* **[Question 2]**

~~Starts from any vertex in the BFS is a linear time algorithm, which has time complexity O(V+E). Based on the theorem in the textbook, there will be E + 1 edges in the graph.~~

* **[Question 3]**

Since the question cares about the non-empty binary tree only, thus:

* The base case is: a tree with height 1 (one root node that has one or two leaves)
* The inductive hypothesis is: for any tree with height k ≥ 1, the number of nodes with two children is exactly one less than the number of leaves.
* The inductive step is: prove that for a tree with height k + 1, the number of nodes with two children is exactly one less than the number of leaves.

Now prove the base case. So, if a tree has height 1, then there are two cases:

Case 1: the root node has one child.

Then, there is 0 node with two children, and 1 leave.

Case 2: the root node has two children.

Then, there are 1 node with two children, and two leaves.

So the hypothesis holds in the base case.

Now prove when the tree has height k + 1:

Case 1: the root node has one child.

Then, both the number of nodes that have two children, and the number of leaves, are the same to the previous step when the height is k. Since the hypothesis holds true when height is k, then when height is k + 1 the hypothesis still holds true.

Case 2: the root node has two children.

Let us denote N = number of nodes with two children.

And since the root node has two children as well, thus:

N of such tree = N of its left subtree + N of its right subtree + 1

Let us denote L = total number of leaves. Thus:

L of such tree = L of its left subtree +L of its right subtree.

And, since the hypothesis holds true when height is k, thus:

L of such tree = N of its left subtree + 1 + N of its right subtree + 1

Therefore, we can see that:

N of such tree + 1= L of such tree

* **[Question 5]**

If *n* operations total, then there are power of 2s, and let *k* denote that. Moreover, since the others cost 1 only, thus the total cost is:

So, the cost of each operation is:

, but we let (approximately), then we have:

=

Therefore, the amortized cost of each operation is .

* **[Question 6]**

Each time when the table with size *n* is full, we need to:

1. Create a new table with size *n + 2* , which has the cost *n + 2*
2. Copy the content of the old table to this new table, which has total cost *n*
3. Then, for the future *2* insertions, each will have cost 1 until the table is full again.

Thus the amortized cost of each is:

Therefore, the amortized cost of each operation is .

* **[Question 7]**

We can solve the problem by doing the following:

1. Sort all edges in decreasing order. Let T be the top-most edge (with the largest weight) in this sequence, and B be the bottom-most edge (with the smallest weight) in the sequence.
2. Start from the edge in the middle *m*, which , do binary search:
   1. If there exists a loop from one vertex of *m* that can loop back to the other vertex of *m*, and along the loop can go through the vertices *s* and *t*, and all edges along this loop have larger weight than the starting edge, then let edge *m* be the new *B*.
   2. Otherwise, let edge *m* be the new *T - 1.*
   3. Repeat until .

Once done the above, then *B* is the answer we need. The time complexity is O()