Group members:

- Titas Geryba tg1404
- Shunya Watanabe sw3212
- Christine Dah-In Chung cic266

Problem 1

(1). Prove that the number of leaves in a complete binary tree with n nodes is $\Omega(logn)$.

For this problem we have to consider the way nodes are inserted into the tree. Since the tree has to be complete, we have to fill in each depth of the tree one at a time, going from left to right. This means that upon addition of a new node we have to first make sure the rightmost node in the lowest depth without two children has to be filled. When considering such a node we have two cases. If the node has 0 children, then the addition of a new node increases the count of leaves by 0, as the parent node is no longer a leaf, but its new left-child now is. In the other case we have that the node has 1 child. Now the addition of a new node to its children creates a right-node, which has nothing but its parent attached to it, meaning it is a leaf. Thus the second addition increases the leaf count by 1.

The number of leaves L can thus be described as follows in terms of n (the number of nodes in the tree):

$$L(n) = 0 \text{ when } n = 0, 1$$

$$L(n) = \lfloor \frac{n+1}{2} \rfloor \text{ for } n > 1$$

$L(n) = 0$ when $n = 1$ since the root alone is not considered a leaf. Then for all n $>$ 1 we have:

$$L(n) = \lfloor \frac{n+1}{2} \rfloor \geq \frac{n}{2} = \Omega(n)$$

implying that:

$$\lfloor \frac{n+1}{2} \rfloor = \Omega(n)$$

(2). Prove that the height of a complete binary tree with n nodes is $O(logn)$.

We begin by examining each node. In a complete binary search tree all of the depths of the tree before the lowest depth d of the tree are filled. Since at each depth greater than $d-1$ we have that each node branches of twice, we can write:

$$\text{Number of nodes up to and including depth d-1} = \sum_{i=0}^{d-1} 2^i = 2^{d-1} - 1$$

Now since all node spots available at depth d are not necesseraly filled and each node at depth $d-1$ can branch out twice, we have that the total number of nodes including depth d has to be:

$$2^{d-1} \leq \text{Number of nodes up to and including depth d} \leq 2^d - 1$$

From here we can see that the function that models this behavior is

$$d = \lfloor log(n) \rfloor$$

And following this:

$$d = \lfloor log(n) \rfloor < log(n) = O(logn)$$

Implying that

$$\lfloor log(n) \rfloor = O(logn)$$

(3). Prove that the average height of the nodes in a complete binary tree is $O(1)$.

By our earlier consideration in problem 1.2 we have seen that each depth $i < d$ (where d is the lowest depth) in a complete binary tree has 2^i nodes. By the definition of the average height and the fact that the highest depth has a height of 0, we know that nodes at depth d will contribute 0 to the average. With n nodes we have then:

$$nE[X] = \sum_{i=0}^{d} 2^i(d-i) = \sum_{i=0}^{\lfloor log(n) \rfloor} 2^i(\lfloor log(n) \rfloor - i)$$
$$= \lfloor log(n) \rfloor \sum_{i=0}^{\lfloor log(n) \rfloor} 2^i - \sum_{i=0}^{\lfloor log(n) \rfloor} 2^i i$$

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

Now we examine S(t) = $\sum_{i=0}^{k} it^i$:

$$S(t) = t + 2t^2 + ... + (k-1)t^{k-1} + kt^k$$
$$tS(t) = t^2 + 2t^3 + ... + (k-1)t^k + kt^{k+1}$$
$$(1-t)S(t) = t(1 + t + t^2 + ... + t^{k-1}) - kt^{k+1}$$
$$S(t) = \frac{t(\frac{1-t^k}{1-t}) - kt^{k+1}}{1-t}$$

We now let $t = 2$:

$$S(2) = \frac{2(\frac{1-2^k}{-1}) - k2^{k+1}}{-1} = 2(1-2^k) + k2^{k+1}$$
$$= 2 + 2^{k+1}(k-1)$$

Now letting $k = \lfloor log(n) \rfloor$:

$$2 + 2^{k+1}(k-1) = 2 + 2^{\lfloor log(n) \rfloor + 1}(\lfloor log(n) \rfloor - 1)$$

We have then that:

$$nE[X] = \lfloor log(n) \rfloor (2^{\lfloor log(n) \rfloor + 1} - 1) - 2 - 2^{\lfloor log(n) \rfloor + 1}(\lfloor log(n) \rfloor - 1)$$
$$= \lfloor log(n) \rfloor 2^{\lfloor log(n) \rfloor + 1} - \lfloor log(n) \rfloor - 2 - \lfloor log(n) \rfloor 2^{\lfloor log(n) \rfloor + 1} + 2^{\lfloor log(n) \rfloor + 1}$$
$$= -\lfloor log(n) \rfloor - 2 - +2^{\lfloor log(n) \rfloor + 1}$$
$$\leq -\lfloor log(n) \rfloor - 2 - +2^{log(n)+1}$$
$$= \lfloor log(n) \rfloor - 2 - +2n$$

We finally have

$$E[X] < \frac{\lfloor log(n) \rfloor - 2 - +2n}{n} = 2 - \frac{2}{n} - \frac{\lfloor log(n) \rfloor}{n} < 2$$

Implying that:

$$E[X] = O(1)$$

Problem 2

The idea here is to have a part of the tree that causes the depth of the tree to be larger than its average. A tree that we have observed having depth that is not O(logn) is a one consisting of a straight line of nodes going down. For this reason we will make the left subtree of our binary tree a list going downwards, each node in this subtree having only one child. If this tree contains the deepest node of the tree, the depth of the tree becomes the number of nodes in this subtree, let us denote it k_1.

Now we examine the average depth that the left subtree will have. Since every node in that tree has a weight of its depth, we simply have:

$$\frac{1}{n}\sum_{i=1}^{k_1} i = \frac{k_1(k_1+1)}{2n} = \frac{k_1^2+k_1}{2n}$$

This suggests that, in order for the list to remain $O(logn)$ we have to have $k_1^2 = O(nlogn)$. At the same time we must have $k_1 \neq O(logn)$. For this reason, the function $k_1 = \sqrt{nlogn}$ is chosen. $\sqrt{nlogn}$ is not $O(logn)$ because:

$$\log n < n \Rightarrow \log^2 n < n \log n \Rightarrow \log(n) < \sqrt{n \log n}$$

Now we examine the right subtree, with the leftover $n - k_1 - 1 = k_2$ nodes. Here we begin assembling a complete tree, making it out of the k_2 nodes as described in problem 1. Analogous to problem 1.3, This means that the average depth of this tree is described by:

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{\lfloor log(k_2)\rfloor} 2^i i &= \frac{2 + 2^{\lfloor log(k_2)\rfloor+1}(\lfloor log(k_2)\rfloor - 1)}{n}\\ &\leq \frac{2 + 2^{log(k_2)+1}(log(k_2)-1)}{n}\\ &= \frac{2 + 2k_2log(k_2) - 2k_2}{n} < K\end{aligned}$$

for some constant K This happens because all of the terms in the final expression are bounded by a constant, so their sum is also bounded by a constant.

So now that the average depth is:

$$\begin{aligned}E[X] &\leq \frac{k_1^2+k_1}{2n} + \frac{2+2k_2log(k_2)-2k_2}{2n}\\ &< \frac{k_1^2+k_1}{2n} + K = \frac{nlogn + \sqrt{nlogn}}{2n} + K\\ &= \frac{logn}{2} + \frac{\sqrt{nlogn}}{2n} + K = O(logn)\end{aligned}$$

So we see that the average depth is $O(logn)$ while the depth is $O(\sqrt{nlogn})$ which is greater than $O(n)$.

Problem 4

In these two loops we have the following occur. The first loop is run n times as i is incremented by one each iteration. The second loop runs a total of $\lceil\frac{n}{i}\rceil$ as the loop stops after $\lceil\frac{n}{i}\rceil - 1$ increments of j by i. We have then:

$$\sum_{i=1}^{n}\lceil\frac{n}{i}\rceil$$

We can rewrite it as follows:

$$\sum_{i=1}^{n}\lceil\frac{n}{i}\rceil = \sum_{i=1}^{n}\left[\lceil\frac{n}{i}\rceil - \frac{n}{i} + \frac{n}{i}\right]$$

Letting:

$$a_i = \lceil\frac{n}{i}\rceil - \frac{n}{i}$$

We have:

$$\sum_{i=1}^{n}\lceil\frac{n}{i}\rceil = \sum_{i=1}^{n}\frac{n}{i} + \sum_{i=1}^{n}a_i = n + \sum_{i=2}^{n}\frac{n}{i} + \sum_{i=2}^{n}a_i$$

First we examine $\sum_{i=2}^{n}\frac{n}{i}$:

$$\sum_{i=2}^{n}\frac{n}{i} = n\sum_{i=2}^{n}\frac{1}{i} < n\int_{1}^{n}\frac{1}{x}dx = nlog_e(n)$$

Now we look at $\sum_{i=1}^{n}a_i$. Since $a_i \leq 1$ we have that

$$\sum_{i=1}^{n}a_i \leq \sum_{i=1}^{n}1 = n$$

Finally:

$$\sum_{i=1}^{n}\lceil\frac{n}{i}\rceil = \sum_{i=1}^{n}\frac{n}{i} + \sum_{i=1}^{n}a_i < nlog_e(n) + n + n$$

Implying that the upper bound for time complexity is $O(nlogn)$.

We have then also that:

$$\sum_{i=1}^{n}\lceil\frac{n}{i}\rceil \geq \sum_{i=1}^{n}\frac{n}{i} > n\int_{1}^{n}\frac{1}{x}dx = nlog_e(x)$$

Thus we also have that the lower bound for time complexity is $\Omega(nlogn)$

We conclude then that the code runs in $\Theta(nlogn)$ time.