|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Part** | **1** | **2** | **3** | **4** | **Total** |
| *maximum* | **25** points | **25** points | **25** points | **25** points | **100**G101010 pointsG |
| ***Your Score*** |  |  |  |  |  |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Trees**

Reading Assignment: Thoroughly read Chapter 10 in the course textbook.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 1 Glossary Terms**

Define, in detail, each of these glossary terms from the realm of computer programming logic and design and computer topics, in general. If applicable, use examples to support your definitions. Consult your notes or course textbook(s) as references or by visiting Web sites such as: [**http://www.ask.com**](http://www.ask.com),[**http://www.bing.com**](http://www.bing.com), [**http://www.webopedia.com**](http://www.webopedia.com)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(a) Binary Tree**

|  |
| --- |
| A Binary Tree is a tree structure made up of a root and binary nodes that have a data item and leftChild / subtree and rightChild / subtree pointers. In a binary tree, a leftChild is always less than the current current node or root and the right child is always greater than or equal to the current node or root.   Insertions into a binary tree involves recursing through the tree’s nodes to find the correct position for the item being inserted. If the item at the current node is less than the item being inserted, the insertion function goes the the leftChild, and right if it is greater or equal to the current node, and so forth until the proper place is found.  When a binary tree is perfectly balanced, searches and insertions are logarithmic in the worst case. |

**(b) Heap Sort**

|  |
| --- |
| A heap sort algorithm builds a heap (min-heap or max-heap) from a set of data and then repeatedly removes the root item and adds it to the end of a list (Lambert 293). |

**(c) In - order Traversal**

|  |
| --- |
| In-order traversal of a binary tree is ideal when a binary tree is already sorted (as it should be on insertion). This algorithm visits the left subtree, gets data at root node, then right subtree, It moves as far left in the tree as possible before visiting a node:    **def** inOrder(self):  *'''supports inorder traversal on a view of self'''*  lyst = list()  **def** recurse(node):  *# if tree is not empty*  **if** node != None:  *# visit left subtree*  recurse(node.left)  *# append item at root of tree*  lyst.append(node.data)  *# visit right subtree*  recurse(node.right)  recurse(self.root)  **return** iter(lyst)  RETURNS (tree is sideways)  | | | 3  | | 2  | 1  0  | -1  | | -3  | | | -5  | | | | -7 |

**(d) Parse Tree**

|  |
| --- |
| A “parse tree” holds the information about the syntactic and semantic structure of a sentence and is used for recognizing, parsing, and interpreting sentences in a language (Lambert 306-307) |

**(e) Preorder Traversal**

|  |
| --- |
| PreOrder traversal in a binary tree visits the root node and then visits the left and right subtree.    **def** \_\_iter\_\_(self):  *''' PRE ORDER traversal is default traversal, use probe based loop to visit the nodes,*  *along with a stack to support returns to parent nodes during the traveral.*  *Upon each visit to a node, its item is yielded.*  *'''*  *# create a stack, push the root node, if there is one, onto the stack*  stack = ArrayStack()  stack.add(self.root)  *# while the stack is not empty*  **while** stack.isEmpty() **is** False:  *# pop a node from the stack*  node = stack.pop()  *# yield the item in the node*  **yield**(node.data)  *# push the node's right and left children, if they exist, in that order onto the stack*  **if**(node.right):  stack.push(node.right)  **if**(node.left):  stack.push(node.left)  **for** i **in** bst:  **print**(i)  RETURNS  0  -1  -3  -5  -7  1  2  3  4 |

**Part 2 True / False Exercises**

For each of these exercises, enter True or False in the spaces provided.

**FALSE** **(1)** With trees, each item, including the first and last, have a distinct successor.

**TRUE (2)** A file system is similar to a binary search tree.

**TRUE (3)** An access, an insertion, or a removal of a node in vine - like tree with N nodes and a height of N − 1 would be linear in the worst case.

**TRUE (4)** A min - heap is a binary tree in which each node is less than or equal to both of its children.

**TRUE (5)** The level order traversal guides the visits to items from left to right through the levels of the tree, much like reading lines of text in a document.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 3 Multiple Choice Exercises**

Select the correct response or responses.

**(1)** In a tree, which of the following is true?

(a) all items have a distinct predecessor and successor

**(b) each item can have multiple children**

(c) each item can have multiple parents

(d) the root has exactly one parent

**(2)** Which of the following is the topmost node in a tree and does not have a parent?

**(a) root**  (b) child

(c) leaf (d) interior node

**(3)** What kind of tree would be useful in analyzing the syntax of a sentence?

(a) binary search tree (b) sorting tree

**(c) parse tree** (d) linear tree

**(4)** Which is true about binary search trees?

(a) they cannot support logarithmic insertions

**(b) they can support logarithmic searches**

(c) they do not work well for sorted collections

(d) each node in the right subtree of a given node is less than that node

**(5)** What is the number of nodes in a full binary tree with a height of 4?

**FORMULA for finding nodes based on tree height (LAMBERT):** N = (2^h+1) -1  
N = (2^4+1)-1   
N = 31

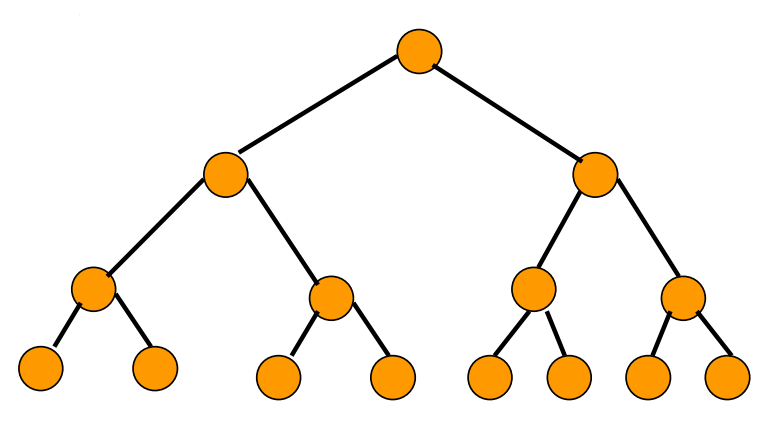
(a) 23 (b) 19 **(c) 31** (d) 47

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 4 Programming Exercises**

**(1)** The formula for determining the number of nodes in a full binary tree where *h* is the height of the tree is ( 2 *h* − 1 ) .

For the tree diagram below, determine the height of the tree and the number nodes in the tree.



**HEIGHT = 3  
FORMULA:** N = (2^**h**+1) -1   
N = (2^**3**+1)-1   
N = (2^4)-1   
N = (16) – 1   
**Nodes = 15**

**(2)** What is the height of a full binary tree with 63 nodes?

**FORMULA: H = log­­­2 (N + 1) -1**

H = log2 (**63** +1) -1

H = (6) – 1

**Height = 5**

Checking work…   
N = (2^5+1) – 1

N = (2^6) – 1

N = (64) – 1   
**N = 63**

**Height is 5 and # of nodes is 63**

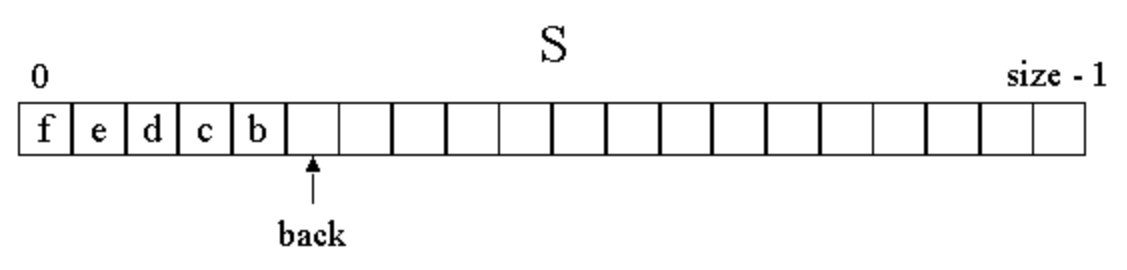
**(3)** A binary tree is a type of tree data structure and is comprised of nodes with at most two children i.e., a right and left child. The node at the top is referred to as the root node. A node without children is known as a leaf node. Some variants of binary trees include, binary search trees and B - trees.

Binary trees are mainly used for searching and sorting since they provide a means to store data hierarchically. Some common operations performed on binary trees include insertion, deletion and traversal.

(a) Explain why a stack is NOT a common application for a binary tree.

Hint: consider the diagram that is given below.

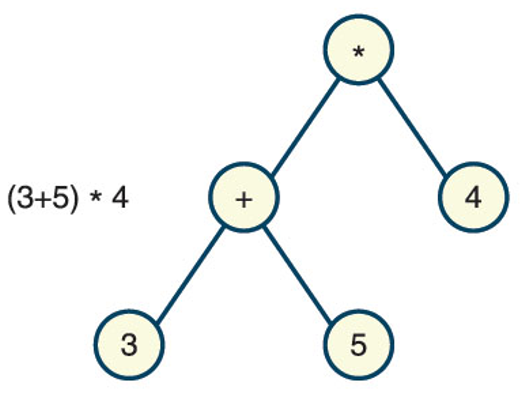
**A stack is not a common application for a binary tree because a stack doesn’t support sorted insertions (it only supports insertions at the front of the stack), selective removals (can only remove from the top of the stack), and sorted traversals (only supports front-back traversal).**



(b) Explain why an expression tree is a common application for a binary tree.

Hint: consider the diagram that is given below.

**An expression tree (used for converting prefix to postfix expressions) is a common application for a binary tree because it involves splitting operators and operands and doing a sorted insert. Perfect for a binary tree.**



**(4)** (a) A programmer wishes to attempt to use a list to implement a binary term, using R1 for row 1 , R2 for row 2 , R for the Root node, L for left - side node and R for the right - side node. The numbers, for the elements of the list, in the list will represent node values.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| element |  | R1 | R | 5 | R2 | L | 2 | R | 3 | R2 | L | 4 |

Will this be a good approach to simulate a binary tree? Explain your answer. **I read that using an array list for a binary tree only applies in special sitatuations (Lambert).**

**This implementation is confusing. I would tell the programmer, why aren’t you using a linkedList, with clear leftChild and rightChild pointers, and a clear hierarchy?**

(b) Examine the program code given below and attempt to draw a diagram which simulates the tree that the code represents.

You can use a hand - drawing or a software tool such as MS Word, MS Visio, Adobe Illustrator, MS Paint or PaintBrush for the MAC to digitally create your tree.

|  |
| --- |
| **class Tree :**  **def \_\_init\_\_(self, value = None, left = None, right = None) :**  **self.left = left**  **self.right = right**  **self.value = value**    **def PrintTree(self) :**  **print (self.value)**  **if self.left :**  **self.left.PrintTree()**  **if self.right :**  **self.right.PrintTree()**  **try :**  **root = Tree(10)**  **root.left = Tree(3)**  **root.right = Tree(8)**  **root.left.left = Tree(1)**  **root.left.right = Tree(6)**  **root.right.right = Tree(12)**  **root.PrintTree()**  **except :**  **print ("An exception has occurred!")** |

lyst = [10, 3, 8, 1, 6, 12]

comparableLyst = []

*for* i *in* range(len(lyst)):

*# comparable class takes a data value and an integer for priority, in this case,*

*# the numbers themselves*

comparableLyst.append(Comparable(lyst[i], lyst[i]))

bst = LinkedBST(comparableLyst)

print(bst)

RETURNS SIDEWAYS TREE, lesser items than root node…

| 12

10

| | 8

| | | 6

| 3

| | 1

PRINTING IN ORDER

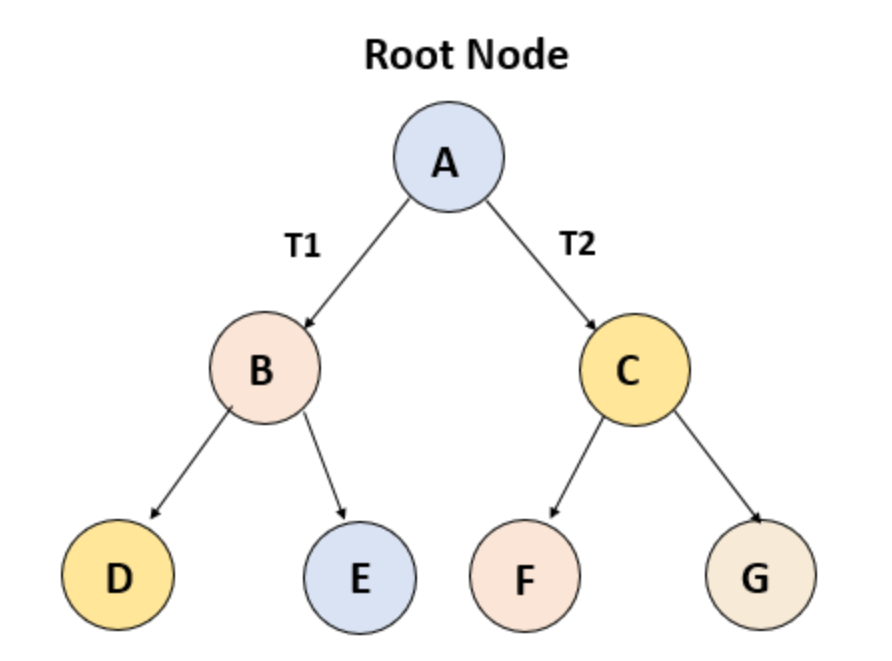
*for* i *in* bst.inOrder():

print(i)

RETURNS

1 3 6 8 10 12

**(5)**



(a) For the above tree enter the values or data that is represented by this data structure.

**A** the root or topmost node

**6** the number of edges or links between two nodes

**E** the value of one particular child node

**B** the value of one particular parent node

**4** the number of leaf nodes

**2** the height, i.e., the length of the longest path to a leaf

**2** the depth or the largest length of a path to its root

(b) An in - order traversal is a type of binary tree traversal that traverses the left subtree, visits the root node and traverses the right subtree.

Can such an in - order traversal be performed on the above tree?

**YES**