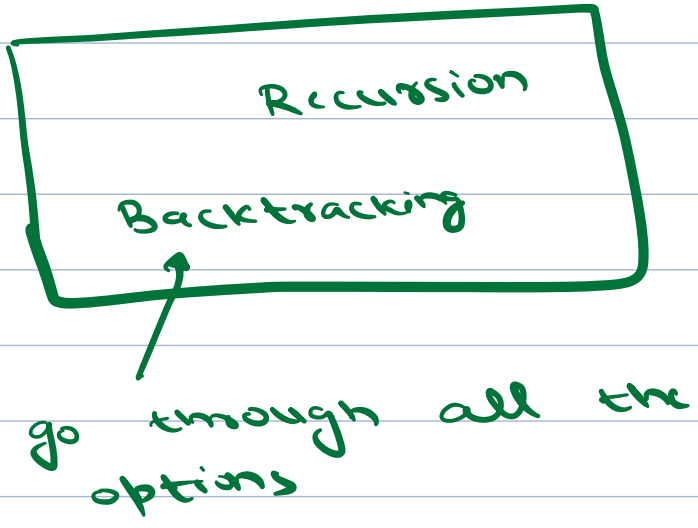
AGENDA

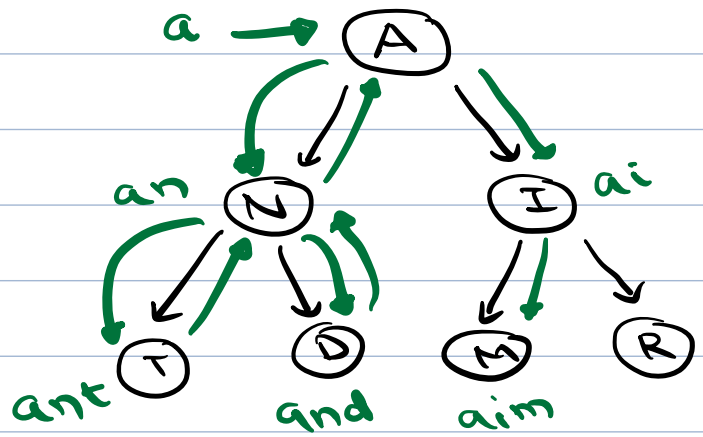
- Backtracking
- Print Valid Parenthesis
- Subsets
- Permutations



Backtracking → Technique to try multible solutions recursively by trying to build a solution, one piece at a time and neglecting any invalid solution.

Given a set of words represented in the form of a tree. Tree is formed such that every path ends in a word.

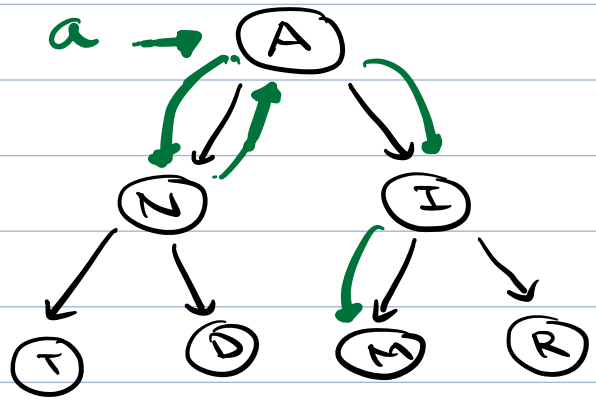
" "



Search for the word 'AIM'

Naive Approach: Traverse every path in tree, Once we reach an end node, then we check our word = target

---



Search for the word 'AIM'

0 1 2
A I M

ans = true

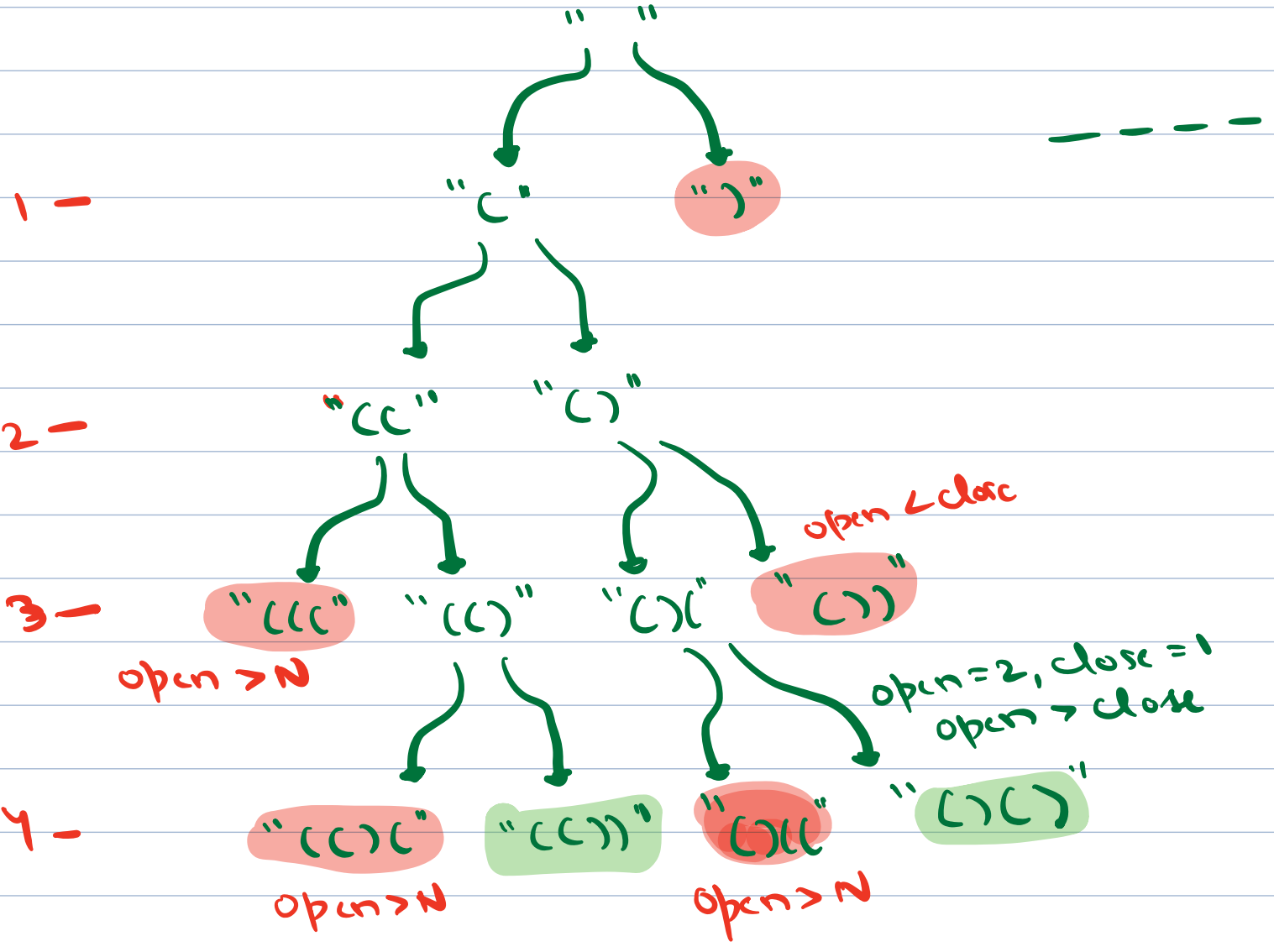
1. Given N, print all valid parenthesis of N pairs.

N = 1 ( )

((
))
( )
) (

N = 2 ( ) ( )
(( ))

)) ((
) ( ) (
) (( )



① opening → closing and vice versa

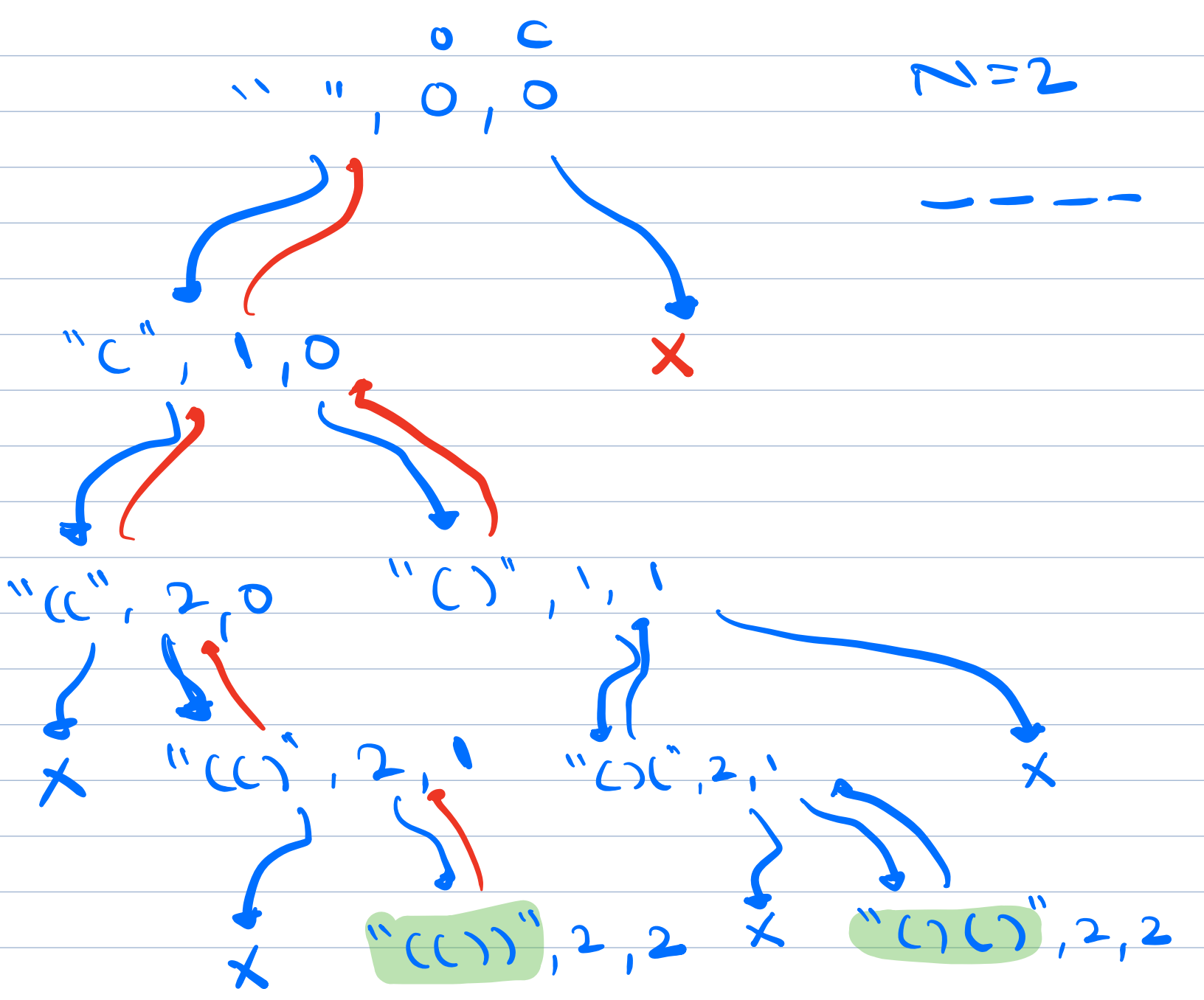
② opening = closing

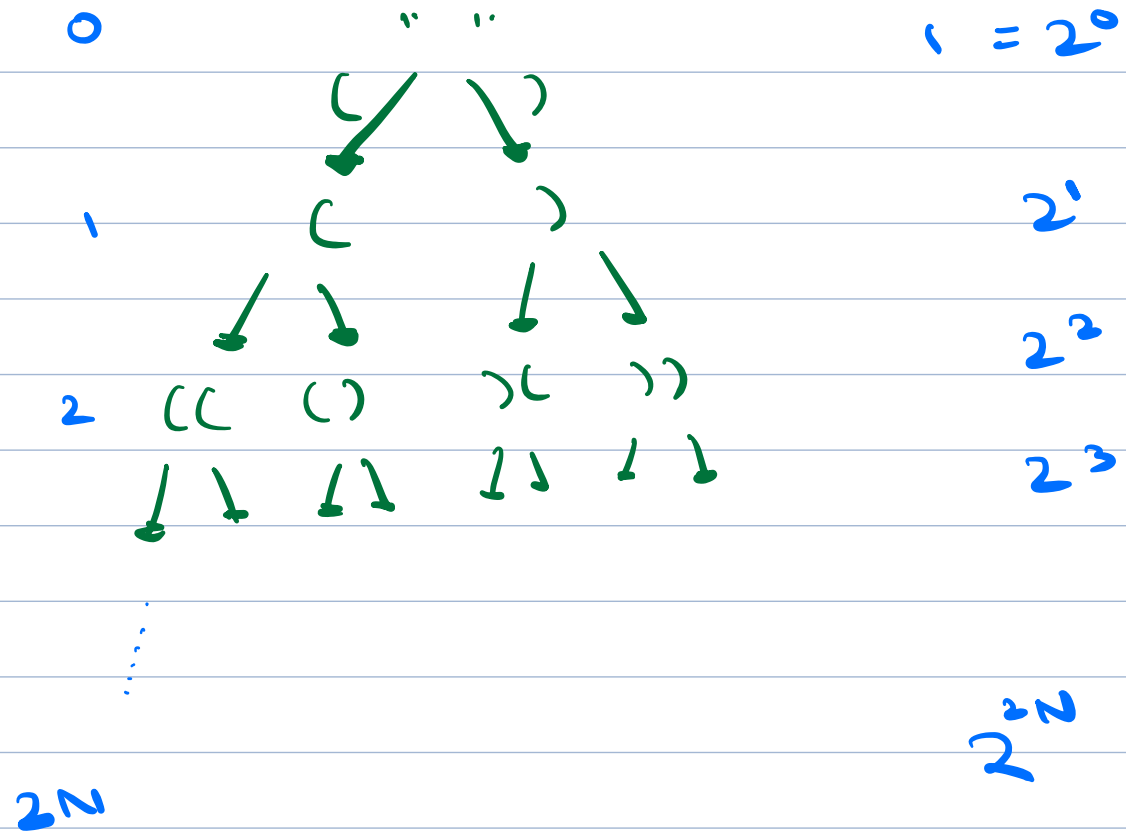
```
void generate (string s, int N, int o, int c) {
    if (s.length() == 2N) {
        print(s)        return
    }
    // add an opening bracket
    if (o < N)
     generate (s+"(", N, o+1, c)

    // add a closing bracket
      if (o > c)
       generate(s+")", N, o, c+1)

}

generate("", N, 0, 0)
```


o c
"", 0, 0
N=2
"(", 1, 0
X
"((", 2, 0
"()", 1, 1
X
"(()", 2, 1
"()(", 2, 1
X
X
"(())", 2, 2
X
"()()", 2, 2



Total fn calls = $2^0 + 2^1 + 2^2 + \dots + 2^{2N}$

$TC = O(2^{2N})$

$SC = O(N)$ 2N function calls on 1 path

10:18

https://notability.com/n/2anmcilplDH8KjCqT6e6oV

Subset → collection of continuous/non-continuous el

A = [1 5 2] No ordering

<7>	<1, 5>
<1>	<5, 2>
<5>	<1, 2>
<2>	<1, 5, 2>

ans = 8

N size array → 2^N subsets

A = [1 5 2 6 3]

<1 2 6 3> ✓

<2 4> ✗

<5 6 3> ✓
<6 3 5> ✓
<3 5 6> ✓

Same subset

[1 5 2]

✗ ✓ ✗ ✓ ✗ ✓

2 . 2 . 2 = 2^3 subsets

Subsequence → ① Collection of continuous / non-continuous el

② order of el should be same as array

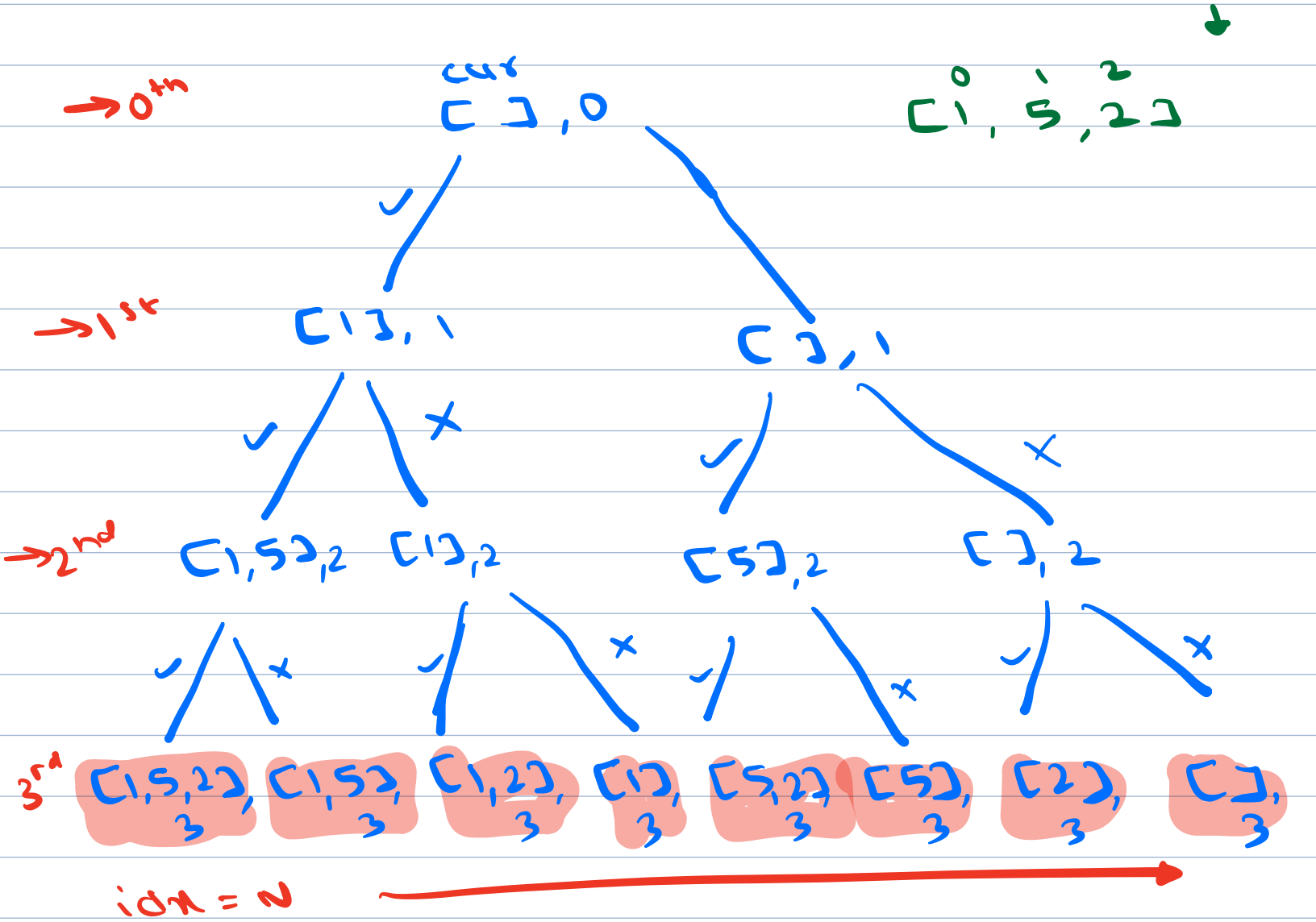
A = [1 2 6 5]

<1 2 5> subseq ✓

<5 1 2> subseq ✗

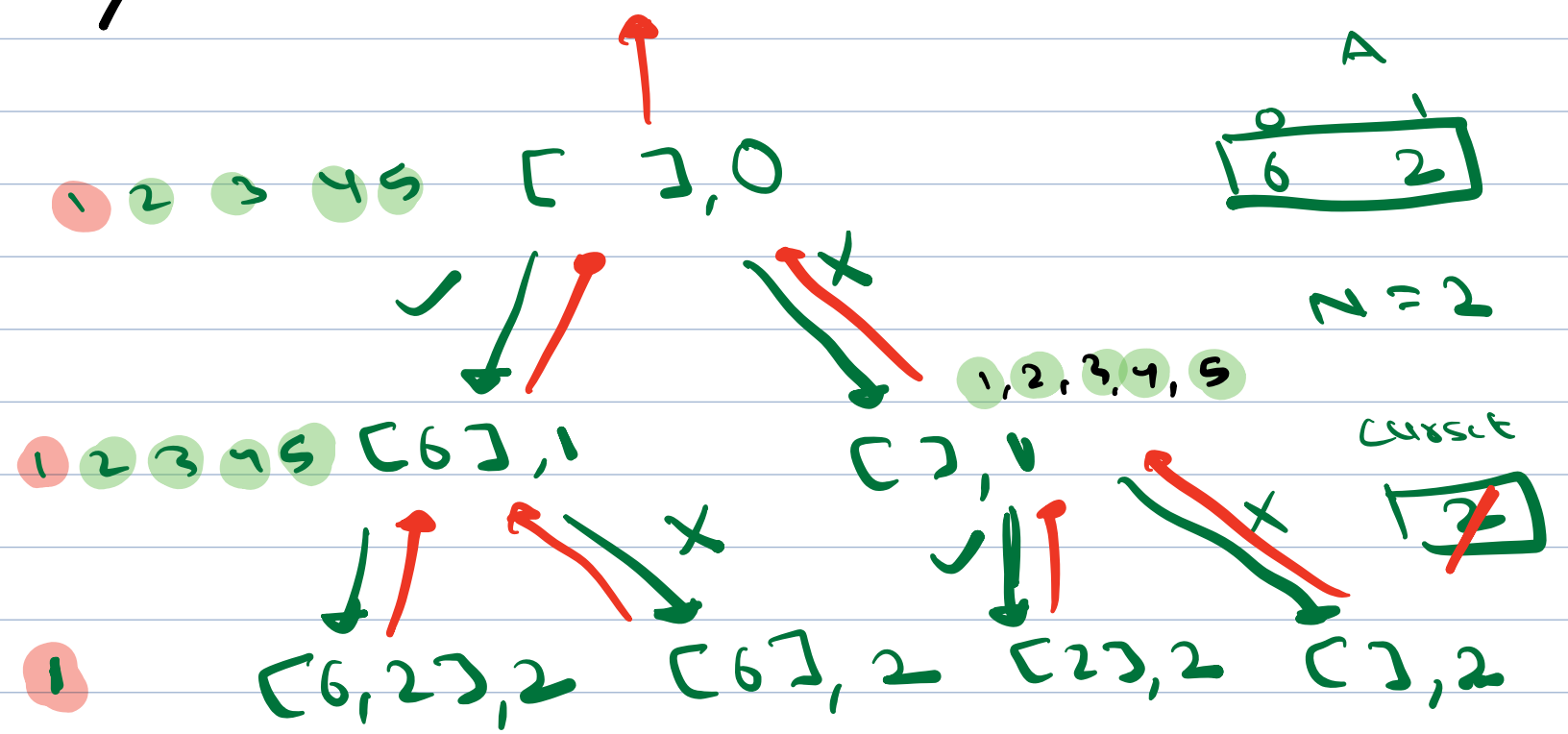
N size array → 2^N subsequence

2. Given an array of distinct integers, generate all subsets using recursion.

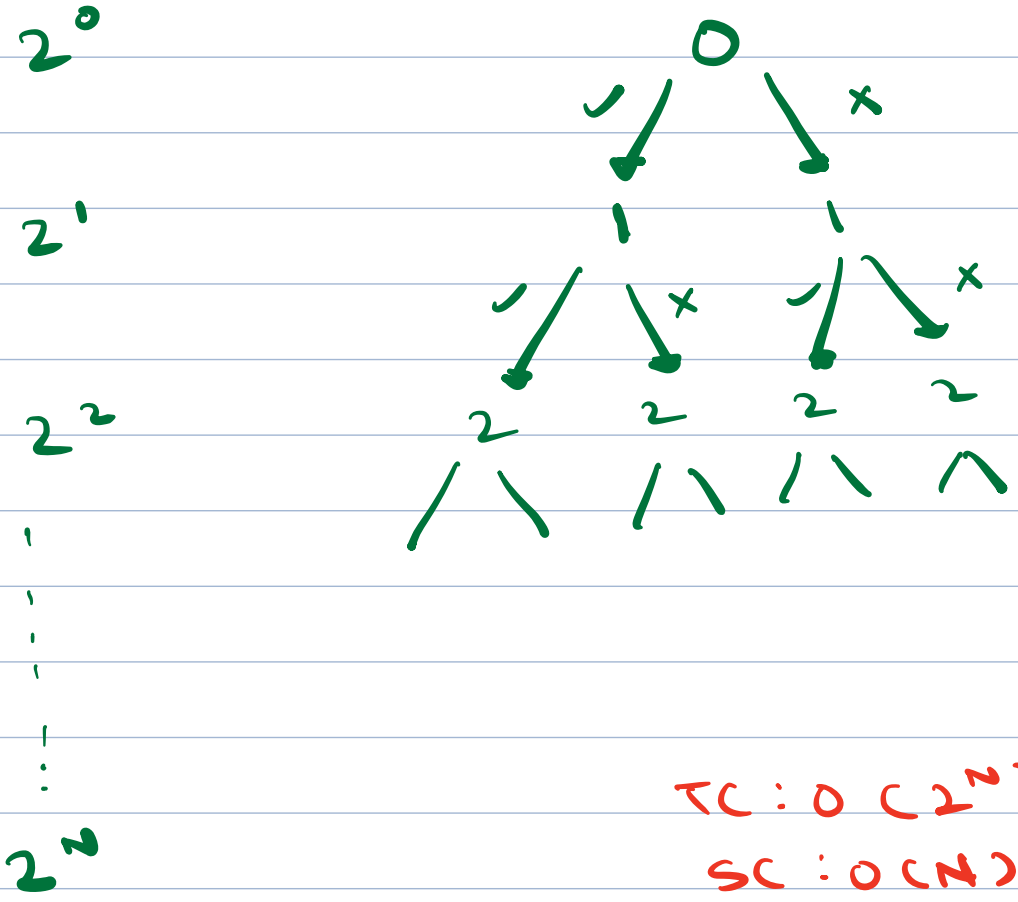


list < list <int>> ans

```
void Subsets (int A[], int N, int idx, list<int> curset){
    if (idx == N) {
        print (curset)        return
    }
    // include idx elt in subset
    curset.add (A[idx])
    Subsets (A, N, idx+1, curset)
    curset. removeLast()
    // exclude idx elt in subset
    Subsets (A, N, idx+1, curset)
}
```



O/P : [6,2] [6] [2] [ ]



Ex : a b c

Permutations

↓

Arrangements

N = 3 ans = 6

a b c
a c b
b a c
b c a
c a b
c b a

$3 \times 2 \times 1 = 6$

a < b c

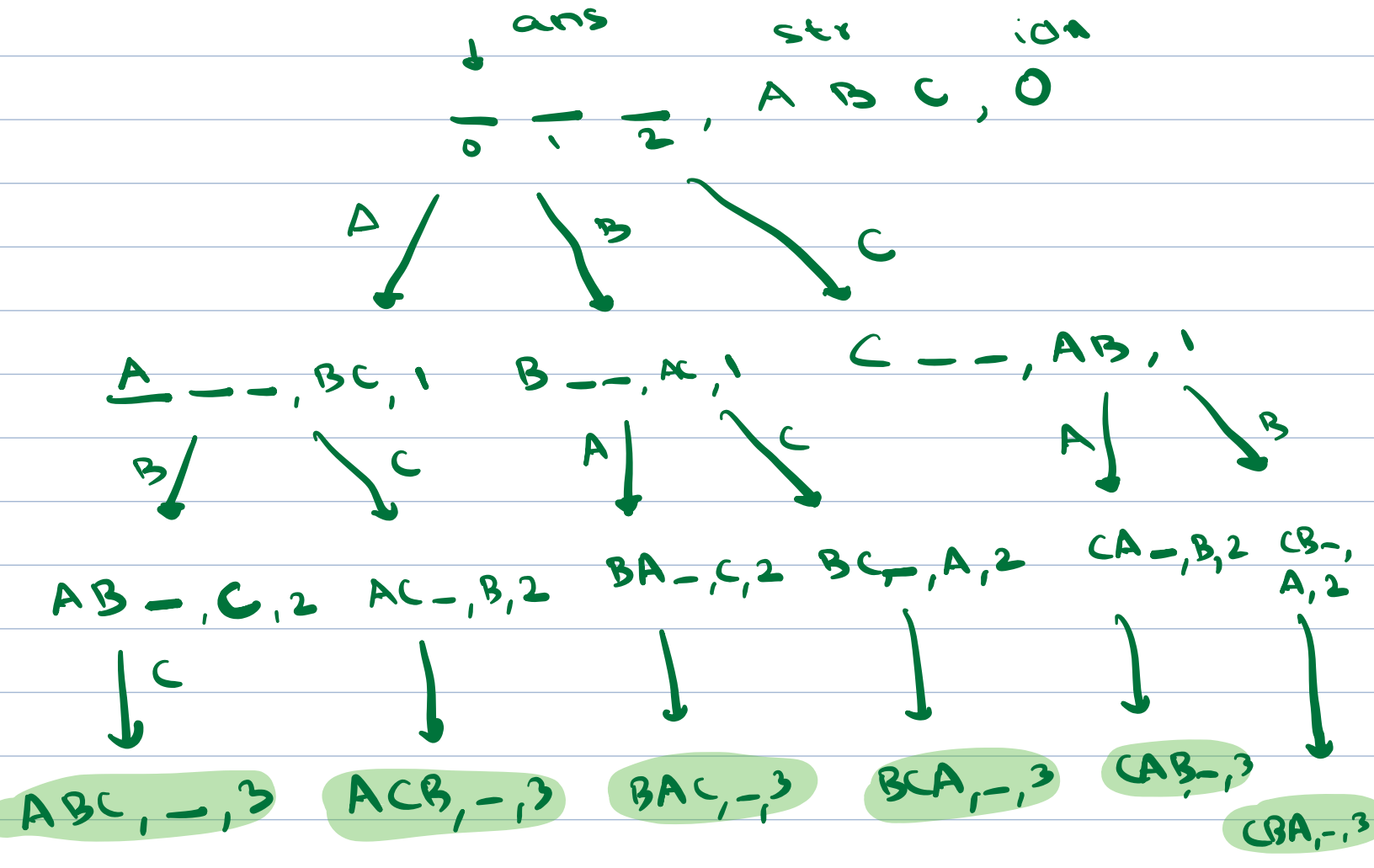
b < c a

c

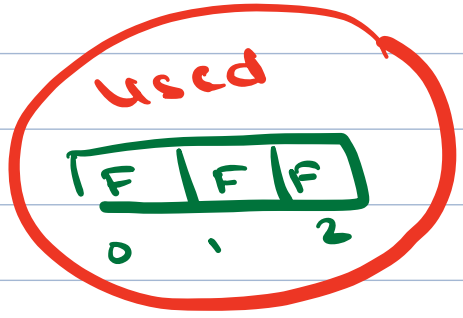
$N \times N-1 \times N-2 \times \ldots\ldots 1$

N string → N! permutations

Str = A B C



if (idx == N) valid permutation



a b c
0 1 2

N = 3

N size array

I/P I/P O C]

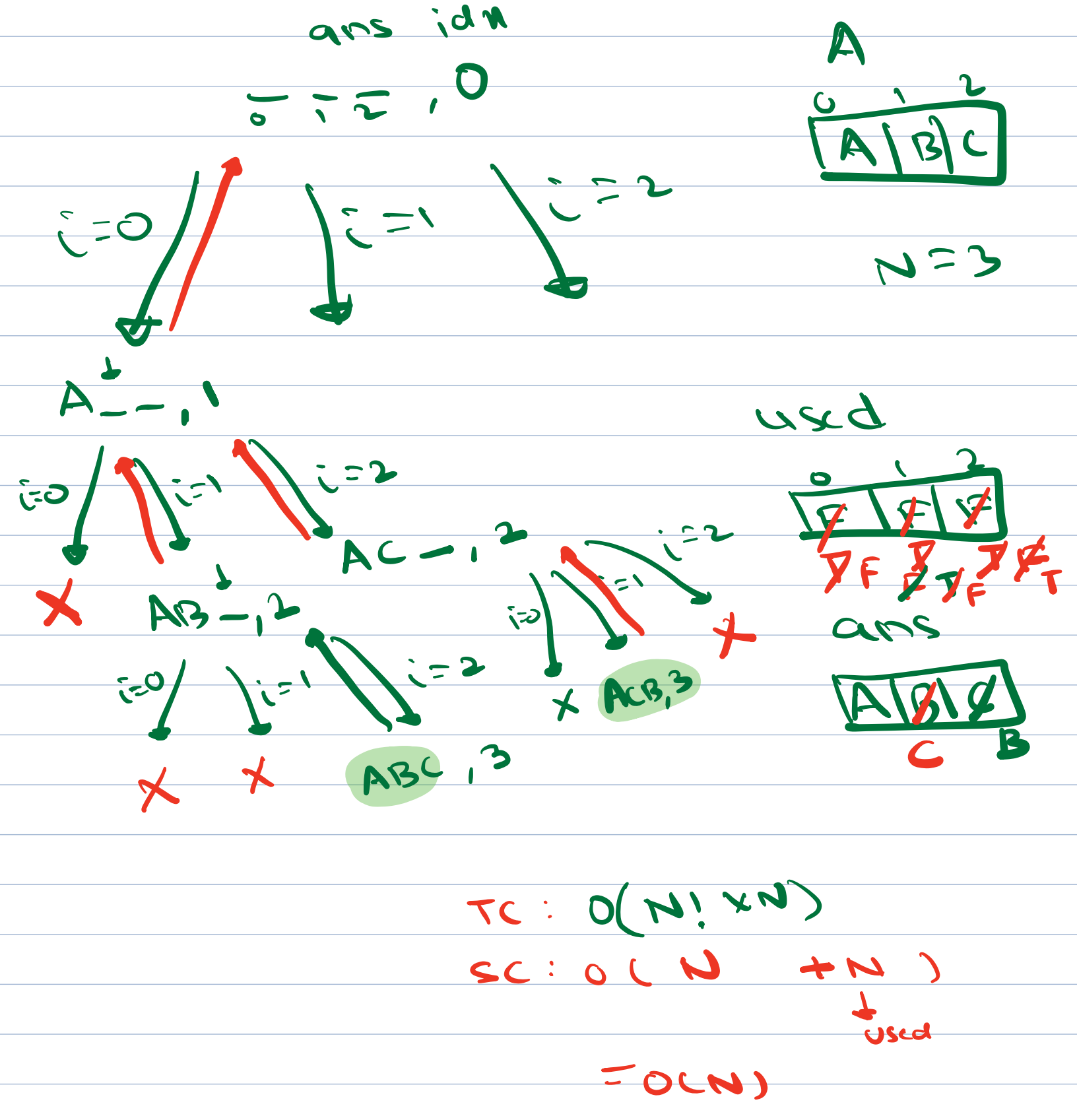
void generate(char A[], int N, int idx, char[] ans,
bool used[]) {

① if (idx == N) {
    print(ans) return
}
// try all options

② for (i=0 ; i<N ; i++) {
    if (used[i] == false) {
        ans[idx] = A[i]
        used[i] = true
        generate(A, N, idx+1, ans, used)
        used[i] = false
    }
}

fn( ) {

// go to all options
// do changes
// recursive fn
// undo changes
}

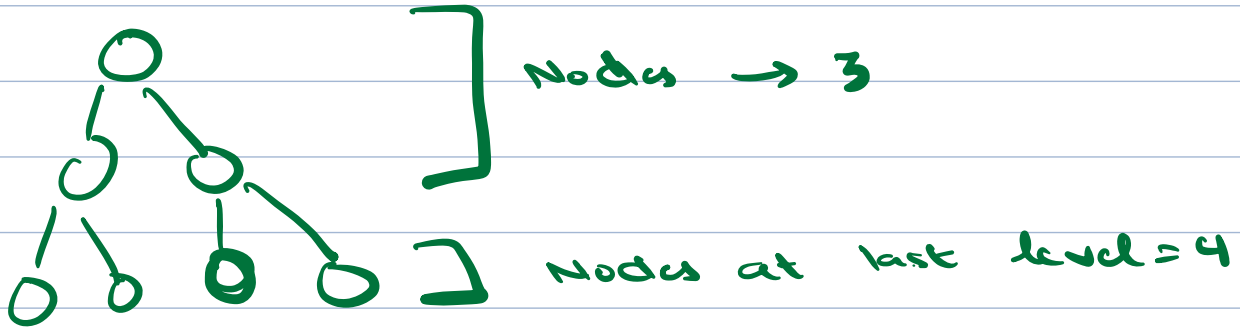


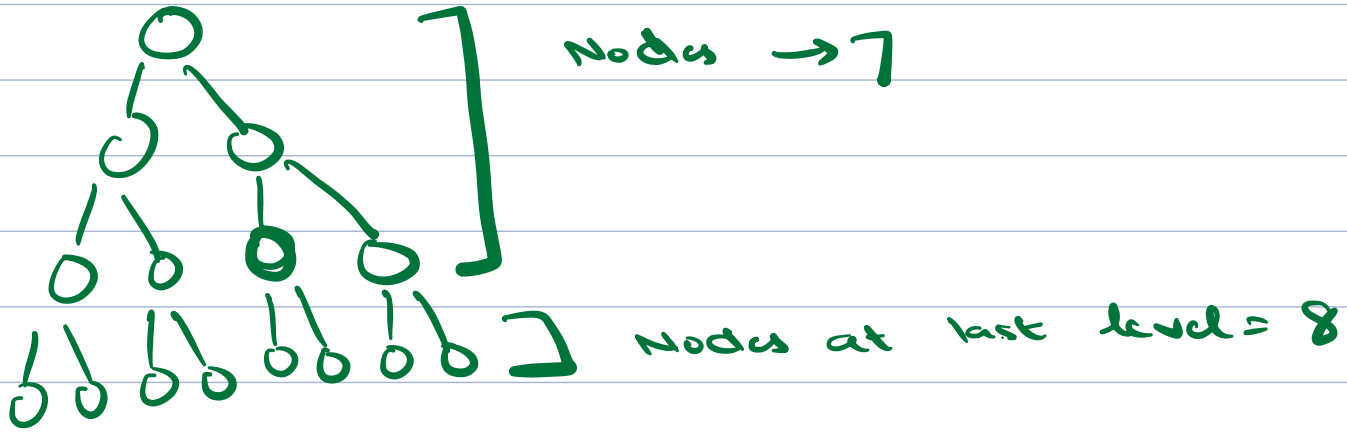
TC : $O(N! \times N)$

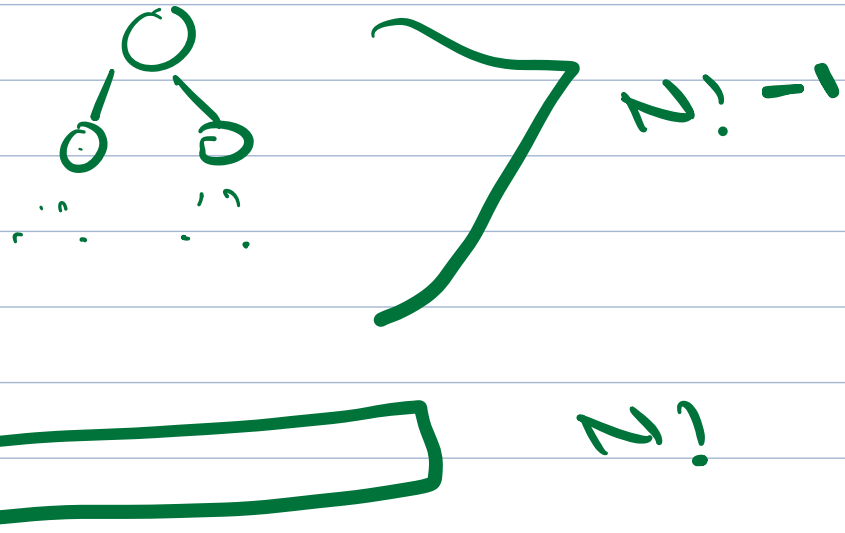
SC : $O(N + N)$

↓ used

$= O(N)$







TC : O(N! * N)

SC : O(N)

N = 2

( ) < >
< > ( )
< ( ) >
( ) < >

[ ] ( )
( ) [ ]
[( )]
([ ])

[ ] < >
< > [ ]
[< >]
< [ ] >

[((( )))]
[( )] [ ]
[[ ]] [ ]
< < > >
< > < >