Construct all possible BSTs for keys 1 to N

In this article, first count of possible BST (Binary Search Trees)s is discussed, then construction of all possible BSTs.

How many structurally unique BSTs for keys from 1..N?

```
For example, for N = 2, there are 2 unique BSTs
     1               2
      \             /
       2           1

For N = 3, there are 5 possible BSTs
  1              3        3         2      1
    \           /        /        /  \      \
     3        2         1        1    3      2
    /       /             \                   \
   2      1                2                   3
```

We strongly recommend you to minimize your browser and try this yourself first.

We know that all node in left subtree are smaller than root and in right subtree are larger than root so if we have ith number as root, all numbers from 1 to i-1 will be in left subtree and i+1 to N will be in right subtree. If 1 to i-1 can form x different trees and i+1 to N can from y different trees then we will have x*y total trees when ith number is root and we also have N choices for root also so we can simply iterate from 1 to N for root and another loop for left and right subtree. If we take a closer look, we can notice that the count is basically n'th Catalan number. We have discussed different approaches to find n'th Catalan number here.

How to construct all BST for keys 1..N?

The idea is to maintain a list of roots of all BSTs. Recursively construct all possible left and right subtrees. Create a tree for every pair of left and right subtree and add the tree to list. Below is detailed algorithm.

```
1) Initialize list of BSTs as empty.
2) For every number i where i varies from 1 to N, do following
......a)  Create a new node with key as 'i', let this node be 'node'
......b)  Recursively construct list of all left subtrees.
......c)  Recursively construct list of all right subtrees.
3) Iterate for all left subtrees
   a) For current leftsubtree, iterate for all right subtrees
        Add current left and right subtrees to 'node' and add
        'node' to list.
```

Below is C++ implementation of above idea.

```
// A C++ prgroam to contrcut all unique BSTs for keys from 1 to n
#include <iostream>
#include<vector>
using namespace std;

//  node structure
struct node
{
    int key;
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp =  new node;
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}
```

```
// A utility function to do preorder traversal of BST
void preorder(struct node *root)
{
    if (root != NULL)
    {
        cout << root->key << " ";
        preorder(root->left);
        preorder(root->right);
    }
}

//  function for constructing trees
vector<struct node *> constructTrees(int start, int end)
{
    vector<struct node *> list;

    /*  if start > end   then subtree will be empty so returning NULL
        in the list */
    if (start > end)
    {
        list.push_back(NULL);
        return list;
    }

    /*  iterating through all values from start to end  for constructing\
        left and right subtree recursively  */
    for (int i = start; i <= end; i++)
    {
        /*  constructing left subtree   */
        vector<struct node *> leftSubtree  = constructTrees(start, i - 1);

        /*  constructing right subtree  */
        vector<struct node *> rightSubtree = constructTrees(i + 1, end);

        /*  now looping through all left and right subtrees and connecting
            them to ith root  below  */
        for (int j = 0; j < leftSubtree.size(); j++)
        {
            struct node* left = leftSubtree[j];
            for (int k = 0; k < rightSubtree.size(); k++)
            {
                struct node * right = rightSubtree[k];
                struct node * node = newNode(i);// making value i as root
                node->left = left;              // connect left subtree
                node->right = right;            // connect right subtree
                list.push_back(node);           // add this tree to list
            }
        }
    }
    return list;
}

// Driver Program to test above functions
int main()
{
    // Construct all possible BSTs
    vector<struct node *> totalTreesFrom1toN = constructTrees(1, 3);


    /*  Printing preorder traversal of all constructed BSTs   */
    cout << "Preorder traversals of all constructed BSTs are \n";
    for (int i = 0; i < totalTreesFrom1toN.size(); i++)
    {
        preorder(totalTreesFrom1toN[i]);
        cout << endl;
    }
    return 0;
}
```

Output:

```
Preorder traversals of all constructed BSTs are
1 2 3
1 3 2
2 1 3
3 1 2
3 2 1
```