Algorithms and Data Structures
Solutions
June 2015

Q1. [Calculation for a new example]

a)

The correct code is:

```
int calculateF(int N){
    int result = 1;
    for (int i=1; i<N; i++)
        result = 2*result + 1;
    return result;
}
```

The errors are:

1. The function should return int and not being void
2. The return inside the for loop should be removed
3. The final return should return the result and not the temp
4. The for loop should increase by 1 and by 2
5. Result = 2*result+1 (2 and not 3)
6. The result should be initialized to 1.

[6]

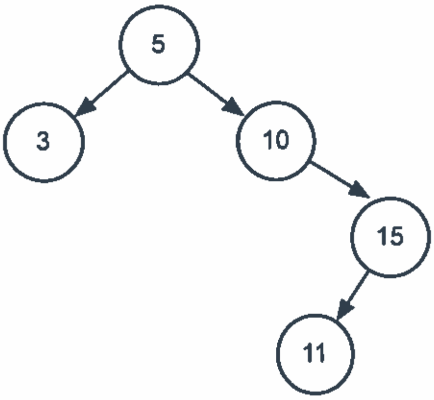
b)

The code is given below:

```
int calculateFR(int n){
    if (n==1)
        return 1;
    else return 2*calculateFR(n-1)+1;
}
```

[6]

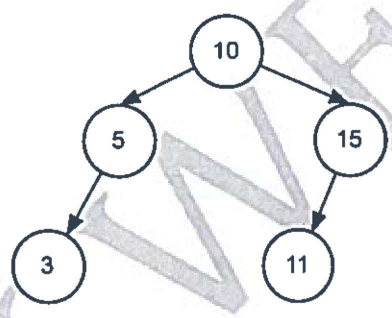
c) i)



The tree in unbalanced (node 10), as the right subtree has height 1, where the left has height -1

[2]

ii)



The AVL tree is a self-balanced tree. So, after each insertion, if the tree becomes unbalanced, the necessary rotations are performed to balance the tree.

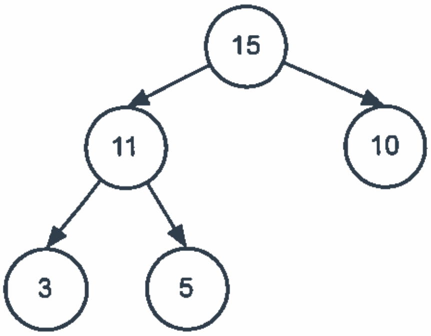
[6]

iii)

The depth of node 3 is 2.

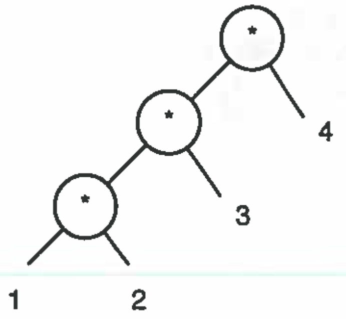
[2]

iv)



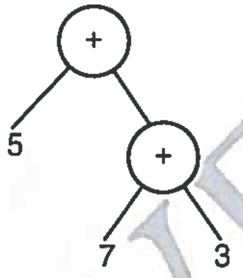
[2]

d) i)



[2]

ii)



[2]

e)

There is a memory leak at the point where p2=p1; [1 mark]

A: x=4, y = 10 [2 marks]

B: x=13, y=3 [2 marks]

[5]

f) i)

```
void returnLastNode(NodePtr hdList, NodePtr &Ptr) {
    Ptr = NULL; // in this case, there is no need to
initialise Ptr
    while (hdList!=NULL){
        Ptr = hdList;  // the pointer Ptr points to the
previous element of hdList
        hdList = hdList->next;
    }
}
```

[3]

ii)

```
void setValuesZeroExceptLast(NodePtr hdList) {
    while (hdList!=NULL){
        if (hdList->next!=NULL)
            hdList->data = 0;
        hdList = hdList->next;
    }
}
```

[4]

Q2) [New application]

a)

```
struct treeNode{
    int data;
    treeNode * left;
    treeNode * right;
    int lowerBound;
    int upperBound;
};

typedef treeNode * TNodePtr;
```

[5]

b)

```
void nodeNumber(TNodePtr hdTree, int & number) {
    if (hdTree != NULL){
        number++;
        nodeNumber(hdTree->left, number);
        nodeNumber(hdTree->right, number);
    }
}
```

[10]

c)

```
void nodeNumberInRange(TNodePtr hdTree, int LB, int UB, int &
number) {
    if (hdTree != NULL){
        if ((hdTree->data >= LB) && (hdTree->data <=UB))
            number++;
        nodeNumberInRange(hdTree->left, LB, UB, number);
        nodeNumberInRange(hdTree->right,LB, UB, number);
    }
}
```

[10]

d)

(it may seem complicated but it is based on the well known insert function, with the extra work to update the lower and upper bounds)

```
void insertNumber(TNodePtr &hdTree, int data) {
    if (hdTree==NULL){
        // Create a new node and update the hdTree
        TNodePtr temp = new treeNode;
        temp->data = data;
        temp->lowerBound = data;
        temp->upperBound = data;
        temp->left = NULL;
        temp->right = NULL;

        hdTree = temp;

    } else {
        if (hdTree->data > data)
        {
            // update the lowerbound
            if (hdTree->lowerBound > data)
                hdTree->lowerBound = data;

            insertNumber(hdTree->left, data);
        }
        else
        {
            // update the upperbound
            if (hdTree->upperBound < data)
                hdTree->upperBound = data;

            insertNumber(hdTree->right, data);
        }
    }
}
```

[10]

e)

```
void updateRange(TNodePtr hdTree){
    if (hdTree!=NULL) {
        updateRange(hdTree->left);
        updateRange(hdTree->right);

        // Update the range for the current node
        if (hdTree->left != NULL)
            hdTree->lowerBound = hdTree->left->lowerBound;
        else
            hdTree->lowerBound = hdTree->data;

        if (hdTree->right != NULL)
            hdTree->upperBound = hdTree->right->upperBound;
        else
            hdTree->upperBound = hdTree->data;
    }
}
```

[10]

f)

```
// You need to initialise maxDensity with zero.
void highestDensityNode(TNodePtr hdTree, TNodePtr &
highestDPtr, float &maxDensity) {
    if (hdTree != NULL) {
        // Find the density of the curren node
        int number = 0;
        nodeNumber(hdTree, number);

        int currentDensity = number / (hdTree->upperBound -
hdTree->lowerBound + 1.0);

        if (currentDensity >= maxDensity){
            maxDensity = currentDensity;
            highestDPtr = hdTree;
        }

        highestDensityNode(hdTree->left, highestDPtr,
maxDensity);
        highestDensityNode(hdTree->right, highestDPtr,
maxDensity);
    }
}
```

[15]