78. Subsets

Given an array of unique elements, we need to find all the subsets. This is a backtracking problem.

Explaination is best seen in code itself.

CODE

```
vector < vector <int>> powerset;
vector <int> subset;
void backtrack ( vector <int> & nums, int start) {
        powerset. push_back (subset);
        for( int i = start , i < nums. size(); i++ ){
                subset. push_back ( nums[i]);
                backtrack ( nums, i+1);
                subset. pop_back ();
        }
vector <vector <int>> subsets ( vector <int> & nums) {
        backtrack (nums, 0);
        return powerset;
};
```

In this solution, we use a depth first approach. We dive into let's say [1] and then add all the subsets that contain 1 before moving on to the subsets that do not contain 1.

Time complexity → $O(2^n)$

space complexity → $O(n)$

↳ excluding the space taken by powerset.

→ Another method is to use a bitstring

For example for $n = 3$, we generate all the bitstrings using backtracking

(0,0,0),

0 0 1

0 1 1

1 1 1

1 1 0

0 1 0

1 0 0

1 0 1

Now, when we have all the bitstrings, we just loop on each bitstring and add the element to subset if bit == 1.