



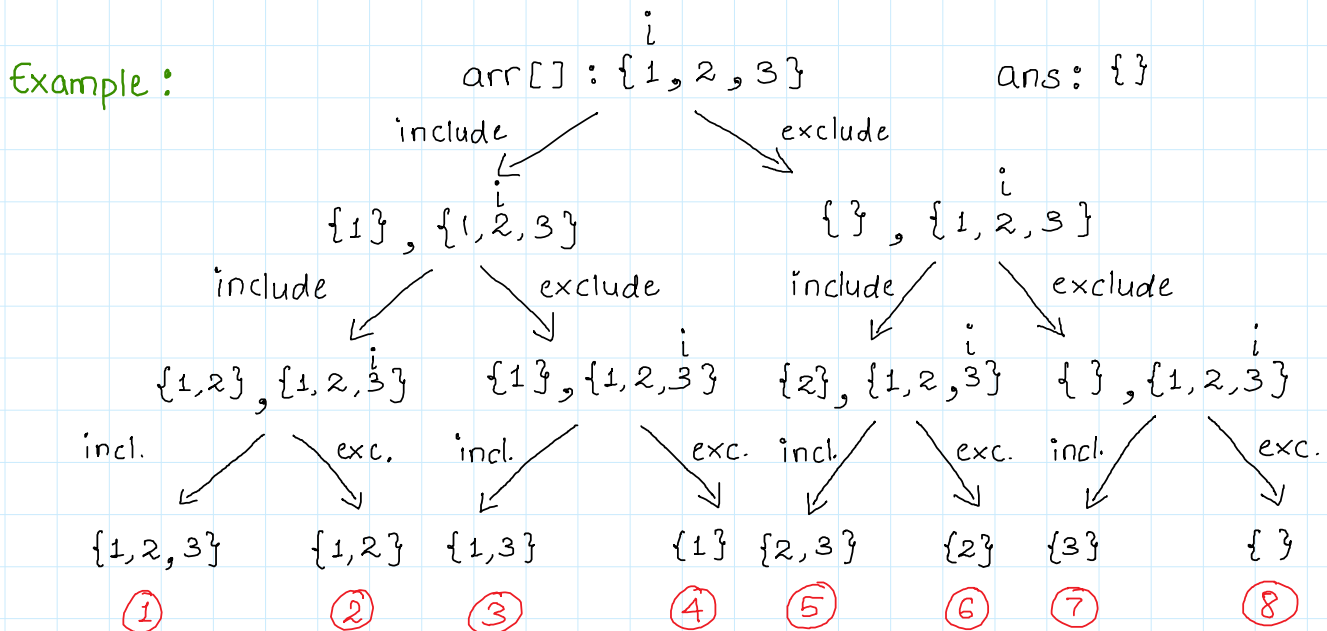
1. Generate all subsets (Power Set)

I/P: [1, 2, 3]

O/P: $\{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}$

Note - We can generate these subsets in any order as long as we cover all the subsets.

Approach: We can either include or exclude an element to generate a subset. Thus, we have 2 options for every element, either include it in the subset or exclude it.



What do you need?

- ① Array
- ② Index
- ③ Answer array.

Code:

```
class Solution {
private:
    void solve(vector<int>& nums, vector<int> output, int i, vector<vector<int>> &ans) {
        if(i >= nums.size()) {
            ans.push_back(output);
            return;
        }
        // exclude this element and move on to the next index
        solve(nums, output, i+1, ans);
        // include this element
        output.push_back(nums[i]);
        solve(nums, output, i+1, ans);
    }
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int>> ans;
        vector<int> output;
        int ind = 0;
        solve(nums, output, ind, ans);
        return ans;
    }
};
```

Checking output array for each recursive call

```
#include <iostream>
#include <vector>
using namespace std;

void solve(vector<int>& nums, vector<int> output, int i, vector<vector<int>>& ans) {
    cout << "output: ";
    for(auto x: output) {
        cout << x << " ";
    }
    cout << endl;
    if (i == nums.size()) {
        ans.push_back(output);
        return;
    }
    // exclude this element and move on to the next index
    solve(nums, output, i + 1, ans);
    // include this element
    output.push_back(nums[i]);
    solve(nums, output, i + 1, ans);
}

vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> ans;
    vector<int> output;
    int ind = 0;
    solve(nums, output, ind, ans);
    return ans;
}
```

```
1 3
2 1 2 3
output.txt
1 output:
2 output:
3 output:
4 output:
5 output: 3
6 output: 2
7 output: 2
8 output: 2 3
9 output: 1
10 output: 1
11 output: 1
12 output: 1 3
13 output: 1 2
14 output: 1 2
15 output: 1 2 3
16
17 3
18 2
19 2 3
20 1
21 1 3
22 1 2
23 1 2 3
24
```

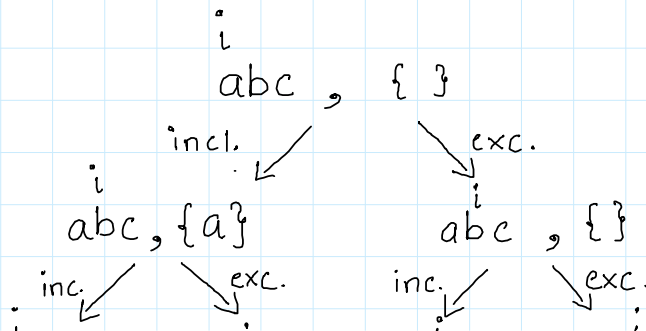
2. Subsequences of String. (slight change)

I/P : "abc"

O/P : {"", "a", "b", "c", "ab", "ac", "bc", "abc"}

Same Approach.

Recursion Tree :



1 1 0 \Rightarrow "ab"
1 1 1 \Rightarrow "abc"

"bc"

So, for a string of length n , we will take an n -bit binary number and iterate through all the 2^n ranging from $\underbrace{000\dots 00}_n$ to $\underbrace{111\dots 11}_n$.

Code :

```
string maskedString(string str, int n) {
    string ans;
    int ind = str.length()-1;
    for(int j=0; j<str.length(); j++) {
        if((1<<j) & n) {
            ans.push_back(str[ind]);
        }
        ind--;
    }
    return ans;
}

vector<string> subsequences(string str){
    vector<string> ans;
    for(int i=0; i<(1<<str.length()); i++) {
        string temp = maskedString(str, i);
        if(!temp.empty())
            ans.push_back(temp);
    }
    return ans;
}
```

1 abc

Output.txt

1 c
2 b
3 cb
4 a
5 ca
6 ba
7 cba
8