IMPERATIVE PROGRAMMING 2

QUESTION 4

(a)
```
class Tree (var datum: Int, var left: Tree, var right : Tree)
```

(b)
```
def traversal (t: Tree): Unit = if (t!= null) {
    traversal (t. left)
    print (t. datum + " ")
    traversal (t. right)
}
```

(c)
```
def MakeTree (a: Array [Int], a : Int, b: Int) : Tree = }
    if ((b-a) > 1)
    {
       var mid = (b-a)/2
       new Tree (a (mid), MakeTree (a, a, mid), MakeTree (a, mid+1, b))
    }
    else if ((b-a) ==1) new Tree (a(a), null, null)
    else null
}
```

(d)
```
def traversal it (t: Tree) : Unit = }
    var current : Tree = t
    while (current != null)
    {
       if (current. left == null) // no left-child
       {   print (current. datum + " ")
           current = current. right
       }
       else
       {
           var leftTree: Tree = current. left
           while ((leftTree . right != null) && (leftTree. right != current))
              leftTree = leftTree. right
```

```
        if (leftTree.right == null)
        {
            leftTree.right = current
            current = current.left
        }
        // restore, otherwise
        else
        {
            leftTree.right = null
            print(current.datum + " ")
            current = current.right
        }
    }
}
```

(e) Each vertex is traversed at most two times before being printed, therefore the algorithm is proportional in the size of the tree.