# Day 14

25 November 2022 19:19

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all* ***unique combinations*** *of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

**Example 1:**

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

# Combination Sum I

☆ candidates = [2, 3, 6, 7]

target = 7

2 + 2 + 3 = 7 ✓

☆ ∈ no. of unlimited times

(7) ✓

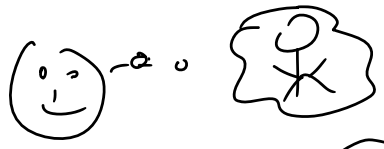
(2)

[[2, 2, 3], [7]]

candidates , target = 8

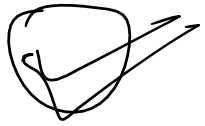
[2, 3, 5]

[2, 2, 2, 2] →
[3, 3, 2] →
[5, 3] →
] (3)

kartik → 

utkarsh 

| 2 | 3 | 6 | 7 |

kartik  50%

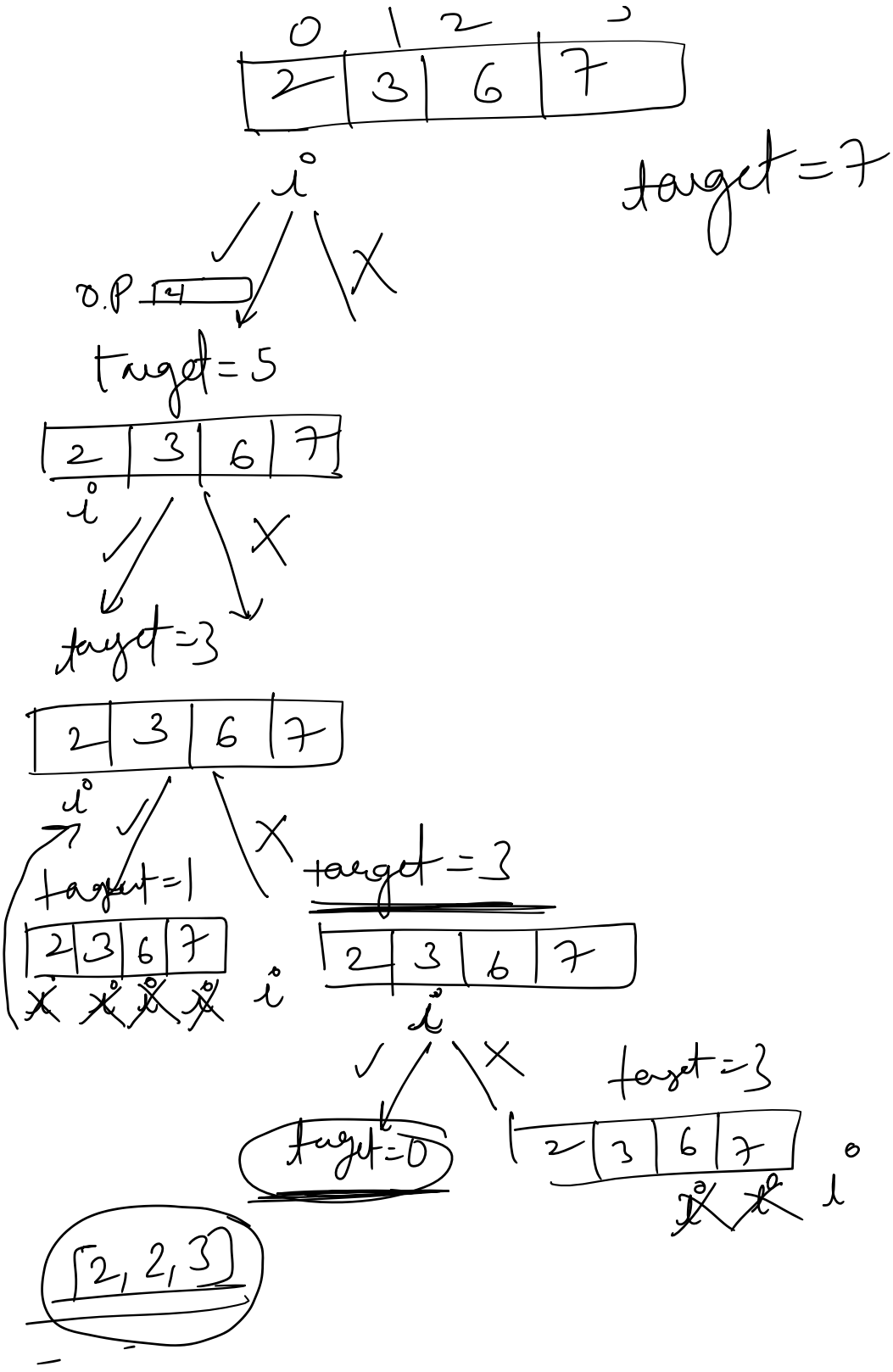
| 2 | 3 | 6 | 7 | target = 7

Choice Diagram

arr[i] ≤ target
① Choice to Include it
② Not to Include it

//arr[i] > target
① Not to Include it

| 0 | 1 | 2 | 3 |
| --- | --- | --- | --- |
| 2 | 3 | 6 | 7 |

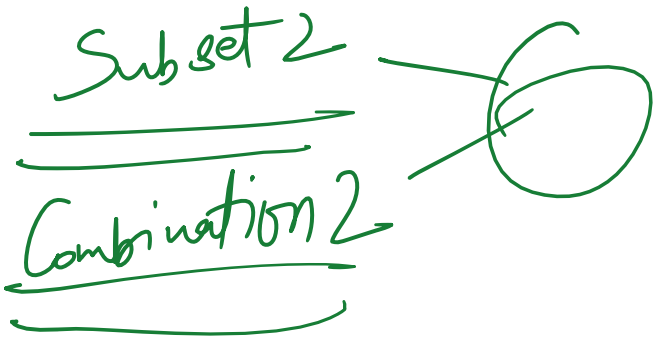



```
class Solution {
public:
    void helper(int i,vector<int> &temp_output,vector<vector<int>> &answer_array,vector<int>
&candidates,int target)
    {
        // sum mera target k equal ho gaya
        if(target == 0){
            answer_array.push_back(temp_output);
            return;
        }

        if(i == candidates.size()){
            return;
```

```
        }

        if(i == candidates.size()){
            return;
        }
        // []
        // [2,---]
        if(candidates[i] <= target){
            temp_output.push_back(candidates[i]);
            // include
            helper(i,temp_output,answer_array,candidates,target-candidates[i]);
            // []
            temp_output.pop_back();
            // not include
            helper(i+1,temp_output,answer_array,candidates,target);
        }else{
            helper(i+1,temp_output,answer_array,candidates,target);
        }
    }
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> answer_array;
        vector<int> temp_output;
        helper(0,temp_output,answer_array,candidates,target);
        return answer_array;
    }
};
```


Approach

```
class Solution {
public:
    void helper(int i,vector<int> &temp_output,vector<vector<int>>
&answer_array,vector<int> &candidates,int target)
    {
        // sum mera target k equal ho gaya
        if(target == 0){
            answer_array.push_back(temp_output);
            return;
        }

        if(i == candidates.size()){
            return;
        }
        // []
        // [2,---]
        if(candidates[i] <= target){
            temp_output.push_back(candidates[i]);
            // include
            helper(i+1,temp_output,answer_array,candidates,target-candidates[i]);
            // []
            temp_output.pop_back();
            // not include
            while(i + 1 < candidates.size() && candidates[i+1] == candidates[i]){
                i++;
```

```
            }
            helper(i+1,temp_output,answer_array,candidates,target);
        }else{
            helper(i+1,temp_output,answer_array,candidates,target);
        }
    }
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        sort(candidates.begin(),candidates.end());
        vector<vector<int>> answer_array;
        vector<int> temp_output;
        helper(0,temp_output,answer_array,candidates,target);
        return answer_array;
    }
};
```

T.C – O(2^N)

S.C – O(N)

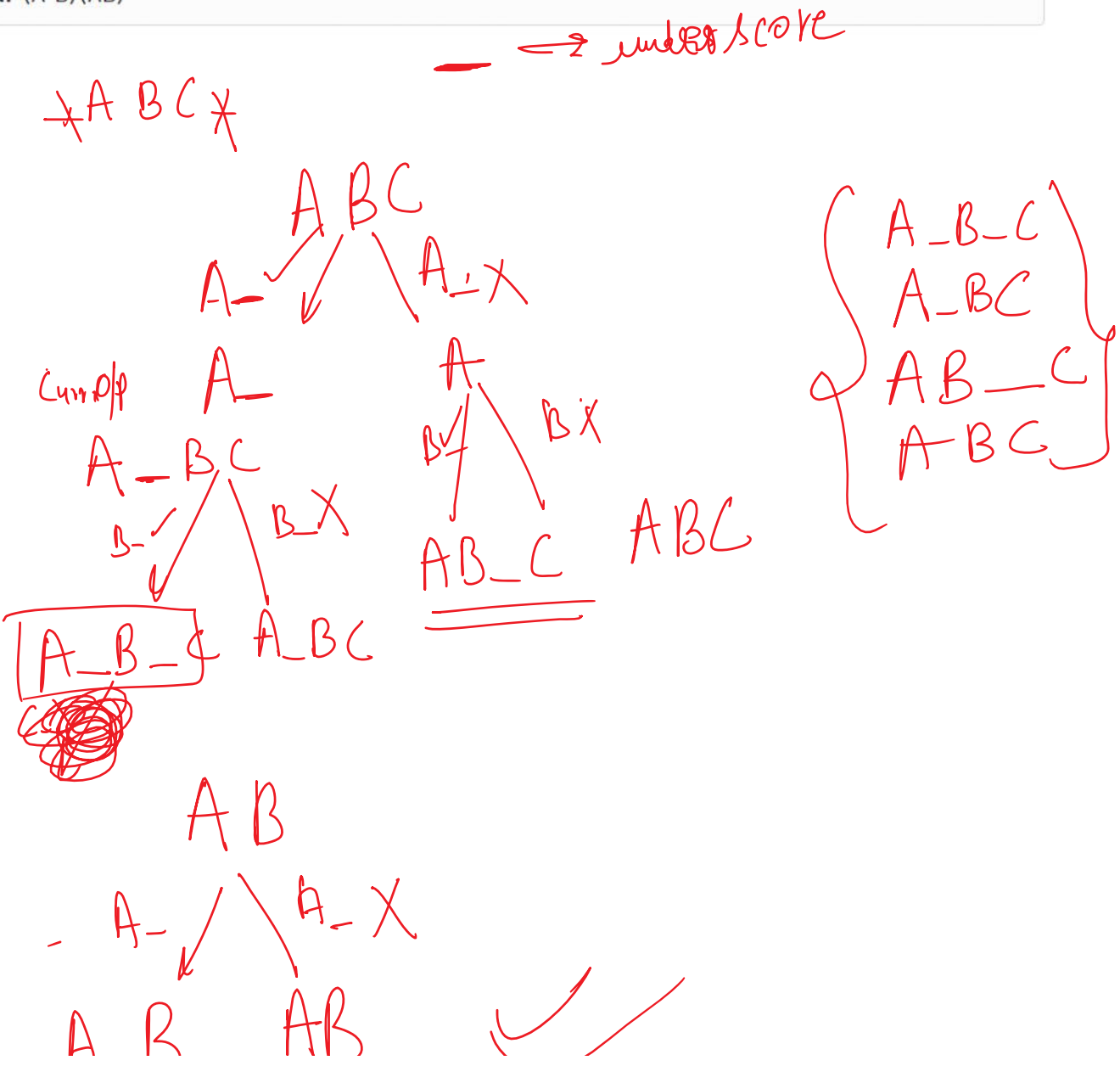
# Permutation with spaces

**Example 1:**

**Input**:
S = "ABC"
**Output:** (A B C)(A BC)(AB C)(ABC)
**Explanation**:
ABC
AB C
A BC
A B C
These are the possible combination of "ABC".

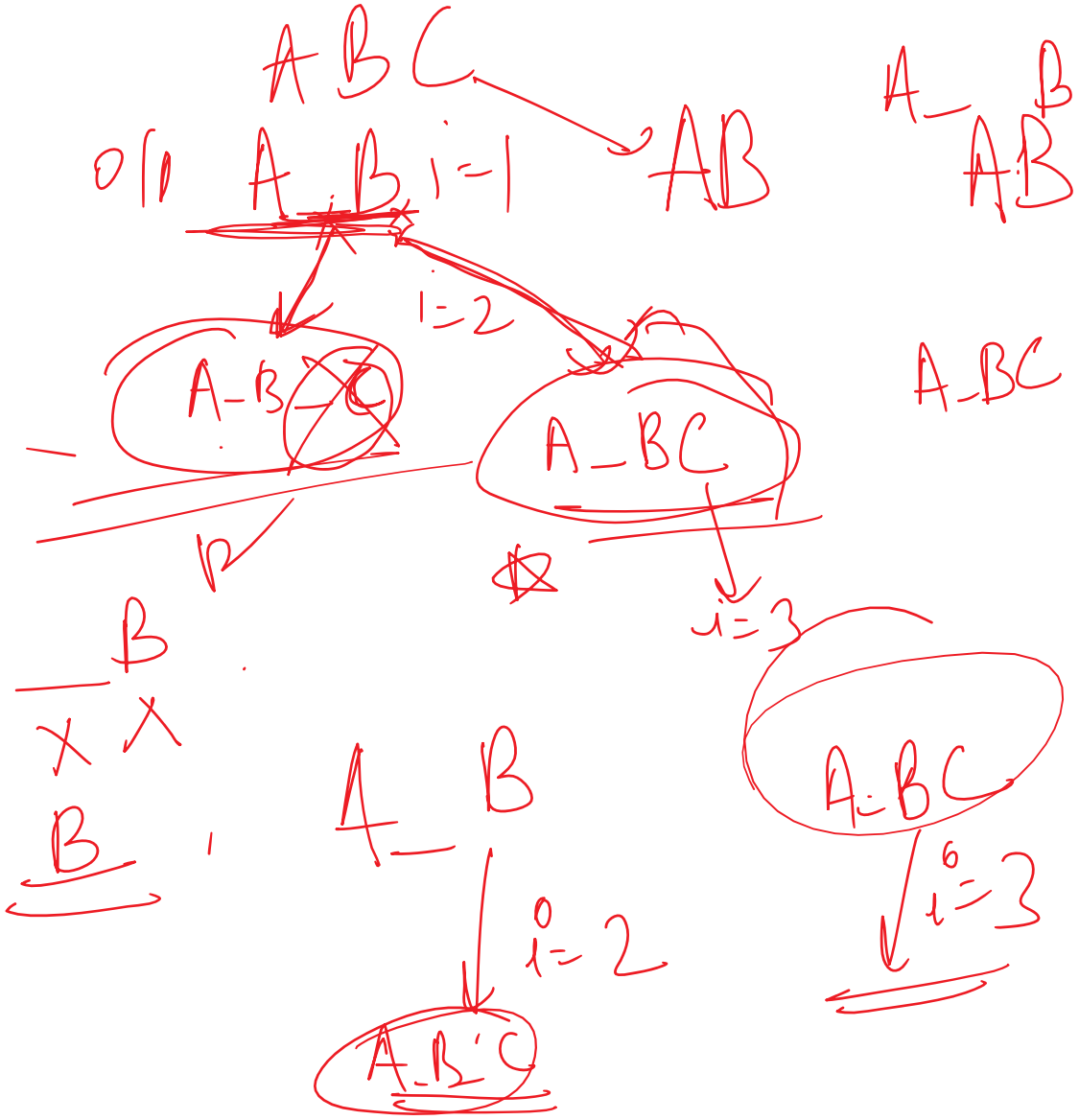
**Example 2:**

**Input:**
S = "AB"
**Output:** (A B)(AB)

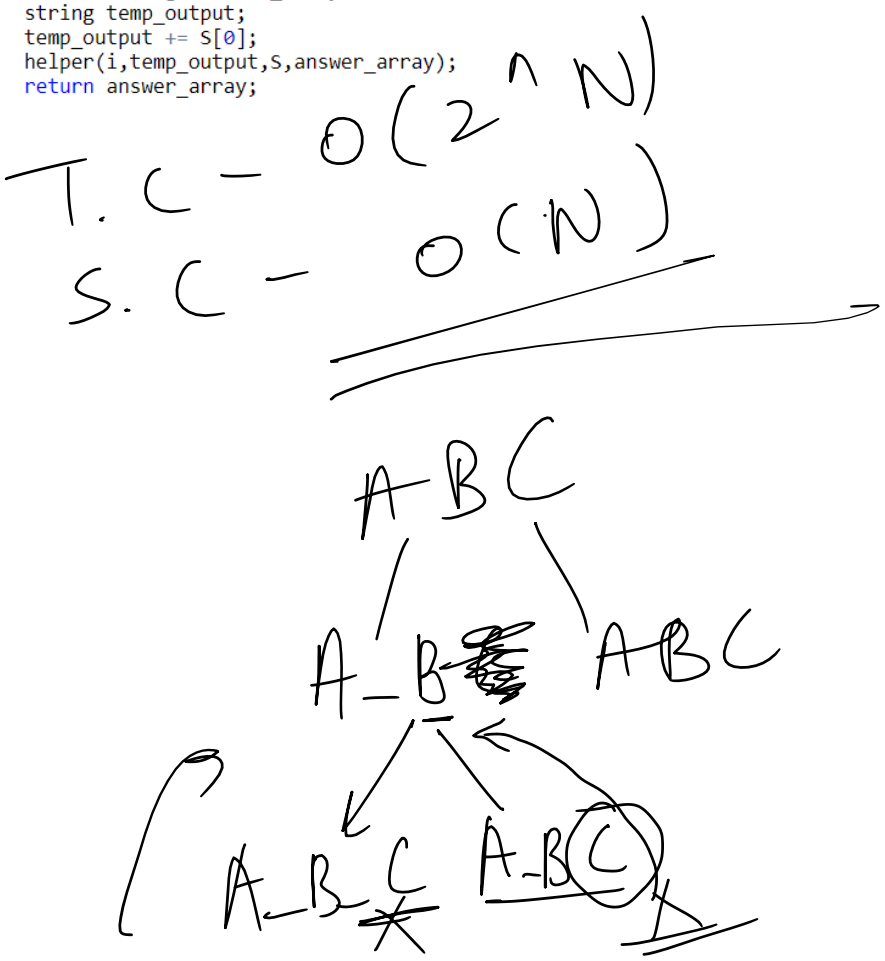


A_B AB

Auxillary stack space



```cpp
void helper(int i, string output,string &input,vector<string> &answer_array){
    if(i == input.size()){
        answer_array.push_back(output);
        return;
    }
    // included
    string op1 = output;
    op1 +=' ';
    op1 += input[i];
    string op2 = output;
    op2 += input[i];
    helper(i+1,op1,input,answer_array);
    // not included
    helper(i+1,op2,input,answer_array);
}
vector<string> permutation(string S){
    int i = 1;
    vector<string> answer_array;
    string temp_output;
    temp_output += S[0];
    helper(i,temp_output,S,answer_array);
    return answer_array;
}
```



```
void helper(int i, string &output,string &input,vector<string> &answer_array){
    if(i == input.size()){
        answer_array.push_back(output);
        return;
    }
    // included
    output+=' ';
    output+= input[i];
    helper(i+1,output,input,answer_array);
    // not included
    output.pop_back();
    output.pop_back();
    output += input[i];
    helper(i+1,output,input,answer_array);
    output.pop_back();
}
vector<string> permutation(string S){
    int i = 1;
    vector<string> answer_array;
    string temp_output;
    temp_output += S[0];
    helper(i,temp_output,S,answer_array);
    return answer_array;
}
```

T.C – $O(2^N)$

S.C – $O(N)$