**Semi-Complete Binary Search Tree Analysis**

One more definition

A perfect binary tree is a full binary tree in which all leaves are at the same depth or same level. A perfect tree always has **2n – 1** nodes.

Variables

**n**: Input (Number of nodes to construct the semi-complete binary search tree).

**N**: Largest possible number such that **N+1** is a power of 2 and **N** is less than **n** (Number of nodes in the largest possible perfect binary tree having less than **n** nodes).

**m**: Number of leaves of a perfect binary tree of **N** nodes **(N + 1) / 2**.

**d**: **n** - **N**.

Solution

Calculate the values of **N**, **m** and **d**, then calculate

Proof

Given a perfect binary tree with **m** leaves, the next level of the tree has **2 \* m** leaves.

To insert a new node to the tree, choose 1 of the **2 \* m** positions. To insert two new nodes to the tree, choose 2 of the **2 \* m** positions. To insert x new nodes, choose x of the **2 \* m** positions.

**1) Adding d new leaves to a perfect binary tree with m leaves gives different binary tree structures.**

When **m** = 4, there are exactly new tree structures after adding **1** node.

**2) Given a structure of a binary tree, there is exactly one way to label the nodes in order to obtain a binary search tree.**

*Why?!*

If **L** is the number of nodes in the left sub-tree, the value of root is **L+1**, since there are exactly **L** values less than root’s value. Hence, root can have exactly one value. By induction, all other nodes can have exactly one value. So there exists exactly one labeling to each tree structure to make it a binary search tree.

Constraints

* **n** < 128
* **N** <= 63
* **m** <= 32
* **d** <= 64
* **2 \* m** = 64
* Values of (**m**, **d**) that maximizes the = (32, 32)
* = 1,832,624,140,942,590,534 < 264 – 1
* Calculation of can result in an internal overflow. Some optimization is needed to prevent that.