Data structures and algorithms

Tutorial 6

Amr Keleg

Faculty of Engineering, Ain Shams University

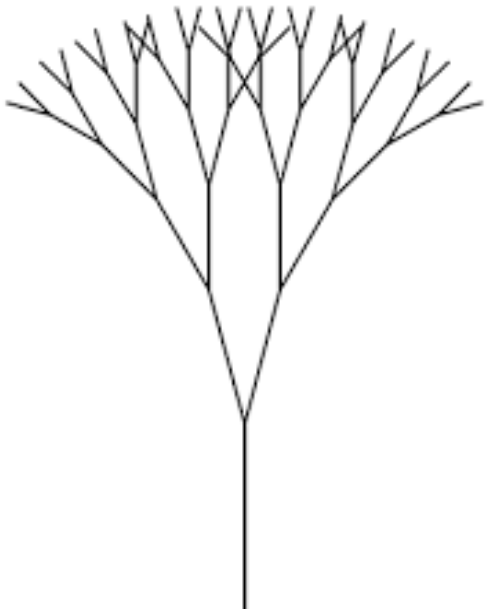
March 27, 2020

Contact: amr_mohamed@live.com

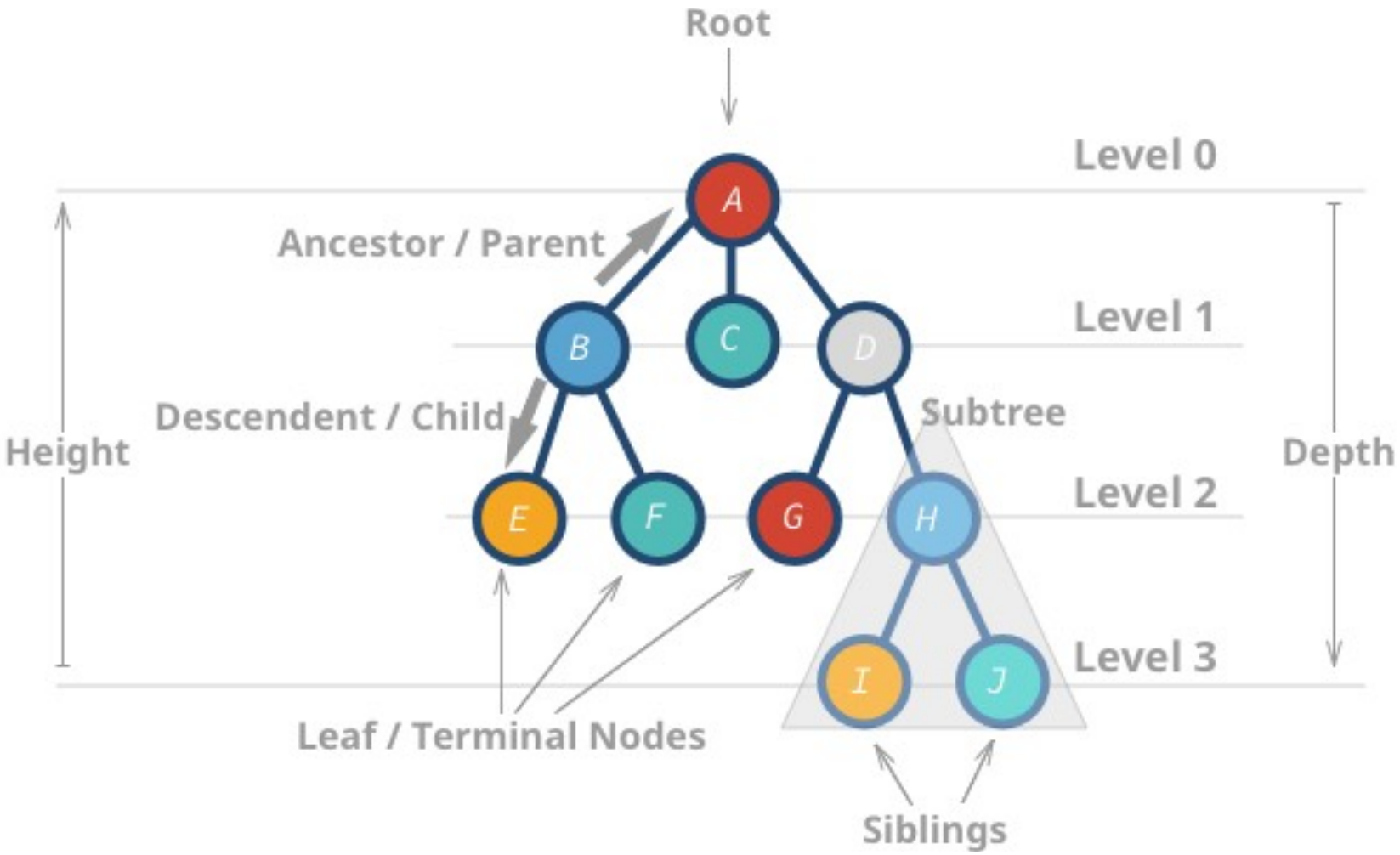
Outline

Outline

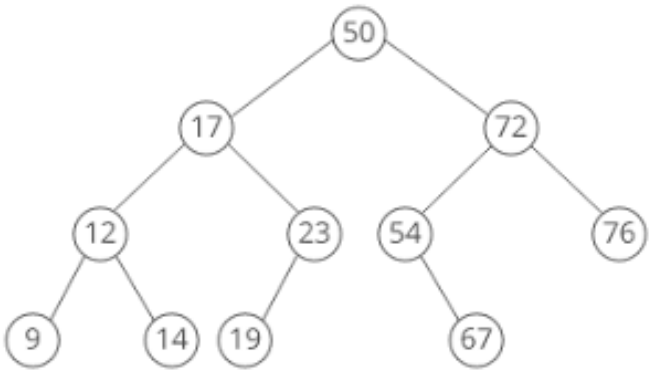
In real-life, a tree has roots and leaves.

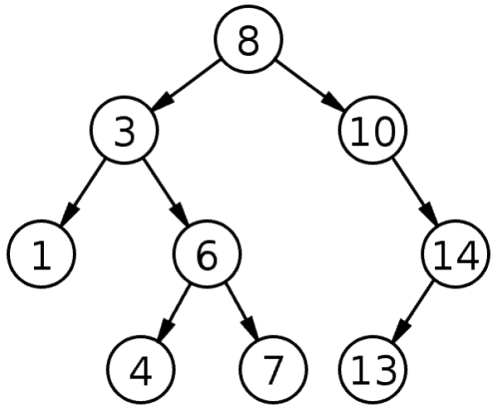


In computer science, the tree is inverted.

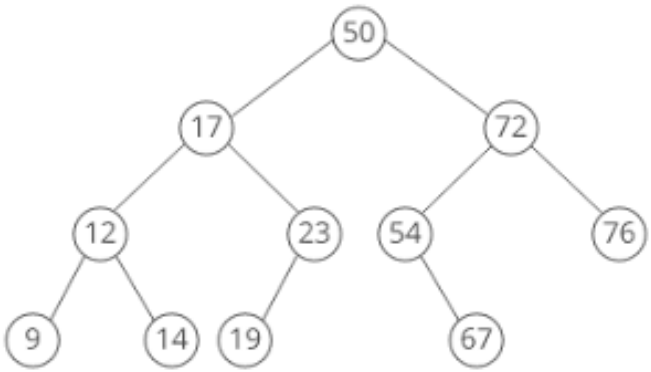


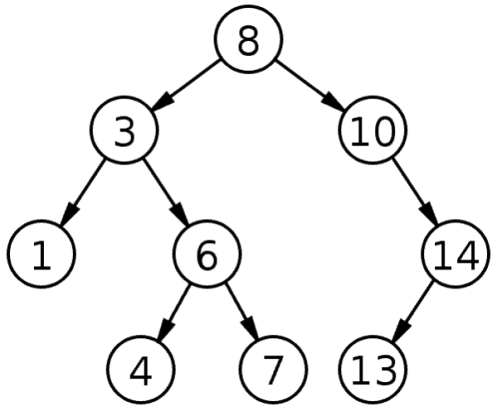
- The important feature of a tree is that each node has one and only one parent except for one special node (WHICH IS?).





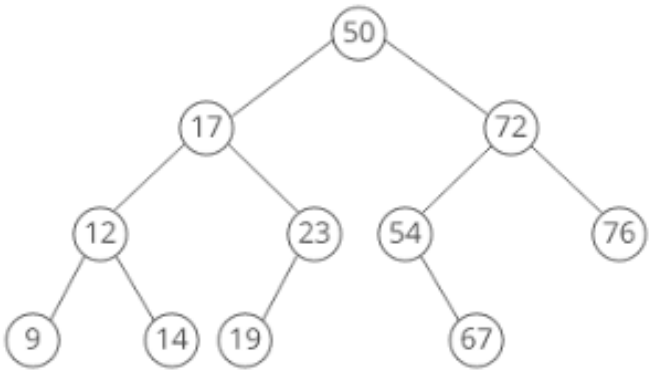
- The important feature of a tree is that each node has one and only one parent except for one special node (WHICH IS?).

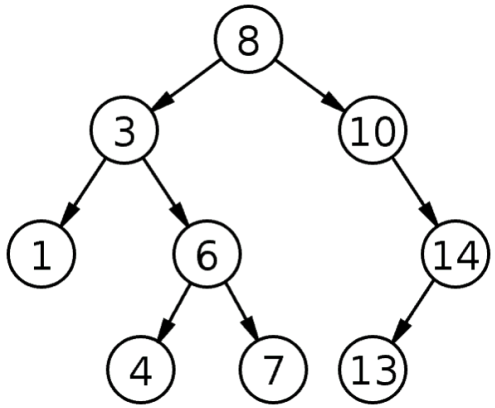




- So if we have a tree with n nodes, What is the number of edges in the tree?

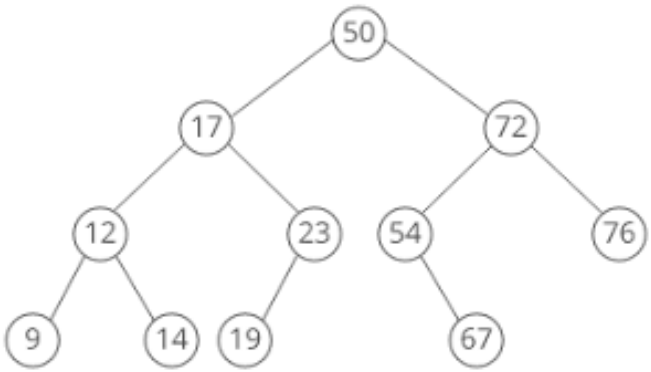
- The important feature of a tree is that each node has one and only one parent except for one special node (WHICH IS?).

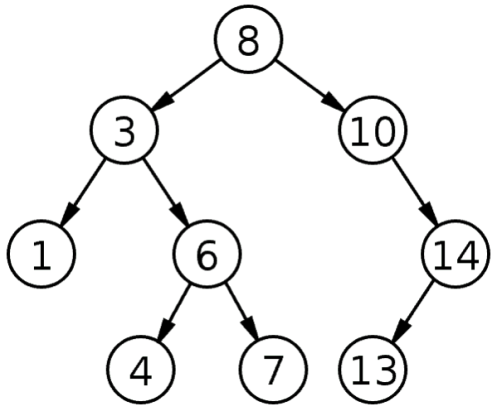




- So if we have a tree with n nodes, What is the number of edges in the tree?It's n-1

- The important feature of a tree is that each node has one and only one parent except for one special node (WHICH IS?).





- So if we have a tree with n nodes, What is the number of edges in the tree?It's n-1
- This fact implies an interesting feature of the tree, There is a unique path from the root to any node of the tree.

Outline

Similar to linked list, we created a class for the Node and a class for the Linkedlist that had a pointer to the head node of the list.

Similar to linked list, we created a class for the Node and a class for the Linkedlist that had a pointer to the head node of the list.

```
#define MAX_NO_OF_CHILDREN 100

class Node{
  /* datatype can be a primitive type (int- ..)
  or an object */
  datatype data;
```

Similar to linked list, we created a class for the Node and a class for the Linkedlist that had a pointer to the head node of the list.

```
#define MAX_NO_OF_CHILDREN 100

class Node{
  /* datatype can be a primitive type (int- ..)
  or an object */
  datatype data;

  Node * children[MAX_NO_OF_CHILDREN];
  // OR a linked list of children
  list<Node *> children;
};
```

Similar to linked list, we created a class for the Node and a class for the Linkedlist that had a pointer to the head node of the list.

```
#define MAX_NO_OF_CHILDREN 100

class Node{
  /* datatype can be a primitive type (int- ..)
  or an object */
  datatype data;

  Node * children[MAX_NO_OF_CHILDREN];
  // OR a linked list of children
  list<Node *> children;
};

class Tree{
```

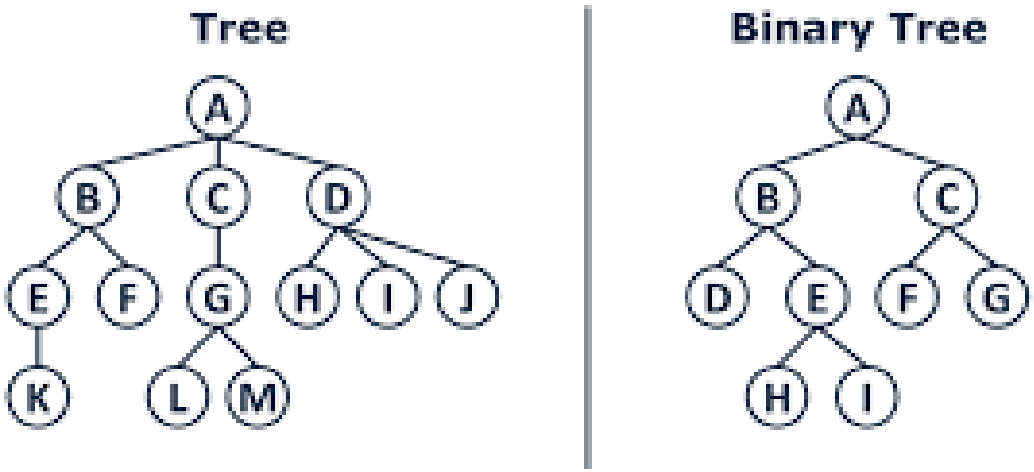
Similar to linked list, we created a class for the Node and a class for the Linkedlist that had a pointer to the head node of the list.

```
#define MAX_NO_OF_CHILDREN 100

class Node{
  /* datatype can be a primitive type (int- ..)
  or an object */
  datatype data;

  Node * children[MAX_NO_OF_CHILDREN];
  // OR a linked list of children
  list<Node *> children;
};

class Tree{
  Node * root;

  // And a set of methods to add/ delete/ print / ...
};
```

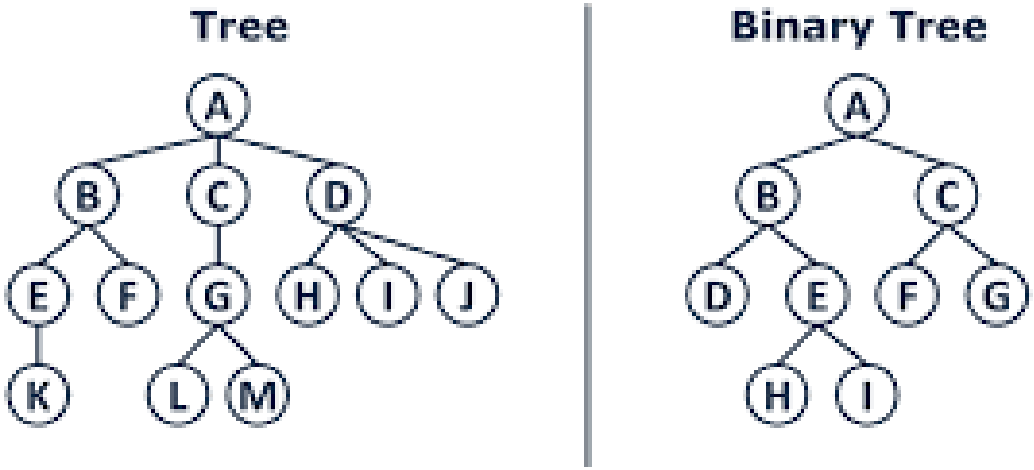
Outline

Outline

A binary tree is a tree where each node has either 0, 1 or 2 children.

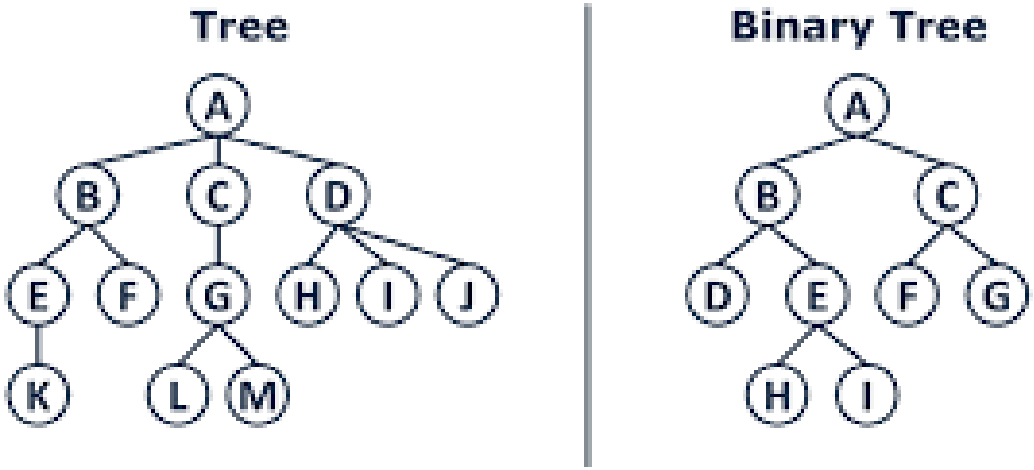


A binary tree is a tree where each node has either 0, 1 or 2 children.




```
class BinaryTreeNode{
  /* datatype can be a primitive type (int- ..)
  or an object */
  datatype data;
```

A binary tree is a tree where each node has either 0, 1 or 2 children.

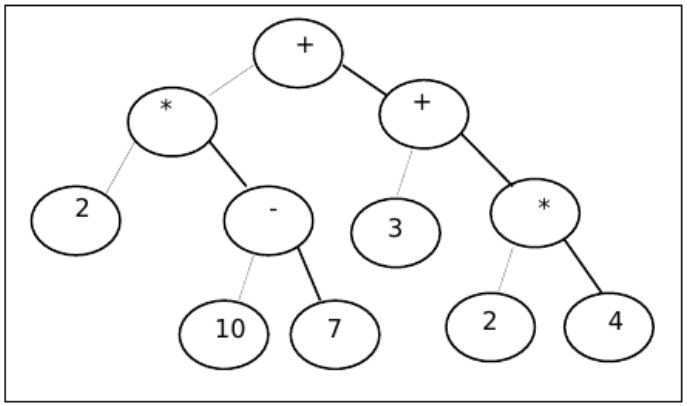



```
class BinaryTreeNode{
  /* datatype can be a primitive type (int- ..)
  or an object */
  datatype data;

  Node * left_child;
  Node * right_child;
};
```

Outline

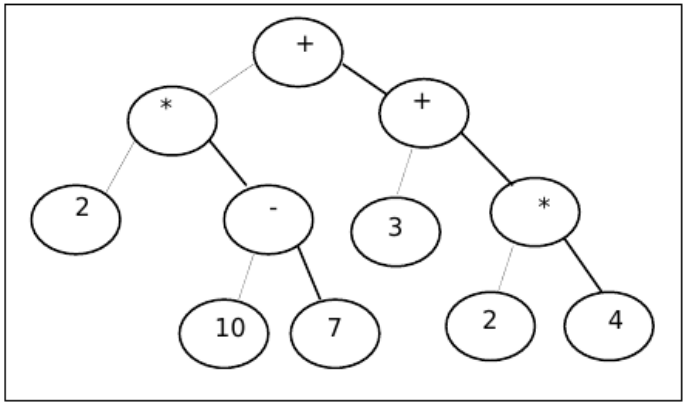
Q1. The following Binary tree holds the mathematical expression,

a) Traverse the tree using in-order, pre-order, post-order
b) Manually evaluate the numeric value of the expression held by the tree
c) Write down a recursive function to evaluate the whole tree or sub-tree of it.



Q1. The following Binary tree holds the mathematical expression,

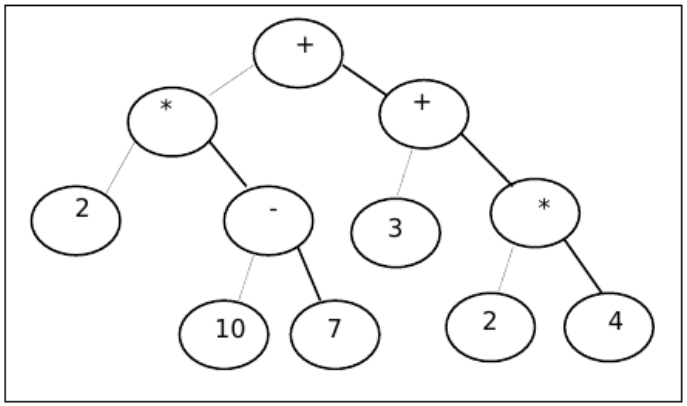
a) Traverse the tree using in-order, pre-order, post-order
b) Manually evaluate the numeric value of the expression held by the tree
c) Write down a recursive function to evaluate the whole tree or sub-tree of it.



pre-order: + * 2 - 10 7 + 3 * 2 4

Q1. The following Binary tree holds the mathematical expression,

a) Traverse the tree using in-order, pre-order, post-order
b) Manually evaluate the numeric value of the expression held by the tree
c) Write down a recursive function to evaluate the whole tree or sub-tree of it.

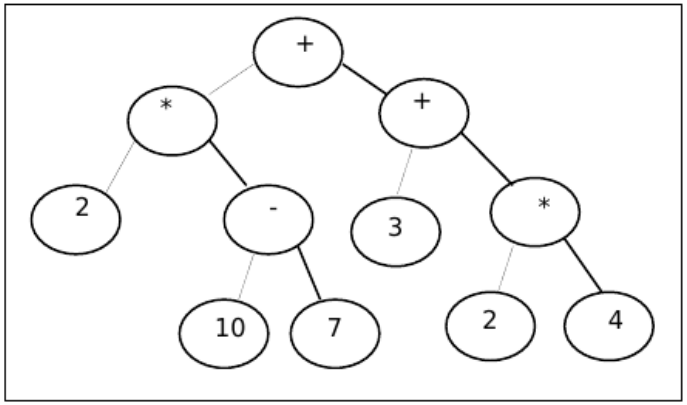


pre-order: + * 2 - 10 7 + 3 * 2 4

in-order: 2 * 10 - 7 + 3 + 2 * 4

Q1. The following Binary tree holds the mathematical expression,

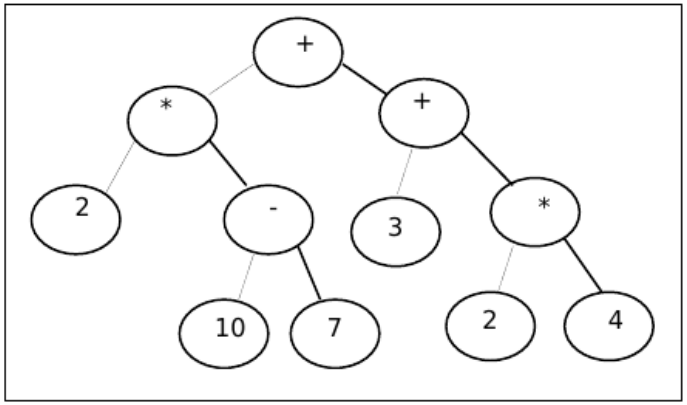
a) Traverse the tree using in-order, pre-order, post-order
b) Manually evaluate the numeric value of the expression held by the tree
c) Write down a recursive function to evaluate the whole tree or sub-tree of it.



pre-order: + * 2 - 10 7 + 3 * 2 4
in-order: 2 * 10 - 7 + 3 + 2 * 4
post-order: 2 10 7 - * 3 2 4 * + +

Q1. The following Binary tree holds the mathematical expression,

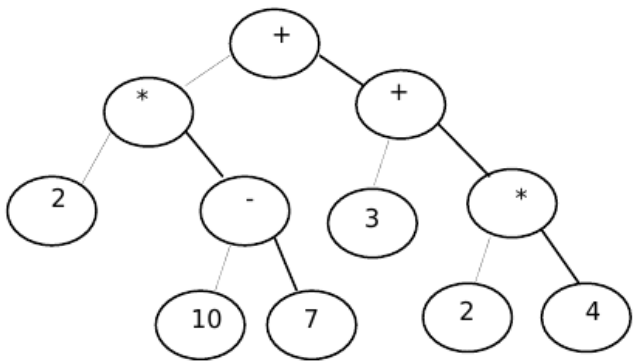
a) Traverse the tree using in-order, pre-order, post-order
b) Manually evaluate the numeric value of the expression held by the tree
c) Write down a recursive function to evaluate the whole tree or sub-tree of it.



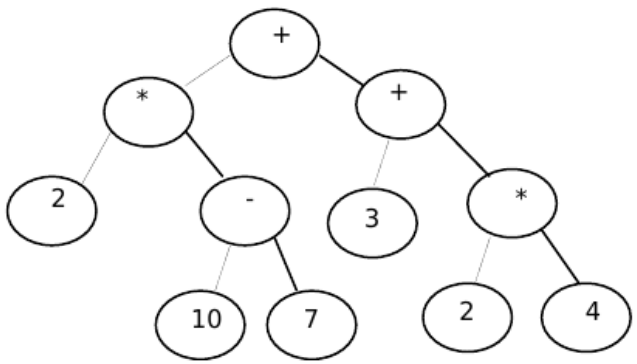
pre-order: + * 2 - 10 7 + 3 * 2 4
in-order: 2 * 10 - 7 + 3 + 2 * 4
post-order: 2 10 7 - * 3 2 4 * + +

```
#include<cstdlib> // for atoi
class Node{
  public:
    string value;
    Node * left;
    Node * right;

    bool is_numeric(){
      for (auto c: value){
        // If c is not in range[0,9]
        if (! (c>='0' && c<='9'))
          return 0;
      }
      return 1;
    }

    int to_numeric(){
      return atoi(value.c_str());
    }
};
```

```
int eval(int left_operand, int right_operand,
      string operator){
  if (operator == "+")
    return left_operand + right_operand;
  if (operator == "-")
    return left_operand - right_operand;

  // .....
}
```


+
*
+
2
-
3
*
10
7
2
4

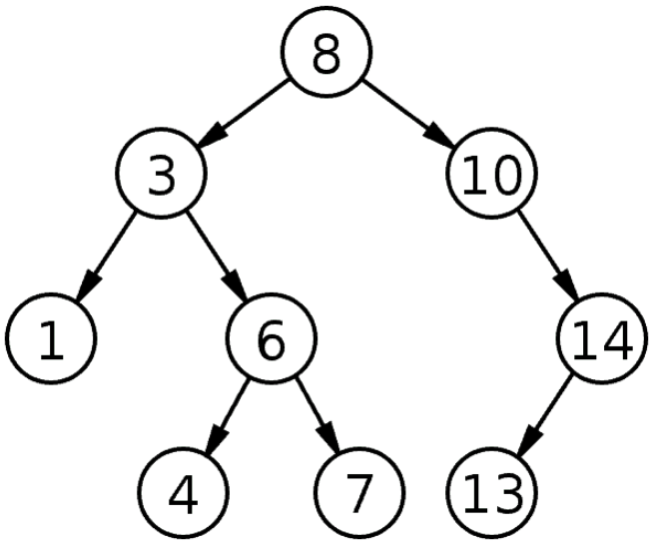
```
int evaluate(Node * subtree_root){
  if(subtree_root->is_numeric())
    return subtree_root->to_numeric();

  // Evaluate the left and right subtrees
  int left_subtree = evaluate(subtree_root->left);
  int right_subtree = evaluate(subtree_root->right);

  return eval(left_subtree, right_subtree,
        subtree_root->value);
};
```

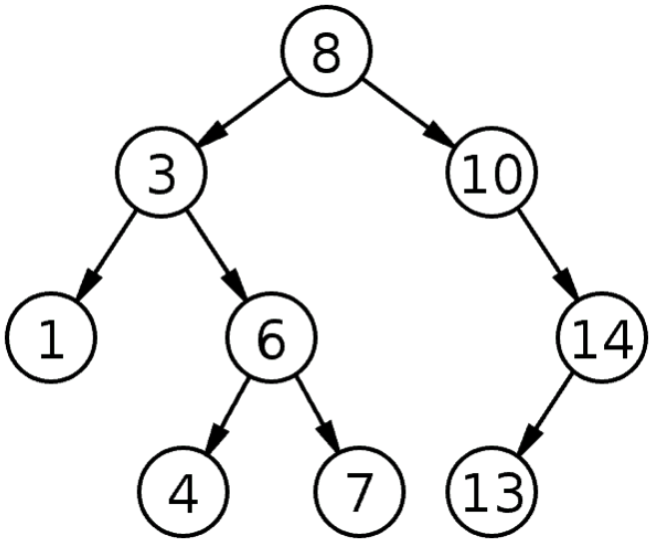

+
*
+
2
-
3
*
10
7
2
4

Outline

Write a method that counts the number of [leaf] nodes in a binary tree.

```
class Tree{
  public:
    int number_of_leaves();
};
```


8
3
10
1
6
14
4
7
13



Let's add a new private method to count the leaves in a subtree.

```
class Tree{
  private:
    int count_leaves_in_subtree(Node * subtree_root);
  public:
    int number_of_leaves();
};
```

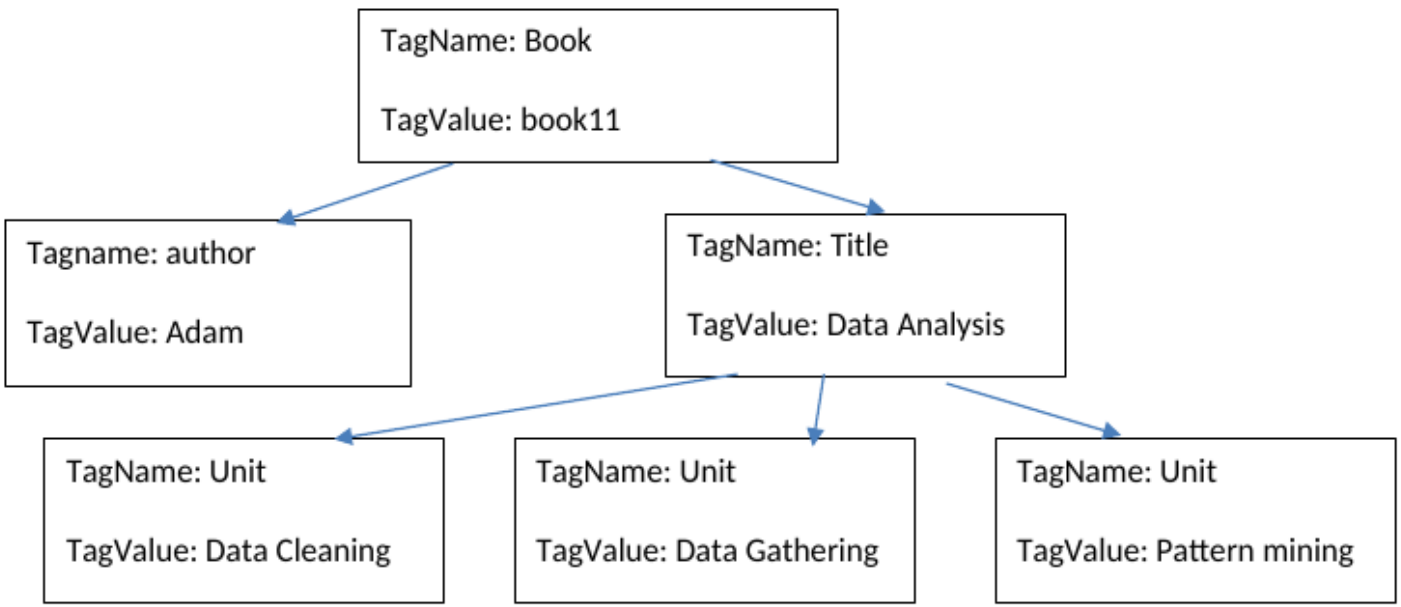
```
int Tree::count_leaves_in_subtree(Node * subtree_root){
  if (subtree_root->left == nullptr &&
    subtree_root->right == nullptr)
    return 1;

  int leaves_in_subtree = 0;
  if (subtree_root->left !=nullptr)
    leaves_in_subtree +=
      count_leaves_in_subtree(subtree_root->left);

  if (subtree_root->right !=nullptr)
    leaves_in_subtree +=
      count_leaves_in_subtree(subtree_root->right);

  return leaves_in_subtree;
}
```

Outline

Q6. Write down a program to translate a tree into XML text file. each tree node has two strings one is the tag name and the other is the tag value, each node has also a number of strings pointing to its children. Traversing the XML file tree will be translated into text a file as follows: <tag> value </tag>. When a node has children the XML appears as follows:

<parenttag> parentvalue <childtag> child 1 </childtag><childtag> child 2 </childtag> </parenttag>

The tree could be as following figure:



```
// Assume that the node class has the
// following attributes
class Node{
  public:
    string TagName;
    string TagValue;
    Node * children[100];
};

void print_XML(Node * node){
  cout<< "<" << node->TagName << ">";
  cout<< node->TagValue;

  for (int i=0; i< 100; i++){
    if (children[i] != nullptr)
      print_XML(children[i]);
  }

  cout<< "</" << node->TagName << ">";
}
```

Outline

Outline

- It's a binary tree.
- Each node has a key and a value.
- The left child node should have a key that is smaller than its parent's key.
- The right child node should have a key that is larger than its parent's key.

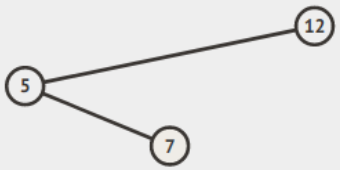
Outline

Create a BST and add to it the following numbers 12, 5, 7, 9, 13, 100, 80, 16, 18, 55, 10, 11, Show the tree after deleting nodes 10, 13, 100. Do you think a different sequence of numbers would be used to have a more compressed tree (same number of elements with minimum depth)

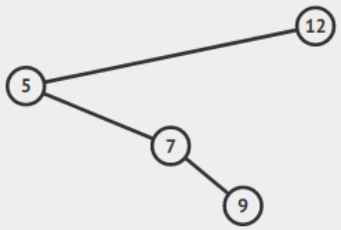
Insert 12

Insert 5

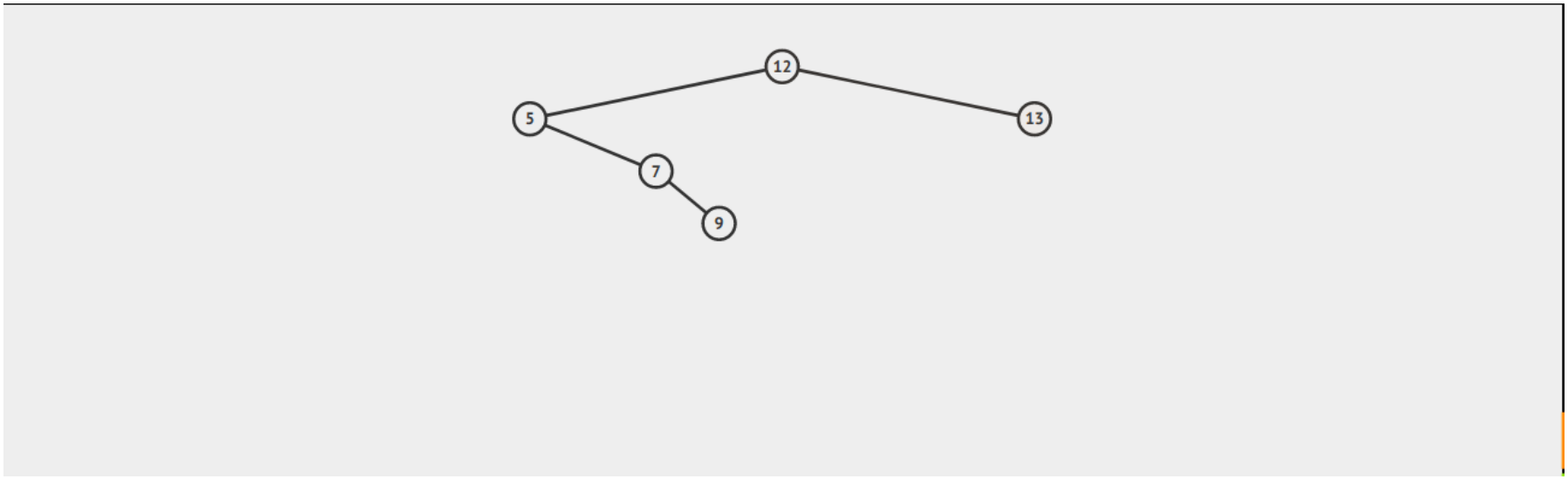
Insert 7



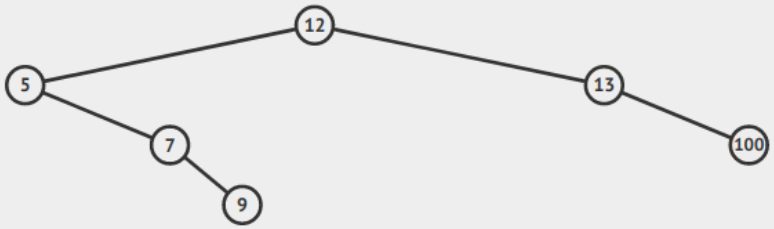
Insert 9



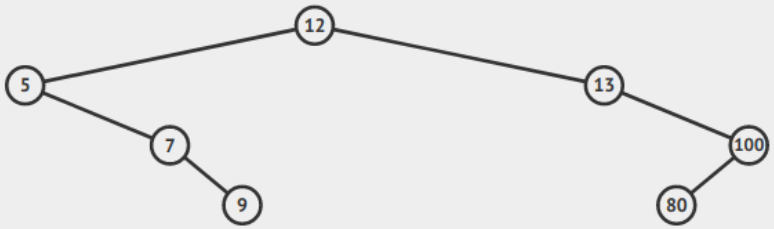
Insert 13



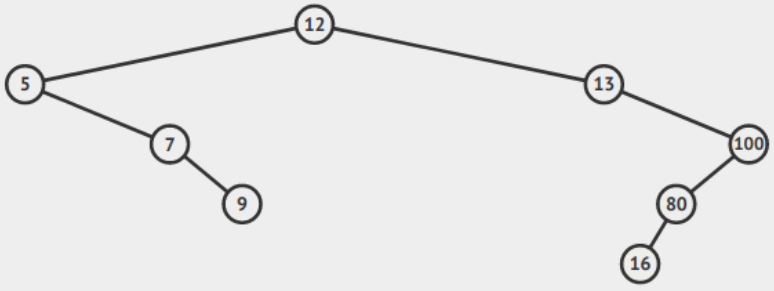
Insert 100



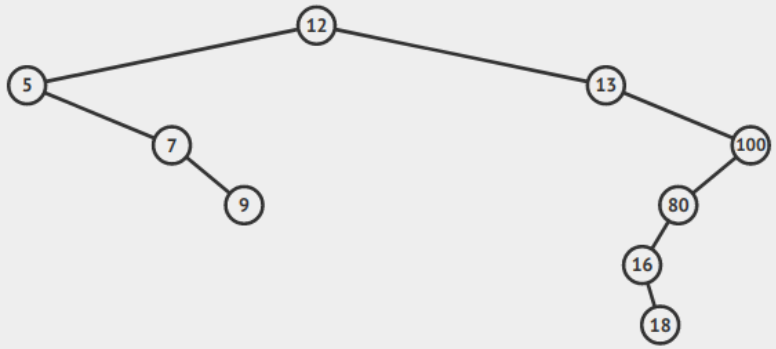
Insert 80



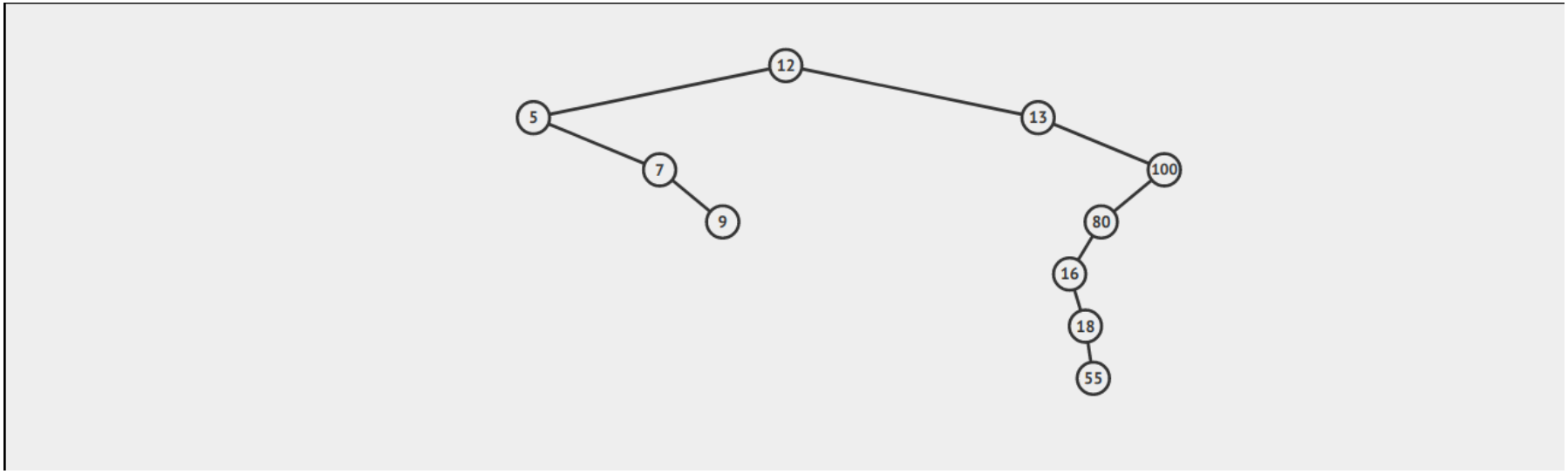
Insert 16



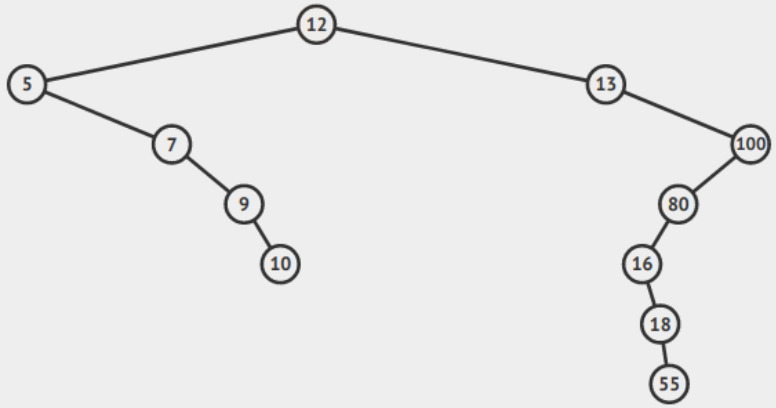
Insert 18



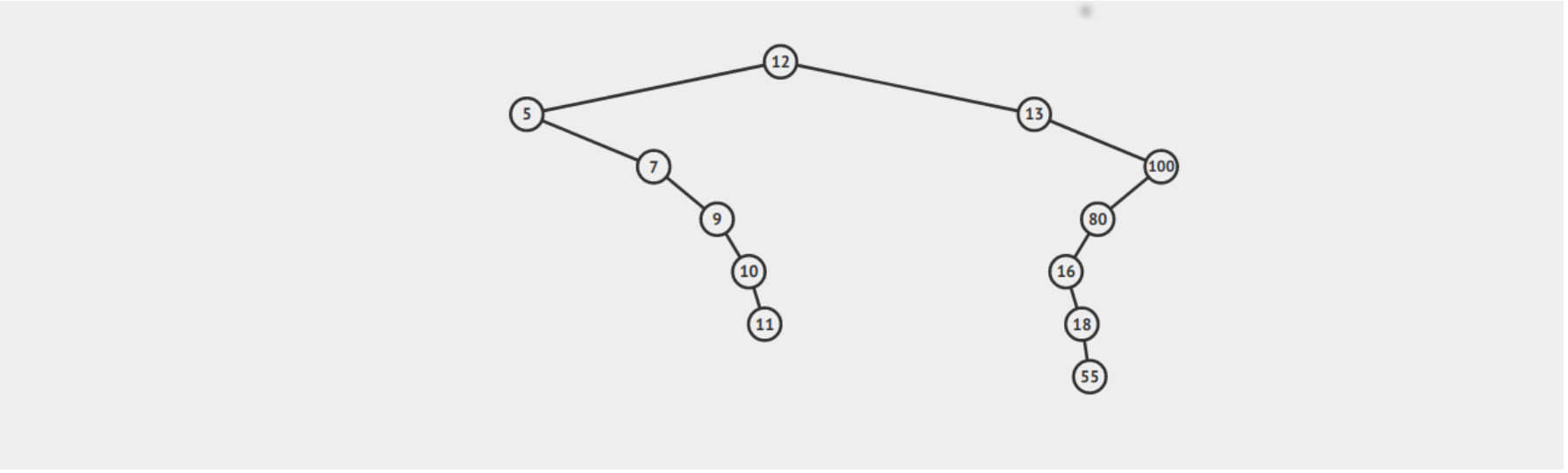
Insert 55



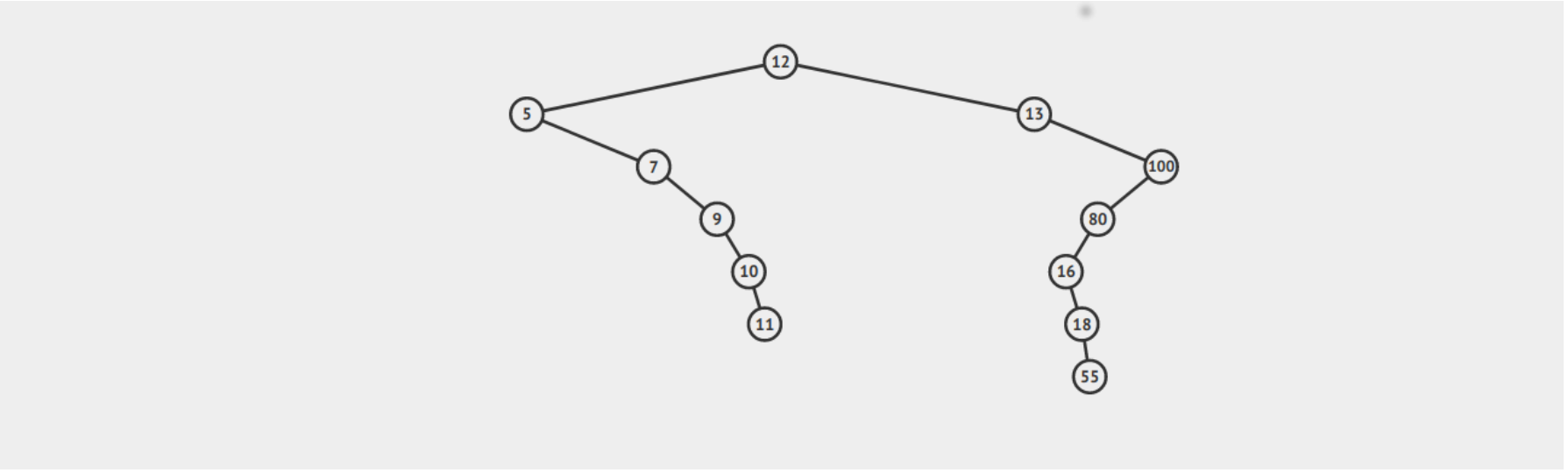
Insert 10



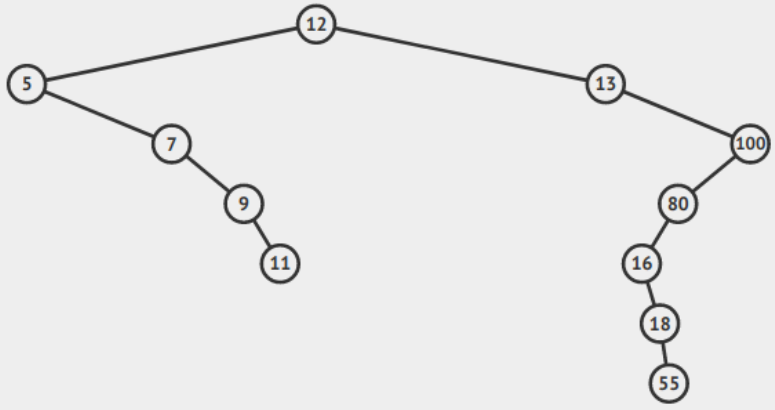
Insert 11



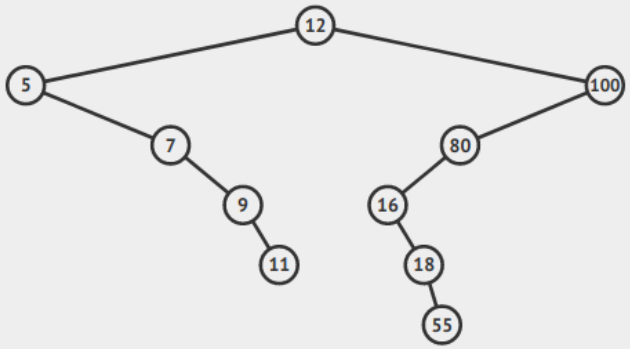
Delete 10



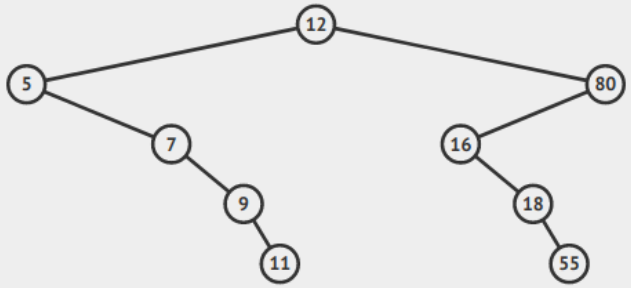
Delete 10



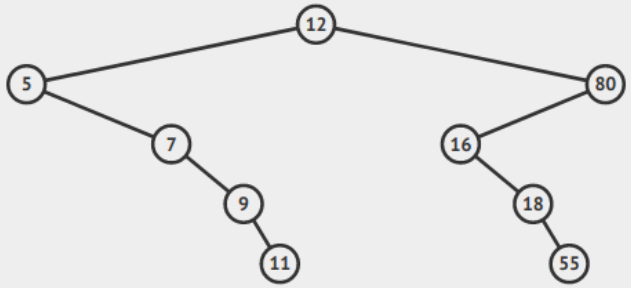
Delete 13



Delete 100



What if we deleted 11 (a leaf)?



What if we deleted 11 (a leaf)?

What if we deleted 12 (a node with two children)?

How to change the order of these values? 12, 5, 7, 9, 13, 100, 80, 16, 18, 55, 10, 11

How to change the order of these values? 12, 5, 7, 9, 13, 100, 80, 16, 18, 55, 10, 11

Sorted values: 5, 7, 9, 10, 11, 12, 13, 16, 18, 55, 80, 100

How to change the order of these values? 12, 5, 7, 9, 13, 100, 80, 16, 18, 55, 10, 11

Sorted values: 5, 7, 9, 10, 11, 12, 13, 16, 18, 55, 80, 100

- Find the value that divides the inputs into two halves (12 or 13), let's pick the smallest one.

How to change the order of these values? 12, 5, 7, 9, 13, 100, 80, 16, 18, 55, 10, 11

Sorted values: 5, 7, 9, 10, 11, 12, 13, 16, 18, 55, 80, 100

- Find the value that divides the inputs into two halves (12 or 13), let's pick the smallest one.
- We now have two halves (5, 7, 9, 10, 11) and (13, 16, 18, 55, 80, 100).

How to change the order of these values? 12, 5, 7, 9, 13, 100, 80, 16, 18, 55, 10, 11

Sorted values: 5, 7, 9, 10, 11, 12, 13, 16, 18, 55, 80, 100

- Find the value that divides the inputs into two halves (12 or 13), let's pick the smallest one.
- We now have two halves (5, 7, 9, 10, 11) and (13, 16, 18, 55, 80, 100).Do the same for each half.

How to change the order of these values? 12, 5, 7, 9, 13, 100, 80, 16, 18, 55, 10, 11

Sorted values: 5, 7, 9, 10, 11, 12, 13, 16, 18, 55, 80, 100

- Find the value that divides the inputs into two halves (12 or 13), let's pick the smallest one.
- We now have two halves (5, 7, 9, 10, 11) and (13, 16, 18, 55, 80, 100).Do the same for each half.
- Left half order: 9, 5, 7, 10, 11

How to change the order of these values? 12, 5, 7, 9, 13, 100, 80, 16, 18, 55, 10, 11

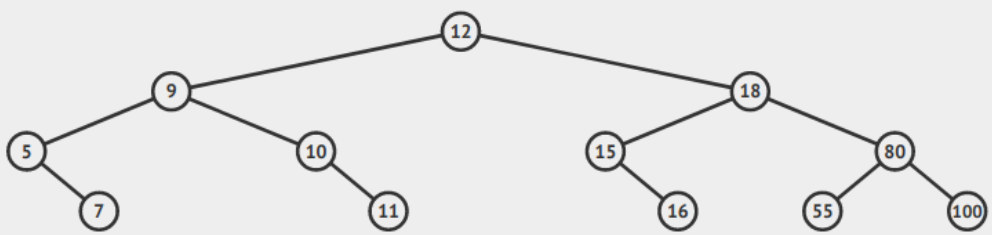
Sorted values: 5, 7, 9, 10, 11, 12, 13, 16, 18, 55, 80, 100

- Find the value that divides the inputs into two halves (12 or 13), let's pick the smallest one.
- We now have two halves (5, 7, 9, 10, 11) and (13, 16, 18, 55, 80, 100).Do the same for each half.
- Left half order: 9, 5, 7, 10, 11
- Right half order: 18, 13, 16, 80, 55, 100

How to change the order of these values? 12, 5, 7, 9, 13, 100, 80, 16, 18, 55, 10, 11

Sorted values: 5, 7, 9, 10, 11, 12, 13, 16, 18, 55, 80, 100

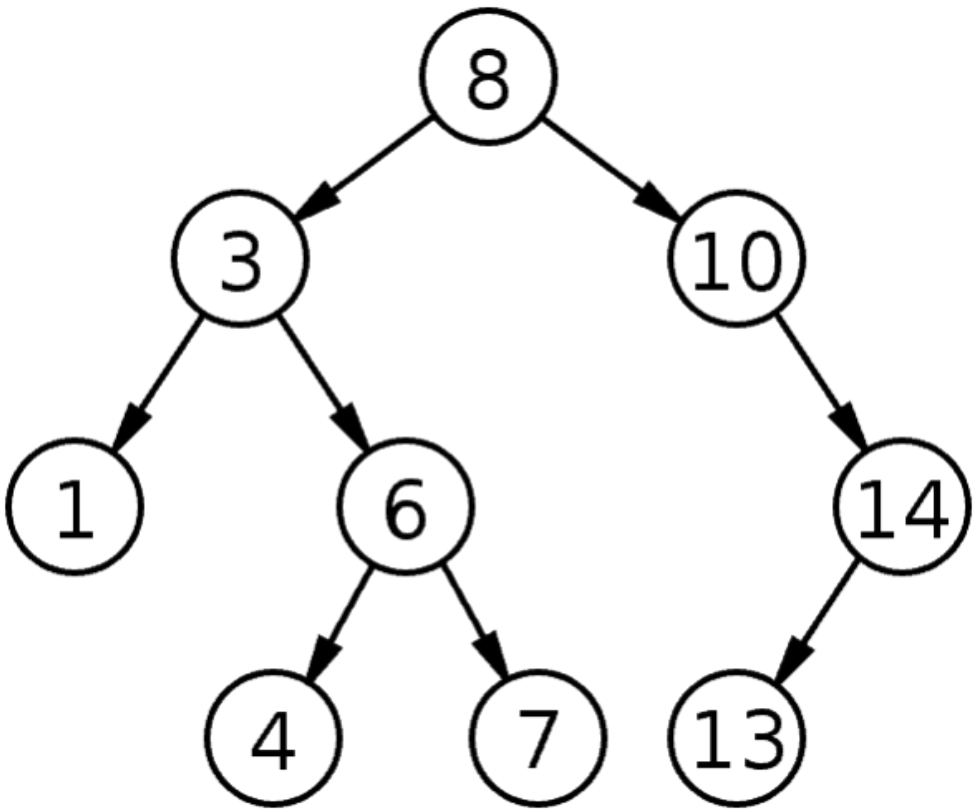
- Find the value that divides the inputs into two halves (12 or 13), let's pick the smallest one.
- We now have two halves (5, 7, 9, 10, 11) and (13, 16, 18, 55, 80, 100).Do the same for each half.
- Left half order: 9, 5, 7, 10, 11
- Right half order: 18, 13, 16, 80, 55, 100
- Order: 12, 9, 5, 7, 10, 11, 18, 13, 16, 80, 55, 100



Outline

Write down a function that returns the maximum and minimum values within a BST.

```
int tree_max(Node * root);
int tree_min(Node * root);
```



```
int tree_max(Node * root){
  if (root->right == nullptr)
    return root->value;

  return tree_max(root->right);
}
```

```
int tree_min(Node * root){
  if (root->left == nullptr)
    return root->value;

  return tree_min(root->left);
}
```

Outline

Using a BST and an unsorted array of integers, develop an algorithm to sort the array.

- Insert all the values into a BST using a for loop.
- Use in-order traversal to generate a sorted version of the array.
- NOTE: in-order traversal: Left subtree - Current node - Right Subtree

Outline

- Insert: Best

- Insert: BestO(log(n)) Worst O(n)

- Insert: BestO(log(n)) Worst O(n)
- Delete: Best

- Insert: BestO(log(n)) Worst O(n)
- Delete: BestO(log(n)) Worst O(n)

- Insert: BestO(log(n)) Worst O(n)
- Delete: BestO(log(n)) Worst O(n)
- Insert n elements to a BST: Best

- Insert: BestO(log(n)) Worst O(n)
- Delete: BestO(log(n)) Worst O(n)
- Insert n elements to a BST: BestO(nlog(n)) Worst

- Insert: BestO(log(n)) Worst O(n)
- Delete: BestO(log(n)) Worst O(n)
- Insert n elements to a BST: BestO(nlog(n)) WorstO(n*n)

Why is the depth of a balanced BST equal to log(n)??

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	1
2	

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	1
2	$1 + 2 = 3$
3	

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	1
2	$1 + 2 = 3$
3	$1 + 2 + 4 = 7$
4	

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	1
2	$1 + 2 = 3$
3	$1 + 2 + 4 = 7$
4	$1 + 2 + 4 + 8 = 15$
n	

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	1
2	$1 + 2 = 3$
3	$1 + 2 + 4 = 7$
4	$1 + 2 + 4 + 8 = 15$
n	$1 + 2 + \ldots + 2^{n-1} = 2^n$ -1 (Geometric series)

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	1
2	$1 + 2 = 3$
3	$1 + 2 + 4 = 7$
4	$1 + 2 + 4 + 8 = 15$
n	$1 + 2 + \ldots + 2^{n-1} = 2^n$ -1 (Geometric series)

If we had N nodes, we need L levels such that: 2^L -1 >= N

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	1
2	$1 + 2 = 3$
3	$1 + 2 + 4 = 7$
4	$1 + 2 + 4 + 8 = 15$
n	$1 + 2 + ... + 2^{n-1} = 2^n$ -1 (Geometric series)

If we had N nodes, we need L levels such that: 2^L -1 >= N
2^L >= N +1 (Taking log to base of 2)

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	1
2	$1 + 2 = 3$
3	$1 + 2 + 4 = 7$
4	$1 + 2 + 4 + 8 = 15$
n	$1 + 2 + ... + 2^{n-1} = 2^n$ -1 (Geometric series)

If we had N nodes, we need L levels such that: 2^L -1 >= N
2^L >= N +1 (Taking log to base of 2)
L >= log(N+1)

Why is the depth of a balanced BST equal to log(n)??

No of levels	No of nodes in a complete tree
1	1
2	$1 + 2 = 3$
3	$1 + 2 + 4 = 7$
4	$1 + 2 + 4 + 8 = 15$
n	$1 + 2 + ... + 2^{n-1} = 2^n$ -1 (Geometric series)

If we had N nodes, we need L levels such that: 2^L -1 >= N
2^L >= N +1 (Taking log to base of 2)
L >= log(N+1)
Therefore, the No of levels is O(log(N)) where N is the No of nodes.

Outline

- BST visualization: `https://visualgo.net/en/bst`
- Feedback form: `https://forms.gle/cD1bS8jYBQ1n6EMu7`
- CSE 2016 - Eng. Eslam Mounier's solutions: `https://drive.google.com/drive/folders/0BxD159ORSl13THZrVm1oQTFzUTA?usp=sharing`
- Q&A Google Doc.: `https://docs.google.com/document/d/1258ERLzJRreQN8VJbe5lFsvce6IWxLqAWpKFrB1K8_4/edit`
- Tree problems on Hackerrank: `https://www.hackerrank.com/domains/data-structures?filters%5Bsubdomains%5D%5B%5D=trees`