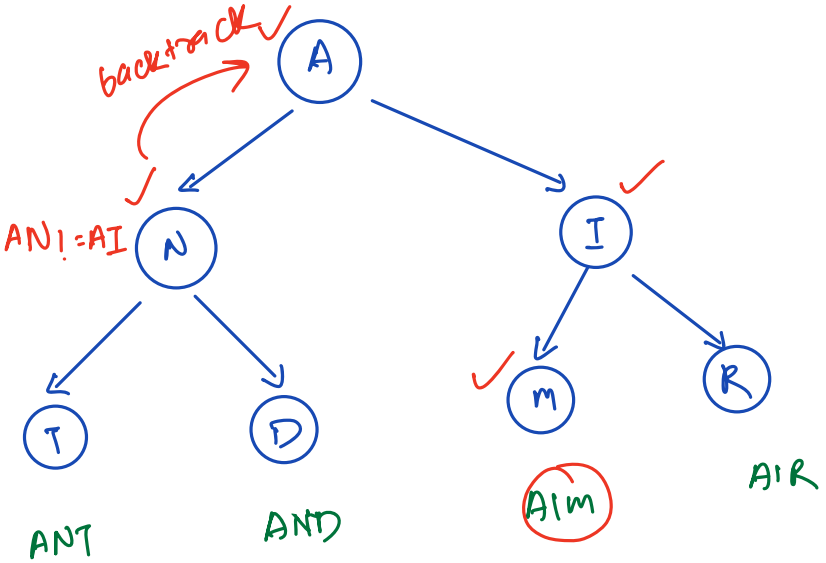
Backtracking 1



AIm ?

Question

Given an integer A. Write a function to generate all combinations of well-formed parantheses of length 2*A.

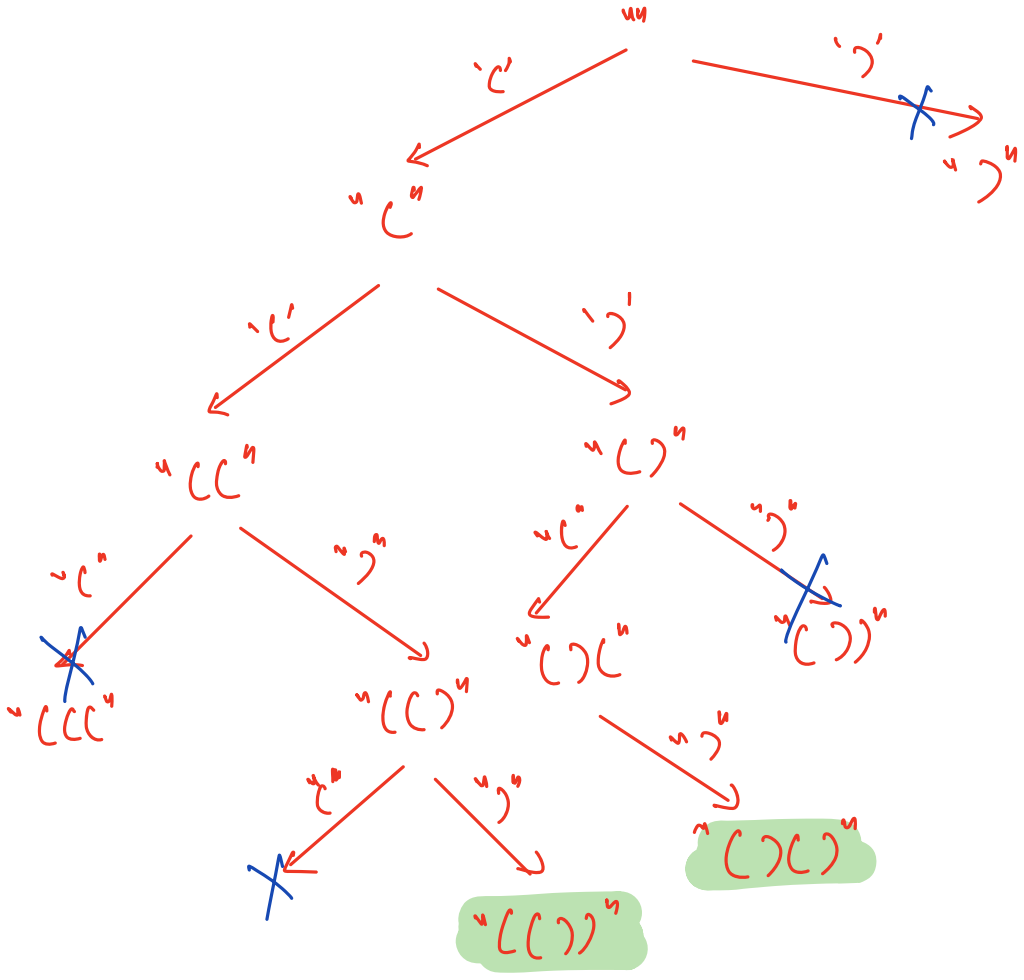
well-formed parantheses

- total no. of open = total no. of close
- if you go from left to right, no. of open >= no. of close

A = 3

"((()))", "(()())", "(())()", "()(())", "()()()"

N = 2

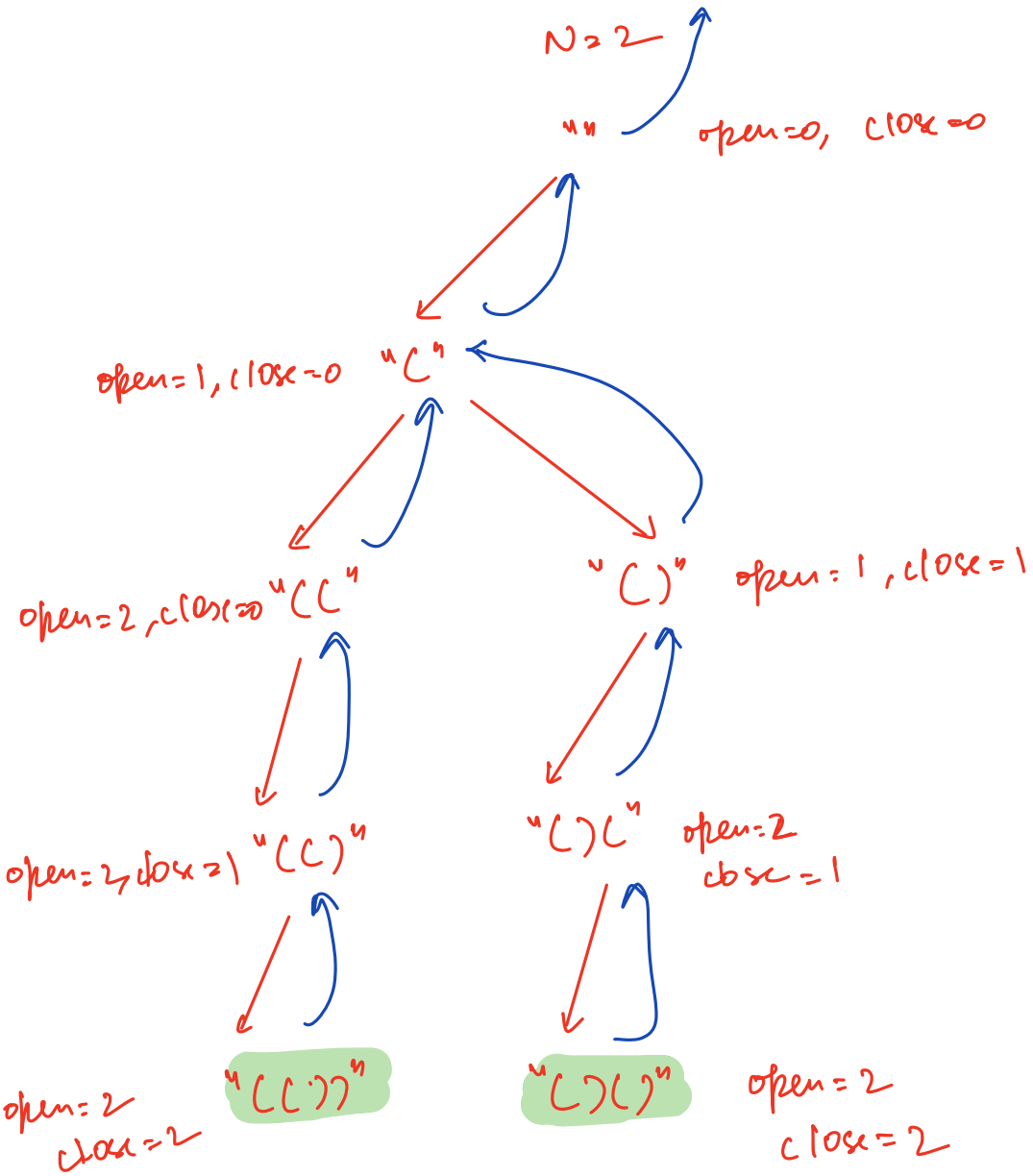


solve("", N, 0, 0)

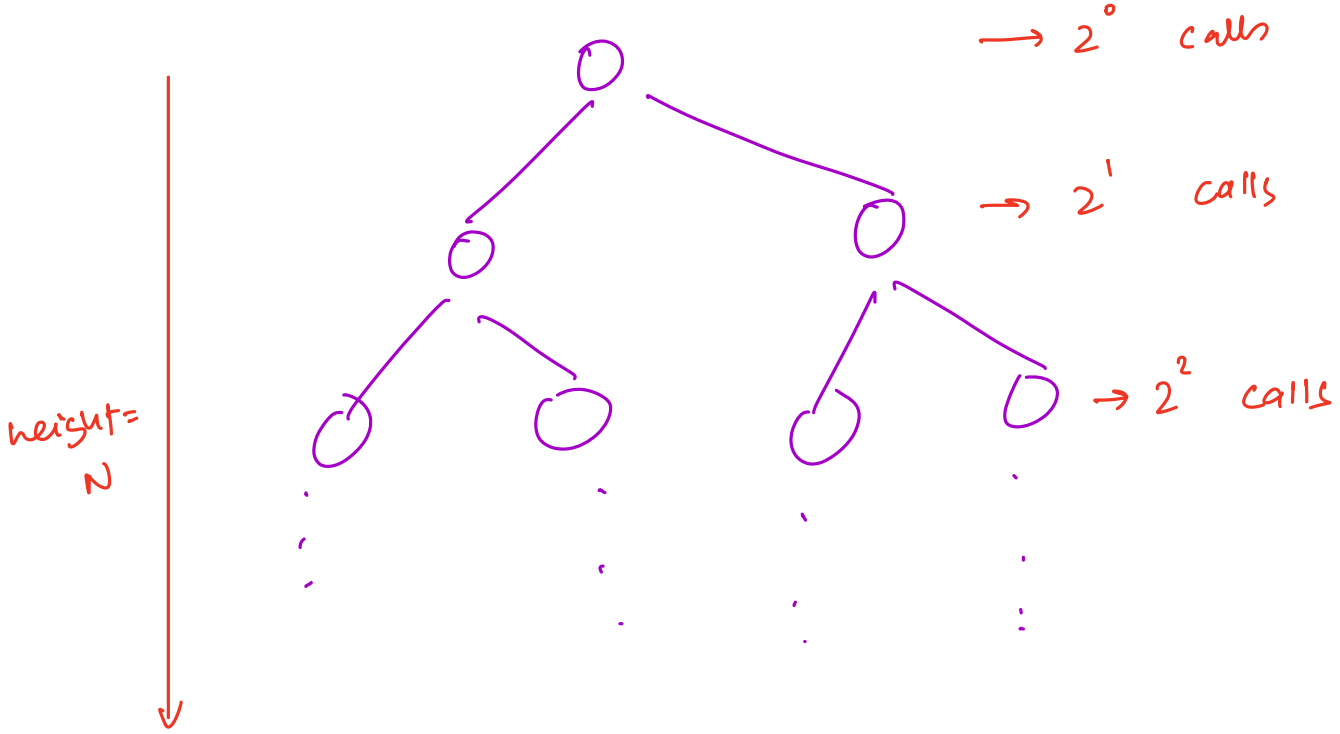
Code

```
def solve(str, N, open, close) {
    if(open == N && close == N) {
        print(str)
        return
    }
    if(open < N) {
        solve(str + "(", N, open+1, close);
    }
```

```
if ( close < open ) {        // open >= close
    solve( str + ")", N, open, close+1 );
}
}
```



time complexity



$2^0 + 2^1 + 2^2 + \cdots + 2^{N-1} \Rightarrow 2^0\left(\frac{2^N - 1}{2-1}\right) = 2^N - 1$

total TC = $O(2^N)$

N = 10 $\approx 10^3$

N = 20 $\approx 10^6$

SC = $O(N)$

Subset vs Subsequence vs Subarray

Subset → any possible combination of original array

A = [1 2 3 4 5]

subset : [1 4],

[1 2 5],

[3 1]

ORDER doesn't MATTER

Subsequence → sequence of array elements after deleting some elements.

A = [1 2 3 4 5]

subsequence : [1 4],

[1 2 5],

~~[3 1]~~

in sequence, order matters

Subarray → continous part of array

Every Subarray is a subsequence but not opposite

Every subsequence is a subset but not opposite

Question

Given an array of distinct elements.

Return all subsets using recursion.

A = [1 2 3]

o/p : {}

{1}

{1 2}

{1 2 3}

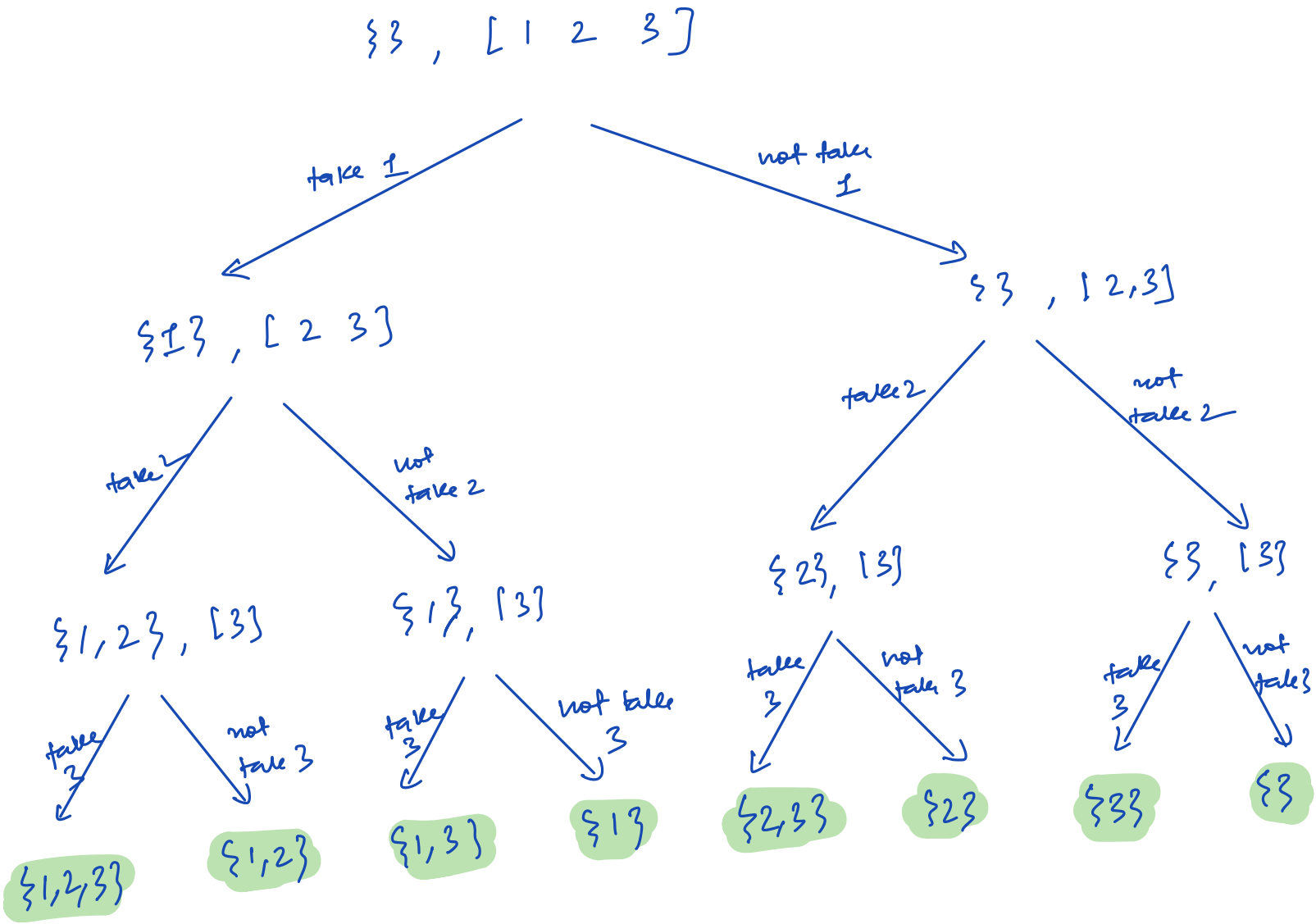
{2}

{2 3}

{3}

{1,3}

count = 8



for every element → 2 options (take OR not-take)

for N elements → $2 \times 2 \times 2 \times \dots$

→ 2^N options

Code // return all subsets

⇒ we need array of arrays.

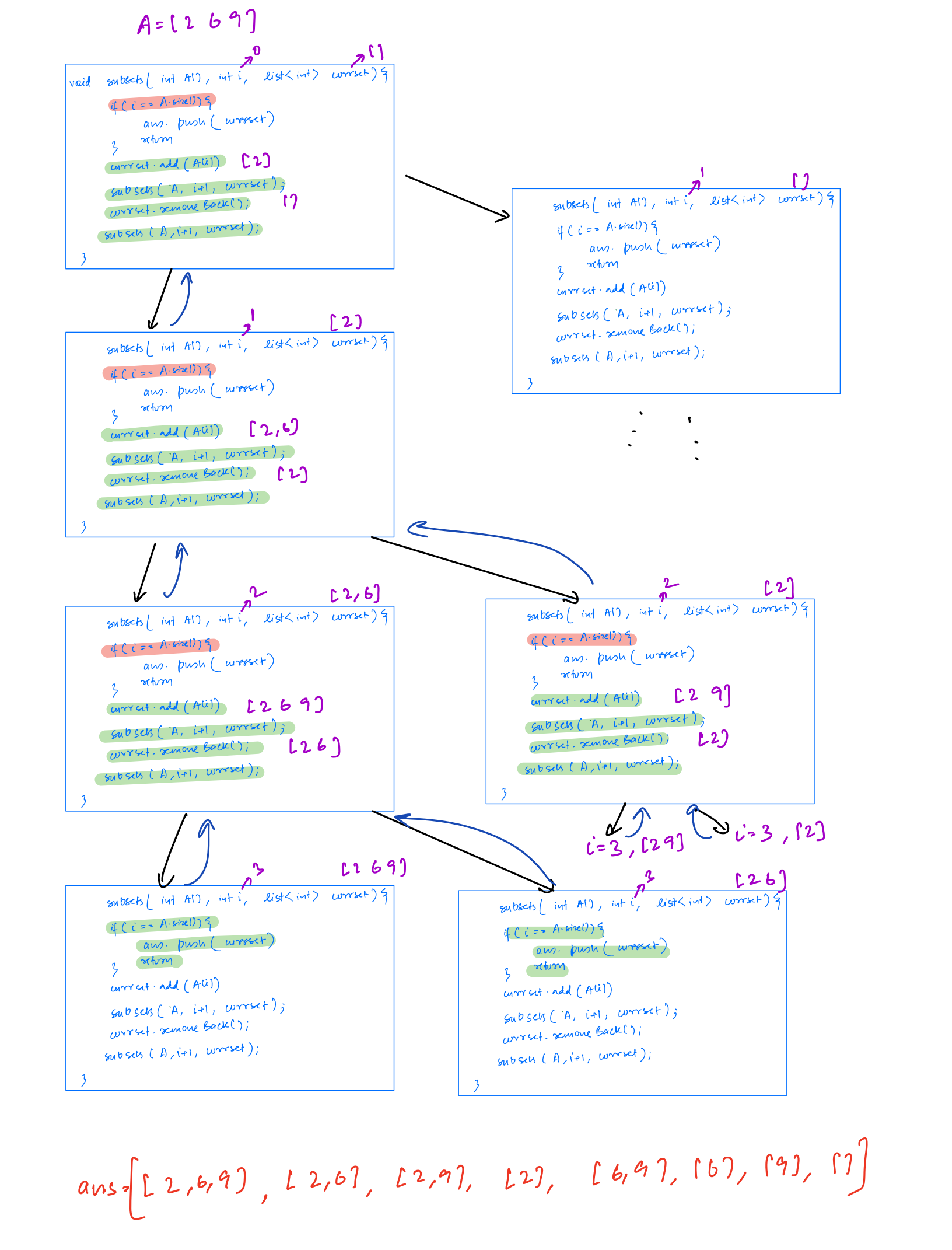
list < list < int > > ans; → global variable

subsets(A, 0, [])

```
void subsets( int A[], int i, list<int> currset) {
    if ( i == A.size()) {
        ans.push ( currset)
        return
    }
    // two options
    1. Pick A[i] in subset
        currset.add ( A[i])
        subsets ( A, i+1, currset );

    2. don't pick A[i]
        currset.removeBack(); → property of backtracking
        subsets ( A, i+1, currset );
}
```

```
subsets( int A[], int i, list<int> currset) {
if ( i == A.size()) {
    ans.push ( currset)
    return
}
currset.add ( A[i])
subsets ( A, i+1, currset);
currset.removeBack();
subsets ( A, i+1, currset);
}
```


A=[2 6 9]
void subsets(int A[], int i, list<int> currset){
if(i == A.size()){
ans.push(currset)
return
}
currset.add(A[i])
subsets(A, i+1, currset);
currset.removeBack();
subsets(A, i+1, currset);
}
0
[]
[2]
[]
1
[]
[2]
[2,6]
[2]
2
[2,6]
[2 6 9]
[2 6]
2
[2]
[2 9]
[2]
i=3, [2 9]
i=3, [2]
3
[2 6 9]
3
[2 6]
ans=[[2,6,9], [2,6], [2,9], [2], [6,9], [6], [9], []]

Time Complexity $\rightarrow O(2^N)$

SC $= O(N)$

Total permutations of string with unique characters.

abc $\rightarrow$ abc bac cab ($3! = 6$)
acb bca cba

fitbit wants to arrange workouts in different order.

eg A = Push-ups
B = Squats
C = Burpees
D = Planks

A B C D
A C B D
D A B C

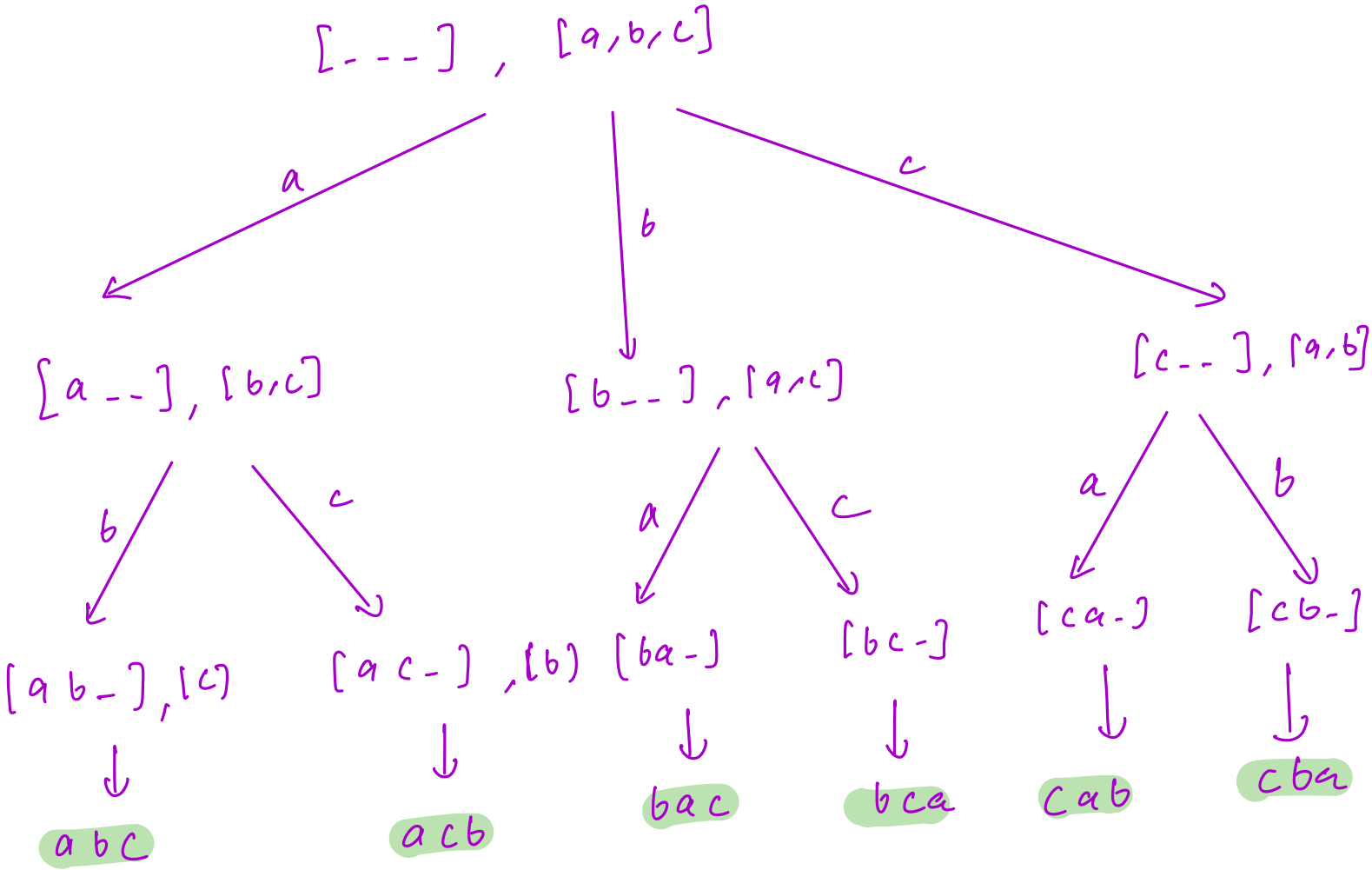
A D B C

⋮

Given a string with distinct characters, print all permutations.

abc → abc bac cab (3! = 6)
acb bca cba

String of size N → N! permutations



permutations("abc", 0, [---], [0,0,0])

Code

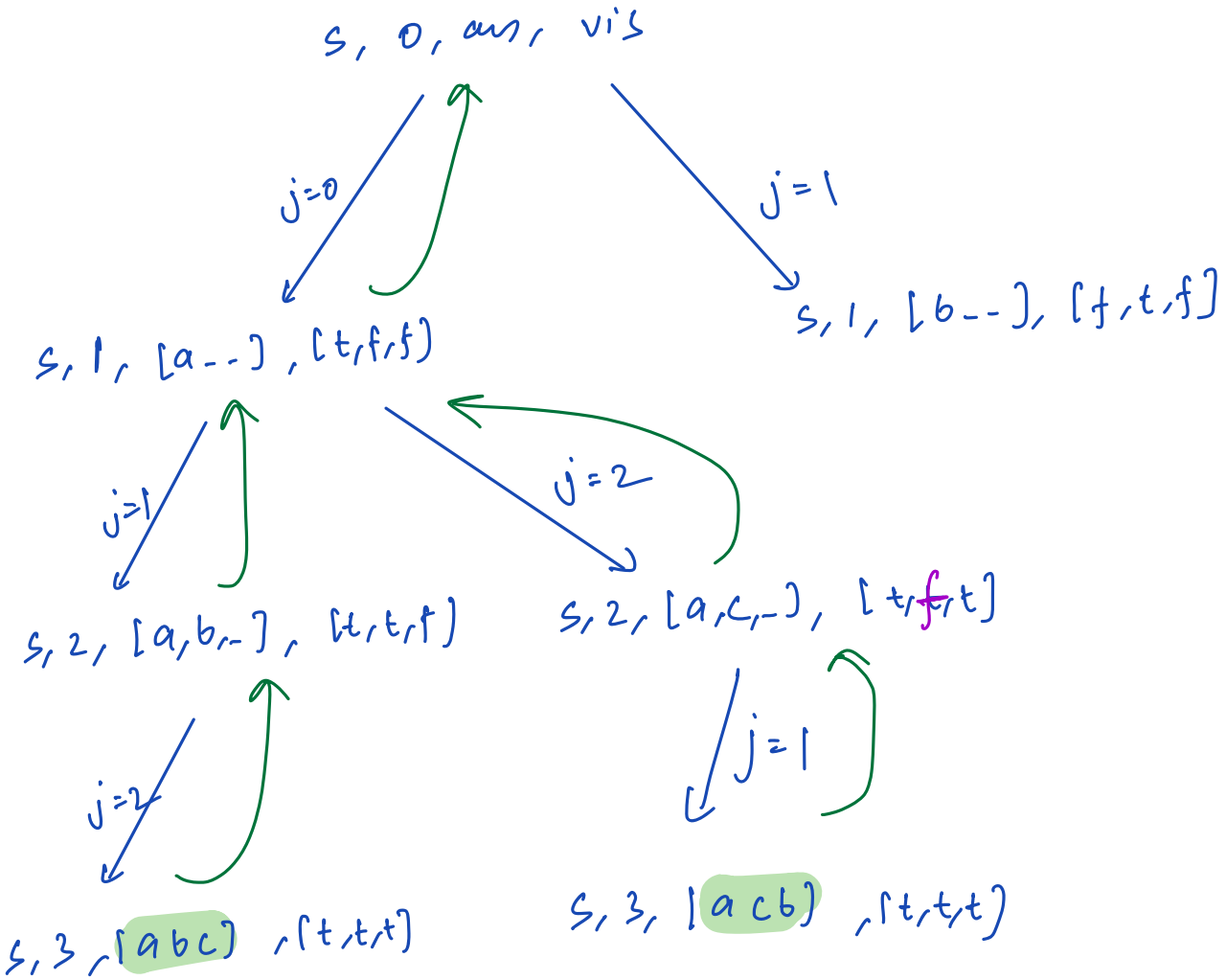
```
void permutations(s, i, ans, vis){
    if(i == s.size()){
        print(ans)
        return
    }
    for(j=0 to n-1){
        if(vis[j] == false){
            ans[i] = s[j]
            vis[j] = true
            permutations(s, i+1, ans, vis);
            vis[j] = false
        }
    }
}
```

vis: bool

s = "abc"

vis = [f, f, f] (0, 1, 2)

ans = [| |] (0, 1, 2)



Time complexity →

total f^n calls * TC of one f^n call

N! * N

TC = $O(N * N!)$ ⇒ $O((N+1)!)$

SC = $O(N)$