1. (*10 Points*) State the precise mathematical definitions for $O(g(n))$, $\Omega(g(n))$, and $\Theta(g(n))$.

   Solution:

   $f(n)$ is $O(g(n))$ if there exist positive numbers c and N such that $f(n) \leq cg(n)$ for all $n \geq N$.

   $f(n)$ is $\Omega(g(n))$ if there exist positive numbers c and N such that $f(n) \geq cg(n)$ for all $n \geq N$.

   $f(n)$ is $\Theta(g(n))$ if there exist positive numbers c_1, c_2, and N such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq N$.

2. (*10 Points*) Prove that $f(n) = 2^{\sqrt{\lg(n)}}$ is $O(n^a)$ for any positive number a.

   Solution:

   We need to find numbers c and N such that $2^{\sqrt{\lg(n)}} \leq cn^a$ for all $n \geq N$ and any positive number a. Following the below series of inequalities,

$$
\begin{aligned}
2^{\sqrt{\lg(n)}} &\leq cn^a \\
\lg\left(2^{\sqrt{\lg(n)}}\right) &\leq \lg\left(cn^a\right) \\
\sqrt{\lg(n)} &\leq \lg(c) + a\lg(n) \qquad \text{let } c = 1 \\
\sqrt{\lg(n)} &\leq a\lg(n) \\
\frac{1}{a} &\leq \sqrt{\lg(n)} \\
\frac{1}{a^2} &\leq \lg(n) \\
2^{1/a^2} &\leq n
\end{aligned}
$$

   So we can let $c = 1$ and $N = 2^{1/a^2}$.

3. (*10 Points*) Find an exact closed form formula for the number of times the inner loop body is executed and state the computational complexity of the loop.

   (a)
```
for (int cnt = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= n; j++)
        cnt++;
```

   Solution: If we let t be the number of iterations of the outside loop then the outside loop is done as long as $2^t \leq n$, hence $t \leq \lg(n)$ and thus $t = \lfloor \lg(n) \rfloor$. The number of iterations of the outside loop is one more then this since the first iteration is when $i = 1 = 2^0$. So the total number of iterations of the outside loop is $\lfloor \lg(n) \rfloor + 1$. The inside loop is done n times for each iteration of the outside loop, hence the total number of times the inner loop body is done is $n(\lfloor \lg(n) \rfloor + 1)$. Since $n(\lfloor \lg(n) \rfloor + 1) = n\lfloor \lg(n) \rfloor + n \leq 2n\lfloor \lg(n) \rfloor \leq 2n\lg(n)$ the complexity is $O(n\lg(n))$.

   (b)
```
for (int cnt = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= i; j++)
        cnt++;
```

   Solution: The outside loop here is the same as the previous exercise and the last iteration of the loop is when $i = 2^{\lfloor \lg(n) \rfloor}$. The inside loop is done i times on each iteration of the outside i loop. So the number of times the inside loop body is done is,

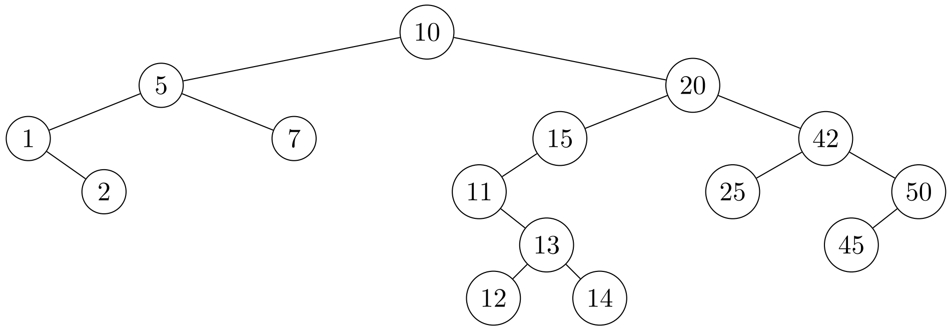
$$
1 + 2 + 4 + \cdots + 2^{\lfloor \lg(n) \rfloor} = \sum_{i=0}^{\lfloor \lg(n) \rfloor} 2^i = 2^{\lfloor \lg(n) \rfloor + 1} - 1
$$

   For the complexity note that $2^{\lfloor \lg(n) \rfloor + 1} - 1 < 2^{\lfloor \lg(n) \rfloor + 1} = 2 \cdot 2^{\lfloor \lg(n) \rfloor} \leq 2 \cdot 2^{\lg(n)} = 2n$ hence the complexity is $O(n)$.

4. (*10 Points*) Draw the following tree after

   (a) A delete by merging of 10. Use the successor node.

   (b) A delete by copy of 10. Use the predecessor node.



Solution:

Delete by Merge

```
        50
            45
    42
        25
20
    15
                14
            13
                12
        11
                7
            5
                    2
                1
```

Delete by Copy

```
            50
                45
        42
            25
    20
        15
                    14
                16
                    12
            11
7
    5
            2
        1
```

5. (*10 Points*) Implement the displayPreOrder recursive function for this class.

Solution:

```
template<class T>
void BinaryTree<T>::displayPreOrder(TreeNode *nodePtr) const {
    if (nodePtr) {
        cout << nodePtr->value << endl;
        displayPreOrder(nodePtr->left);
        displayPreOrder(nodePtr->right);
    }
}
```

6. (*10 Points*) Write both the specification and implementation of an iterative preorder display function.

   Solution:

```
void iterativePreorder();

template<class T>
void BinaryTree<T>::iterativePreorder() {
    deque<TreeNode*> stack;

    TreeNode *nodePtr = root;
    if (nodePtr) {
        stack.push_back(nodePtr);
        while (!stack.empty()) {
            nodePtr = stack.back();
            stack.pop_back();
            cout << nodePtr->value << endl;
            if (nodePtr->right)
                stack.push_back(nodePtr->right);
            if (nodePtr->left)
                stack.push_back(nodePtr->left);
        }
    }
}
```

7. (*10 Points*) Write both the specification and implementation of a recursive function to count the number of leaves in a binary tree. Include a non-recursive function that initiates the recursive version that could be called from the main.

   Solution:

```
int numLeaves();
int numLeavesrec(TreeNode*);

template<class T>
int BinaryTree<T>::numLeaves() {
    return numLeavesrec(root);
}

template<class T>
int BinaryTree<T>::numLeavesrec(TreeNode *t) {
    if (!t)
        return 0;
    else if (!t->left && !t->right)
        return 1;
    else
        return numLeavesrec(t->left) + numLeavesrec(t->right);
}
```

8. (*10 Points*) Write both the specification and implementation of an iterative function to count the number of leaves in a binary tree.

   Solution: You can use the iterative preorder code from the previous exercise and simply change the visit function to a leaf counter. Another approach would be to use the slightly simpler breath first traversal of the tree with a leaf counter for the visit function.

```
int numLeaveIter();

template<class T>
int BinaryTree<T>::numLeaveIter() {
    int count = 0;
    deque<TreeNode*> queue;
    TreeNode *t = root;

    if (t) {
        queue.push_back(t);
        while (!queue.empty()) {
            t = queue.back();
            queue.pop_back();
            if (!t->left && !t->right)
                count++;
            if (t->left)
                queue.push_back(t->left);
            if (t->right)
                queue.push_back(t->right);
        }
    }
    return count;
}
```

9. (*10 Points*) Write both the specification and implementation of a function that will load the values of a tree into a vector so that the resulting vector is sorted. If you write this recursively include a non-recursive function that initiates the recursive version that could be called from the main.

   Solution:

```
void loadVector(vector<T>&);
void loadVectorRec(TreeNode*, vector<T>&);

template<class T>
void BinaryTree<T>::loadVector(vector<T> &v) {
    loadVectorRec(root, v);
}

template<class T>
void BinaryTree<T>::loadVectorRec(TreeNode *t, vector<T> &v) {
    if (t) {
        loadVectorRec(t->left, v);
        v.push_back(t->value);
        loadVectorRec(t->right, v);
    }
}
```

10. (*10 Points*) Write both the specification and implementation of a function that will create a complete binary tree with each node holding the level it is at, root level will be 1, it's children 2, and so on. The function should take in a single parameter that specifies the height of the final complete tree. You may assume that the tree is empty before calling this function. If you write this recursively include a non-recursive function that initiates the recursive version that could be called from the main.

Solution:

```
void buildCompleteTree(int);
void buildCompleteTreeRec(TreeNode*&, int, int);


template<class T>
void BinaryTree<T>::buildCompleteTree(int ht) {
    buildCompleteTreeRec(root, 1, ht);
}

template<class T>
void BinaryTree<T>::buildCompleteTreeRec(TreeNode *&t, int h, int max) {
    if (h > max)
        return;

    TreeNode *newNode = new TreeNode;
    newNode->value = h;
    t = newNode;
    buildCompleteTreeRec(t->left, h + 1, max);
    buildCompleteTreeRec(t->right, h + 1, max);
}
```