# L13 Subsets II or Subsets Sum II

Friday, December 23, 2022 7:40 PM

Given an integer array nums that may contain duplicates, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

**Example 1:**
**Input:** nums = [1,2,2]
**Output:** [[],[1],[1,2],[1,2,2],[2],[2,2]]

↳ this problem is the extended version of previous problem that we learn Subset Sum I

→ Previous problem our task is finding All the possible subset and this problem finding all the possible subset without any dublicate subset ☺

arr = [1,2,2]
n = 3

| [1, 2, 2] | | Subsets |
|---|---|---|
| x x x | = | [] |
| ✓ x x | = | [1] |
| ✓ ✓ x | = | [1,2] |
| ✓ ✓ ✓ | = | [1,2,2] |
| x ✓ x | = | [2] |
| x ✓ ✓ | = | [2,3] |

6 unique subset possible

Important Point

→ Previously we learn if any particular Array subset find then at Max possible subset is $2^n$

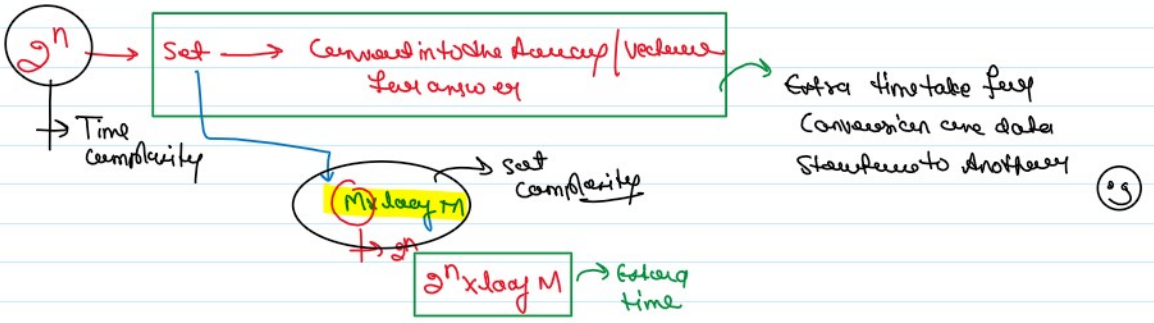
According to formula = $2^n = 2^3 = 8$ but my answer is = 6

↳ because this Array contain dublicate element so dublicate subset are also exist and problem state that only unique subset point that's why answer is 6 ☺

# Thought Process

→ Again we follow the concept of pick and Not pick previously concept that we learn
→ But we used that concept when dublicate subset are also exist then

↳ used a set data structure for Remove the Dublicate Subset




→ So Interviewer Not Happy with this Approach and ask removed that extra time and Optimized that Solution Means (Optimized Recursion)

Solution Name (Optimized Recursion)

# Thought Process

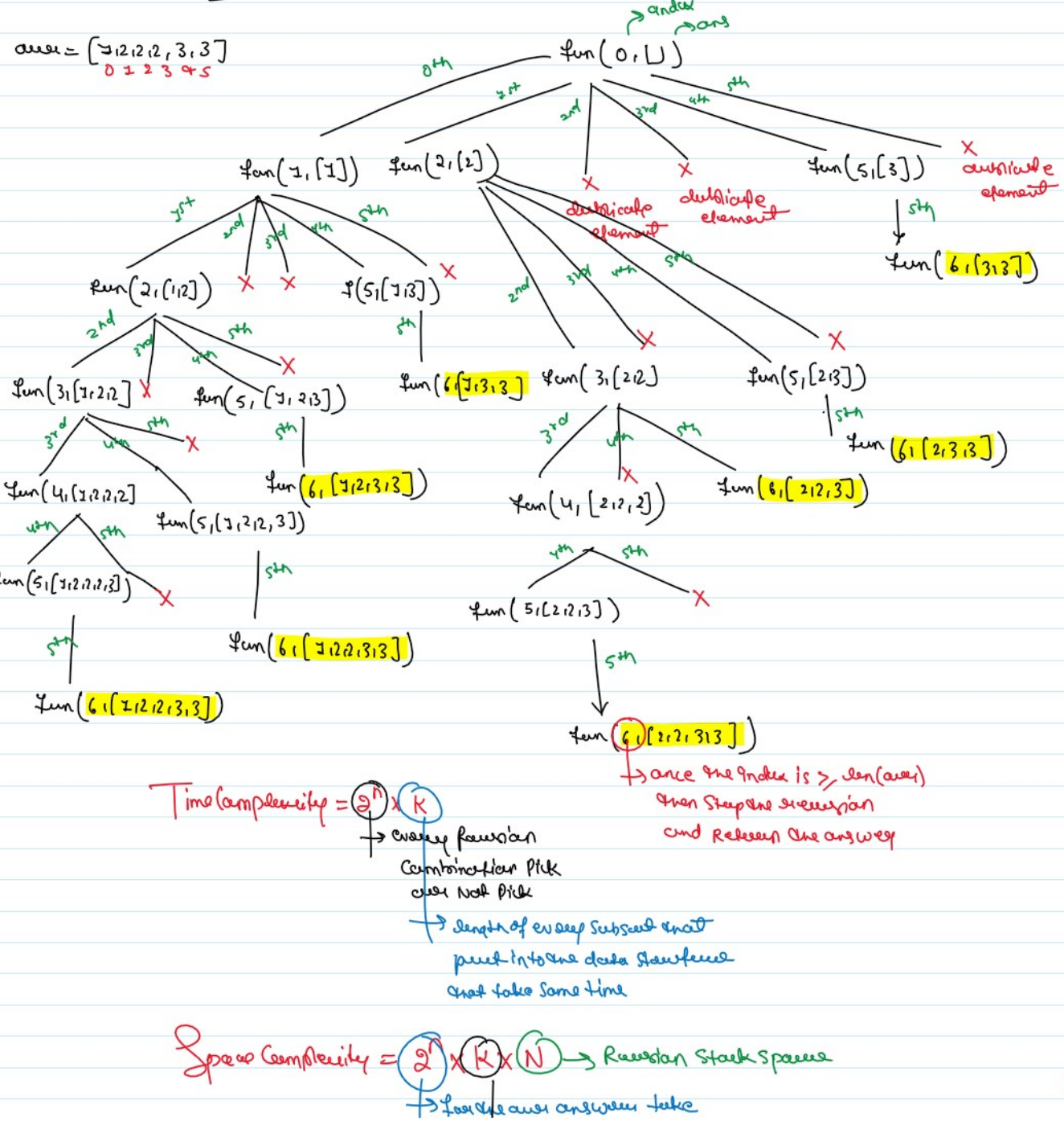
→ We Goto write a Recursion Such a way that every steps its generate a unique list

arr = [1,2,2,2,3,3]
0 1 2 3 4 5

N = 6

→ Now here we used the concept that we learned in Combination Sum II for finding every unique Combination

→ Stand with Empty list and push into the answers and check every combination like we do in the Combination Sum II




Time Complexity = $2^n \times K$

→ every Recursion Combination Pick one Not Pick

→ length of every Subset that push into the data structure that take some time

Space Complexity = $2^n \times K \times N$ → Recursion Stack Space

→ For the ans answers take

Space Complexity = $(2^N) \times (K) \times (N)$ → Recursion Stack Space

→ For the ans answer take space

→ every single subset size that we generate

```
myfunction (index, nums, ans, ds ){
```

Same Such case as Combination Sum II only few steps are changed

```
ans.append(ds);
for i in range(index, len(nums)){
    if( i!= index and nums[i]== nums[i-1]) {
        continue;
    }
    ds.append(nums[i]);
    myfunction(i+1, nums, ans, ds);
    ds.remove(nums[i]);
}
```

```python
class Solution:

    def findAllUniquesubsets(self, index, nums, ans, ds):
        ans.append(ds[:])

        for i in range(index, len(nums)):
            if i != index and nums[i] == nums[i - 1]:
                continue
            ds.append(nums[i])

            self.findAllUniquesubsets(i + 1, nums, ans, ds)
            ds.remove(nums[i])

    def subsetsWithDup(self, nums):

        nums.sort()

        ans = []
        self.findAllUniquesubsets(0, nums, ans, [])
        return ans
```

```java
import java.util.*;
public class L13_Subsets_Sums_II {
    public static void main(String[] args) { System.out.println("L13_Subsets_Sums_II"); }
}

no usages
class Solutions {
    2 usages
    public static void findAllUniquesubsets(int ind, int[] nums, List<Integer> ds, List<List<Integer>> ansList) {
        ansList.add(new ArrayList<>(ds));
        for(int i = ind;i<nums.length;i++) {
            if(i!=ind && nums[i] == nums[i-1]){
                continue;
            }
            ds.add(nums[i]);
            findAllUniquesubsets( ind: i+1, nums, ds, ansList);
            ds.remove( index: ds.size()  1);
        }

    }
    no usages
    public static List<List<Integer>> subsetsWithDup(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> ans = new ArrayList<>();
        findAllUniquesubsets( ind: 0, nums, new ArrayList<>(), ans);
        return ans;
    }
}
```