Implement B-trees as described in class – notice that they are slightly different from the description in the text book, which actually presents what is known as a B plus tree.

Implement methods to the following operations:

1. Insert an element to the B-tree. Make sure you do not allow duplicate elements in the tree and recall that all insertions occur at the leaves.
2. Determine if a given element is in the tree.
3. Given a key k, find the successor of k, that is, the smallest key in the tree that is larger than k.
4. Print all the elements in the B-tree in ascending order.
5. Extract all the elements in the tree into a sorted array.
6. Return the minimum element in the tree.
7. Return the maximum element in the tree.
8. Return the number of nodes in the tree.
9. Return the number of keys in the tree.
10. Return the number of leaves in the tree.
11. Return the number of nodes in the tree that are full.
12. Return the height of the tree.
13. Print the contents of the tree ordered by depth, that is, first print the keys at depth 0, then the keys at depth 1, then the keys at depth 2, and so on. Hint: use a queue.

First build a B-tree using randomly-generated integers as input, and then test each of the methods using that tree. Write a main method that shows the functionality of your program and, as usual, write a report describing your program and experiments. Include in your report results using at least two different values of t.

Here’s an example showing the fields that a node must contain, together with a constructor for an empty leaf node.

```java
public class BTreeNode{
    public int[] key;
    public BTreeNode[] c;
    boolean isLeaf;
    public int n;
    private int t;

    public void BTreeNode(int tnode){
        t = tnode;
        isLeaf = true;
        key = new int[2*t-1];
        c = new BTreeNode[2*t];
        n=0;
    }
}
```