1. Assume a Binary Tree class has a member function call play and a tree has been built as shown.
What's the output of main? Show the execution tree.  

```
int BinTree::play() const {
    int n = 0;
    return helper(root, n);
}

int BinTree::helper(Node* current, int n) const {
    if (current == NULL)
        return n;
    if (current->right != NULL || current->left != NULL)
        return 1 + helper(current->left, n+1) + helper(current->right, n+1);
    return 0;
}
```

```
A,0:  1    +    L      +     R = 1+5+4 = 10
Output: 10
```

2. Give the complexity (tight big-oh) of the following. You need not show work.

(a). Find the smallest value in an AVL tree of n items.  O(log n)

(b). Find the smallest value in a binary heap of n items.  O(1)

(c). Insert one item into a binary heap of n items.  O(log n)

(d). Destructor for a binary search tree of n items.  O(n)

(e). Remove one edge in a graph of n nodes and E edges stored in an adjacency matrix.  O(1)

(f). The maximum number of unique edges in an undirected graph of n nodes.  O(n^2)

(g). Breadth-first ordering on a graph with n nodes and E edges stored in an adjacency matrix.  O(n^2)
3. Given the following characters and frequency of occurrence in a message, use the Huffman encoding algorithm to find a unique encoding for the characters. Show all work. (10 pts)

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>15</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
</tr>
<tr>
<td>d</td>
<td>22</td>
</tr>
<tr>
<td>e</td>
<td>30</td>
</tr>
</tbody>
</table>

Start with these two, a and b, then delete from list, add “ab 25”

These would be next, c and d, delete, add “cd 42”, and so on

Final answer: cdabe(97)

```
0 /       1 a 100
/          abe(55) b 101
/            0 /  
 cd(42) ab(25) \ 1 c 00
 / \       / \  e 11
 0 /   
 c(20) d(22) a(10) b(15) e(30)
```

4. Demonstrate the heap sort (done efficiently) on the values: 10 5 7 11 6 3 2 13 15 1 4. Show only the first three passes of the algorithm (three values are sorted). There is no code writing. Redraw your heaps when needed for clarity. (15 pts)

Build a heap using the O(n) algorithm (work is not shown):

```
Final heap: 1
   / 
  4 2
 / 
11 5
 / 
13 15 6
```

Three deleteMins (work is not shown) (highlighted values are at the end of the array, but not a part of the heap):

```
2
 / 
4 3
 / 
11 5
 / 
13 15 6
```

```
3
 / 
4 6
 / 
11 5
 / 
13 15 2
```

```
4
 / 
5 6
 / 
11 15
 / 
13 3
```

5. Given the following AVL tree structure (letters represent appropriate values).
Suppose a value was added as shown (in the P position).
Show the balanced tree after the insertion. Show each step. (5 pts)

A
 /       \
B       C
 /     /
D   E   F   G
\  \   \  \\  \
H  I  J  K  L
/ /\   /\   /\ \      /\ \
M N   O   K      L
/ \      /\      /\
P

Which node is unbalanced? B

Do a right-right rotation.

Final answer:
A
 /       \
E        C
 /   \    /   \
B     J   F   G
/   \   /   \   /   \   \
D   I   N   O   K      L
\  /   /     /     /     /\ 
H M   P

7. Consider your Polynomial class only the array content contains double: (10 pts)

```cpp
class Poly {
    friend ...

public:
    Poly(int coeff=0, int maxExp=0); // constructor, sets size=maxExp+1
    ~Poly(); // destructor
    Poly(const Poly &); // copy constructor
    void integrate(); // integrate poly
    ...

private:
    double* ptr; // pointer to first array element
    int size; // size of the array
};
```

Write the member function integrate which replaces the current polynomial with its integral (as in calculus). E.g.,

```cpp
Poly A(0,4);
cin >> A; // user enters values so A = +10x^4 -7x^2 +3
A.integrate();
cout << A << endl; // outputs: +2.00x^5 -2.33x^3 +3.00x
```

```cpp
void Poly::integrate() {
    double* temp = new double[size+1];
    temp[0] = 0.0;
    for (int i = 0; i < size; i++) {
        temp[i+1] = ptr[i]/(i+1);
    }
    delete [] ptr;
    ptr = temp;
    temp = NULL;
    size++;
}
```
8. Implement function(s) for a BSTree class that determine whether or not the tree is fully complete (all levels filled completely). Assume usual Node, with member data: NodeData* data, Node* left, Node* right

Any function you use, you must write. For example:

```
T1: A
    / 
   B C
  / \ /
 D E F G

// Sample main:
BSTree T1;
BSTree T2;
bool complete;

// T1 is complete:
complete = T1.isComplete(); // returns true

// T2 is incomplete:
complete = T2.isComplete(); // returns false
```

```
bool BSTree::isComplete() const {
  return depthBalanceHelper(root) != -1;
}
```

```
int BSTree::depthBalanceHelper(Node* current) const {
  if (current == NULL)
    return 0;

  int leftDepth = depthBalanceHelper(current->left);
  if (leftDepth == -1)
    return -1;

  int rightDepth = depthBalanceHelper(current->right);
  if (rightDepth == -1)
    return -1;

  if (leftDepth != rightDepth)
    return -1;

  return leftDepth+1;
}
```