Santhosh 05-Mar | Recursion

05 March 2025 19:48

Basics

fun calls itself

- → Base case / condition
- → Recursive calls (variables) ⇓ Base condition

// Base case

// Recursive Case

unique combination

Combination Sum

- → distinct ✓
- → target ✓
- → unique ✓
- → ~~repition~~ ✗

[$\boxed{2}$ → 2 2 2

uniqueness | different count

How to solve it

[2, 3, 6, 7] for = 7

[(2 2 3) (7)]

[(2 2 3) (7)]

① $\boxed{7}$ ✓ ✗

② (7)

all possibility

Relax conditions

target x

combinations [unique]

[2, 3, 6, 7]

all combinations

2	3, 6	2, 6
2, 3	3, 6, 7	2, 7
2, 3, 6	6, ~~6~~	
2, 3, 6, 7	6, 7	

How to generate all possible combination

Subsequences

2, 3, 7

2	2 7
2 3	3
2 3 7	3 7
	7

Subarray

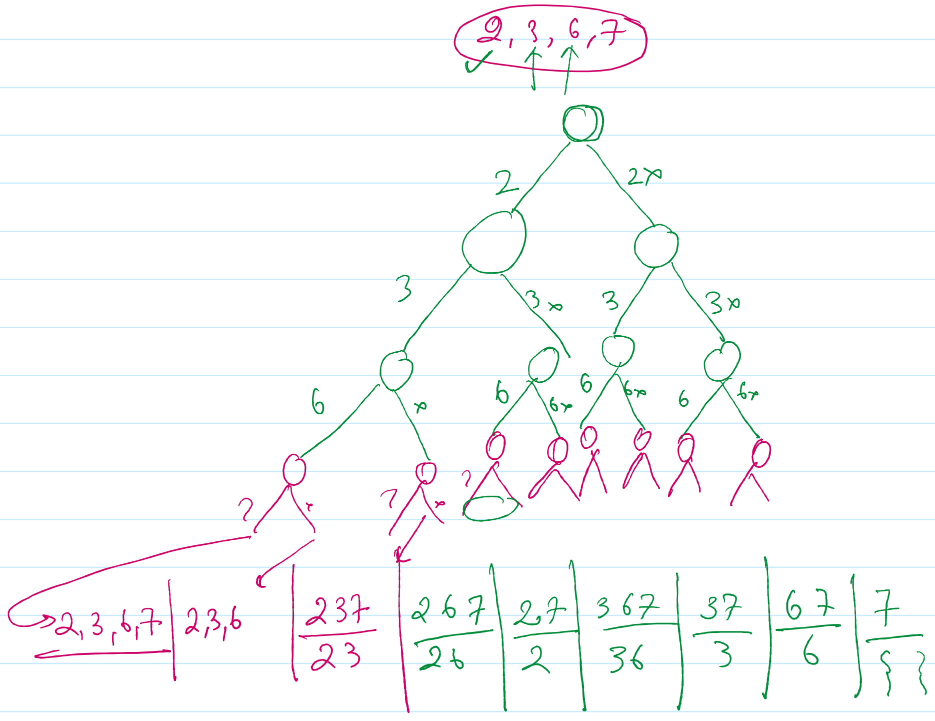
find all subarrays of an array

different problem

Take not take | Take leave | Select, not select

approach

approach



Input [2, 3, 6, 7] | index | store

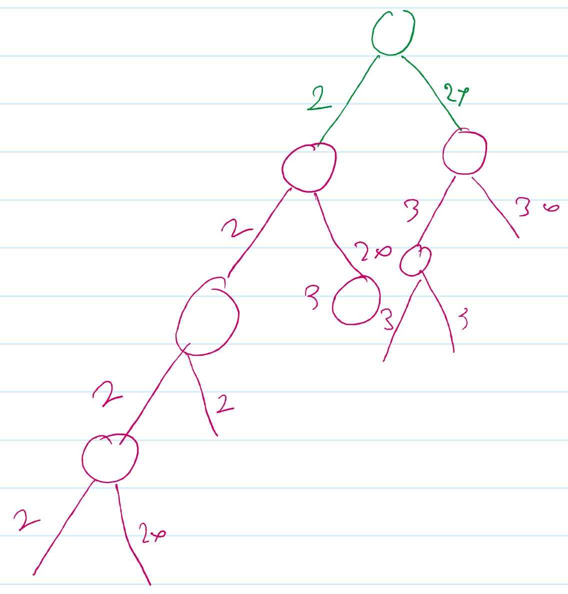
```
ArrayList<ArrayList<int>> ans = new ArrayList<>();

fun( int[] arr , int n,  curr ) → arraylist

// Base Case
if ( n == arr.length)
  {   ans.add(curr);
      return;  }
// Recursive Case

//take
    curr.add (arr[n])
    fun (arr, n+1, curr)
                n ——→ repetion
//not take
    curr.remove(curr.length-1) → remove last element
    fun (arr, n+1, curr)
```

Take an example & Run it

Draw the recursion tree

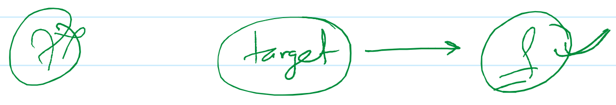
Repitition is allowed

(2, 3, 6, 7)

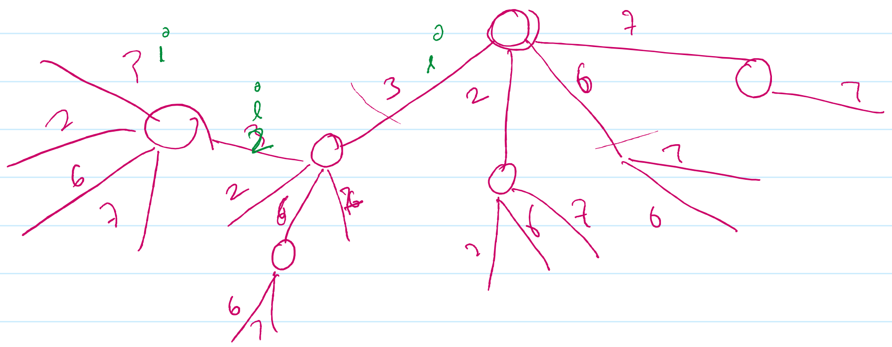


[take → stay at same index
not take → move ahead

gist

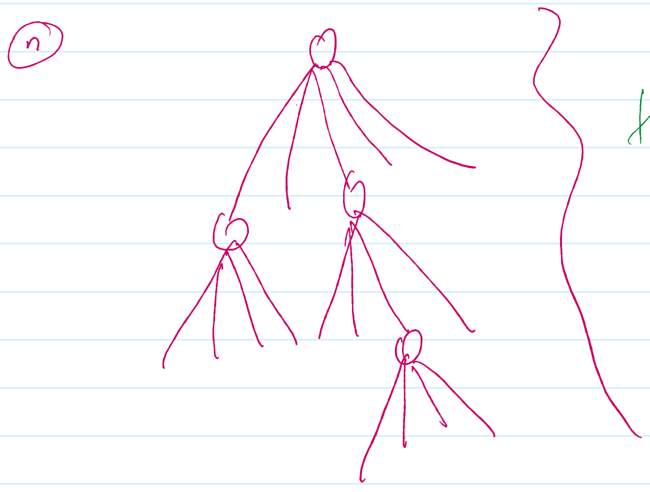


3, 2, 6, 7



target

without repitn



How to code it

[2, 3, 6, 7 - - - - - - - -]

How to code

```
fun( arr, n, curr)
  // Base
    if (n==arr.length) { ans.add(curr); return}

  // Recursn
    for( i=n; i< arr.len; i++)
      curr.add( arr[i])
      fun(arr, i , curr)
      curr.remove _last ele
    }
```