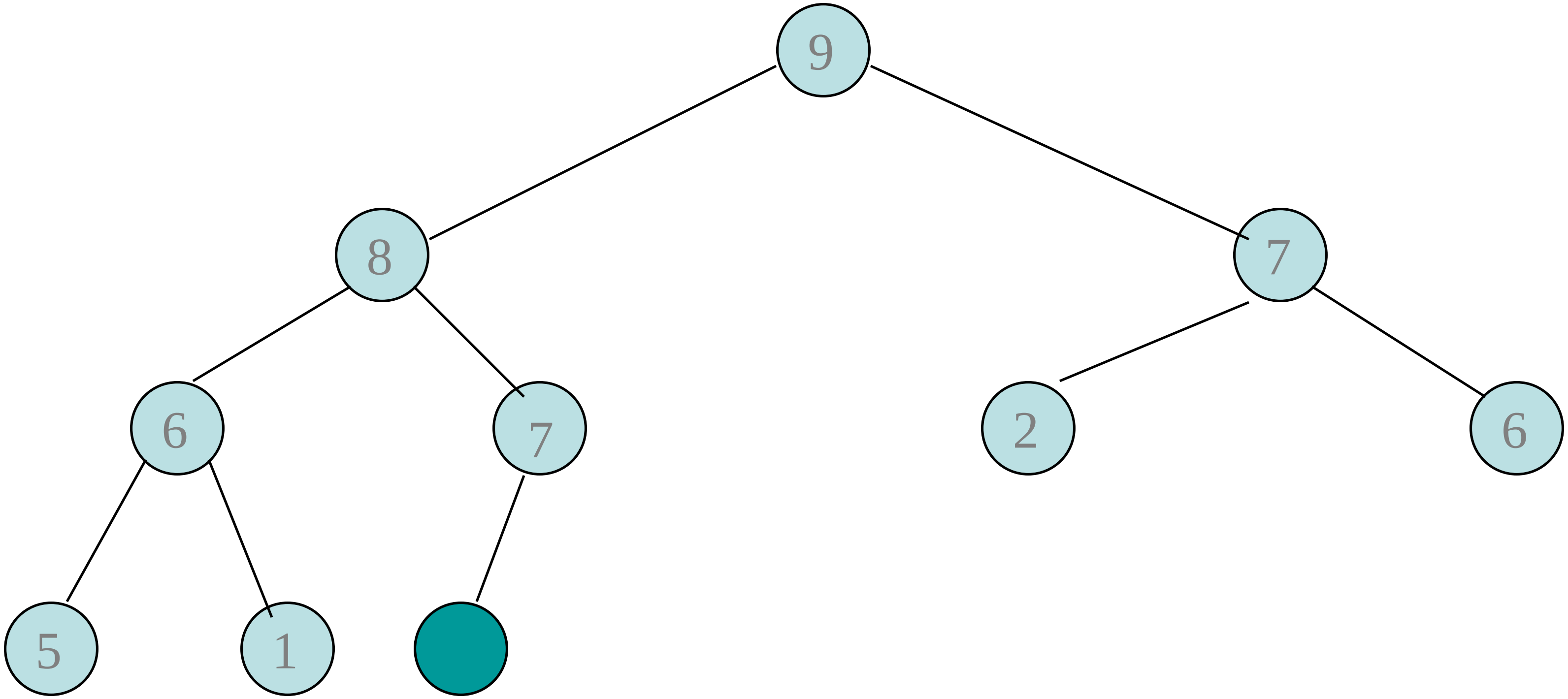
Complete binary tree with 10 nodes.

Inserting An Element Into A Max Heap



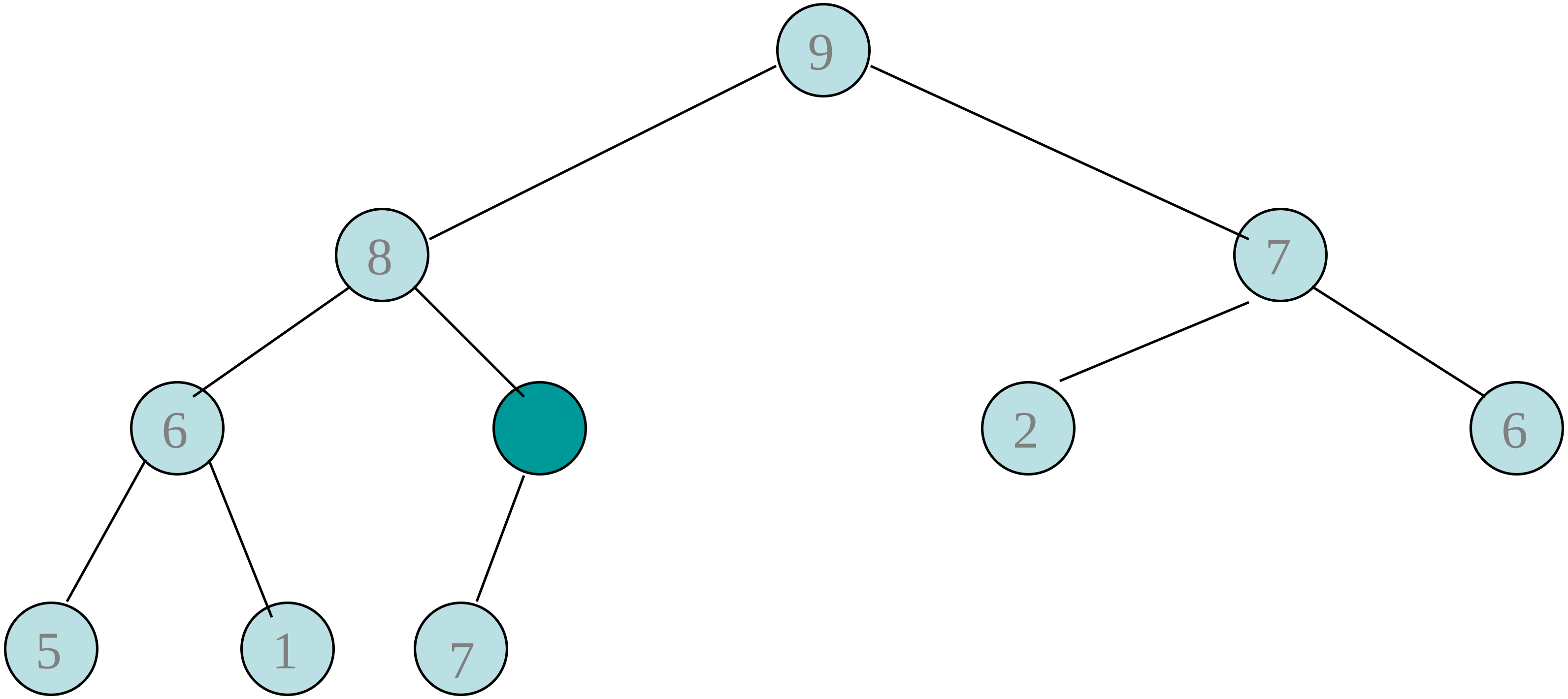
New element is 5.

Inserting An Element Into A Max Heap



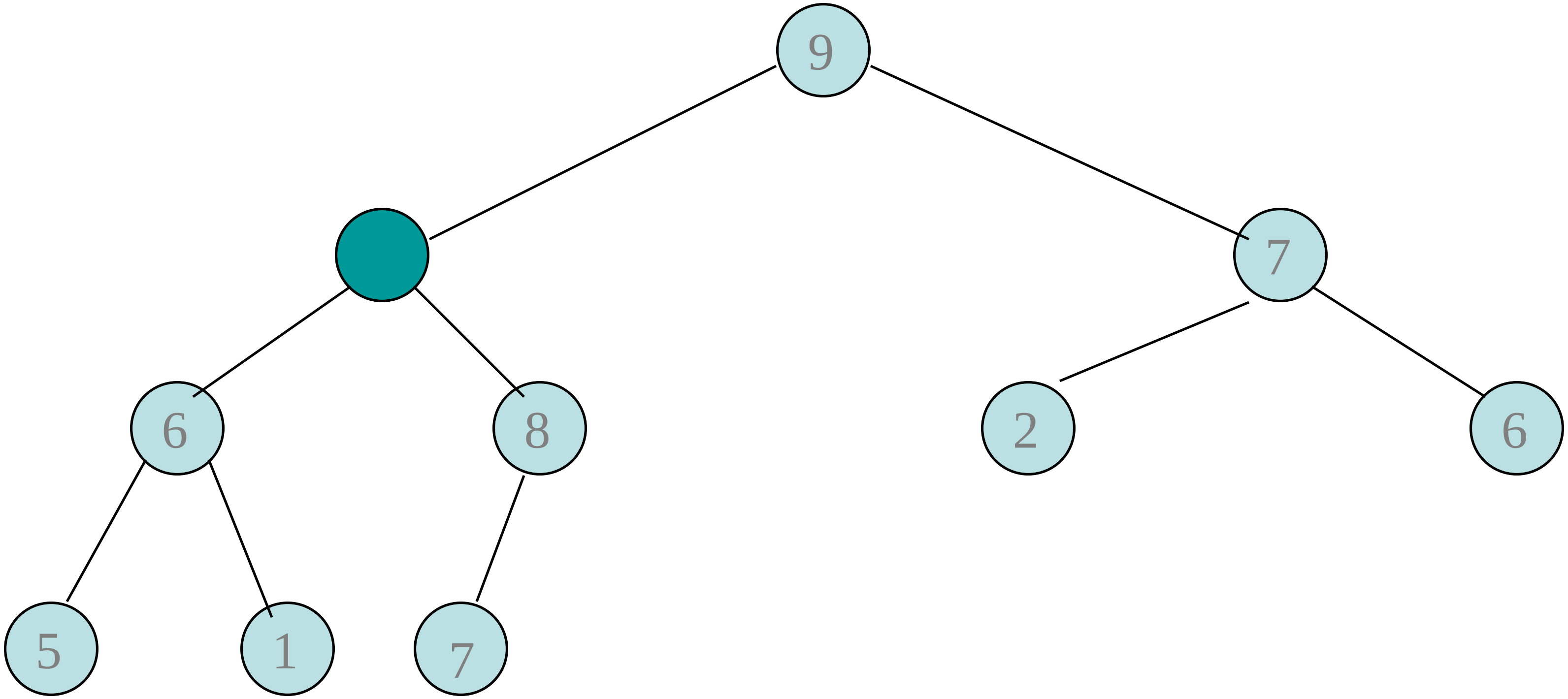
New element is 20.

Inserting An Element Into A Max Heap



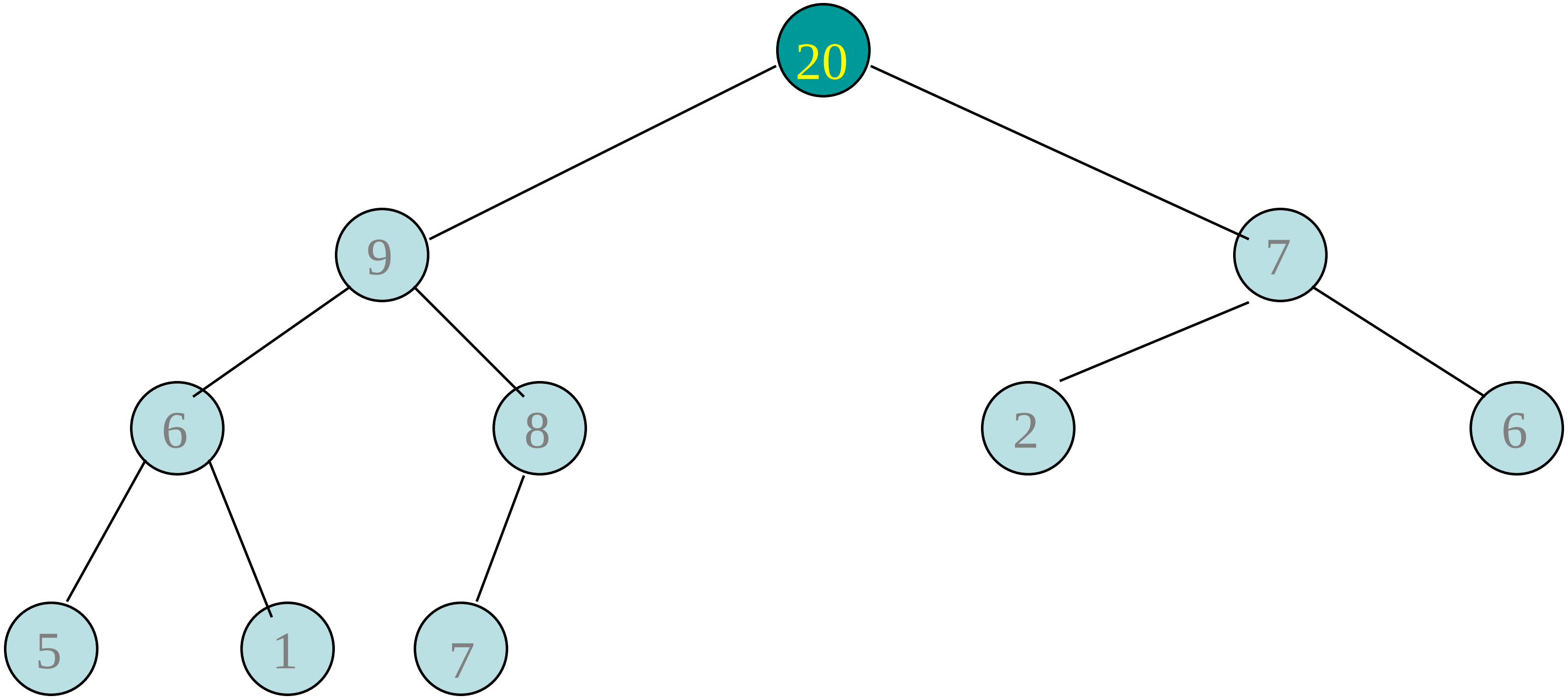
New element is 20.

Inserting An Element Into A Max Heap



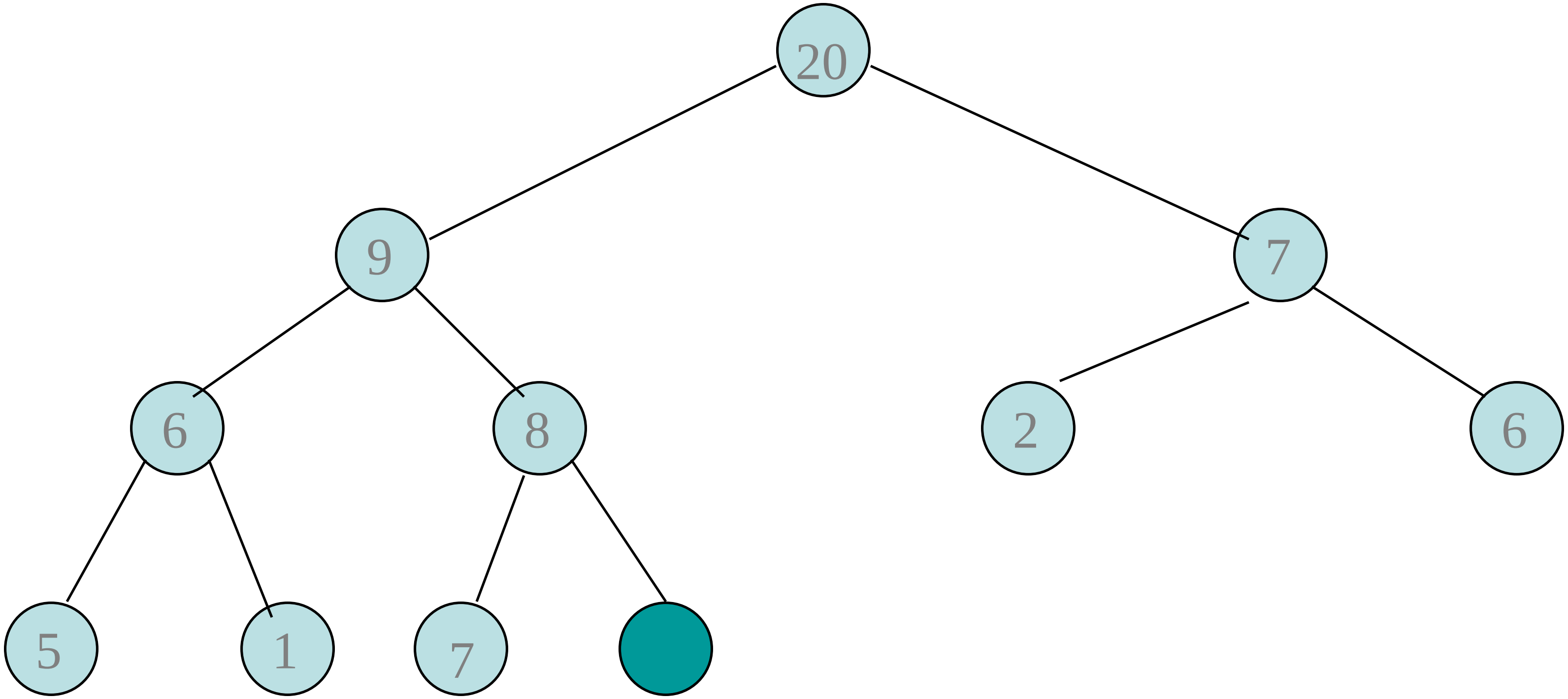
New element is 20.

Inserting An Element Into A Max Heap



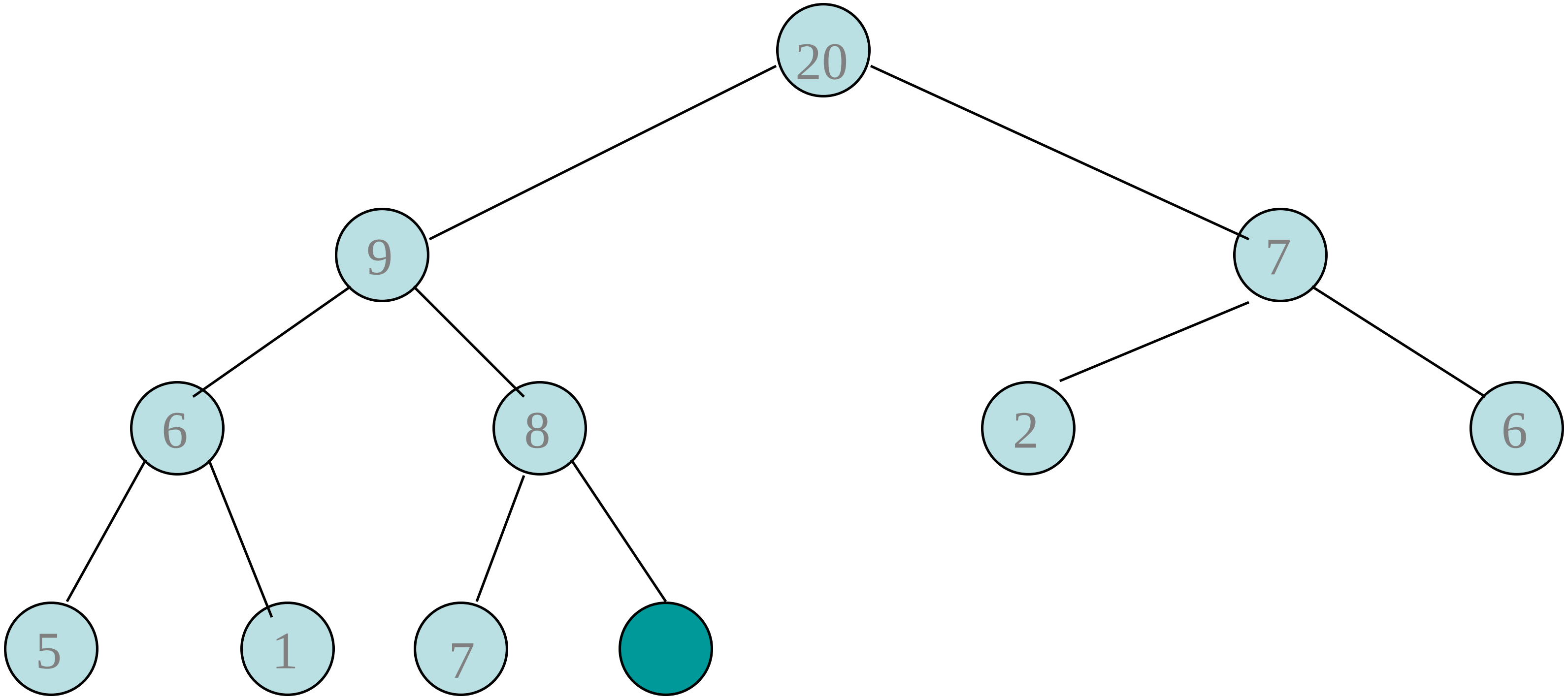
New element is 20.

Inserting An Element Into A Max Heap



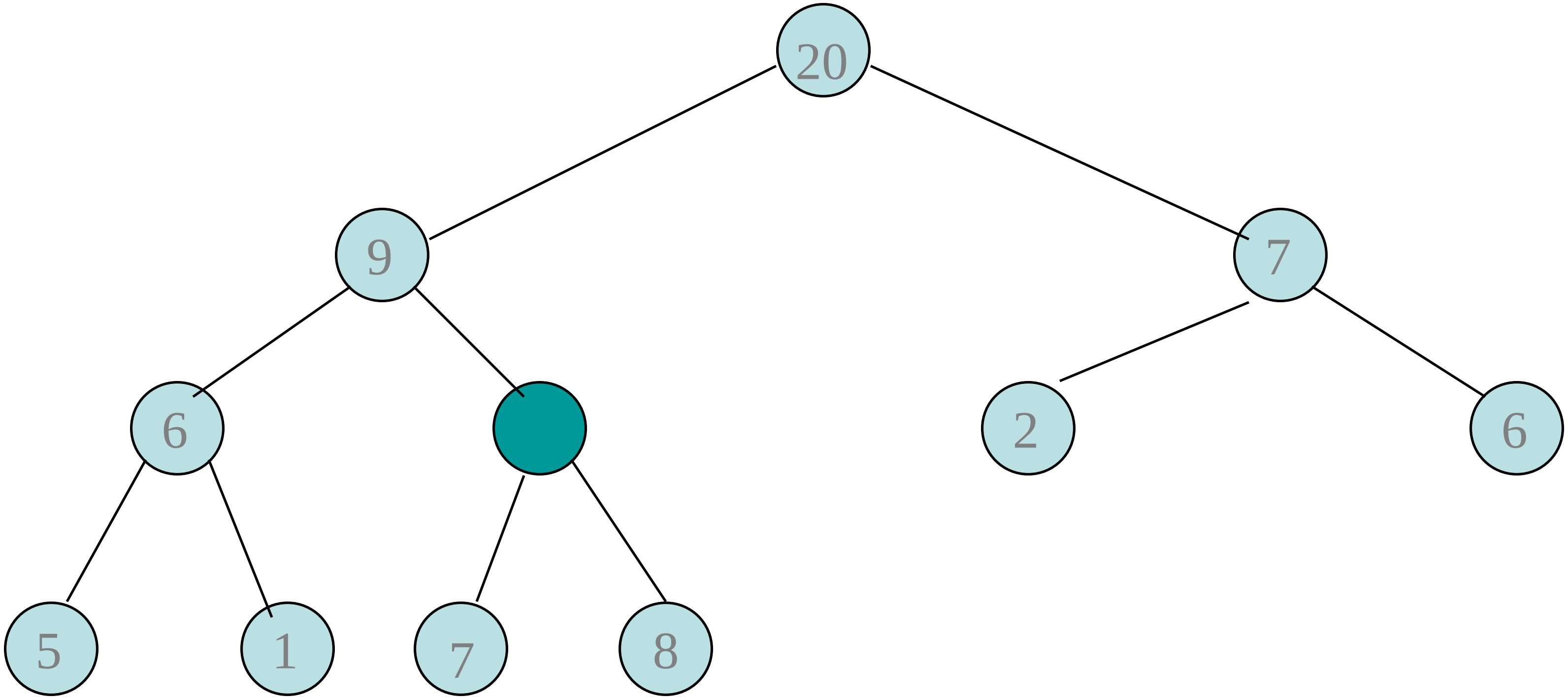
Complete binary tree with 11 nodes.

Inserting An Element Into A Max Heap



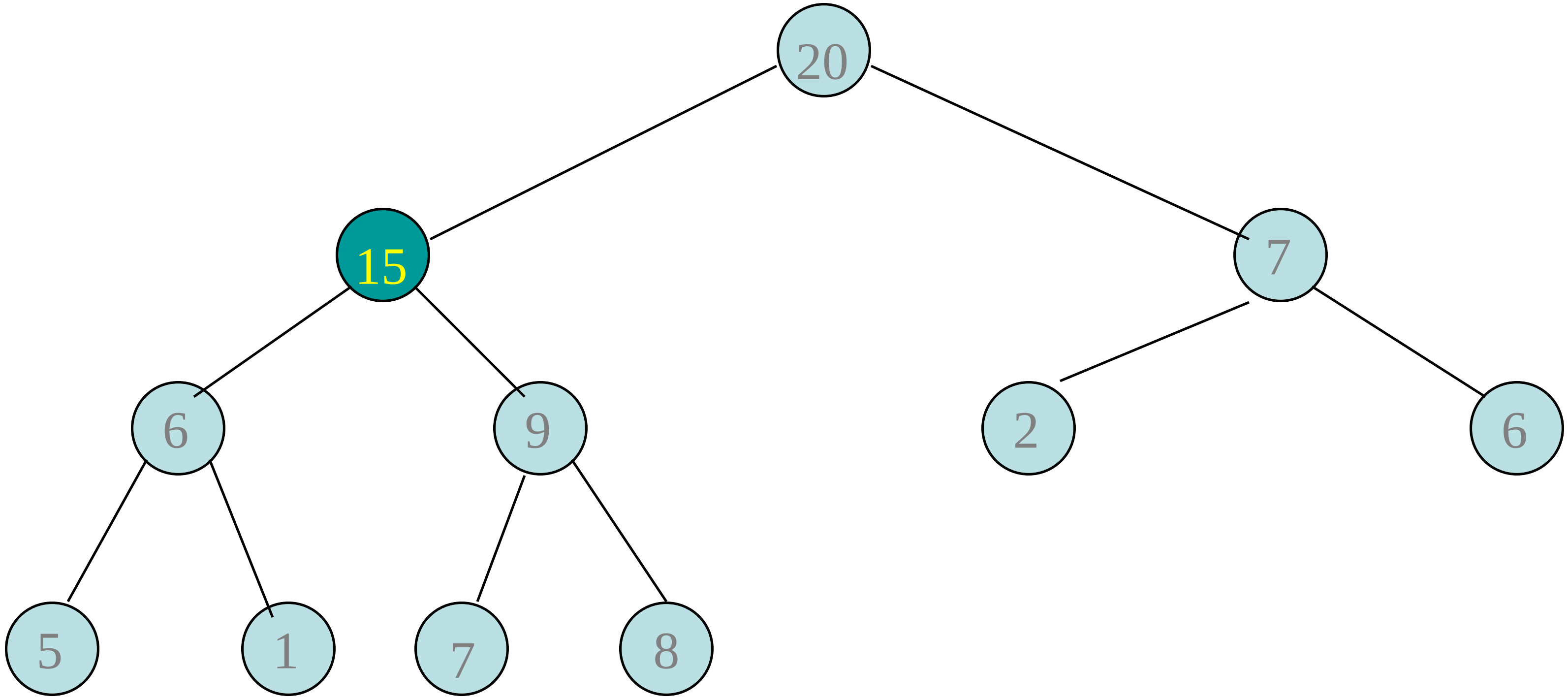
New element is 15.

Inserting An Element Into A Max Heap



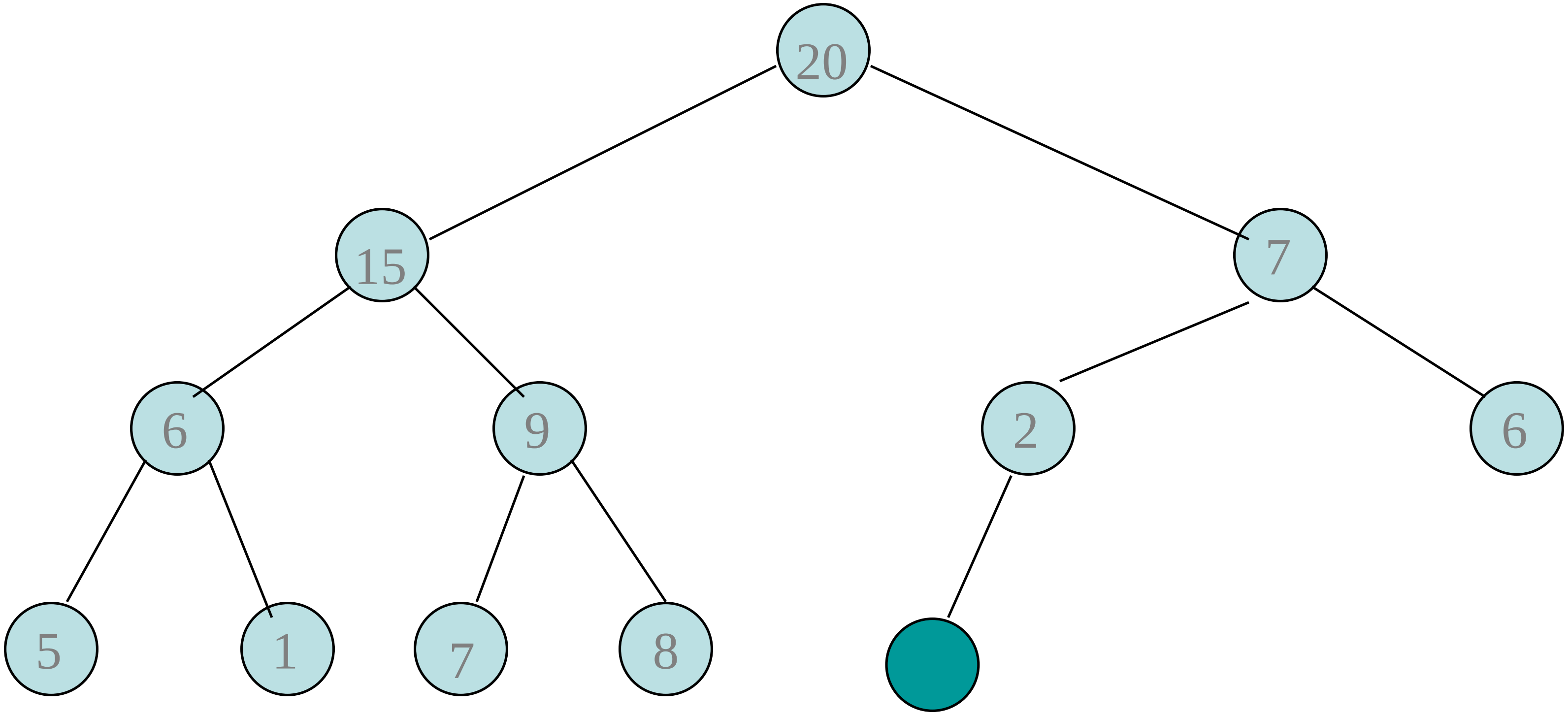
New element is 15.

Inserting An Element Into A Max Heap



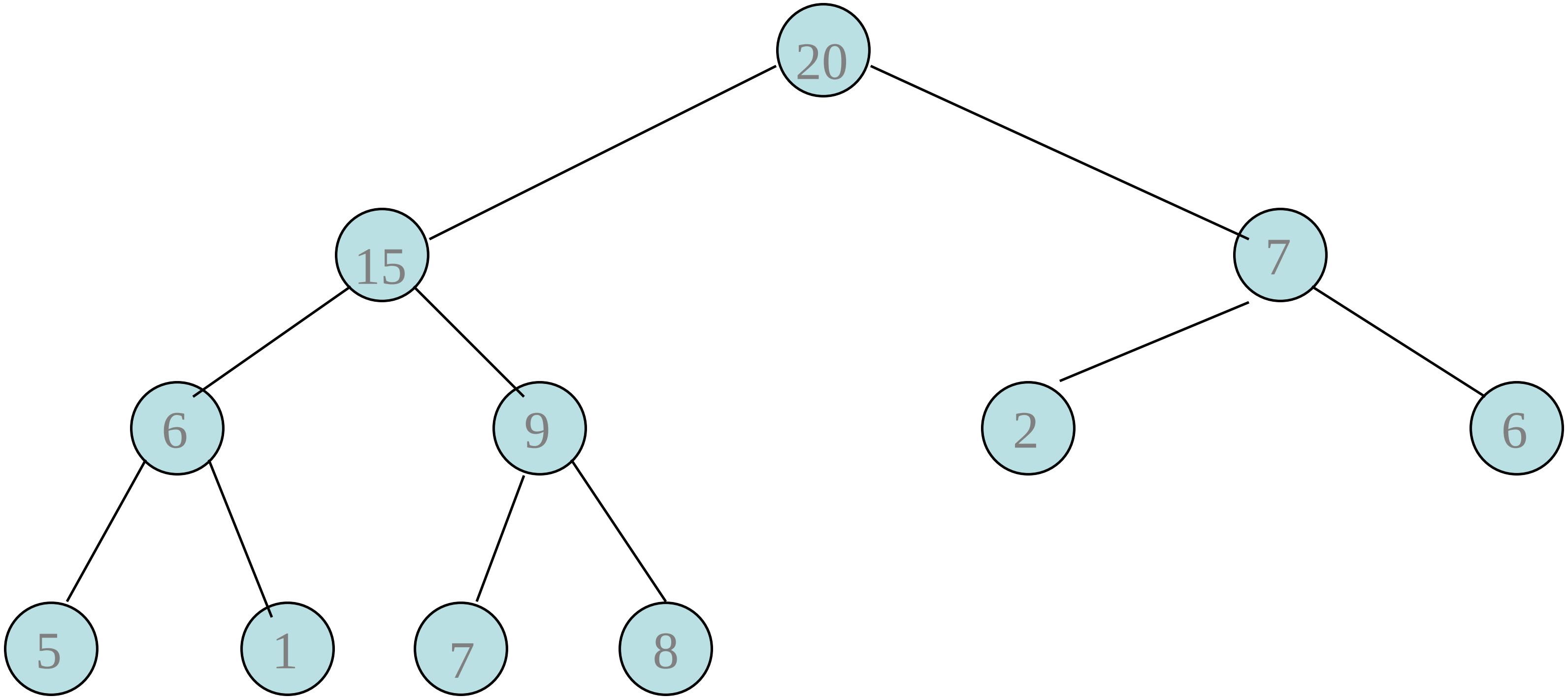
New element is 15.

Complexity Of Insert



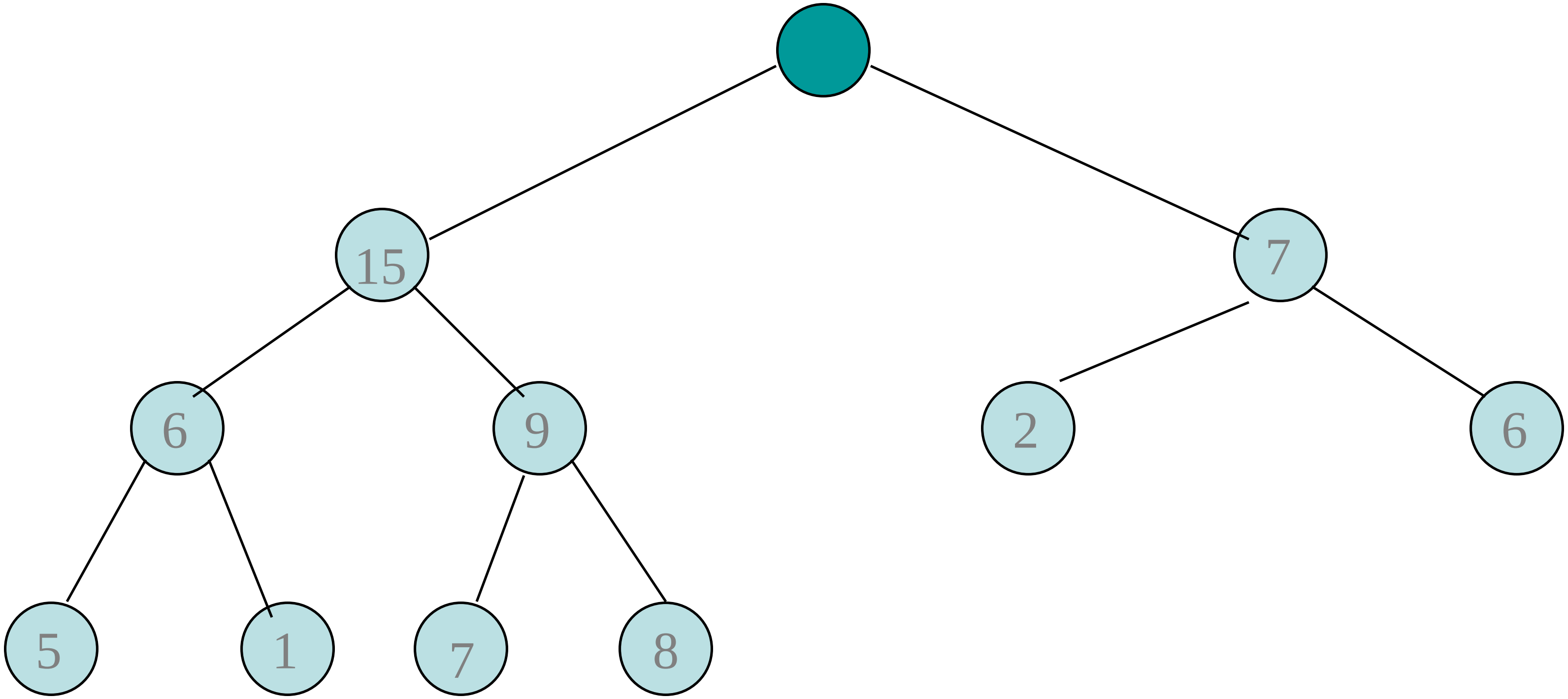
Complexity is O(log n), where n is heap size.

Removing The Max Element



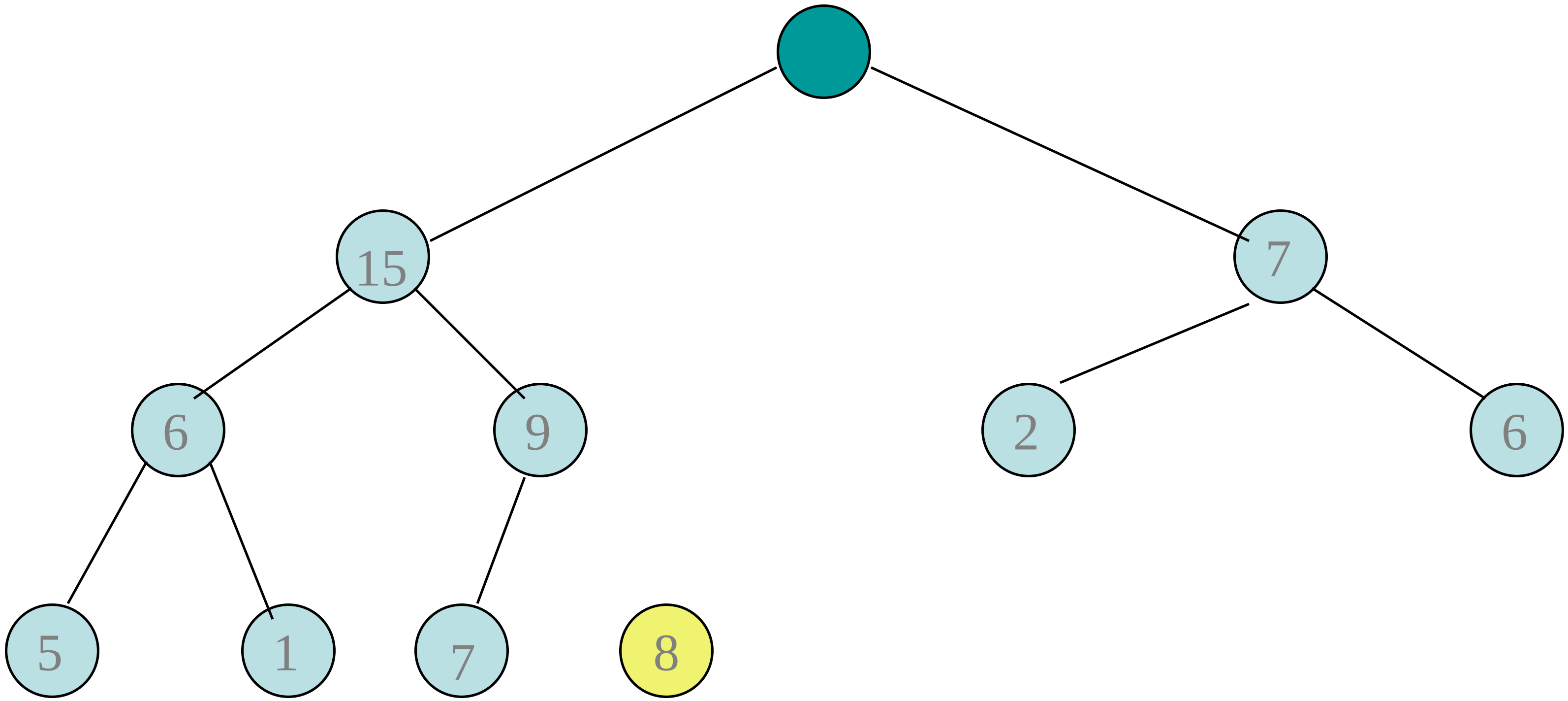
Max element is in the root.

Removing The Max Element



After max element is removed.

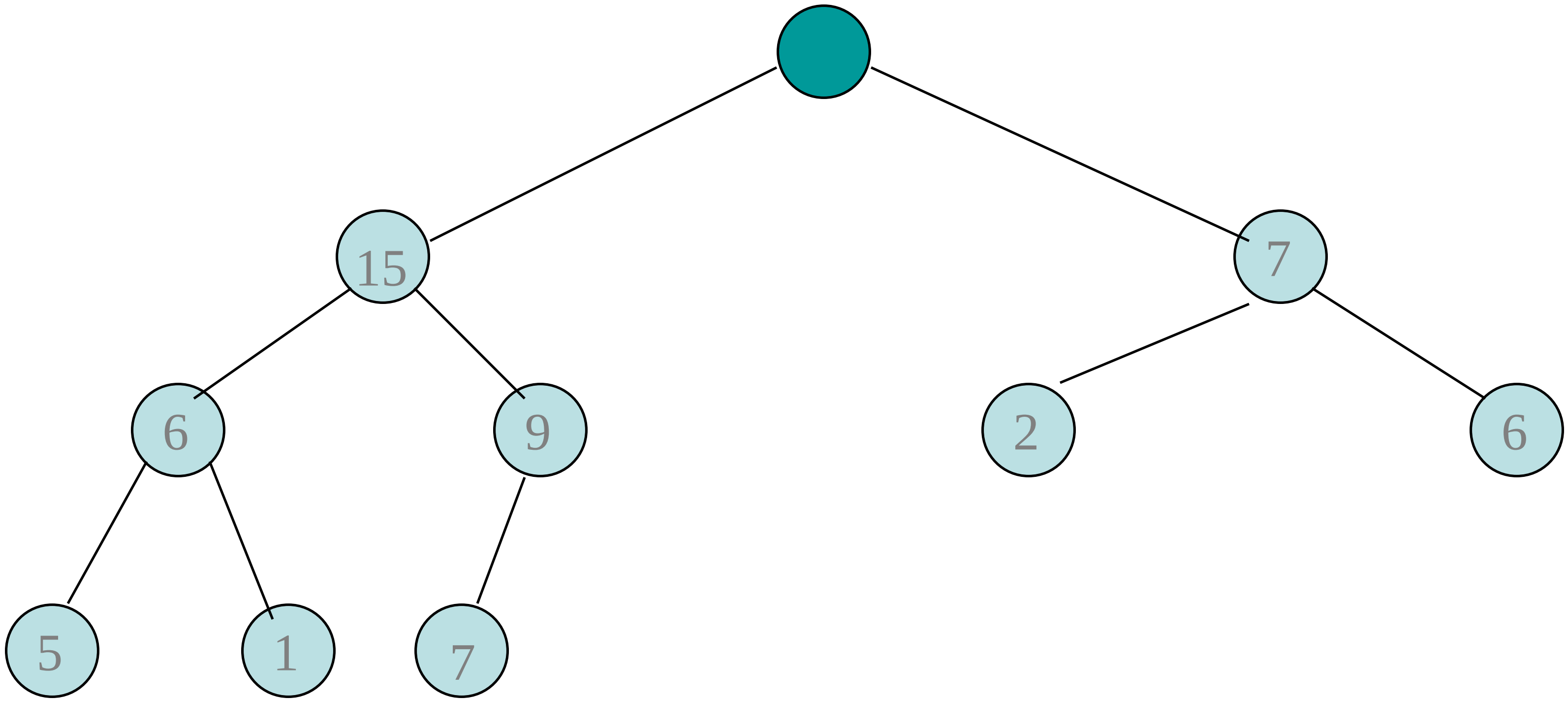
Removing The Max Element



Heap with 10 nodes.

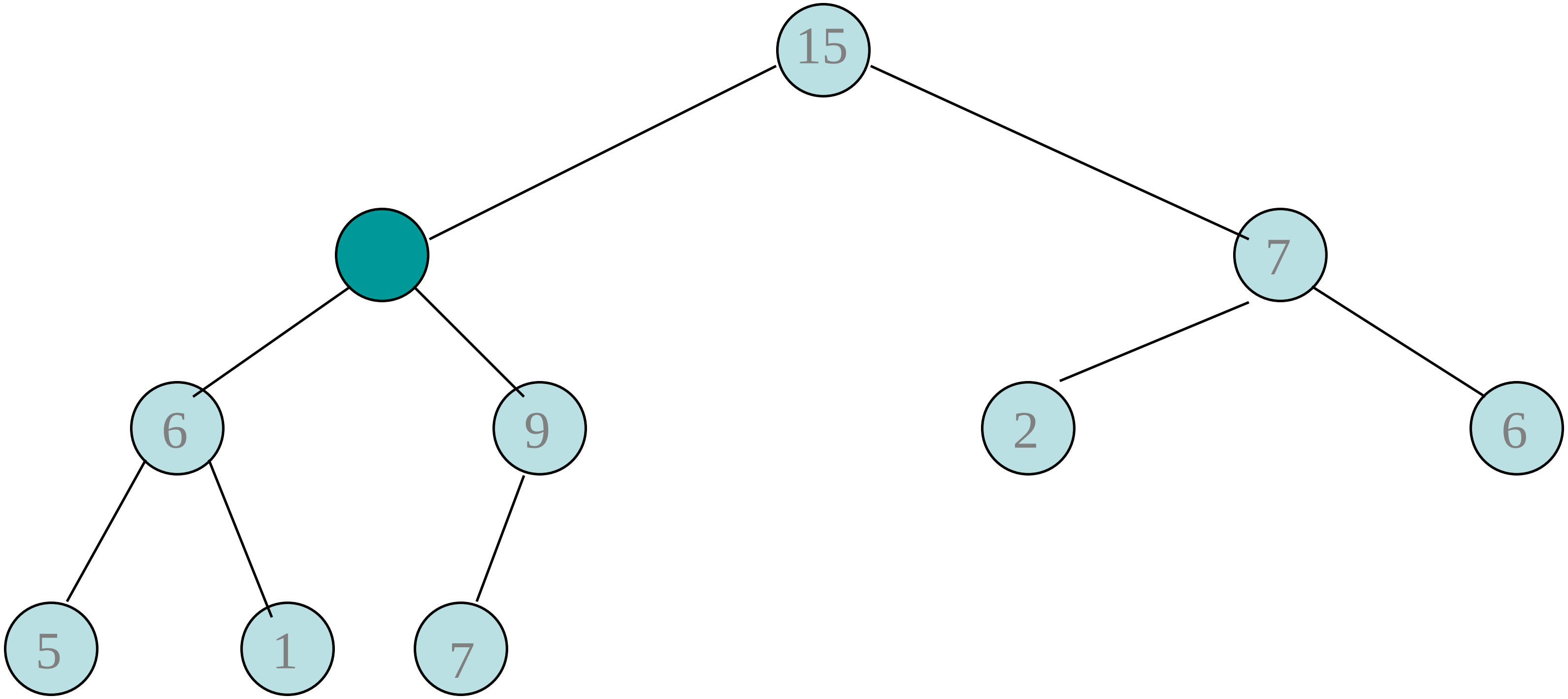
Reinsert 8 into the heap.

Removing The Max Element



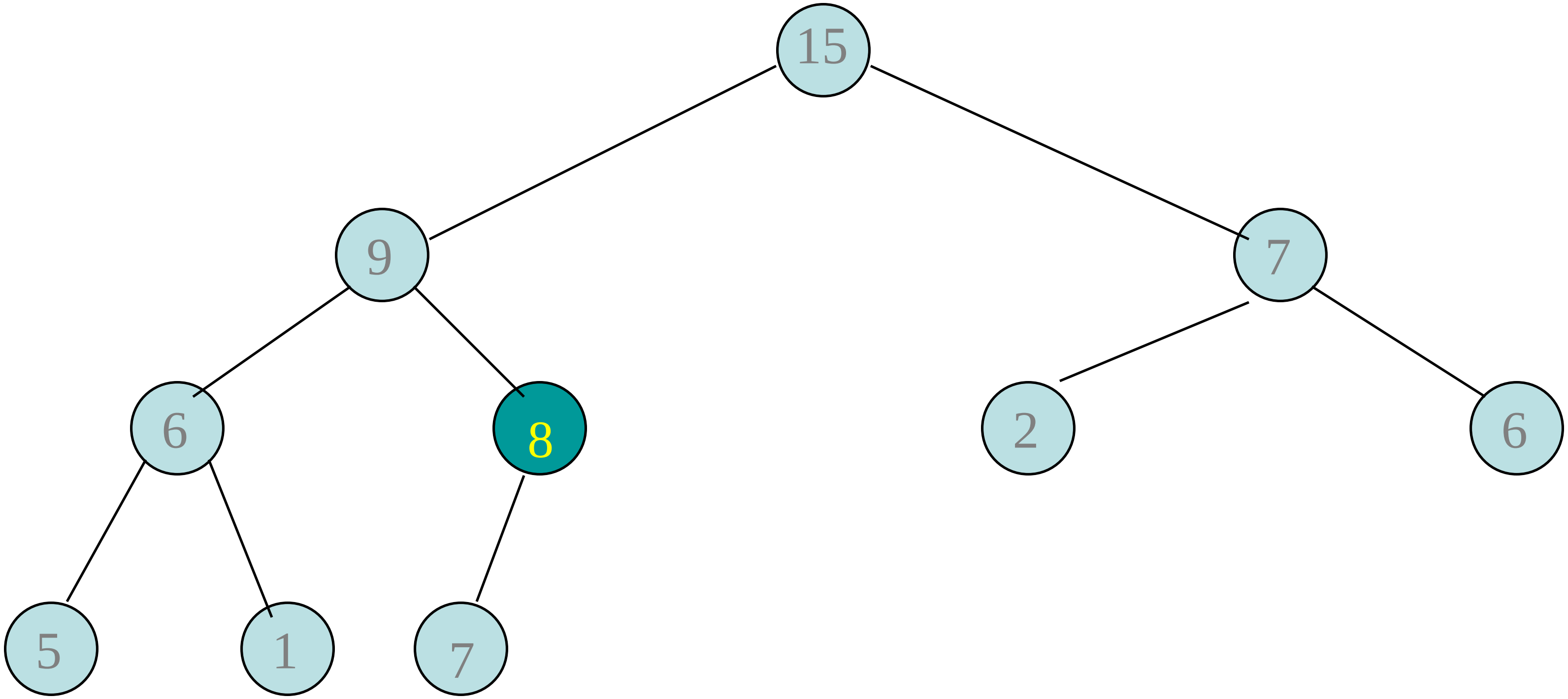
Reinsert 8 into the heap.

Removing The Max Element



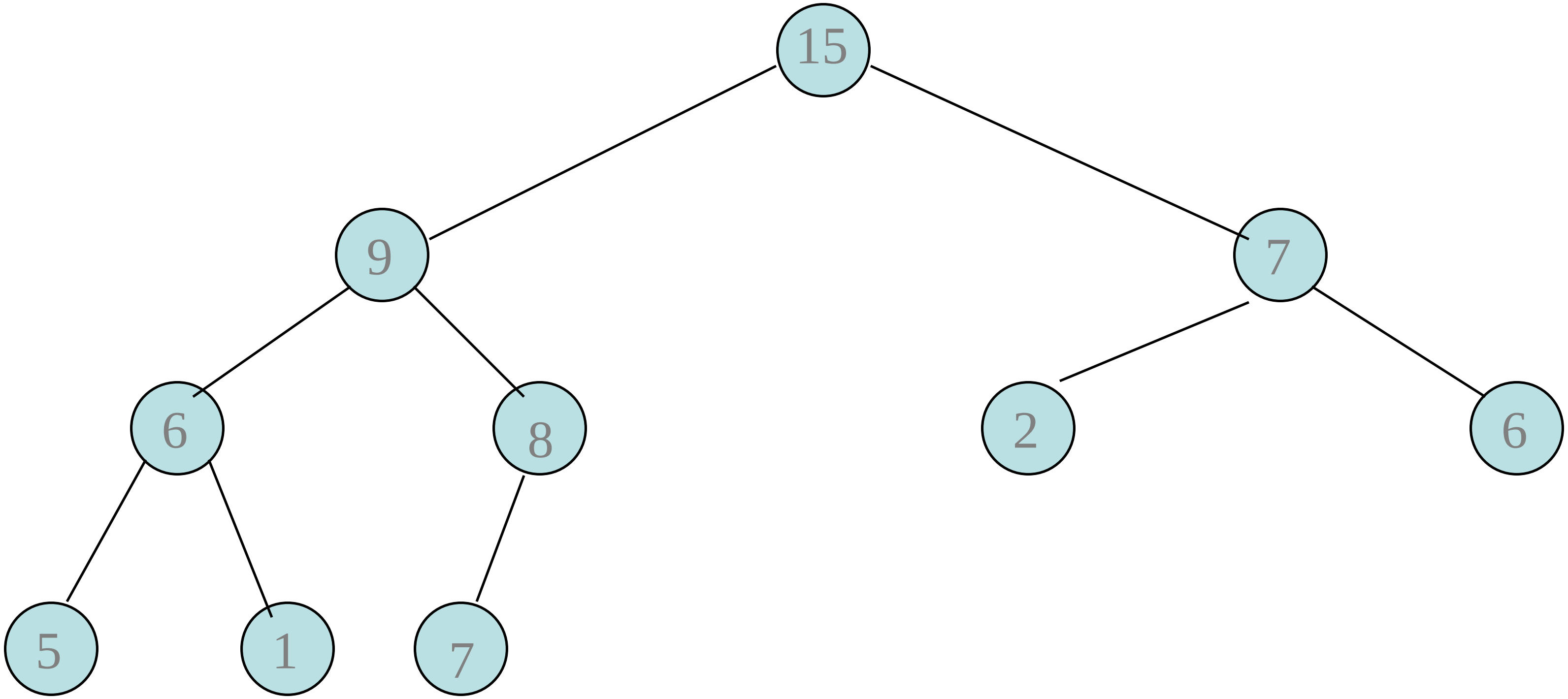
Reinsert 8 into the heap.

Removing The Max Element



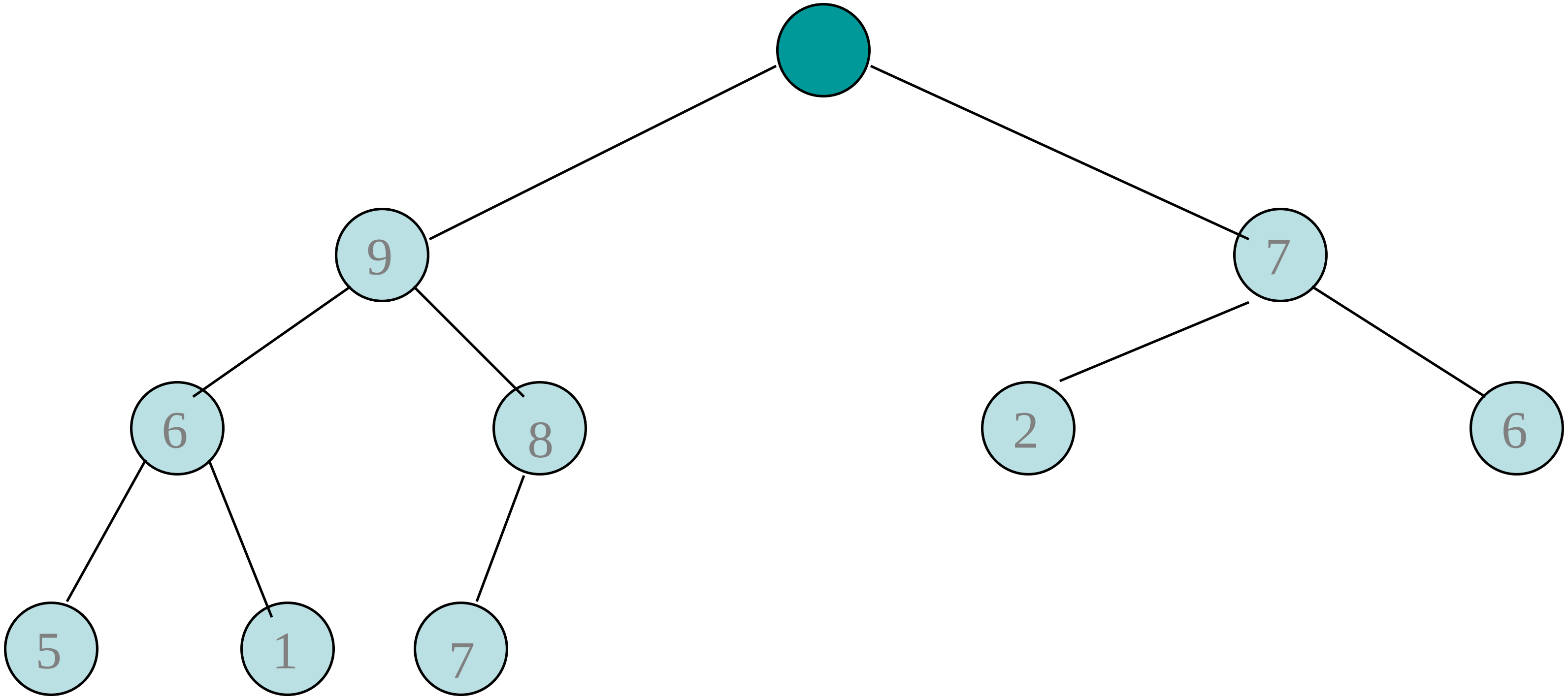
Reinsert 8 into the heap.

Removing The Max Element



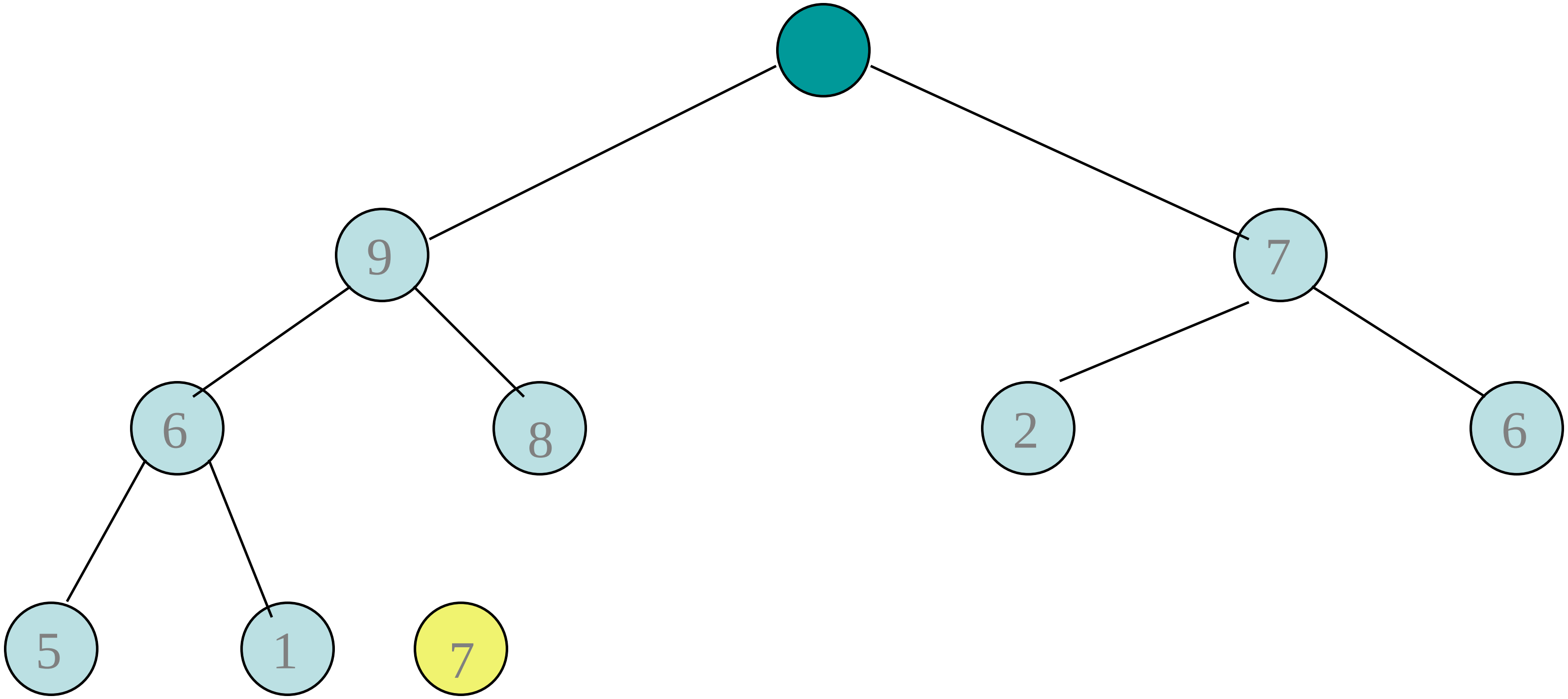
Max element is 15.

Removing The Max Element



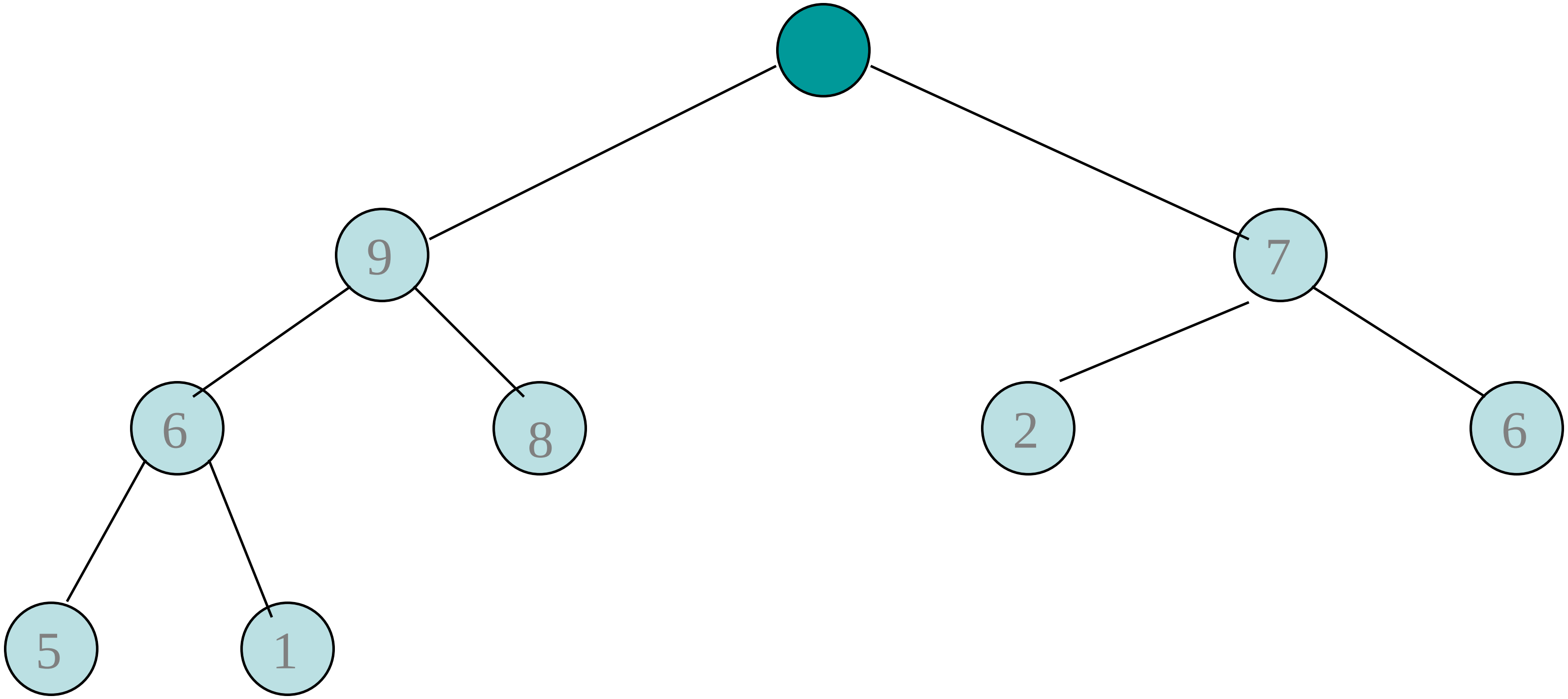
After max element is removed.

Removing The Max Element



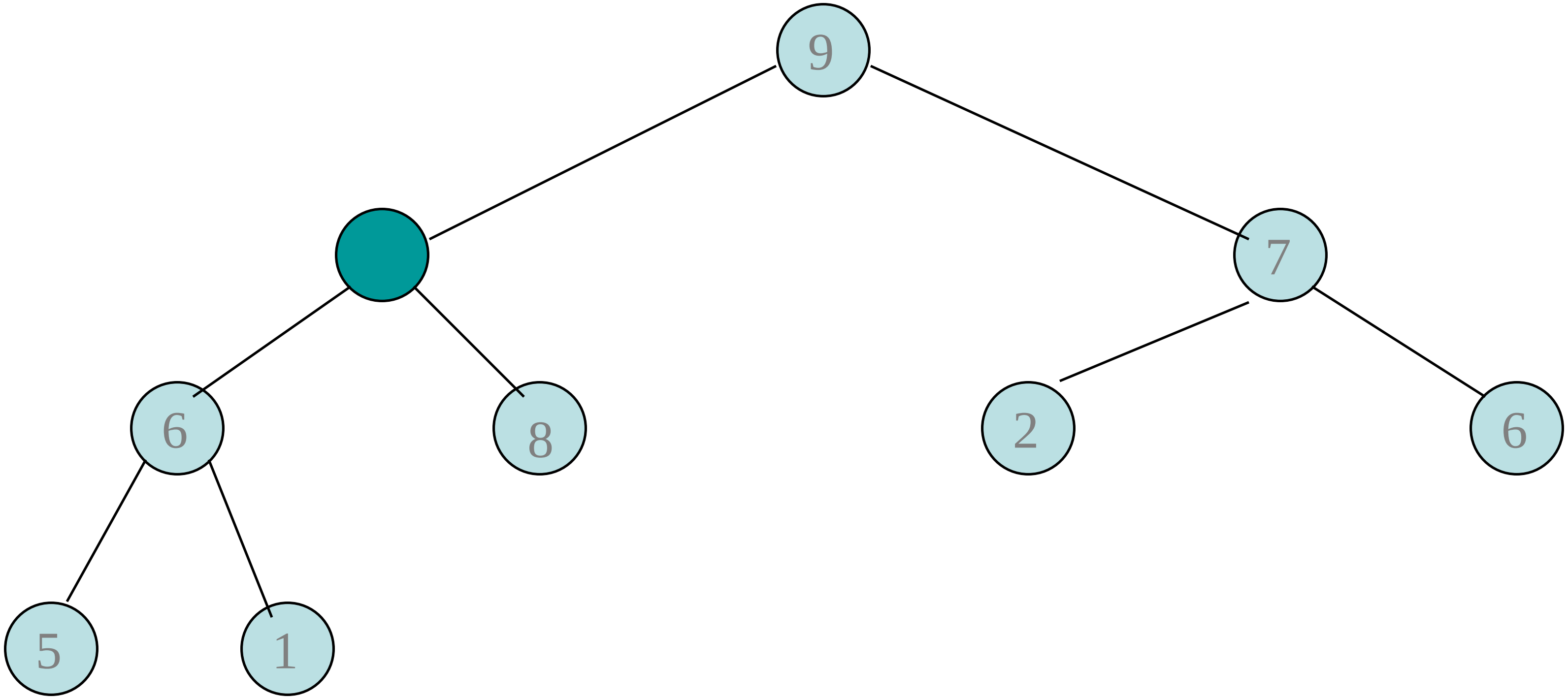
Heap with 9 nodes.

Removing The Max Element



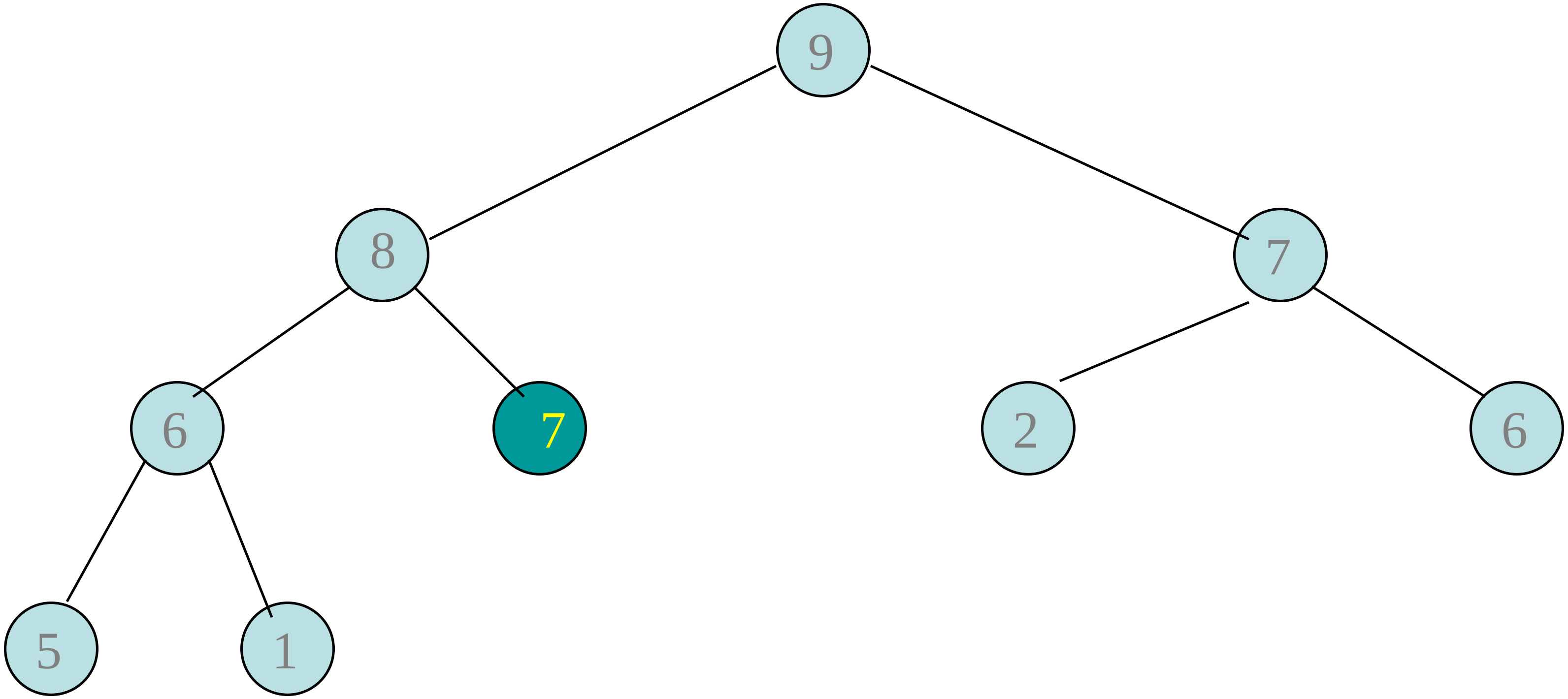
Reinsert 7.

Removing The Max Element



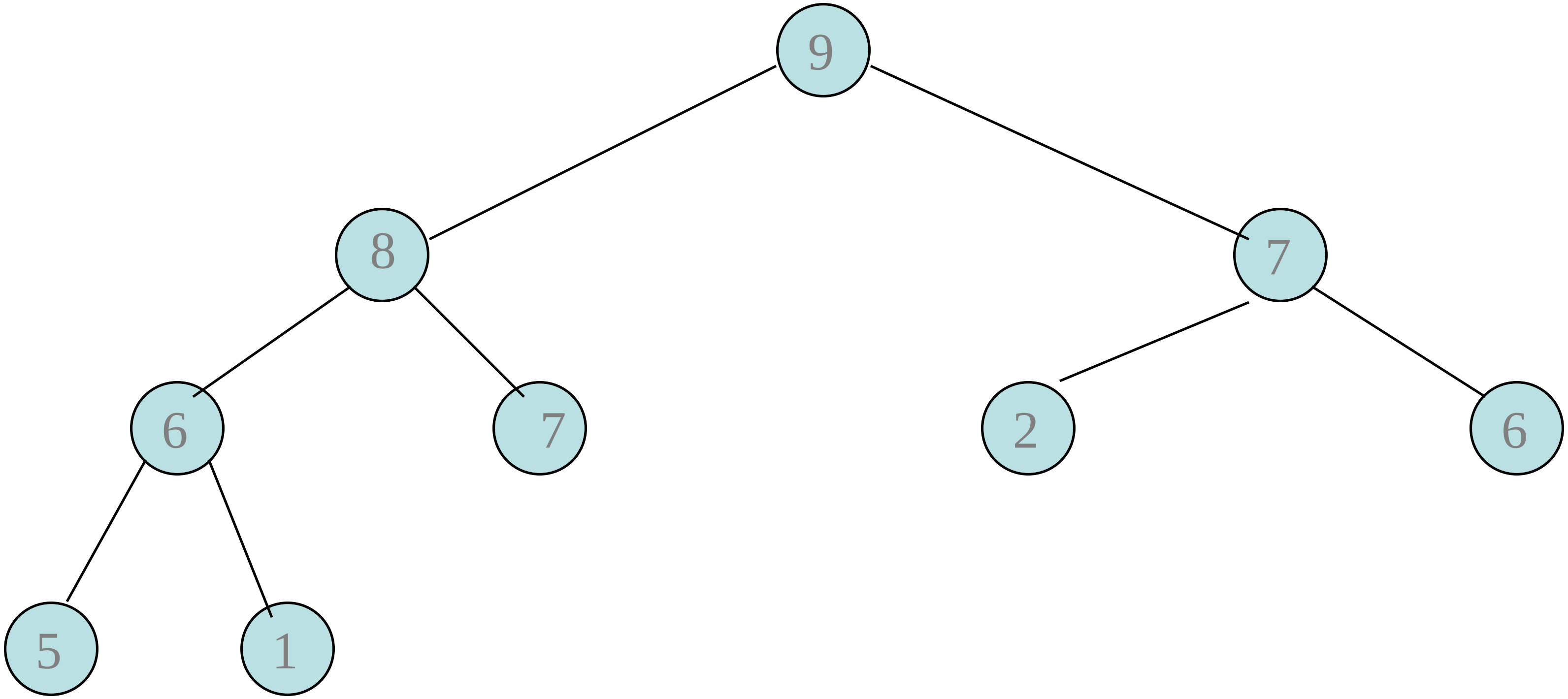
Reinsert 7.

Removing The Max Element



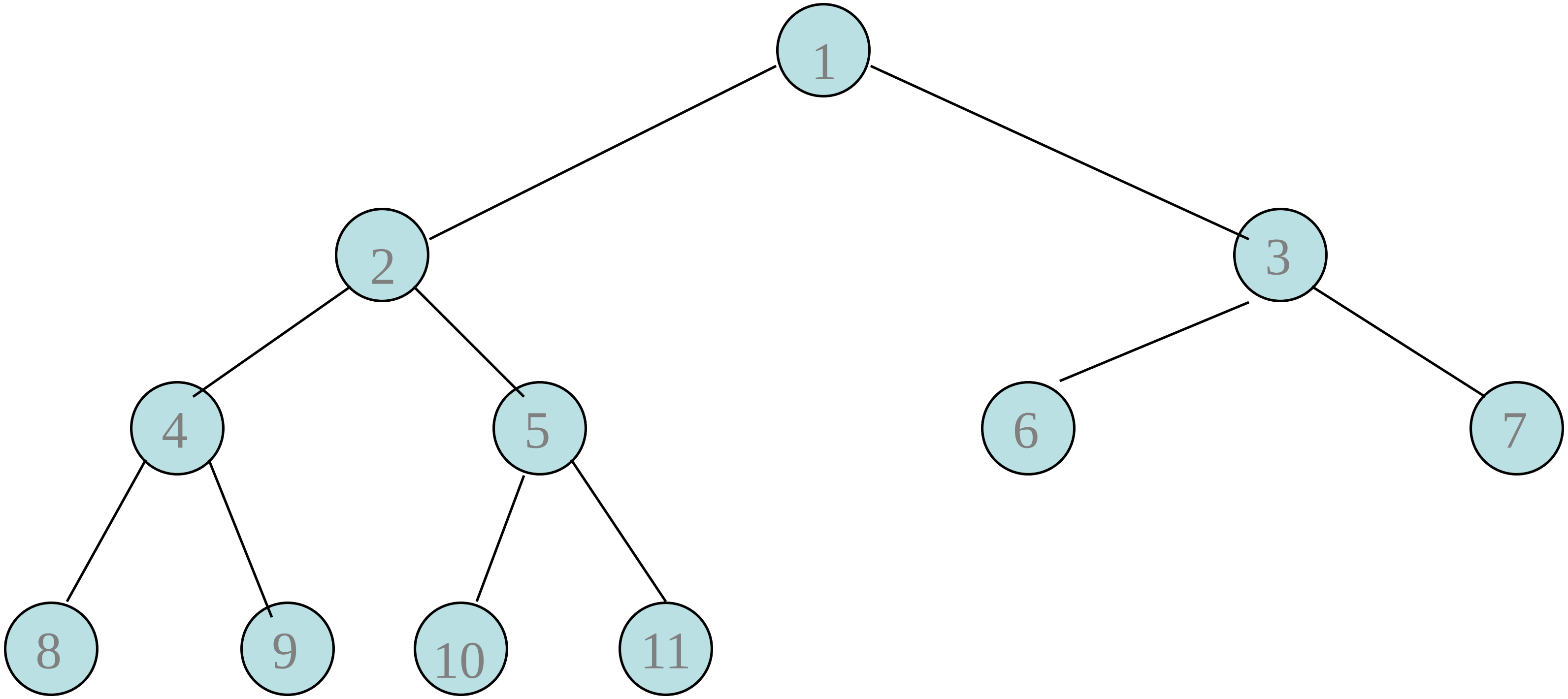
Reinsert 7.

Complexity Of Remove Max Element



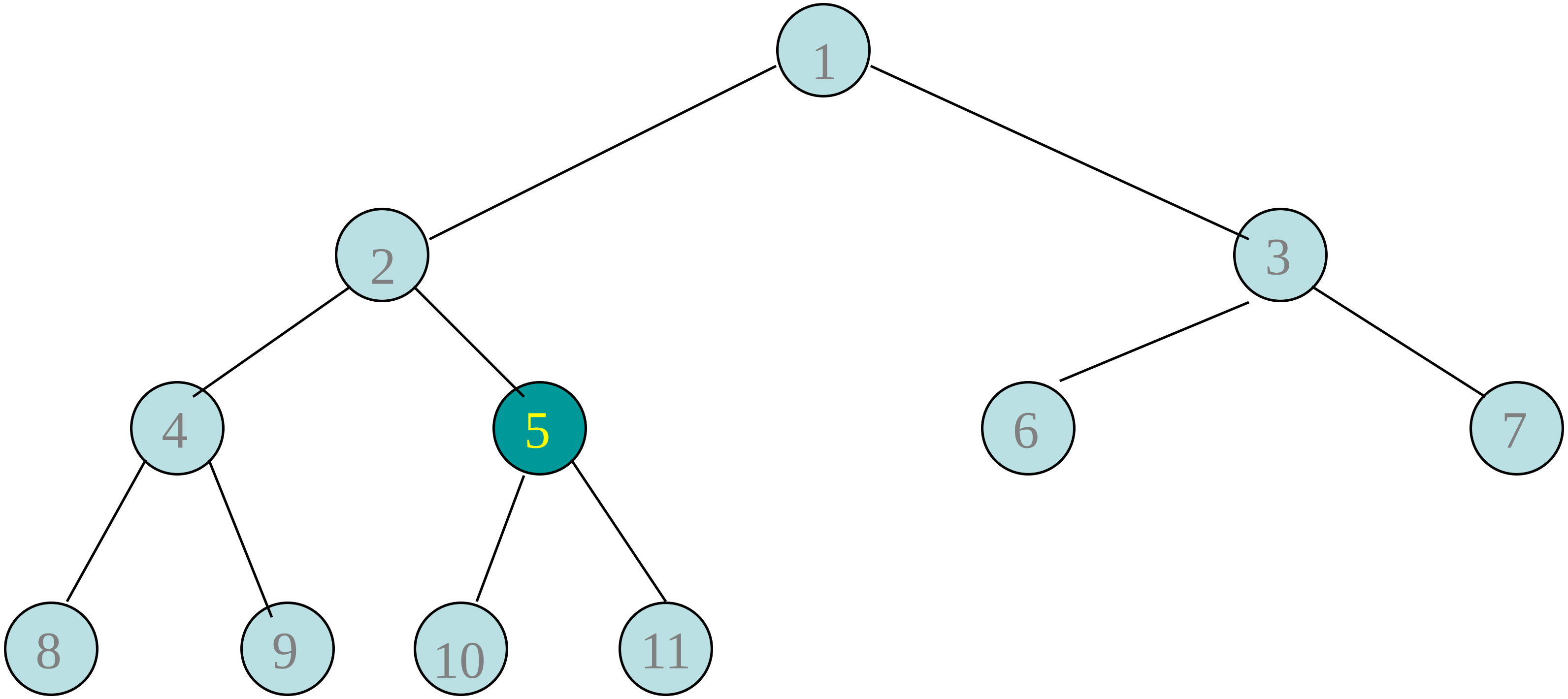
Complexity is O(log n).

Initializing A Max Heap



input array = [-, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

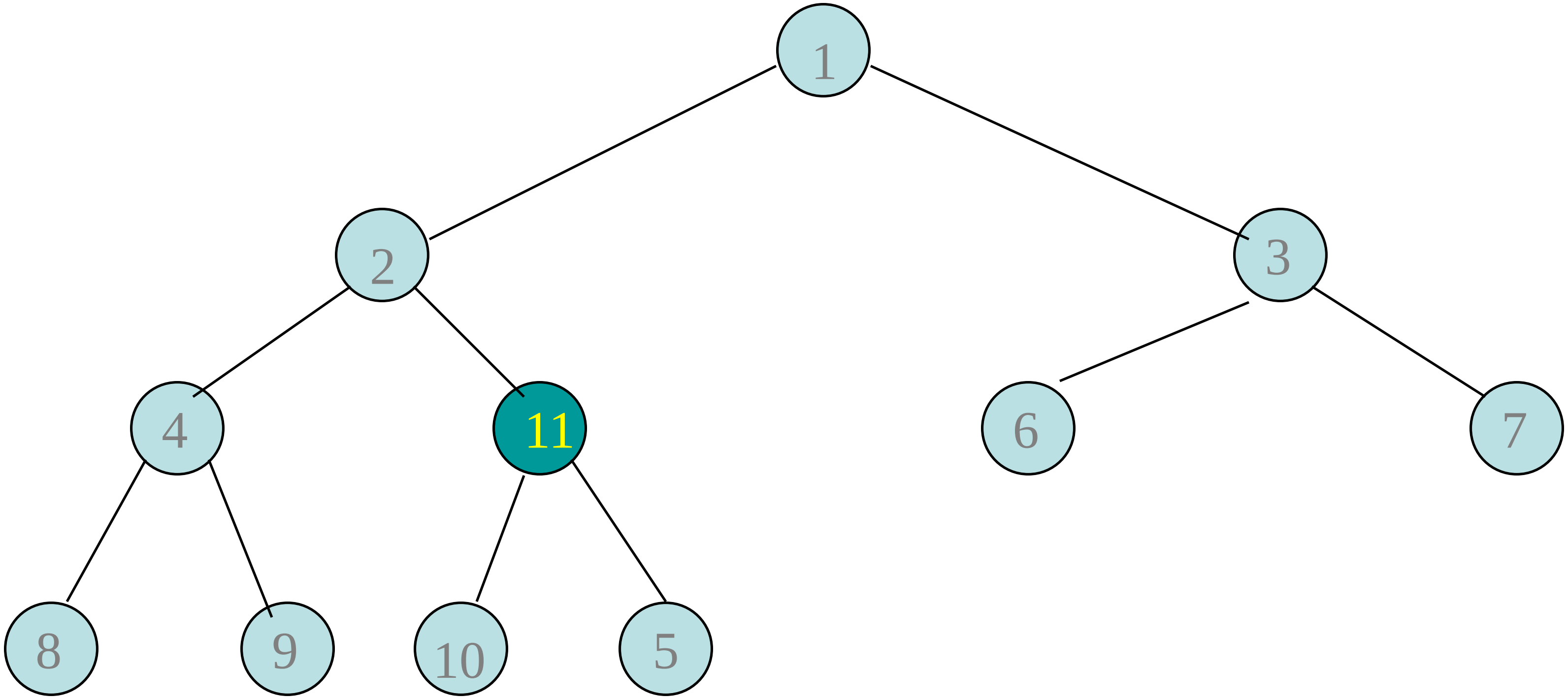
Initializing A Max Heap



Start at rightmost array position that has a child.

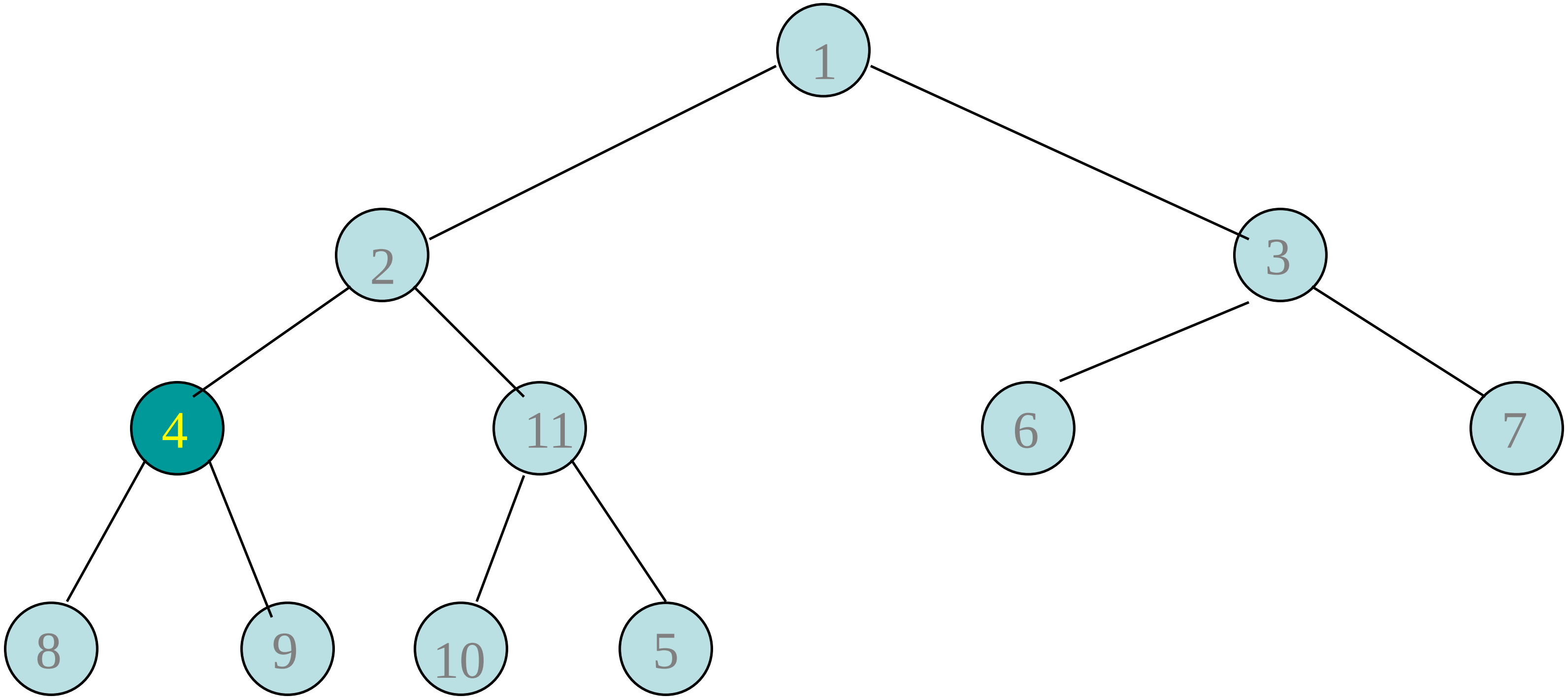
Index is n/2.

Initializing A Max Heap

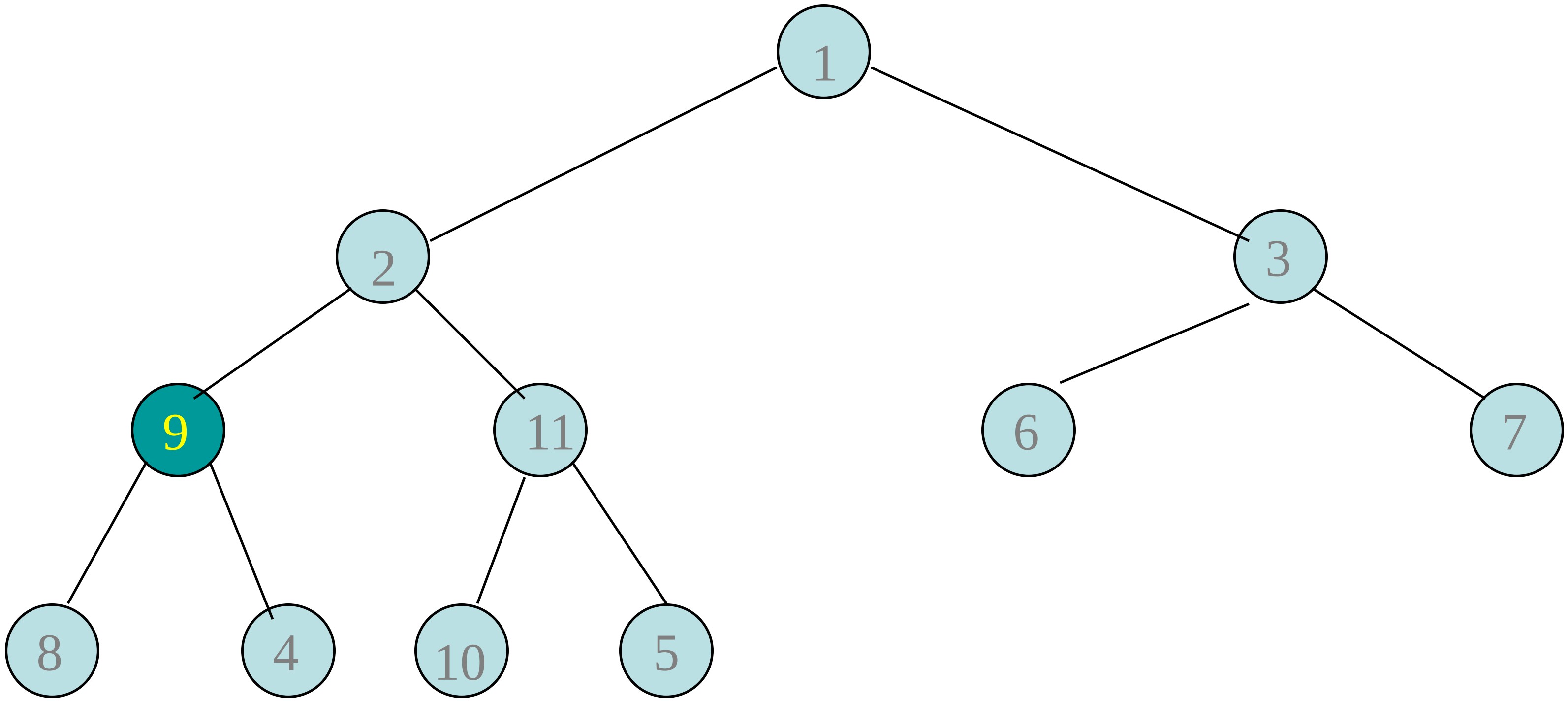


Move to next lower array position.

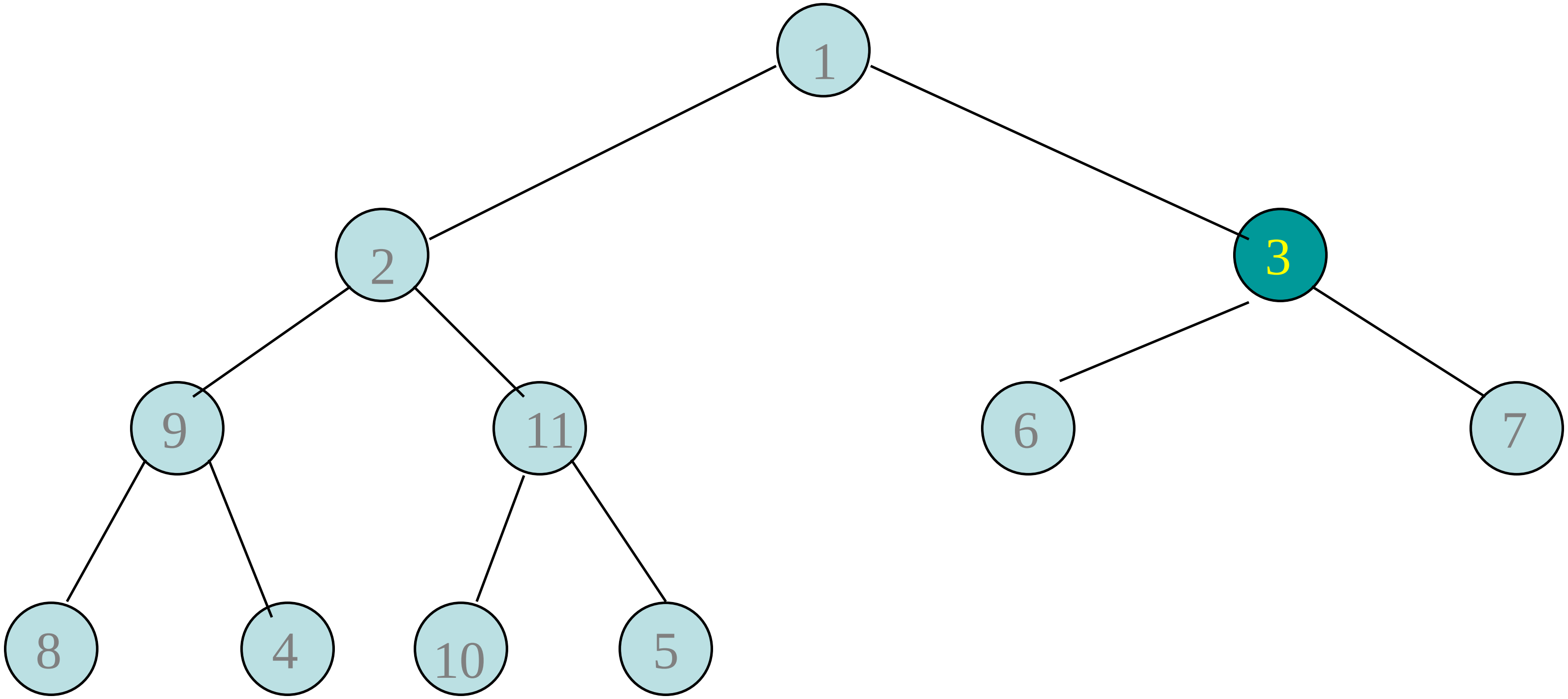
Initializing A Max Heap



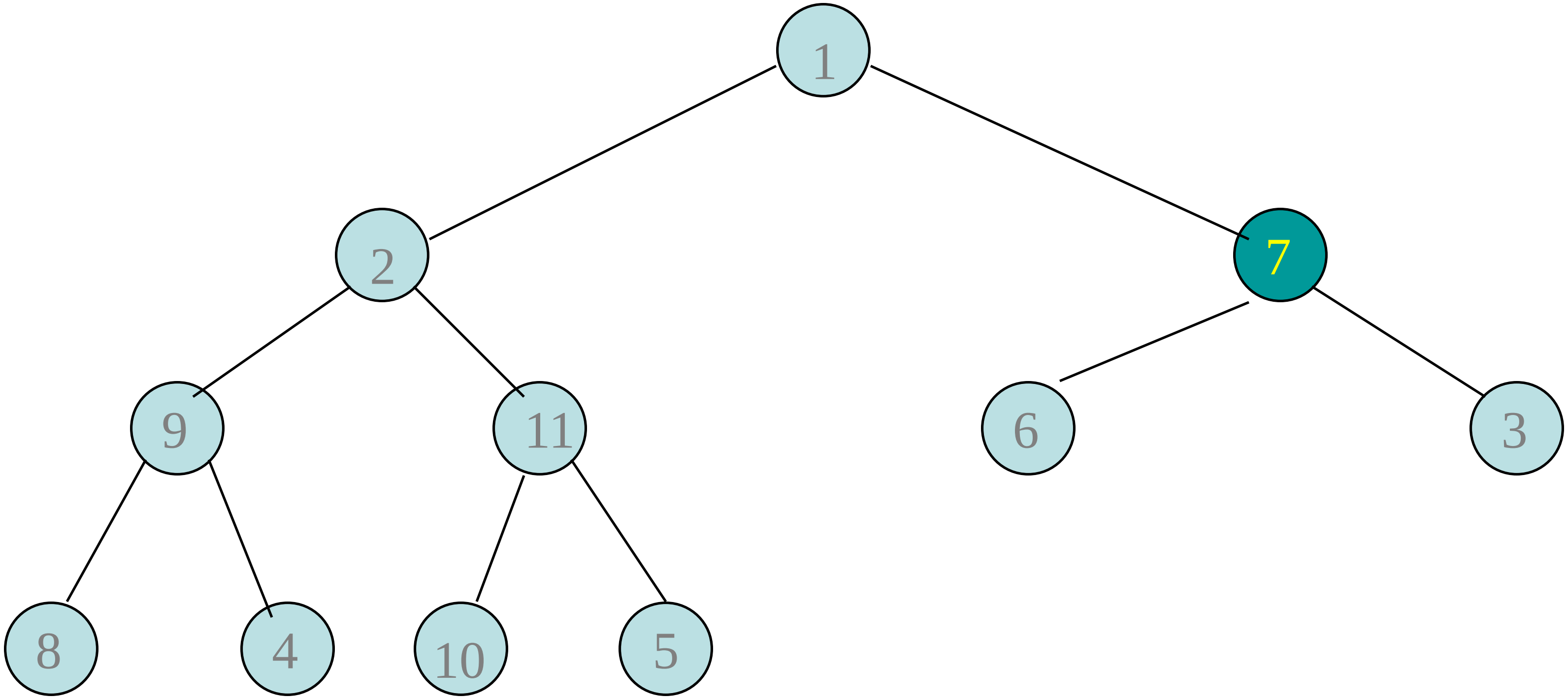
Initializing A Max Heap



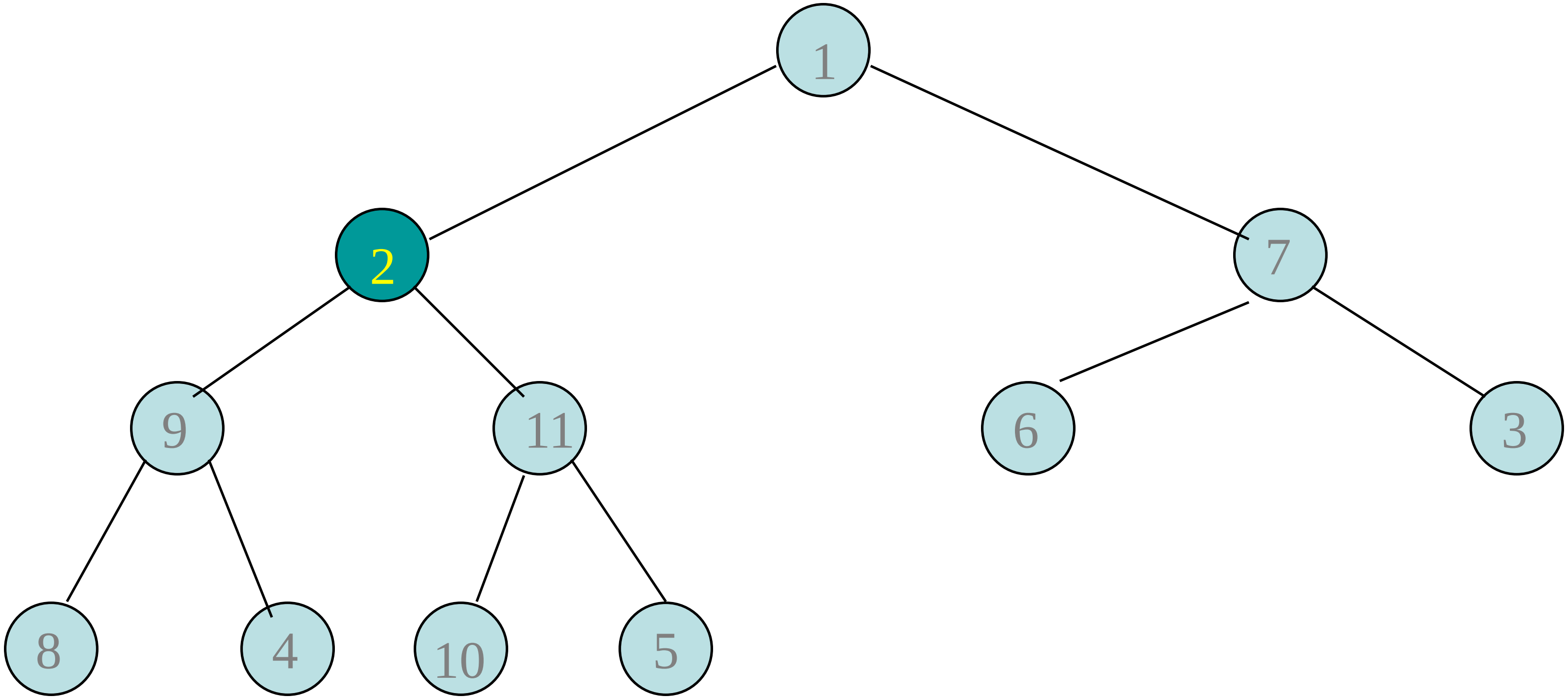
Initializing A Max Heap



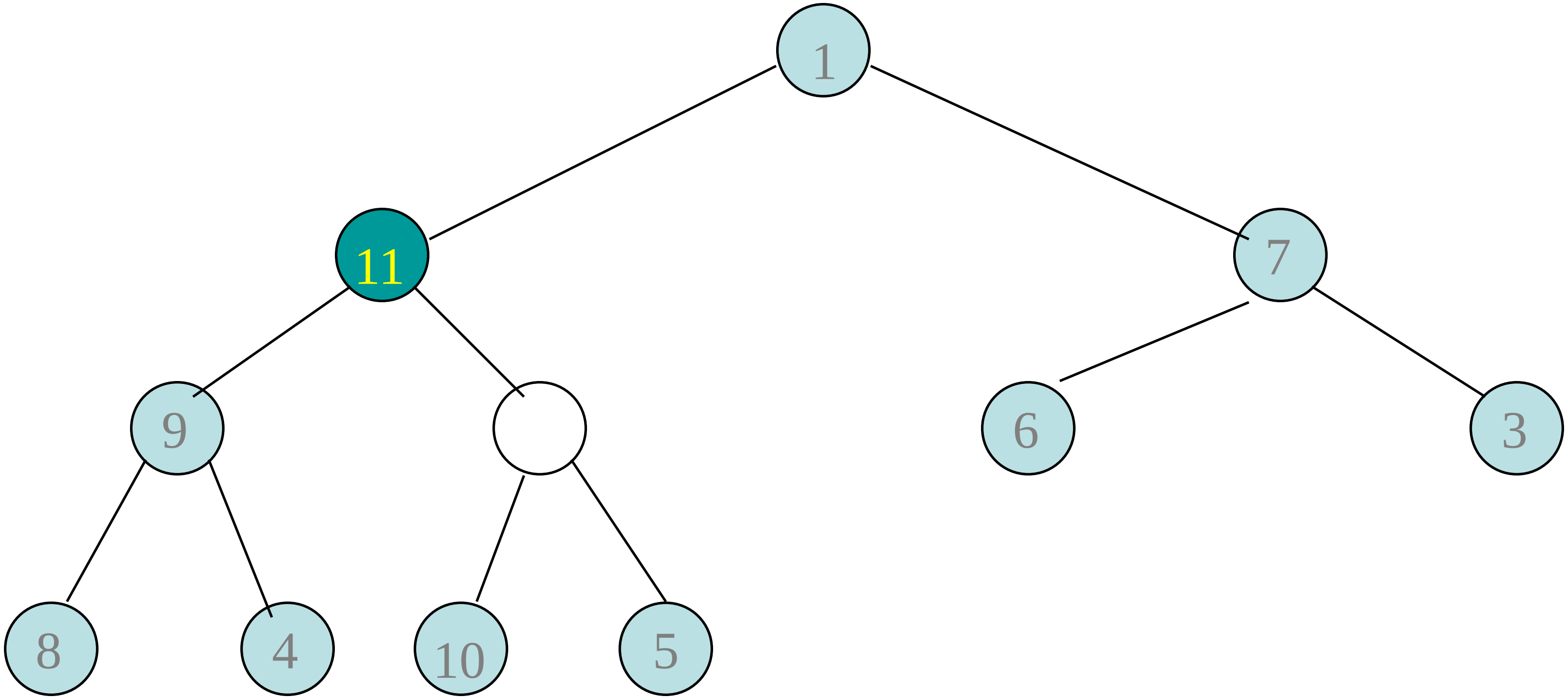
Initializing A Max Heap



Initializing A Max Heap

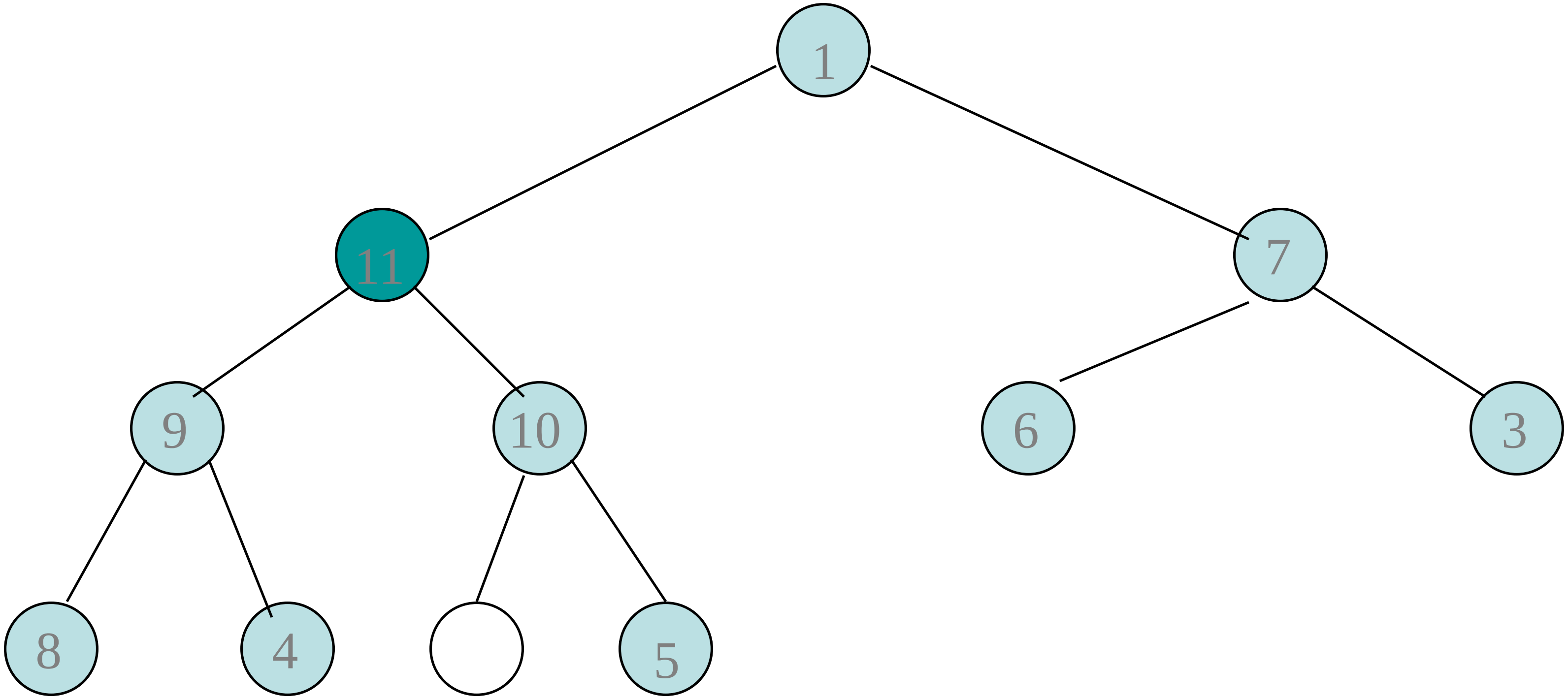


Initializing A Max Heap



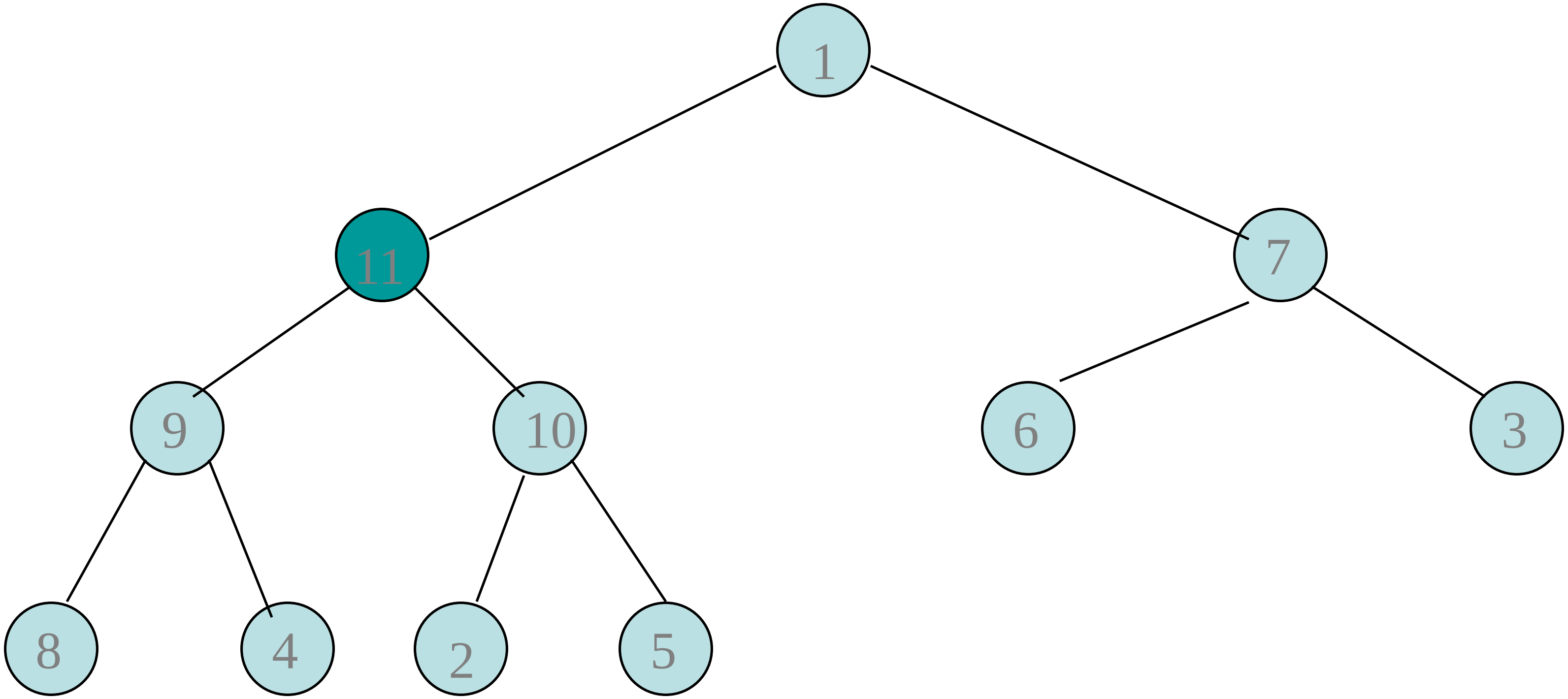
Find a home for 2.

Initializing A Max Heap



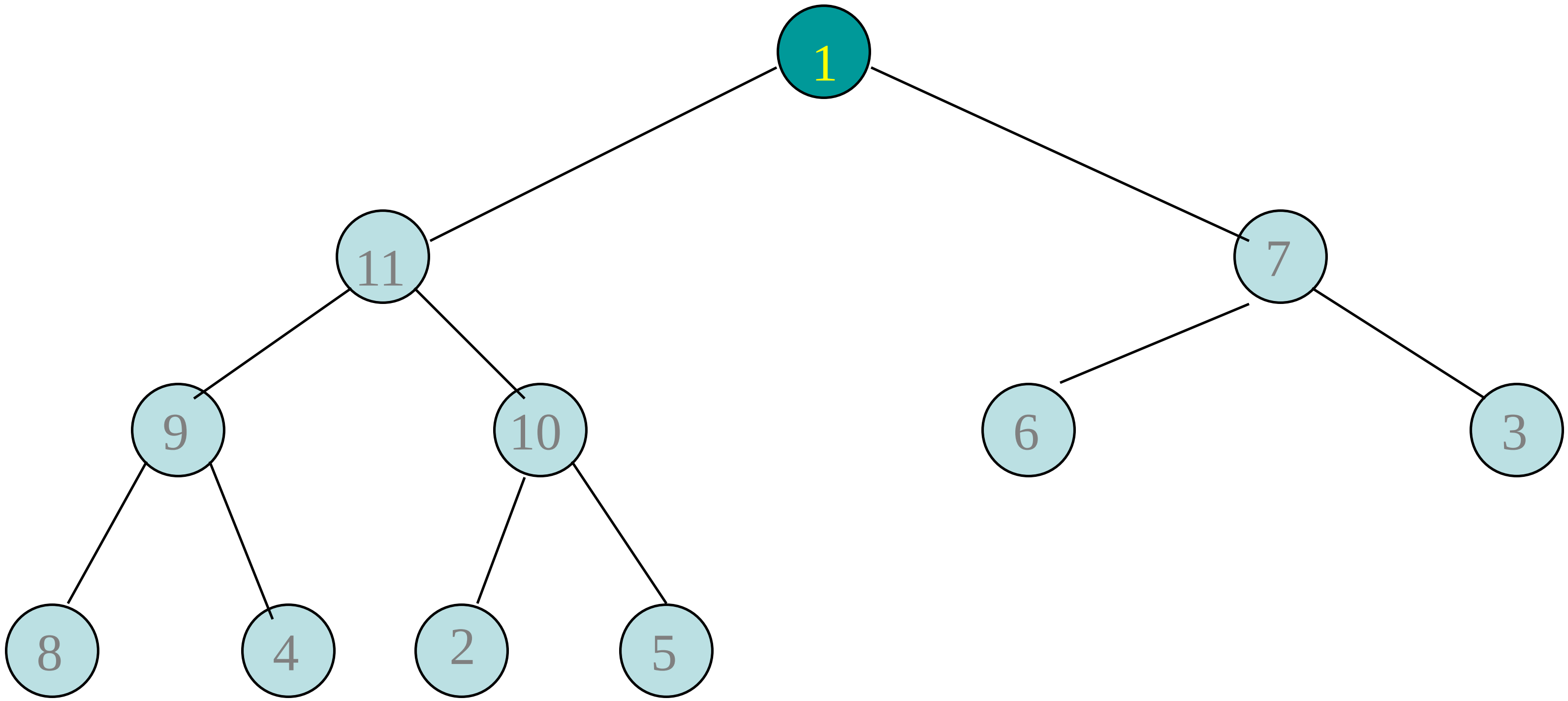
Find a home for 2.

Initializing A Max Heap



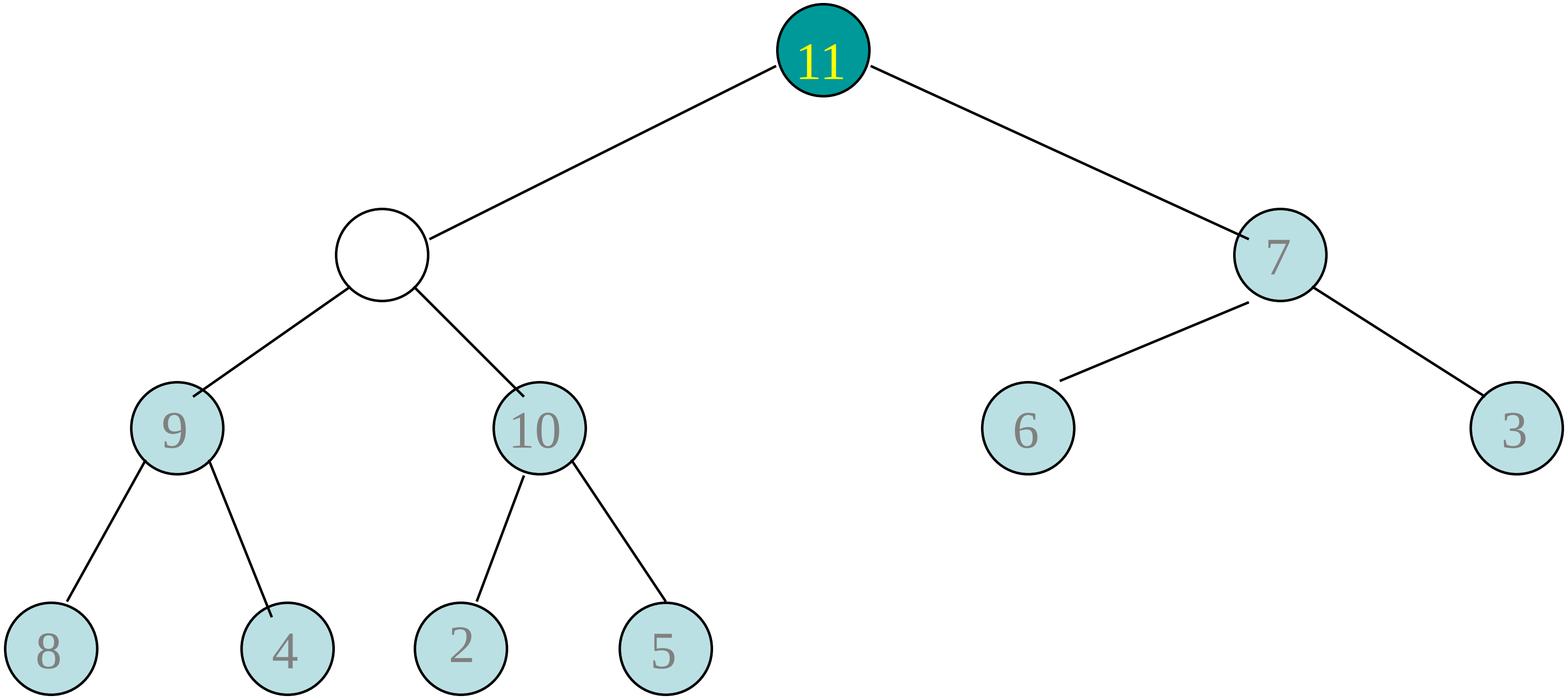
Done, move to next lower array position.

Initializing A Max Heap



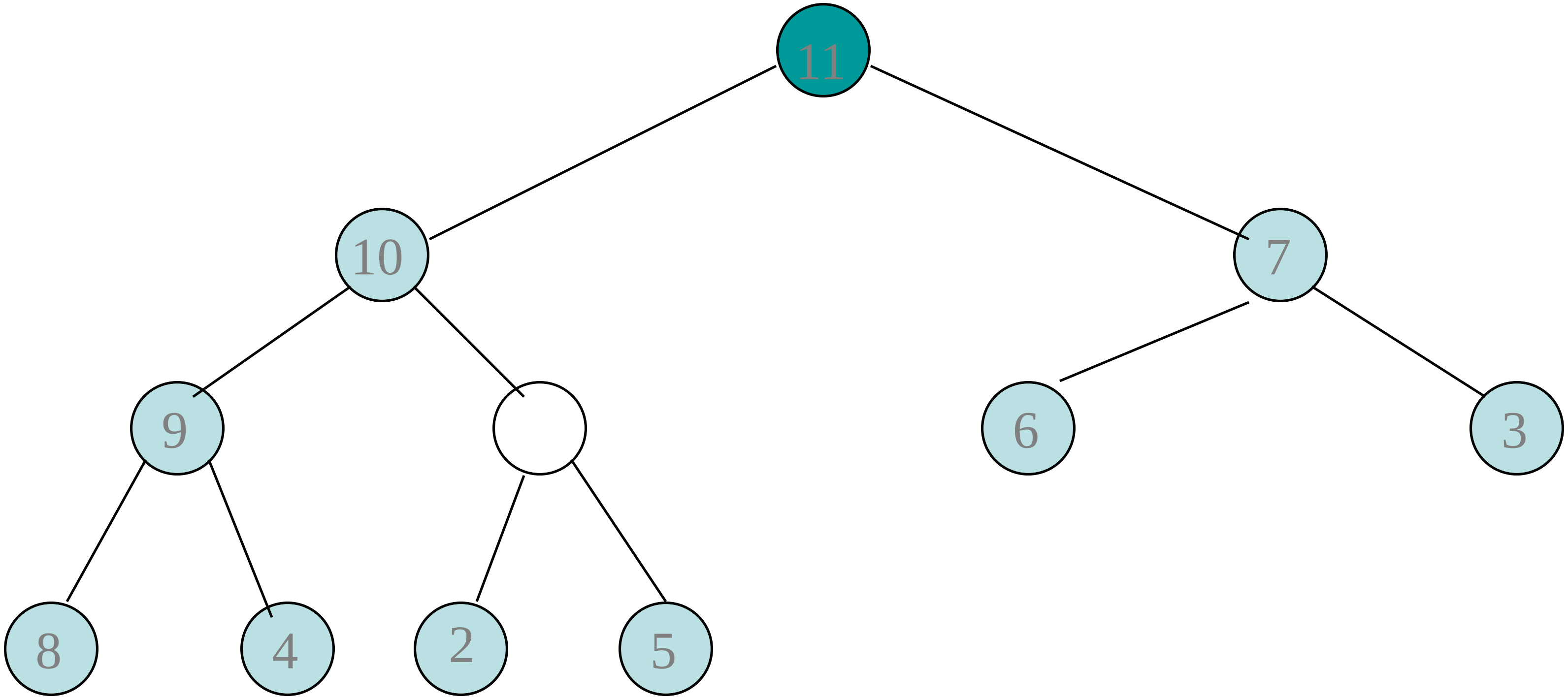
Find home for 1.

Initializing A Max Heap



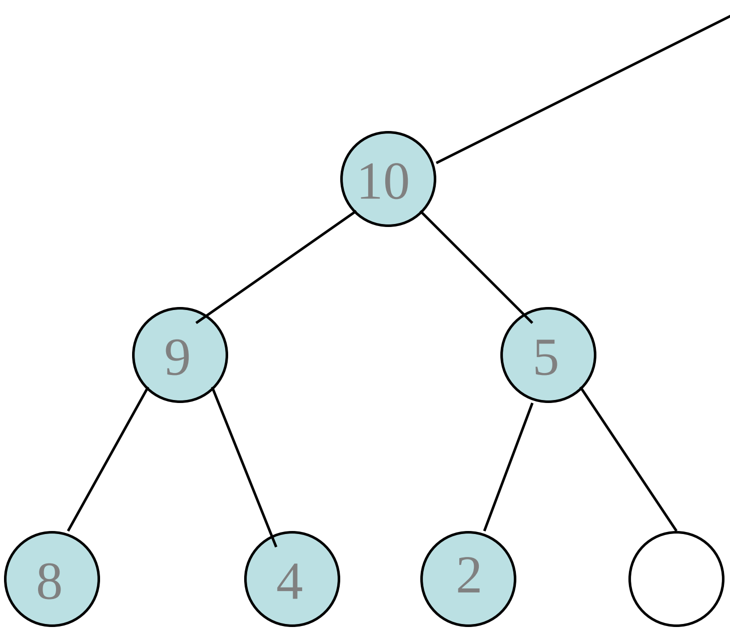
Find home for 1.

Initializing A Max Heap



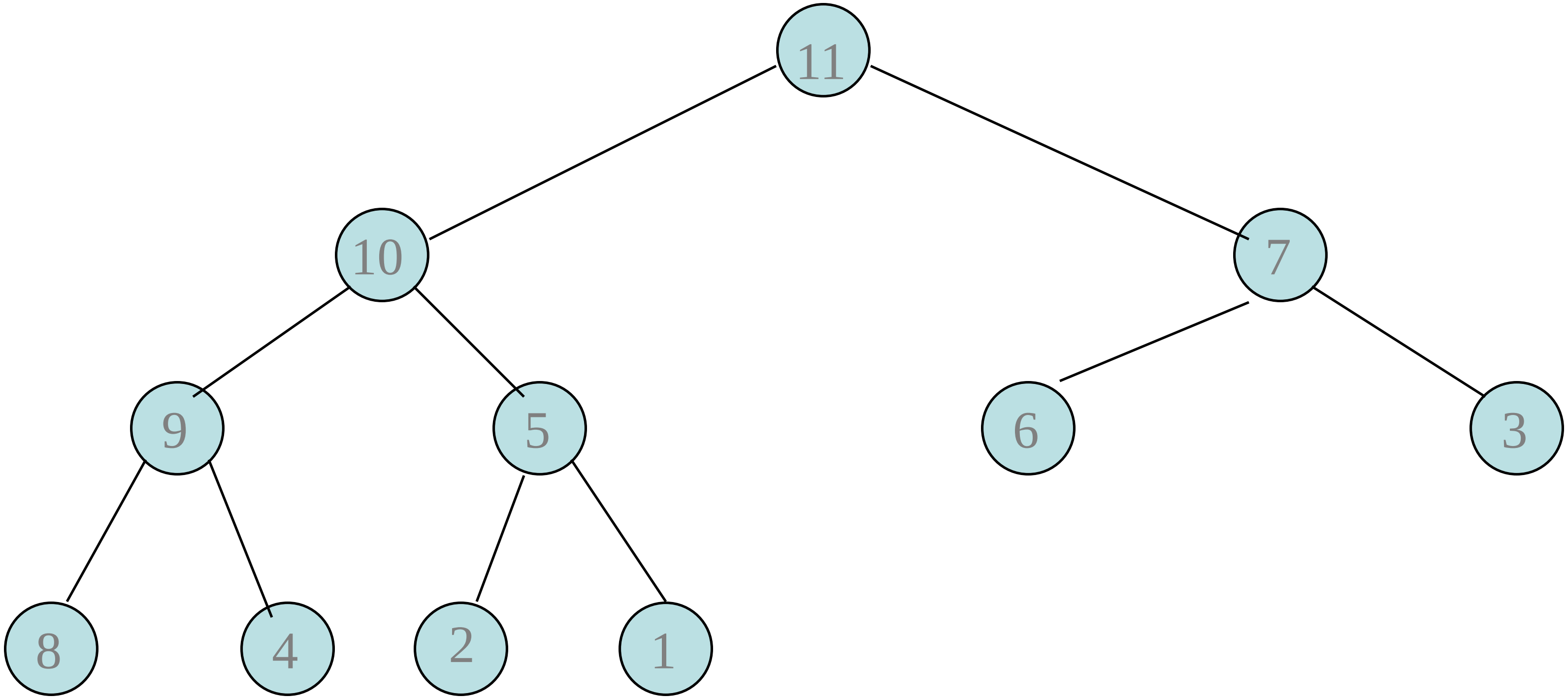
Find home for 1.

Initializing A Max Heap



Find home for 1.

Initializing A Max Heap



Done.