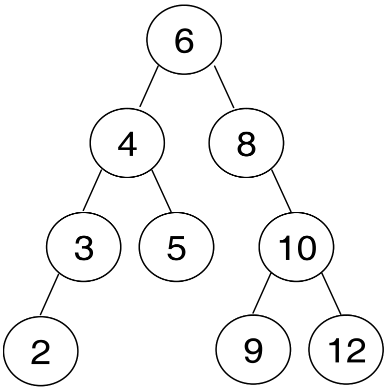

Figure 1

```
int arr[]={-9, -7, -5, -3, 0, 1, 3, 5, 7, 9}, i;
TreeEntry e;
Tree t;
CreateTree(&t);
for(i=0; i<10; i++){
    e.key=arr[i];
    InsertTree(&t, e);
}
```

Code 1.

1. A Binary Search Tree (BST)

- **a)** enables for a fast search.
- **b)** is a structure in which each node cannot have more than two children.
- **c)** is a structure whose root has no parent.
- **d)** has a root with a key value larger than the key value of any node to its left and smaller than the key value of any node to its right.
- **e)** all of the above.

2. A Preorder traversal (VLR) to the tree in Fig. 1 gives:

- **a)** 6, 4, 3, 2, 5, 8, 10, 9, 12.
- **b)** 2, 3, 4, 5, 6, 8, 9, 10, 12.
- **c)** 6, 2, 3, 4, 5, 8, 9, 10, 12.
- **d)** 6, 4, 8, 3, 5, 10, 2, 9, 12.
- **e)** non of the above.

3. Insertion of 7 in the tree in Fig. 1 will be:

- **a)** to the right of 5.
- **b)** to the right of 2.
- **c)** to the left of 8.
- **d)** to the left of 9.
- **e)** non of the above.

4. The depth of the tree in Fig. 1 will be increased by one if an element is inserted to

- **a)** the left of 2.
- **b)** the left of 9.
- **c)** all of the above
- **d)** to the left of 5.
- **e)** all of the above.

5. Using an iterative function to search for 9 in the tree of Fig. 1 requires visiting

- **a)** 9 elements.
- **b)** 8 elements.
- **c)** 5 elements.
- **d)** 4 elements.
- **e)** 1 elements.

6. To delete 8 from the tree of Fig. 1, then the subtree, whose root is 10, will be

- **a)** lost.
- **b)** the right subtree of 5.
- **c)** the left subtree of 5.
- **d)** the right subtree of 6.
- **e)** destroyed to save memory.

7. If a good `Delete` algorithm is used for deleting 6 from the tree of Fig. 1. Then,

- **a)** the left subtree of 6 should be moved to be the left subtree of 9.
- **b)** 2 should replace 6 and catch its two children.
- **c)** 9 should replace 6 and catch its two children.
- **d)** 5 should replace 6 and catch its two children.
- **e)** all of the nodes should be deleted because the root, which is their parent, is deleted.

8. Executing the code in Code 1 results in a tree that

- **a)** looks like a chain (List).
- **b)** has a depth of exactly the number of inserted elements.
- **c)** is not efficient for searching.
- **d)** all of the above.
- **e)** non of the above.

9. The recursive version of the function `InsertTree` is

- **a)** less efficient than the iterative one, because recursion is always less efficient than iteration.
- **b)** less efficient than the iterative one, because of building an unnecessary stack.
- **c)** more efficient than the iterative one, because recursion is always more efficient than iteration.
- **d)** more efficient than the iterative one because the iterative solution also builds a stack.
- **e)** exactly as efficient as the iterative one.