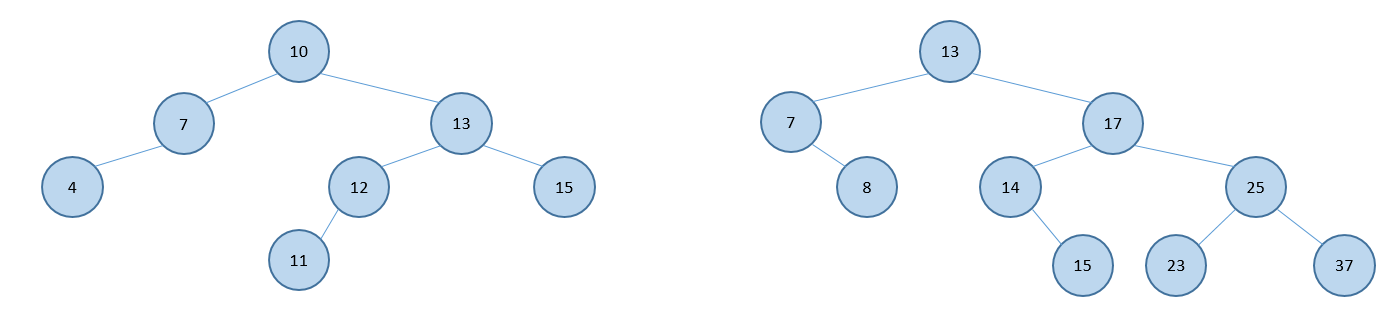
Exercises for lesson 2: Binary trees

**Exercise 3.1**

Draw the two trees as arrays, using computed child links.



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 7 | 13 | 4 | \* | 12 | 15 | \* | \* | \* | \* | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | 7 | 17 | \* | 8 | 14 | 25 | \* | \* | \* | \* | \* | 15 | 23 | 37 |

**Exercise 3.2**

The two arrays represent trees that are implemented with computed child links. Draw the trees.

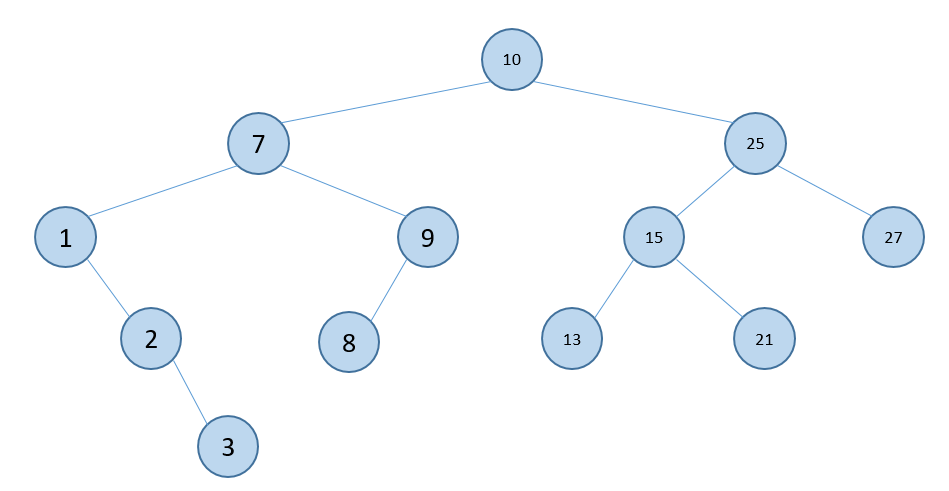
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 10 | 25 | - | 12 | 20 | 27 | - | - | 11 | - | 19 | - |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 10 | 30 | 5 | 15 | - | 35 | - | 6 | - | - | - | - | 33 | - |

**Exercise 3.3**

Write the sequences of numbers resulting from doing a:

1. Pre-order traversal: 10, 7, 1, 2, 3, 9, 8, 25, 15, 13, 21, 27
2. In-order traversal: 1, 2, 3, 7, 8, 9, 10, 13, 15, 21, 25, 27
3. Post-order traversal: 3, 2, 1, 8, 9, 7, 13, 21, 15, 27, 25, 10
4. Level-order traversal: 10, 7, 25, 1, 9, 15, 27, 2, 8, 13, 21, 3



What is the time complexity of the traversals?

**Exercise 3.4**

Write pseudo-code for a non-recursive version of a pre-order traversal. What is the time complexity of your method?

(Hint: Use a stack as an auxiliary data structure)

preOrderTraversal(root):

if root == null:

return

node = root

stack = new stack

while node != null:

visit(node)

if node.right != null

stack push node.right

if node.left != null

node = node.left

else if !stack is empty

node = pop stack

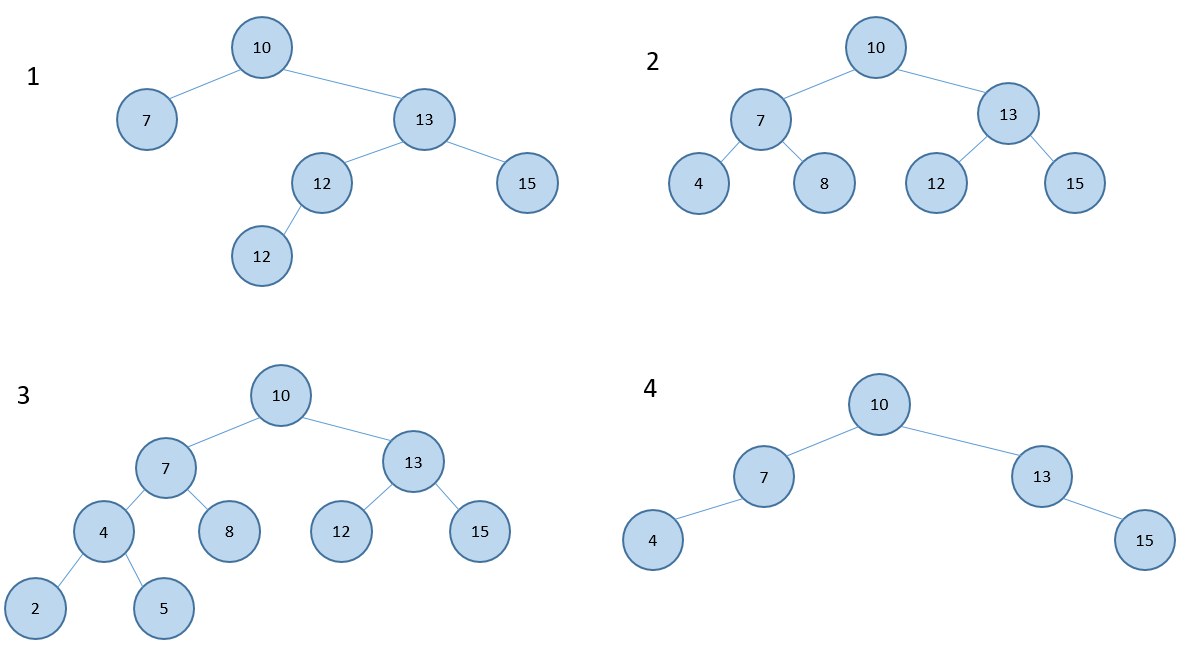
else

node = null

**Exercise 3.5**

For each of the trees determine if it is:

* Full
* Complete



1: none

2: full + complete

3: complete

4: none

**Exercise 3.6**

Create a tree ADT/interface in java, and then implement the tree with a “linked list”-type representation. This is relatively easy to do if you rewrite your implementation of a linked list.

**Exercise 3.7 (challenge exercise)**

The (relative) frequencies of the letters in the English language are[[1]](#footnote-1):

|  |  |  |  |
| --- | --- | --- | --- |
| Letter | Count | Letter | Count |
| E | 21912 | M | 4761 |
| T | 16587 | F | 4200 |
| A | 14810 | Y | 3853 |
| O | 14003 | W | 3819 |
| I | 13318 | G | 3693 |
| N | 12666 | P | 3316 |
| S | 11450 | B | 2715 |
| R | 10977 | V | 2019 |
| H | 10795 | K | 1257 |
| D | 7874 | X | 315 |
| L | 7253 | Q | 205 |
| U | 5246 | J | 188 |
| C | 4943 | Z | 128 |

Based on this table, build a general-purpose Huffman tree and encode the message “ILOVEADS”. If you find the work of building the Huffman tree by hand too tedious, write code to do it for you!

1. https://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html [↑](#footnote-ref-1)