Ceng 111 – Fall 2021
Week 13b

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Stacks in Python (Example - Solution)

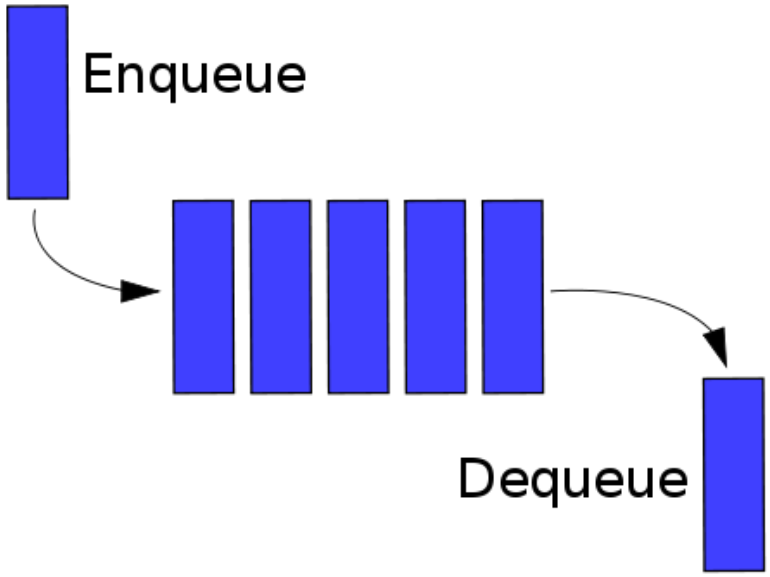
```
def postfix_eval(Exp):
        # Example Exp: "3 4 + 5 6 + *"
        Stack = CreateStack()
        Exp = Exp.split(' ')

        for token in Exp:
                if token.isdigit(): Push(token, Stack)
                else:
                        op2 = Pop(Stack)
                        op1 = Pop(Stack)
                        result = str(eval(op1 + token + op2))
                        Push(result, Stack)

        return Pop(Stack)
```

Queues

- FIFO:
  - First In First Out
- The item that was inserted first is removed first.
- Main operations:
  - Add (enqueue)
  - Remove (dequeue)



Queues in Python

Queue Operation

- Add (enqueue)
- Remove (dequeue)
- Front/Peek
- Is-Empty
- Length

Corresponding Python Op.

- L.append(item)
- L.pop(0)
- L[0]
- L == []
- len(L)

Implementing Queues in Python

```
def CreateQueue():
        """Creates an empty queue"""
        return []

def Enqueue(item, Queue):
        """Add item to the end of Queue"""
        Queue.append(item)

def Dequeue(Queue):
        """Remove and return the item at the front of the Queue"""
        return Queue.pop(0)

def IsEmpty(Queue):
        """Check whether the Queue is empty"""
        return Queue == []

def Front(Queue):
        """Return the value of the current front item without removing it"""
        return Queue[0]
```

Previously on CENG111!

Queues: Formal Definition

$add(item, queue)$  $item \boxplus queue$

- $new() \rightarrow \varnothing$
- $front(\xi \boxplus \varnothing) \rightarrow \xi$
- $front(\xi \boxplus Q) \rightarrow front(Q)$
- $remove(\xi \boxplus \varnothing) \rightarrow \varnothing$
- $remove(\xi \boxplus Q) \rightarrow \xi \boxplus remove(Q)$
- $isempty(\varnothing) \rightarrow$TRUE
- $isempty(\xi \boxplus Q) \rightarrow$ FALSE

Today

- Abstract data types
  - Priority queue
  - Tree

Administrative Notes

- THE3:
  - Deadline: 16 January.
- Final:
  - 5 Feb December, Saturday, 13:30

Priority Queue

- Similar to Queue except that the items in a queue has a priority value based on which they are kept in order!
- Operations:
  - insert(item, priority) → Push item with the given priority
  - Highest() → The item in the queue that has the highest priority
  - Deletehighest() → Delete the item that has the highest priority
  - Is-Empty
  - Length

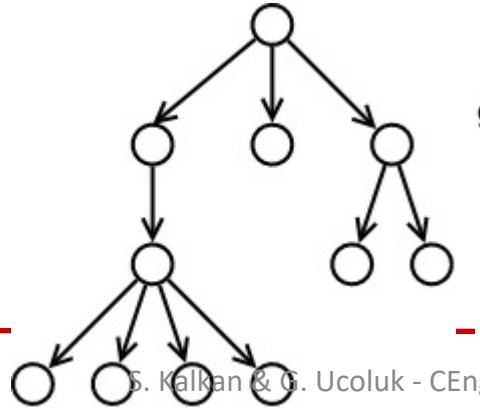
Priority Queues in Python

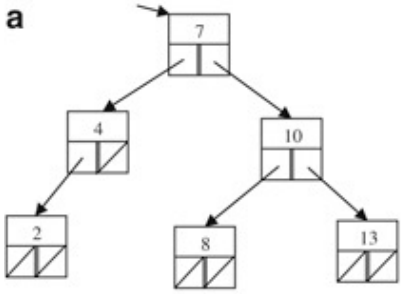
Priority Queue Operation	Corresponding Python Op.
Insert	L.append((item, priority))
Highest	Write a function that finds the max
Delete highest	Write a function that finds the max and deletes it
Is-Empty	L == []
Length	len(L)

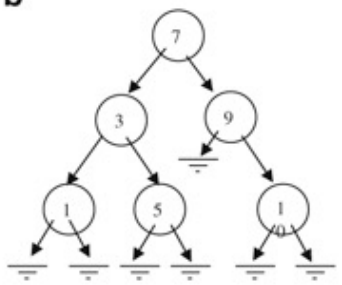
$insert(item, PQ)$  $item \curvearrowright PQ$

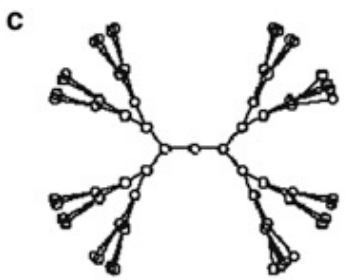
- $new() \rightarrow \langle\rangle$
- $highest(\xi \curvearrowright \langle\rangle) \rightarrow \xi$
- $highest(\xi \curvearrowright PQ) \rightarrow$
  if $priority(\xi) > priority(highest(PQ))$ **then** ξ **else** $highest(PQ)$
- $deletehighest(\xi \curvearrowright \langle\rangle) \rightarrow \langle\rangle$
- $deletehighest(\xi \curvearrowright PQ) \rightarrow$
  if $priority(\xi) > priority(highest(PQ))$ **then** PQ
  else $\xi \curvearrowright deletehighest(PQ)$
- $isempty(\langle\rangle) \rightarrow$ TRUE
- $isempty(\xi \curvearrowright PQ) \rightarrow$ FALSE

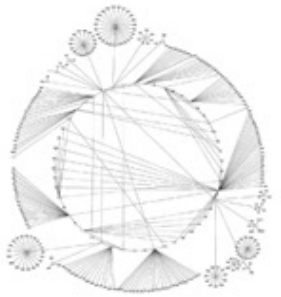
Trees

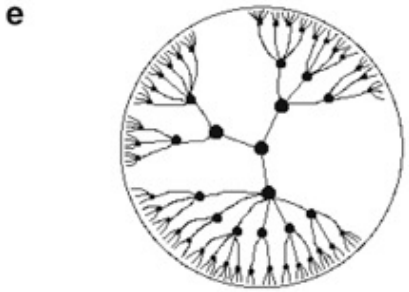


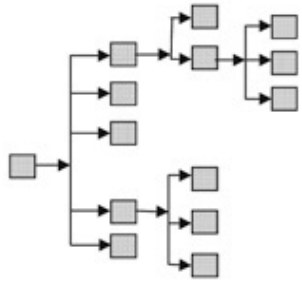

a
7
4
10
2
8
13

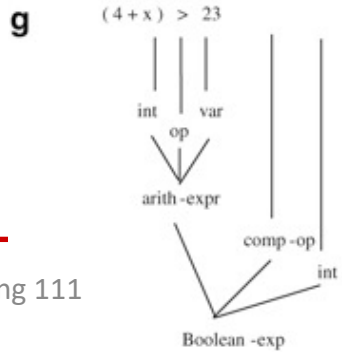

b
7
3
9
1
5
1

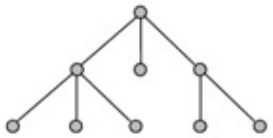

c


d

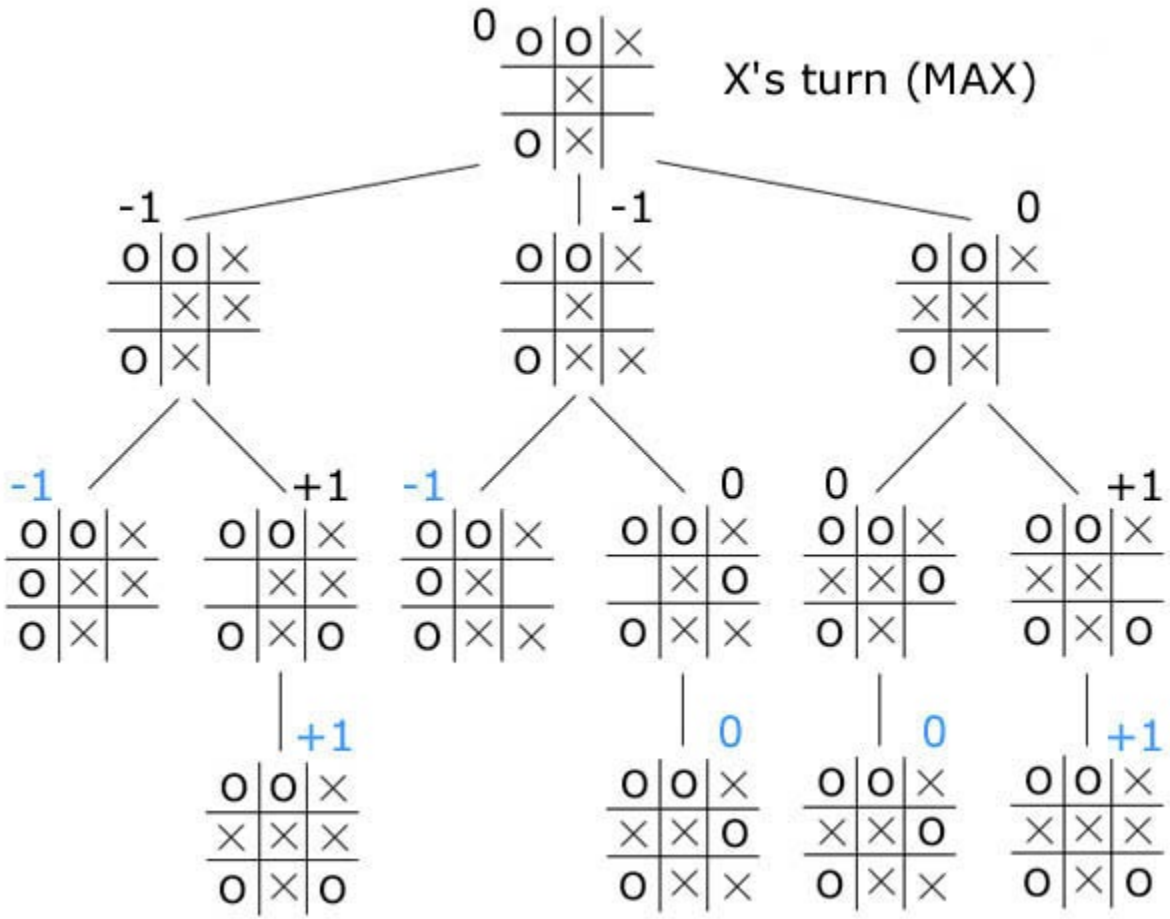

e


f


g
(4 + x) > 23
int
var
op
arith -expr
comp -op
int
Boolean -exp


h

Example for Trees: Decision/Game Tree



Properties of Trees

- A tree is composed of nodes.
- A node can have either no branches, two branches or more than two branches.
- Binary tree: a tree where nodes have two branches.
- The depth of a tree:
  - The number of levels in the tree.

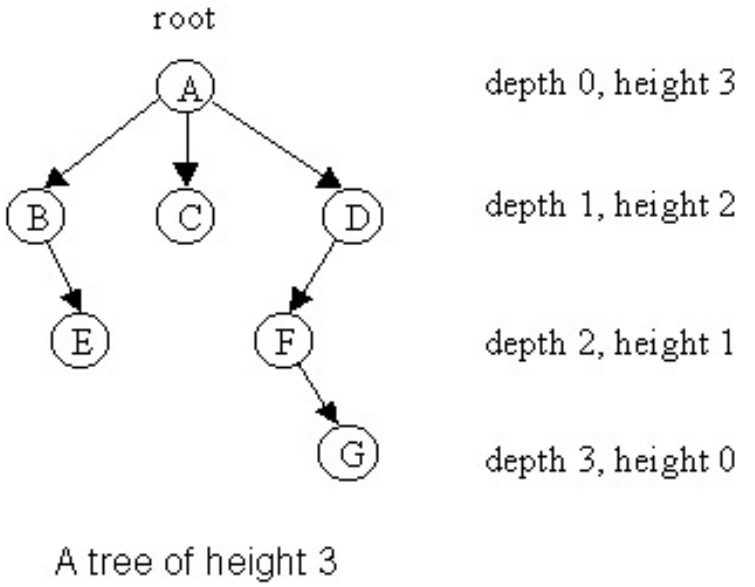
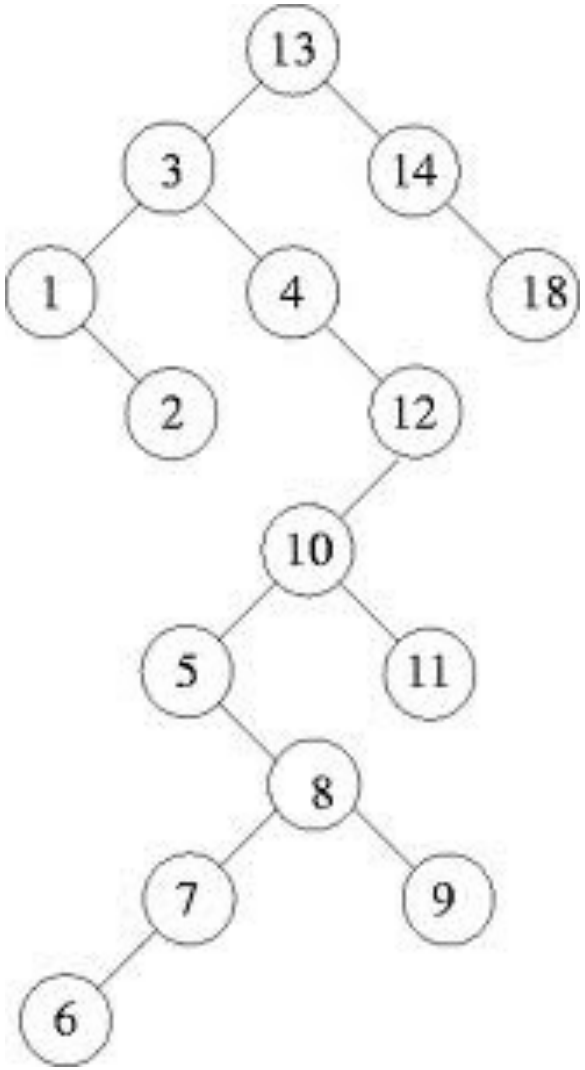


Figure: https://condor.depaul.edu/ntomuro/courses/402/notes/heap.html

Binary Search Tree

- The nodes in the left branch of a node have less value than the node.
- The nodes in the right branch of a node have more value than the node.



How can we represent Trees in Python?

- Nested Lists

vs.

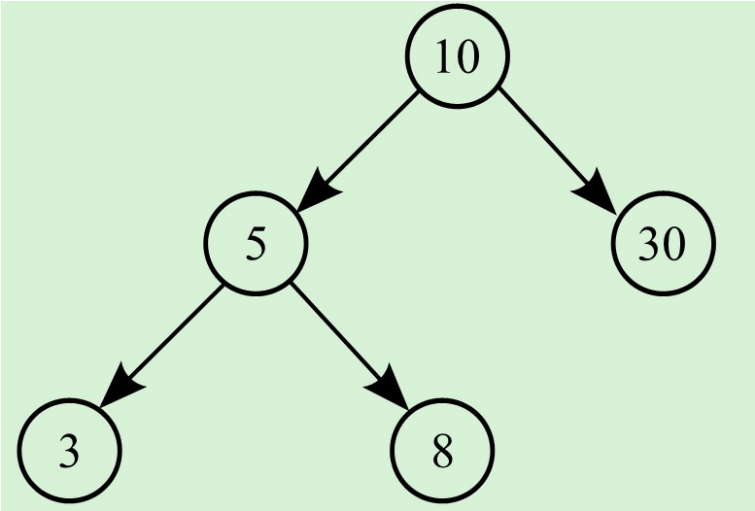
- Nested Tuples

- Tuples / Lists

vs.

- Dictionaries

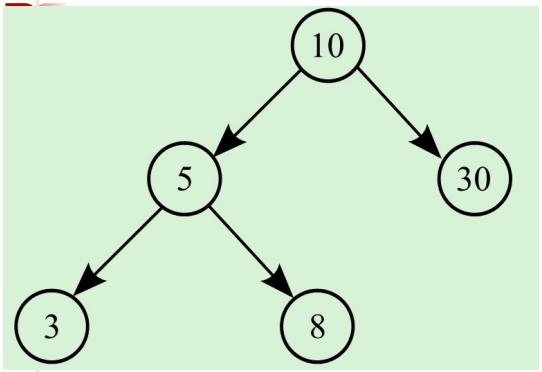
Now, let us see how we can represent Trees in Python



- Using Lists:
  - `[10, [5, [3, [], []], [8, [], []]], [30, [], []]].`
  - `[10, [5, [3, '#', '#'], [8, '#', '#']], [30, '#', '#']],` where the empty branches are marked with `'#'`.
  - `[10, [5, [3], [8]], [30]].`

Now, let us see how we can represent Trees in Python

- Using dictionaries




```
Tree = \
   { 'value' : 10, \
     'left' : {'value': 5, \
               'left': {'value': 3, \
                        'left': {}, \
                        'right': {}},\
               'right': {'value': 8, \
                        'left': {}, \
                        'right': {}}}, \
     'right' : {'value': 30, \
                'left': {}, \
                'right': {}}\
   }
```

Tree operations

- datum()
- isempty()
- left()
- right()
- createNode()

```python
# Return the value stored in the node
def datum(T):
    return T[0] # Assume nested list rep.
```

```python
# Check whether the Tree is empty
def isempty(T):
    return T == [] # Assume nested list rep.
```

```python
# Get the left branch
def left(T):
    #TODO: Throw exception if the tree is empty
    return T[1] # Assume nested list rep.
```

```python
# Get the right branch
def right(T):
    #TODO: Throw exception if the tree is empty
    return T[2] # Assume nested list rep.
```

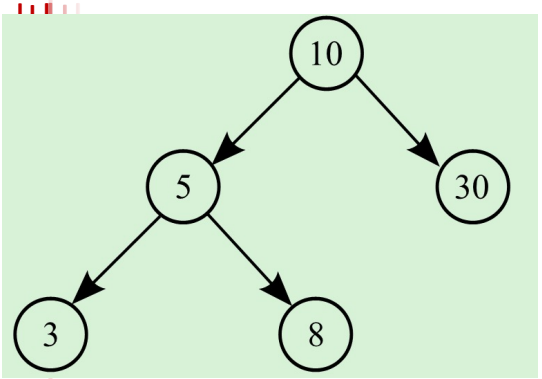
```python
# Create a node
def createNode(datum, left=[], right=[]):
    return [datum, left, right]
```

This creates aliasing ➔
Use the following:

```python
def newNode(datum, left = None, right = None):
    return [datum, left if left else [], right if right else []]
```

Traversing Trees

1. Pre-order Traversal



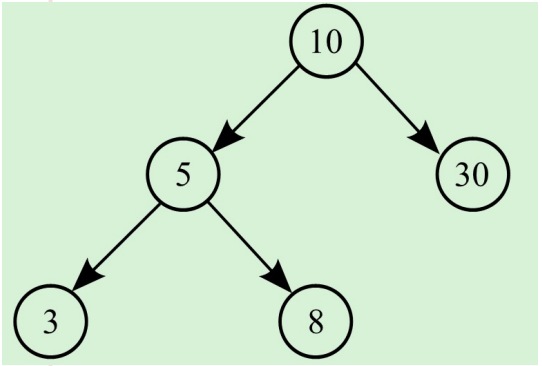

```
[10, [5, [3, [], []], [8, [], []]], [30, [], []]].
```

```
1  def preorder_traverse(T):
2      if isempty(T):
3          return
4
5      print datum(T)
6      preorder_traverse(left(T))
7      preorder_traverse(right(T))
```

```
10 5 3 8 30
```

Traversing Trees

2. In-order Traversal

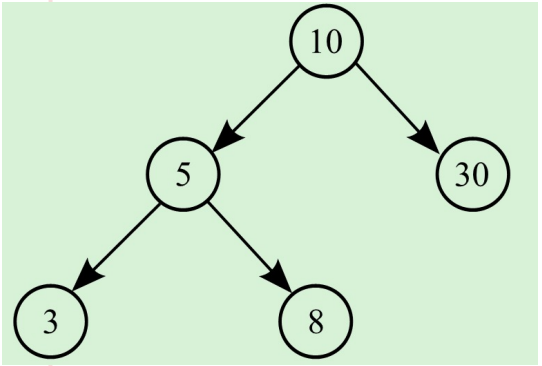



```
1  def inorder_traverse(T):
2      if isempty(T):
3          return
4
5      inorder_traverse(left(T))
6      print datum(T)
7      inorder_traverse(right(T))
```

```
3 5 8 10 30
```

Traversing Trees

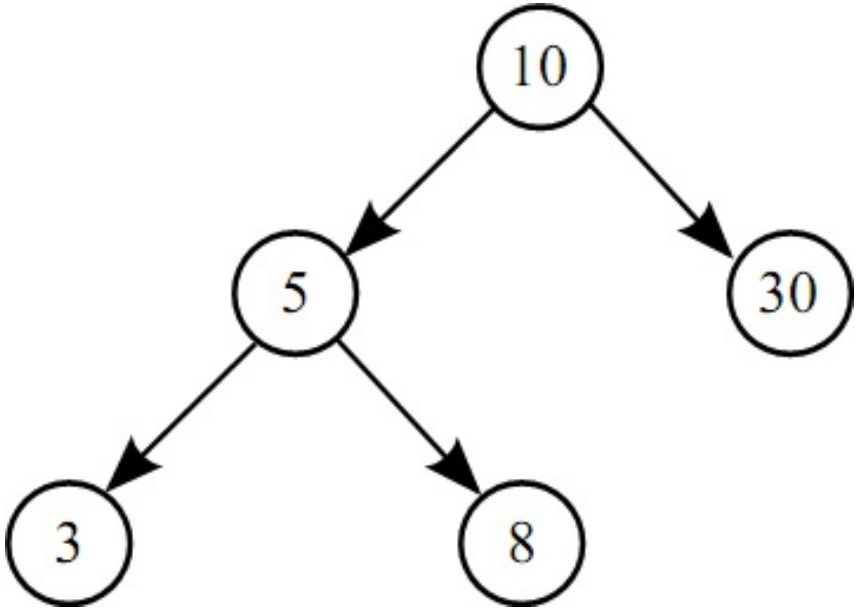
3. Post-order Traversal




```
1  def postorder_traverse(T):
2      if isempty(T):
3          return
4
5      postorder_traverse(left(T))
6      postorder_traverse(right(T))
7      print datum(T)
```

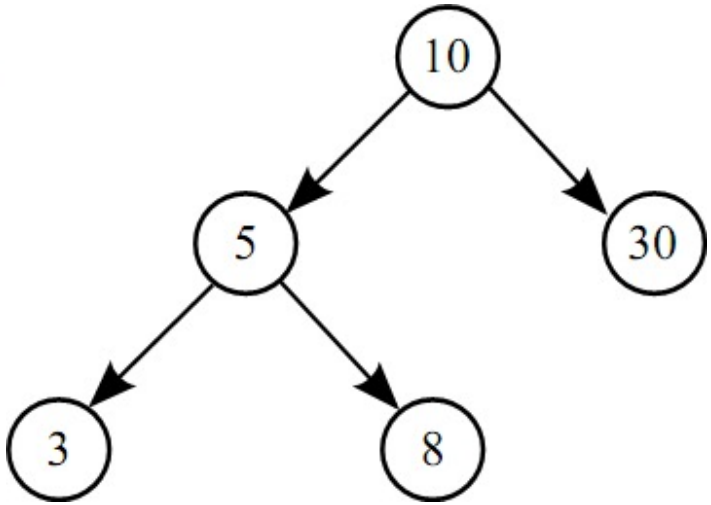
```
3 8 5 30 10
```

Binary **Search** Trees



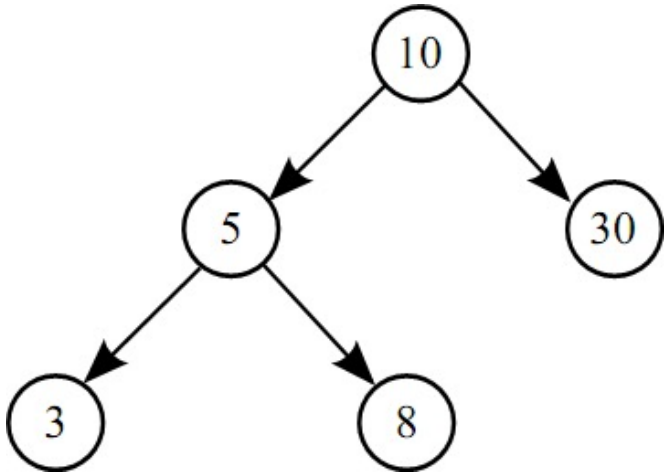
Binary Search Trees: An example


```
1  def search_tree(T, value):
2      '''Search 'value' in binary search tree'''
3      if isempty(T):
4          return False
5      elif datum(T) == value:
6          return True
7      elif value < datum(T):
8          return search_tree(left(T), value)
9      else:
10         return search tree(right(T), value)
```



Binary Search Trees: Insertion

```
1  def insert_node(T, value):
2      '''Insert a node with value to
3              the binary search tree'''
4      if isempty(T):
5          T.extend(createNode(value))
6      elif datum(T) == value: #duplicate
7          return
8      elif value < datum(T):
9          insert_node(left(T), value)
10     else:
11         insert_node(right(T), value)
```




```
# The following can construct the tree on the right
Tree = []
insert_node(Tree, 10)
insert_node(Tree, 30)
insert_node(Tree, 5)
insert_node(Tree, 3)
insert_node(Tree, 8)
```