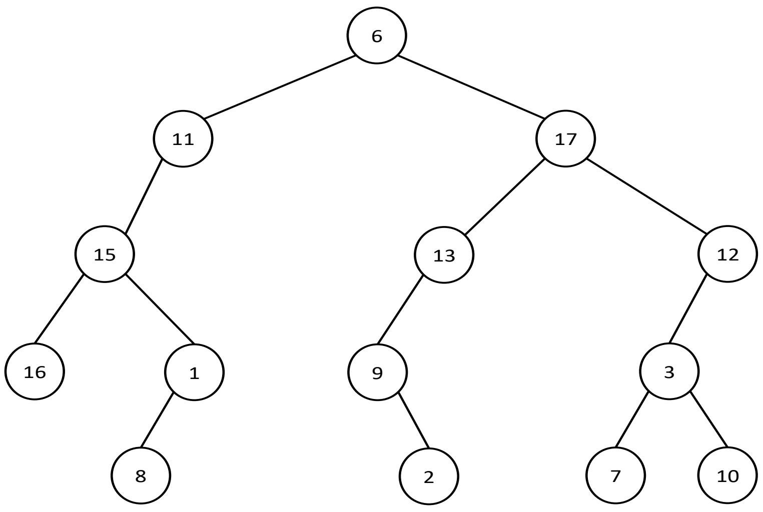
Module IN2002—Data Structures and Algorithms
Exercise Sheet 7

1. Write out a preorder, inorder, postorder and breadth-first traversals of the following binary tree:



Indicate the contents of the node visited and the stack/queue at each point in the preorder, inorder, postorder and breadth-first traversals.

- Preorder: // visit the node, traverse left, then traverse right; use stack to keep track of nodes to visit given the order of the recursive calls

Node visited	Stack
	6
6	11 17
11	15 17
15	16 1 17
16	1 17
1	8 17
8	17
17	13 12
13	9 12
9	2 12
2	12
12	3
3	7 10
7	10
10	

- Inorder: // traverse left, visit node, then traverse right; use stack to keep track of nodes to visit given the order of the recursive calls

Node visited	Stack
	16 15 11 6
16	15 11 6
15	8 1 11 6
8	1 11 6
1	11 6
11	6
6	9 13 17
9	2 13 17
2	13 17
13	17
17	7 3 12
7	3 12
3	10 12
10	12
12	

- Postorder: // traverse left, traverse right, then visit the node; use stack to keep track of nodes to visit given the order of the recursive calls

Node visited	Stack
16	15L 11L 6L
8	1L 15R 11L 6L
1	15R 11L 6L
15	11L 6L
11	6L
2	9R 13L 17L 6R
9	13L 17L 6R
13	17L 6R
7	3L 12L 17R 6R
10	3R 12L 17R 6R
3	12L 17R 6R
12	17R 6R
17	6R
6	

- Breadth-first: // visit one level at a time (left to right); use a queue to keep track of nodes to visit.

Node visited	Queue
	6
6	11 17
11	17 15
17	15 13 12
15	13 12 16 1
13	12 16 1 9
12	16 1 9 3
16	1 9 3
1	9 3 8
9	3 8 2
3	8 2 7 10
8	2 7 10
2	7 10
7	10
10	

2. Write a recursive function that reverses the order of the nodes in each level of a binary tree.

```
void reverse(TreeNode t) {
        if (t != null) {
                tmp = t.left;
                t.left = t.right;
                t.right = tmp;
                reverse(t.left);
                reverse(t.right);
        }
}
```

3. Write a non-recursive function using a stack to compute the size of a binary tree.

- Any traversal could be adapted by creating a counter and adding 1 to it each time a node is visited. The exercise asks for an iterative function using stacks, so preorder or inorder would be appropriate.

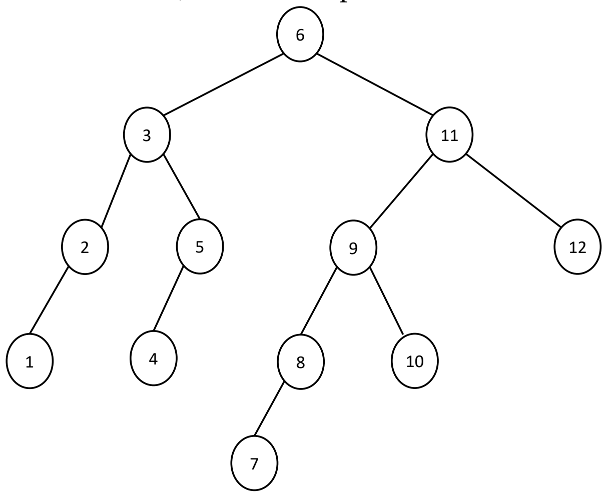
- Using the preorder function:

```
int iterativeSize(TreeNode p) {
   int count = 0; // we create the counting variable
   if (p != null){
         Stack stack = new StackImpl();
         stack.push(p);
         do {
               p = (TreeNode)stack.pop();
```

```
                count ++; // we add 1 when we visit
                if (p.right != null)
                     stack.push(p.right);
                if (p.left != null)
                     stack.push(p.left);
           } while (! stack.isEmpty());
      }
      return count; // return the count
   }
```

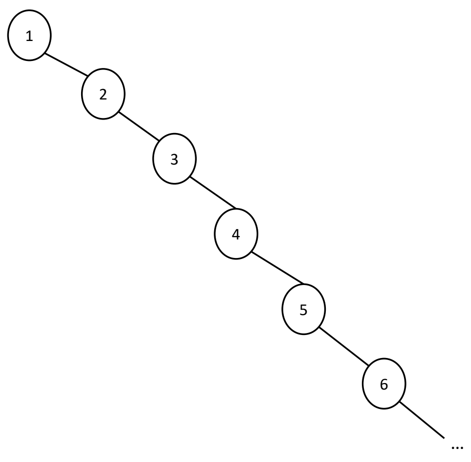
4. Show the binary search tree that results from inserting the following sequence of keys: 6, 3, 11, 9, 8, 5, 4, 2, 12, 7, 1, 10.

- The first element (6) is added at the root, and any subsequent elements are added by following the left branch when their value is lower than the root, or the right branch when it is greater. For example, the last element added was (10), which is larger than 6; so the right branch is followed and 11 is found. 10 is lower than 11, so the left branch is followed and 9 is found. 10 is greater than 9 and nothing is in 9's right branch, so then we place 10 there.



5. What kind of tree is produced if the same keys are inserted in ascending order? What if they are inserted in descending order?

- By placing the keys in order (ascending or descending), a "backbone" tree is produced. For example, in ascending order:

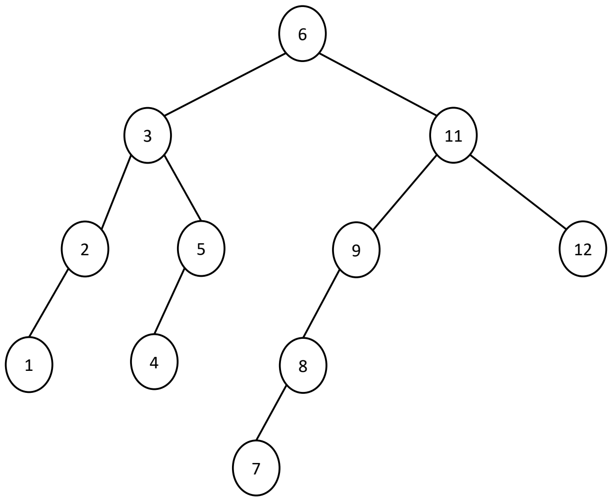

1
2
3
4
5
6
...

6. A possible sorting algorithm is to add all the keys of an array to a binary search tree and then read off its inorder traversal. Is this related to any of the sorting algorithms previously considered?

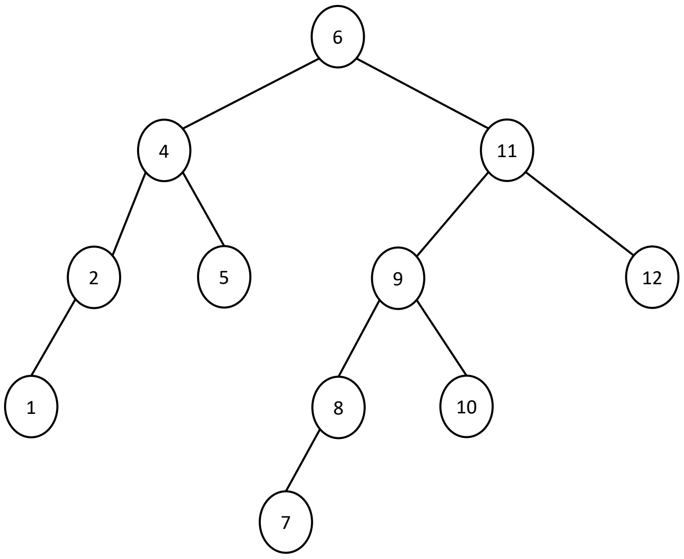
- When adding keys to the binary tree, what is being done can be seen as finding a pivot, placing it at the root, and partitioning the rest by placing the values lower than the root to the left, and the values larger than the root to the right. A pivot is found for each one of these subtrees and a similar partitioning process takes place. The procedure is thus similar to what quicksort does.

7. Show the tree that results from the tree built in exercise 4 if you delete10? What if you deleted 3 instead?

- Removing a leaf node is easy: the element is simply deleted.



- To remove the 3, some procedure should be followed. Since the 3 has the two branches, the “harder case” of deletion seen in the lecture is applied:



- As you may have realised, this is not the only possible way of doing the deletion. In your textbooks you may find other possible algorithms that will, for example, place the 2 where the 3 was, and have the (5) subtree.