# Day 5

28 November 2022 19:39

Given a string `s`, you can transform every letter individually to be lowercase or uppercase to create another string.

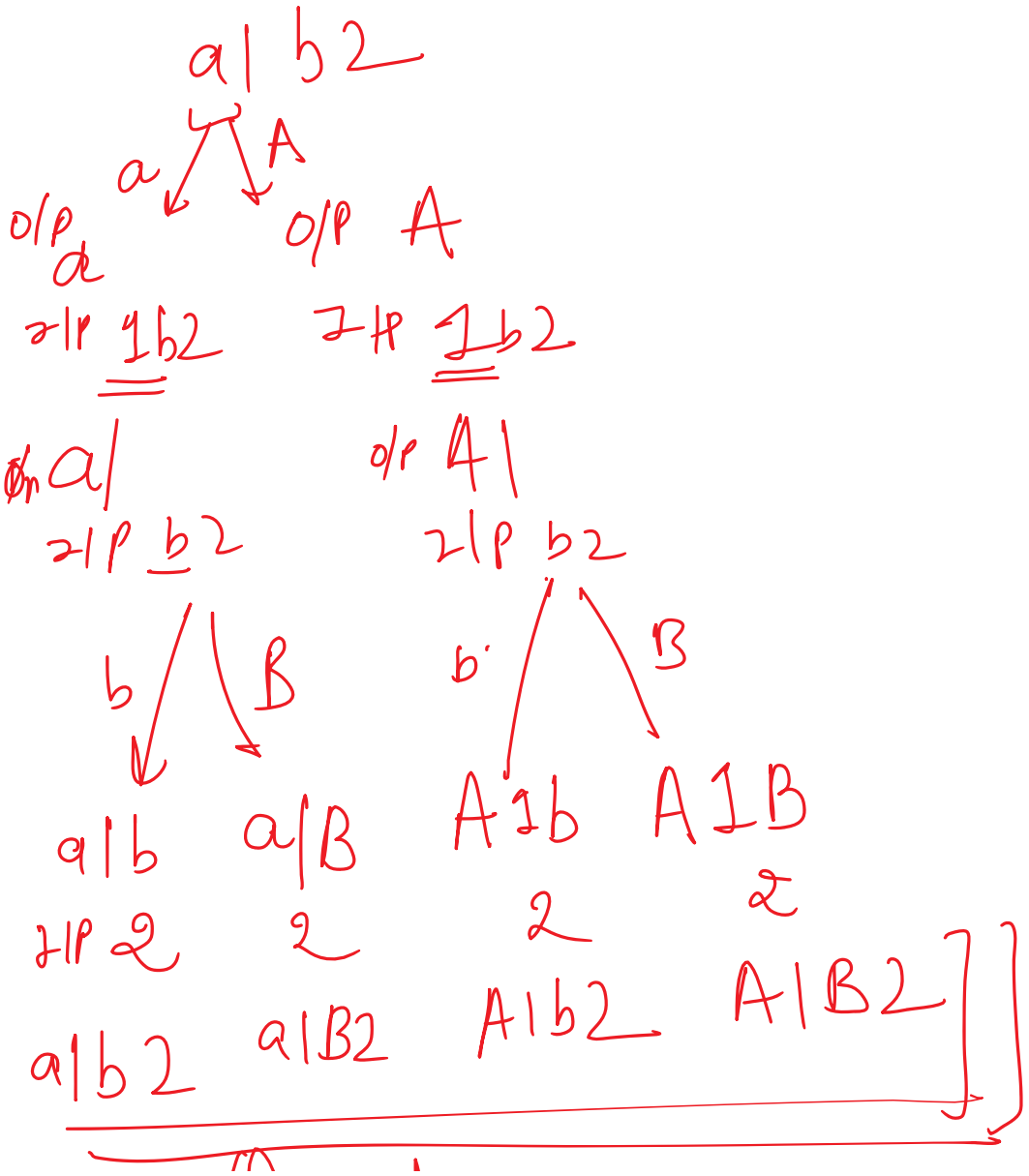
Return *a list of all possible strings we could create*. Return the output in **any order**.

**Example 1:**

```
Input: s = "a1b2"
Output: ["a1b2","a1B2","A1b2","A1B2"]
```

**Example 2:**

```
Input: s = "3z4"
Output: ["3z4","3Z4"]
```



# Questions

space

a

Not

A

Permutation

```
public:
bool ifenglishcharacter(char ch){
    if((ch >= 'a' && ch <= 'z') or (ch >= 'A' && ch <= 'Z')){
        return true;
    }
    return false;
}
    void solve(int i,string &temp_output,vector<string> &Permuatation,string &input_string){
        // base case
        if(i == input_string.size()){
            Permuatation.push_back(temp_output);
            return;
        }
        if(ifenglishcharacter(input_string[i])){
            char lower_char = tolower(input_string[i]);
            char upper_char = toupper(input_string[i]);
            temp_output += lower_char;
            solve(i+1,temp_output,Permuatation,input_string);
            temp_output.pop_back();
            temp_output += upper_char;
            solve(i+1,temp_output,Permuatation,input_string);
            temp_output.pop_back();
        }else{
            temp_output += input_string[i];
            solve(i+1,temp_output,Permuatation,input_string);
            temp_output.pop_back();
        }
    }
    vector<string> letterCasePermutation(string s) {
        vector<string> Permuatation;
        string temp_output;
        solve(0,temp_output,Permuatation,s);
        return Permuatation;
    }
};
```

T.C - O(2^N)  S.C - O(N)

पूछकर आये तब

# Generate Parenthesis

Given `n` pairs of parentheses, write a function to *generate all combinations of well-formed parentheses.*

**Example 1:**

```
Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]
```

**Example 2:**

```
Input: n = 1
Output: ["()"]
```

n=1

( )

जो open हुआ उसको closing भी दो ||

( )

n = 1

( )

n = 2

(( ))

opening = 1 closing = 2

( )

(( ))

opening = 1
closing = 1

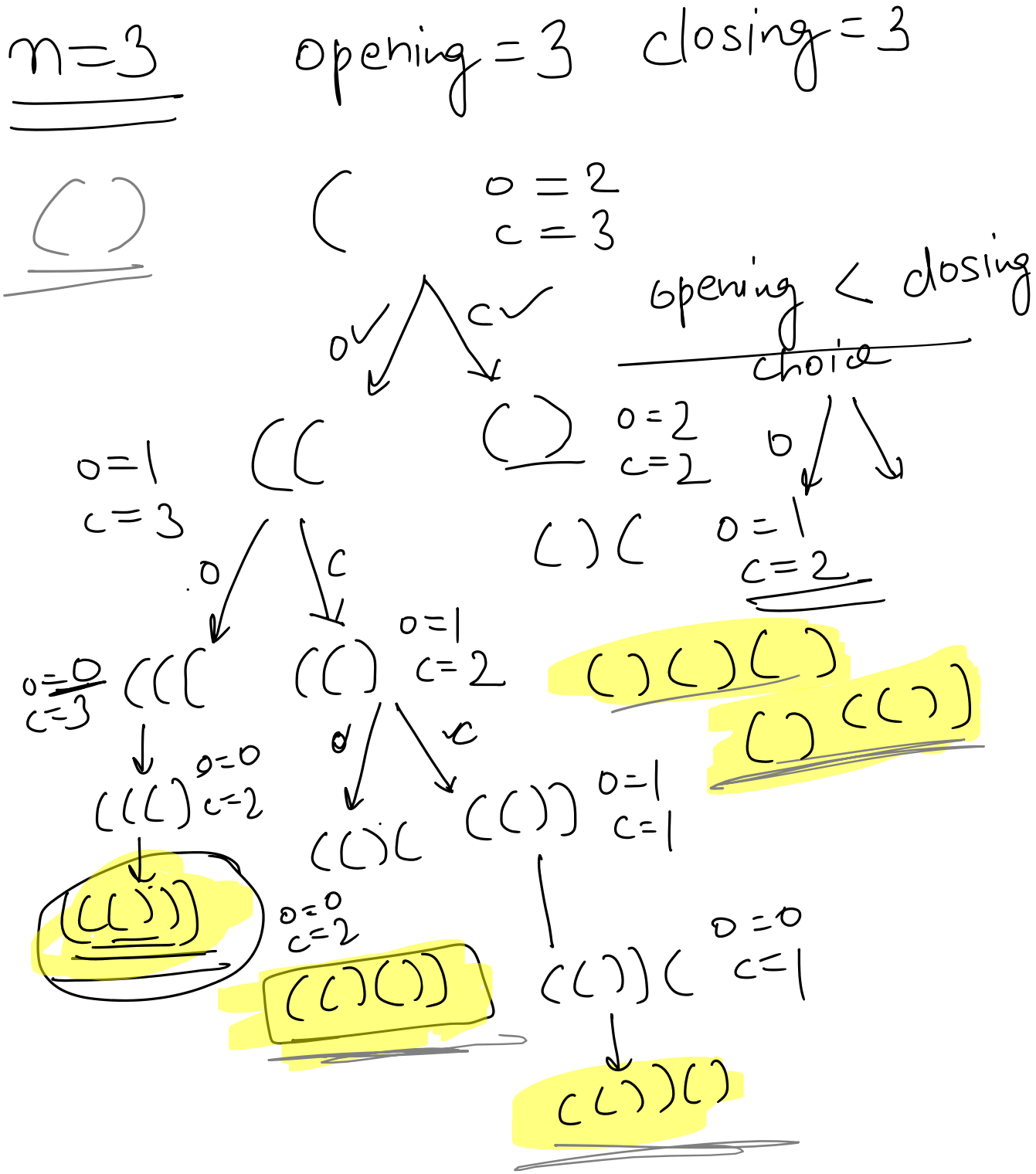
( )

( )

n = 2

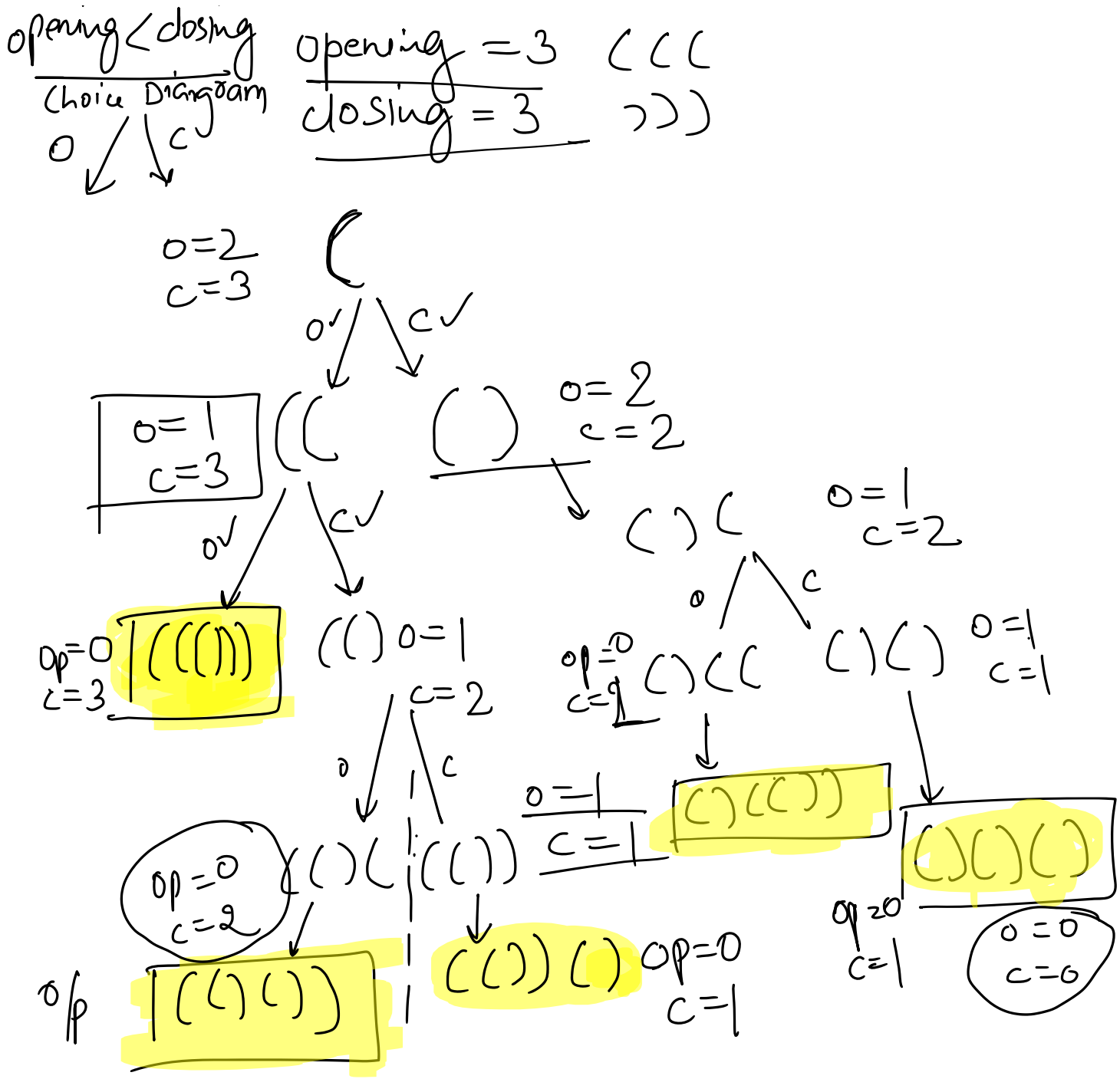
(( ))

( )( )




n=3

opening<closing opening=3 (((


opening < closing
Choice Diagram
opening = 3
closing = 3
(((
)))
o
c
o=2
c=3
(
o✓
c✓
o=1
c=3
((
( )
o=2
c=2
o✓
c✓
op=0
c=3
((()))
(() o=1
c=2
o
c
( )(
o=1
c=2
o
c
op=0
c=1
( )((
( )( )
o=1
c=1
op=0
c=2
( )(( ))
o=1
c=1
( )(( ))
( )( )( )
op=20
c=1
o=0
c=6
op
( )( )
( ( )) ( )
op=0
c=1

```
public:
    void solve(int opening, int closing, int n, vector<string> &Parenthesis,string &temp_output){
        if(opening == 0 and closing == 0){
            Parenthesis.push_back(temp_output);
            return;
        }
        //choice diagram
        if(opening == closing){
            temp_output += '(';
            solve(opening - 1,closing,n,Parenthesis,temp_output);
            temp_output.pop_back();
        }
        else if(opening < closing and opening!= 0){
            // opening
            temp_output += '(';
            solve(opening-1,closing,n,Parenthesis,temp_output);
            temp_output.pop_back();
            // closing
            temp_output += ')';
            solve(opening,closing-1,n,Parenthesis,temp_output);
            temp_output.pop_back();
        }else{
           // closing
           temp_output += ')';
           solve(opening,closing-1,n,Parenthesis,temp_output);
           temp_output.pop_back();
        }
    }
    vector<string> generateParenthesis(int n) {
       vector<string> Parenthesis;
       string temp_output;
       int opening  = n,closing  = n;
       solve(opening,closing,n,Parenthesis,temp_output);
       return Parenthesis;
    }
```

T.C- O(2^N)  S.C- O(N)

## Print N-bit binary numbers having more 1s than 0s

Medium Accuracy: **56.08%** Submissions: **11975** Points: **4**

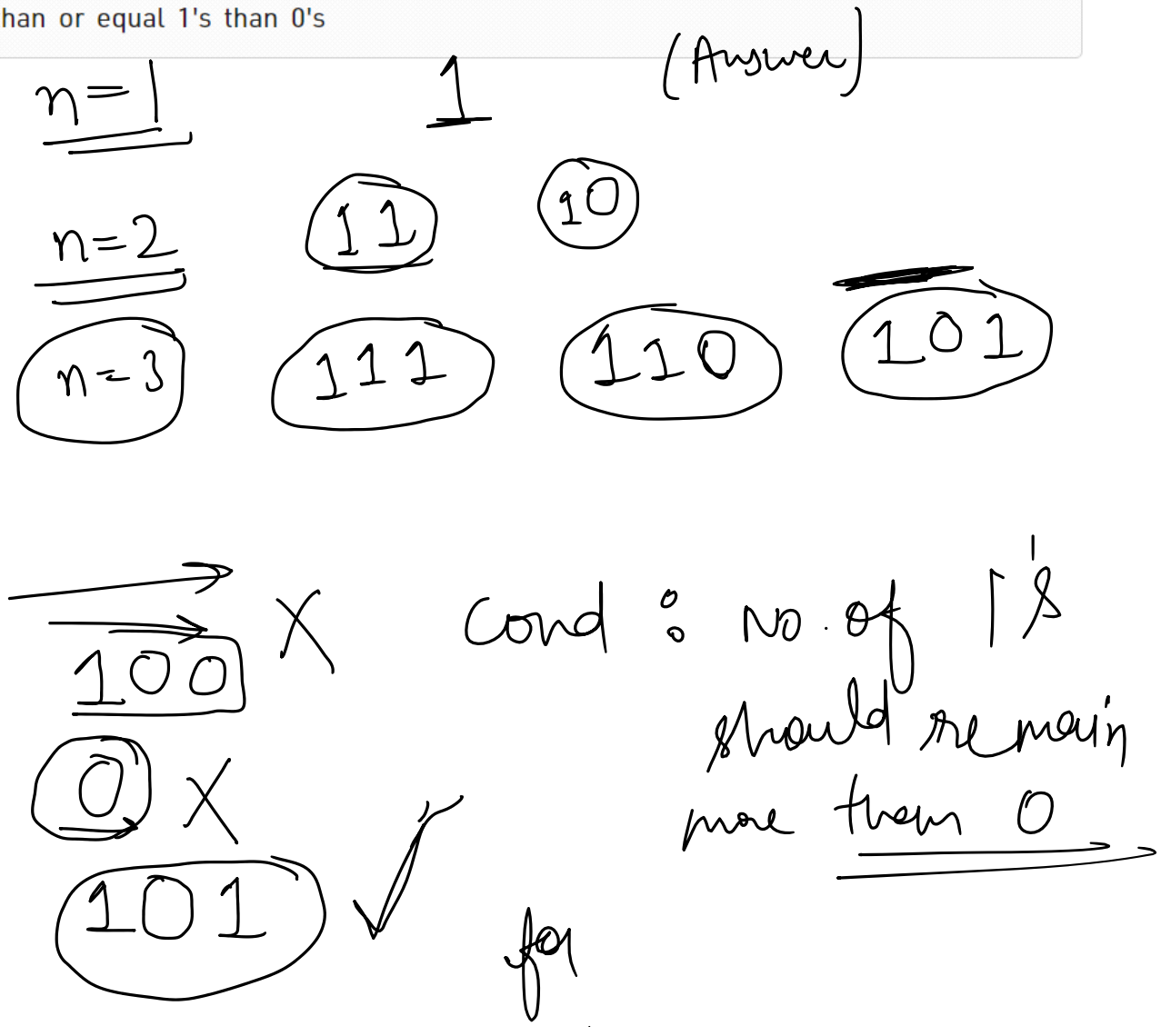
Given a positive integer **N**, the task is to find all the N bit binary numbers having more than or equal 1's than 0's for any prefix of the number.

**Example 1:**

**Input:** N = 2
**Output:** 11 10
**Explanation:** 11 and 10 have more than or equal 1's than 0's

**Example 2:**

**Input:** N = 3
**Output:** 111 110 101
**Explanation:** 111, 110 and 101 have more than or equal 1's than 0's

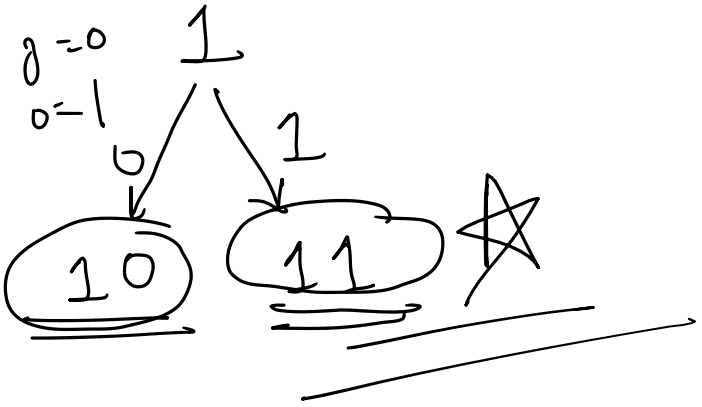


$n=1$ → 1

$n=2$ → 0 X

zero = 0
one = 0 ]

zero < one
0 1

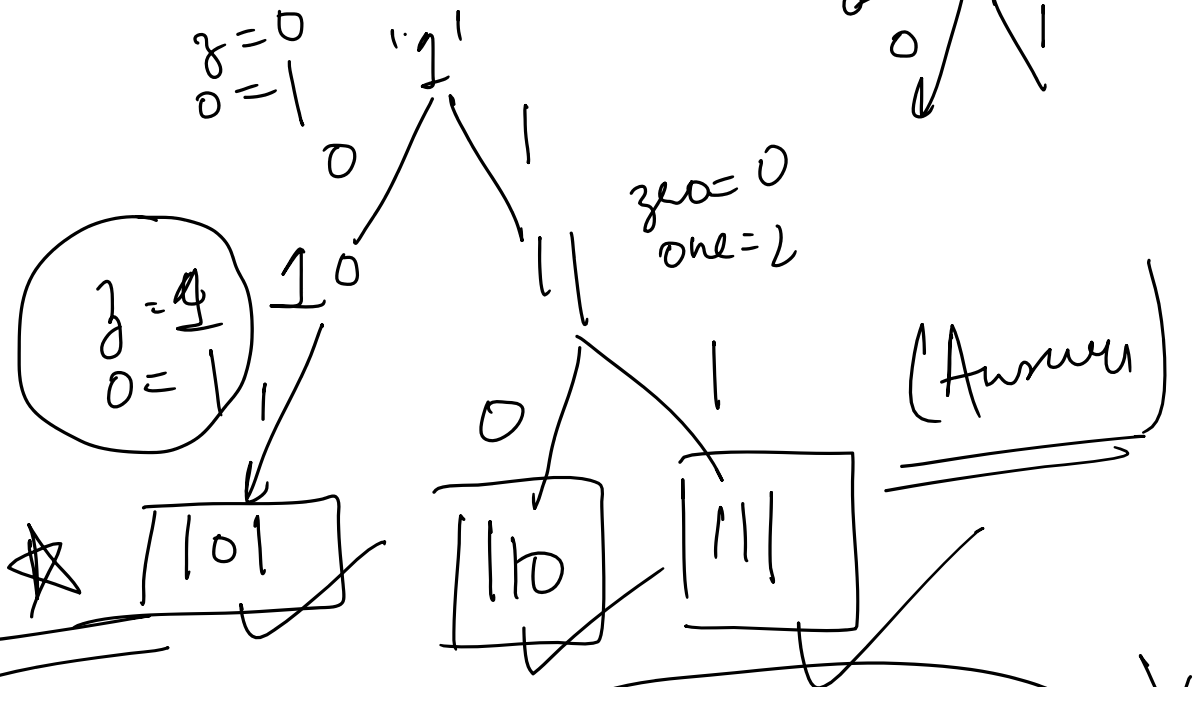



$n=3$

zero = 0
ones = 0

zero == one

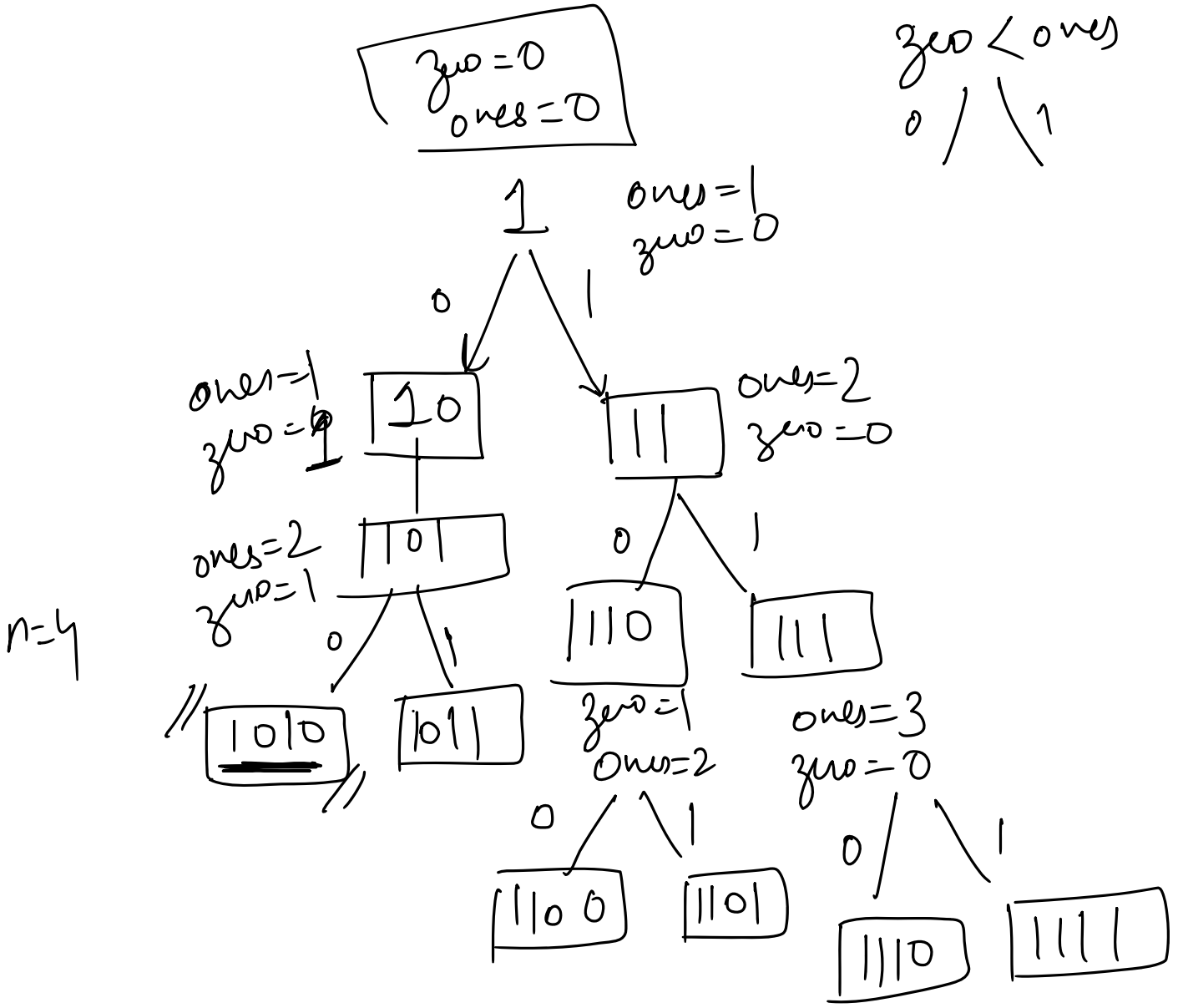
zero < one
0 1



n=4

→ n 0's > n 1's

n 1's ≥ n 0's



```
void solve(string &temp_output,int n,vector<string> &output,int ones, int zeros){
    if(temp_output.size() == n){
        output.push_back(temp_output);
        return;
    }
    // choice diagram
    if(ones == zeros){
        // ones
        temp_output.push_back('1');
        solve(temp_output,n,output,ones+1,zeros);
        temp_output.pop_back();
    }else{
        // ones
        temp_output.push_back('1');
        solve(temp_output,n,output,ones+1,zeros);
        temp_output.pop_back();
        // zeros
        temp_output.push_back('0');
        solve(temp_output,n,output,ones,zeros+1);
        temp_output.pop_back();
    }
}
vector<string> NBitBinary(int N)
{
    vector<string> output;
    string temp_output;
    solve(temp_output,N,output,0,0);
    return output;
}
```

T.C - O(2^N)  S.C - O(N)