Foundations of Computer Science (Semester 2) – 2015

Assessed Exercise Sheet 3 – 10% of Continuous Assessment Mark

Deadline : 11pm Sunday 8th February, via Canvas

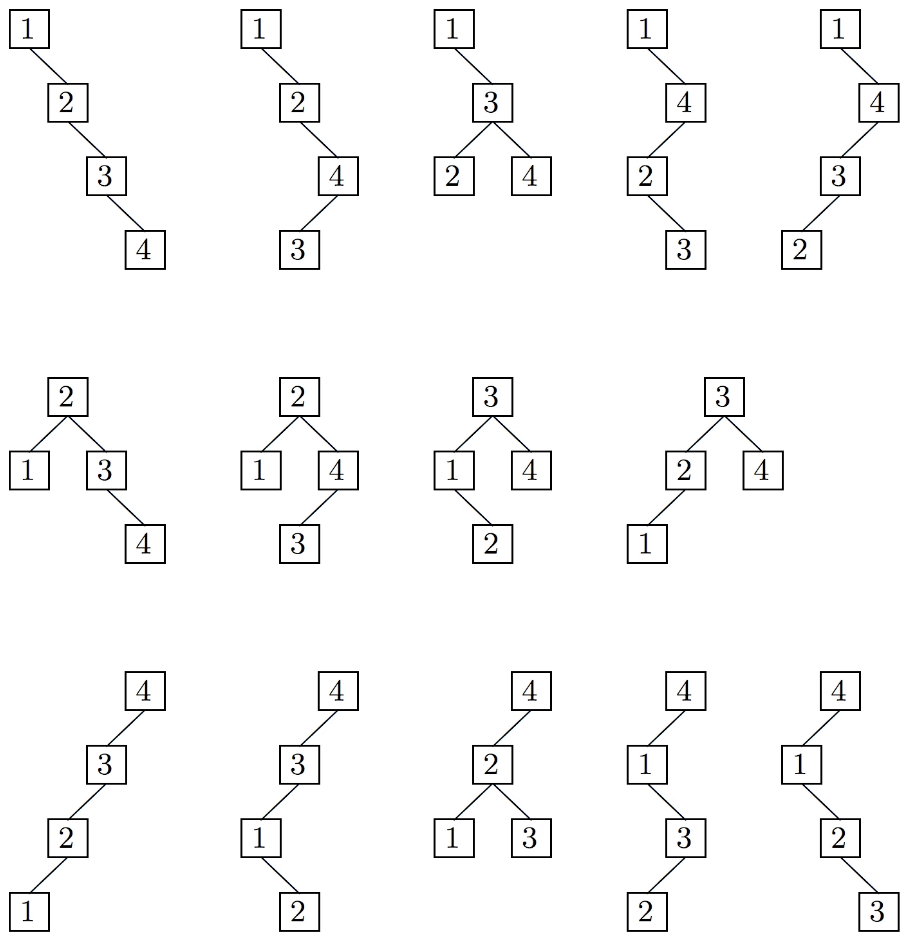
Question 1 (20 marks)

How many different orderings of the four numbers {1, 2, 3, 4} are there?

4! = 24

By considering all those possible orderings, draw all possible binary search trees of size four with nodes labeled by the four numbers {1, 2, 3, 4}. After discarding any duplicate trees, how many different binary search trees of size four are there?

Out of the 24, there are 14 different trees:



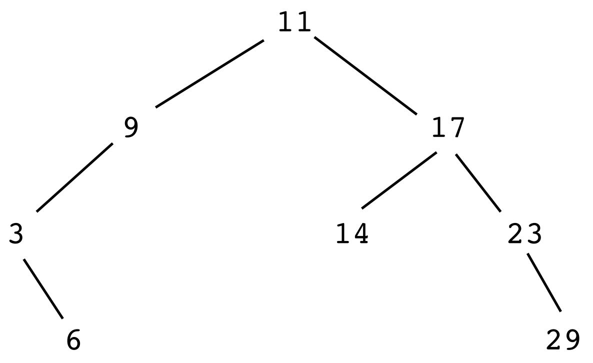
For each different tree, state its height, how many leaf nodes it has, and whether it is perfectly balanced.

Height: 3, 3, 2, 3, 3; 2, 2, 2, 2; 3, 3, 2, 3, 3.
Leafs; 1, 1, 2, 1, 1; 2, 2, 2, 2; 1, 1, 2, 1, 1.
Balanced: N, N, N, N, N; Y, Y, Y, Y; N, N, N, N, N.

Question 2 (16 marks)

One way of storing binary trees is as a table with four columns: the record numbers, the node values, pointers to the record number of the left sub-tree, and pointers to the record number of the right sub-tree. A "null pointer" means the sub-tree is empty. Draw the binary tree represented by the following table when the root is stored in record number 6.

Record number	Node value	Left sub-tree	Right sub-tree
1	23	null	5
2	9	4	null
3	14	null	null
4	3	null	8
5	29	null	null
6	11	2	7
7	17	3	1
8	6	null	null

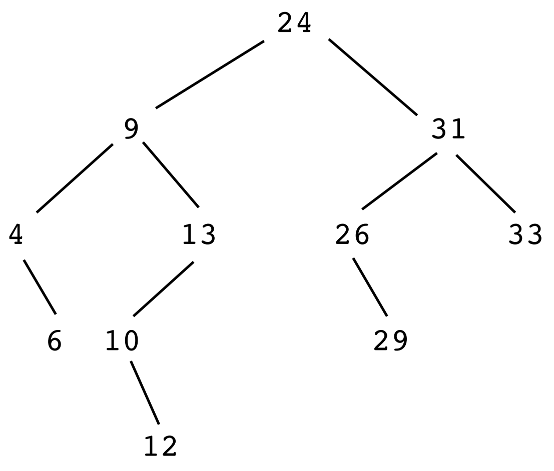


Is that tree a valid binary search tree? Explain why.

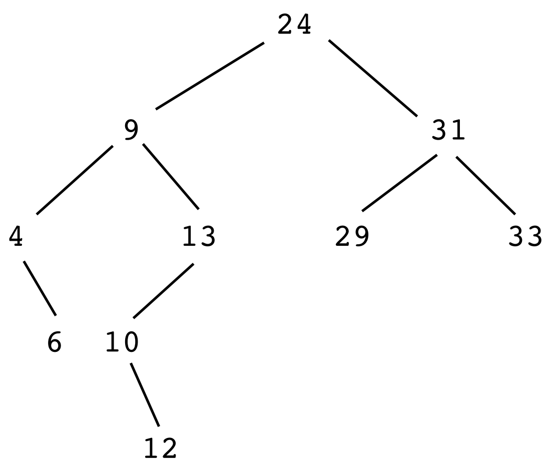
Yes, it is, because each node has a value that is bigger than all node values in its left sub-tree and smaller than all node values in its right sub-tree.

Question 3 (24 marks)

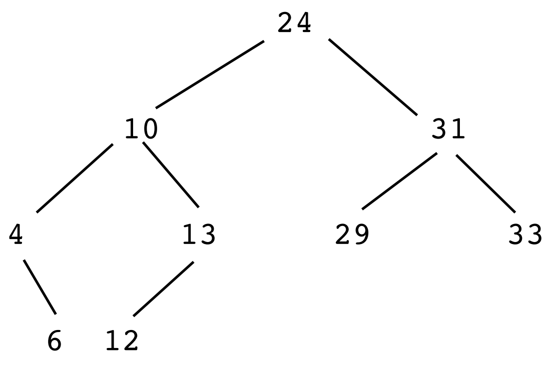
Draw the binary search tree that results from inserting the items [24, 9, 13, 4, 31, 26, 6, 10, 29, 33, 12] in that order into an initially empty tree.



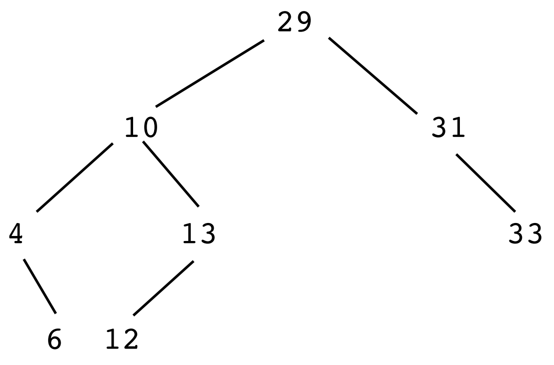
Draw tree that results from deleting the item 26 from that tree using the `delete` algorithm in the lecture notes (Section 6.6).



Now draw the tree that results from deleting the item 9 from that tree.



Finally, draw the tree that results from deleting the item 24 from that tree.



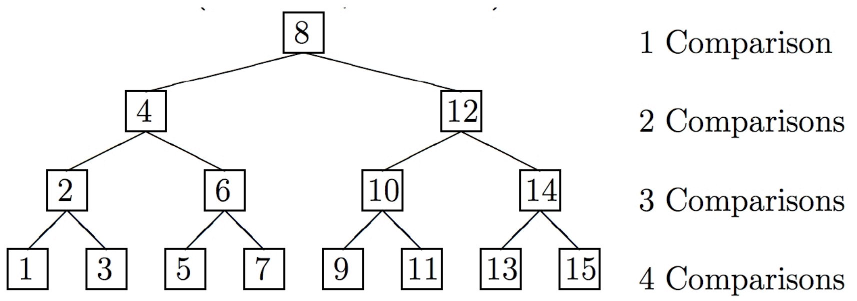
You have now drawn three new trees, each one smaller than the one before. Are they all valid binary search trees? If not, give an example where they are not. Are these the same trees that you would have got if you had built a new binary search tree from the reduced list of numbers each time? If not, give an example of where they are different.

Yes, they are all valid binary search trees. No, they are not the same as building new trees from scratch. For example, after deleting items 26, 9 and 24 from the list, item 13 would be the root of the new tree drawn from scratch, not item 29.

Question 4 (16 marks)

The 15 numbers {1, 2, 3, …, 15} can be stored in a perfectly balanced binary search tree of height 3. Draw that tree.

Suppose the determination of "bigger, equal or smaller" is counted as one "comparison". For each of the above 15 numbers, state how many comparisons are required to determine whether that number is in the search tree.



Compute the total number of comparisons *C* required to search for all the numbers, and hence compute the average number of comparisons *A* required to search for one number.

$$C = (1 \times 1) + (2 \times 2) + (4 \times 3) + (8 \times 4) = 49$$

$$A = 49/15 = 3.267$$

Question 5 (24 marks)

Derive expressions for *C(h)* and *A(h)* which generalize the total and average number of comparisons computed in Question 4 to the case of a full binary search tree of any height *h*.

Hint: This question is quite challenging! If your mathematics is strong, it is possible to write *C(h)* as the sum of the number of comparisons at each level, and then simplify that using the usual algebraic tricks for summing series. Or, you may find it easier to proceed in the same way as in lecture notes Section 5.5 where an expression was derived for the tree size *s(h)*, i.e. the total number of nodes in a full binary tree of height *h*. First think about how *C(h+1)* is related to *C(h)* by adding in the extra row *h*+1, then think about how the height *h*+1 tree's *C(h+1)* is related to its two sub-tree's *C(h)*, and finally combine the two equations you get for *C(h+1)* to give an equation for *C(h)*. You can use the expression for *s(h)* from the lecture notes. Whichever approach you use, show all the steps involved.

Algebraic summation of series solution:

Level *i* contains $2^i.(i+1)$ comparisons, so the total number of comparisons is:

$$C(h) = \sum_{i=0}^{h} 2^i.(i+1)$$

Summing this series algebraically can be done in various ways. One approach is to use

$$\frac{d}{dx} x^{i+1} = x^i.(i+1) \qquad \text{and} \qquad (1-x).\sum_{i=0}^{h} x^{i+1} = x - x^{h+2}$$

which allow the sum for *C(h)* to be written:

$$C(h) = \frac{d}{dx}\sum_{i=0}^{h} x^{i+1}\Bigg|_{x=2} = \frac{d}{dx}\left(\frac{x - x^{h+2}}{1-x}\right)\Bigg|_{x=2} = \left(\frac{x - x^{h+2}}{(1-x)^2} + \frac{1-(h+2)x^{h+1}}{1-x}\right)\Bigg|_{x=2}$$

$$\Rightarrow \quad C(h) = 2 - 2^{h+2} - 1 + (h+2).2^{h+1} = h.2^{h+1} + 1$$

Two expression for C(h+1) solution:

Level *i* contains $2^i.(i+1)$ comparisons, so level $h+1$ contains $2^{h+1}.(h+2)$ comparisons. Adding that to the *C(h)* comparisons of the height *h* tree gives:

$$C(h+1) = C(h) + 2^{h+1}.(h+2)$$

The two height *h* sub-trees within the height *h+1* tree would each have *C(h)* comparisons if the height *h*+1 tree's root node were not there. But that root node is there, so each of the *s(h+1)* nodes in the whole tree has one more comparison due to that root node, so:

$$C(h+1) = 2.C(h) + s(h+1).1$$

Then subtracting one expression for *C(h+1)* from the other gives:

$$C(h) = 2^{h+1}.(h+2) - s(h+1)$$

From the lecture notes, or a similar proof, we have:

$$s(h) = 2^{h+1} - 1$$

so substituting that into the expression for *C(h)* gives

$$C(h) = 2^{h+1}.(h+2) - 2^{h+2} + 1 = h.2^{h+1} + 1$$

which, not surprisingly, is the same expression obtained by the other approach.

Finally, simply divide *C(h)* by the total number of nodes *s(h)* to give the average

$$A(h) = \frac{C(h)}{s(h)} = \frac{h.2^{h+1} + 1}{2^{h+1} - 1}$$

What can you deduce about the average number of comparisons required, and hence the search complexity, for large trees?

For large trees $A(h) \sim h \sim \log_2 s(h)$, so the searching has linear complexity in terms of the height of the tree and log complexity in terms of the size of the tree.