[BACKTRACKING]:

controlled recursion + recursion ke base case se return maarte time kuch work karna !!

print 1 n times :

$n=5$

O/P ⇒ 1 1 1 1 1

```
void print (int n)        // n = 5
  if (n==0) return;
  print (n-1);     0
  cout << "1" << endl;
}
```

```
void print (int n)        // n = 4
  if (n==0) return;
  print (n-1);     0
  cout << "1" << endl;
}
```

```
void print (int n)        // n = 3
  if (n==0) return;
  print (n-1);     0
  cout << "1" << endl;
}
```

```
void print (int n)        // n = 0
  if (n==0) return;
  print (n-1);
  cout << "1" << endl;
}
```

```
void print (int n)        // n = 1
  if (n==0) return;
  print (n-1);     0
  cout << "1" << endl;
}
```

```
void print (int n)        // n = 2
  if (n==0) return;
  print (n-1);     0
  cout << "1" << endl;
}
```

⑥ f(5, sum=0)

```
f (n, sum=0)              // n = 3
if (n==0) ret sum;
sum = sum + n      3
f(n-1, sum);
```

```
f (n, sum=0)              // n = 2, sum = 3 0
if (n==0) ret sum;
sum = sum + n      5
f(n-1, sum);
```

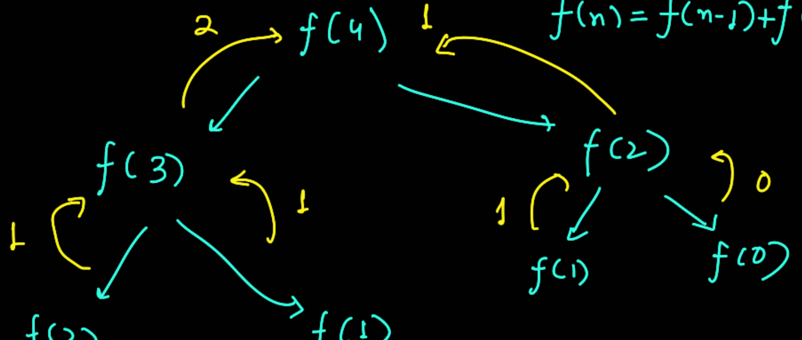
```
f (n, sum=0)              // n = 1, sum = 5 0
if (n==0) ret sum;
sum = sum + n      6
f(n-1, sum);
```

6

```
f (n, sum=0)              // n = 0, sum = 6 0
if (n==0) ret sum;
sum = sum + n
f(n-1, sum);
```

[Recursion tree for fibonacci sequence]:

$f(n) = f(n-1) + f(n-2)$



$f(4) = 3$ and

$$2^0 + 2^1 + 2^2 + \cdots \quad \underbrace{\text{n terms.}}$$

$$\frac{2^0(2^n - 1)}{2-1} = 2^n - 1$$

Time complexity = $O(2^n)$

Recursion on Subsequences : a contiguous/ non-contiguous sequence, which follows the order.

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

all subsequences ⇒

	3	1	2
{ }	×	×	×
{3}	✓	×	×
{1}	×	✓	×
{2}	×	×	✓
{3, 1}	✓	✓	×
{1, 2}	×	✓	✓
{3, 2}	✓	×	✓
{3, 1, 2}	✓	✓	✓

Total = $2^n = 2^3 = 8$

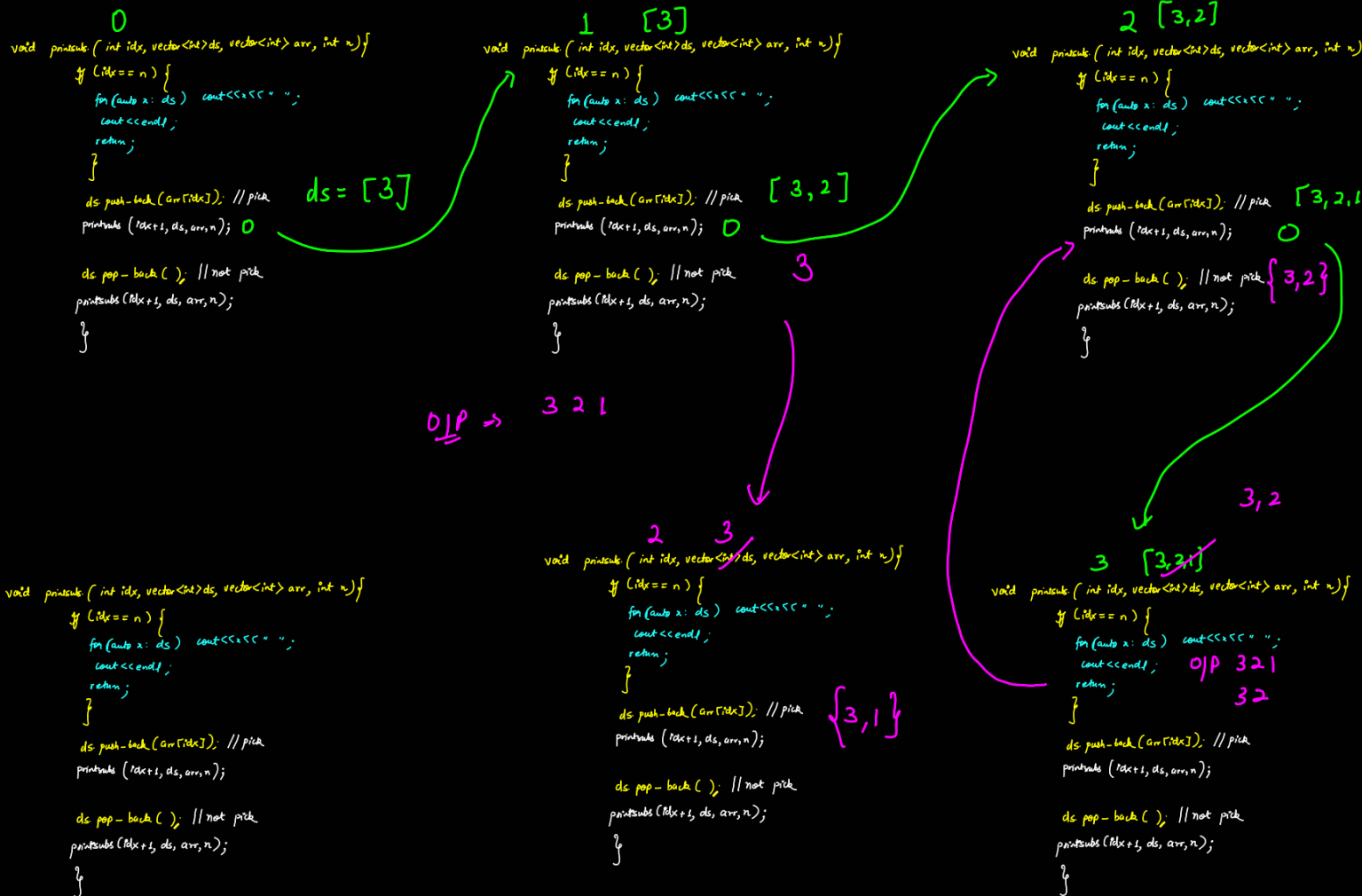
using pick not pick strategy : ⇒ VVImp Piece of code

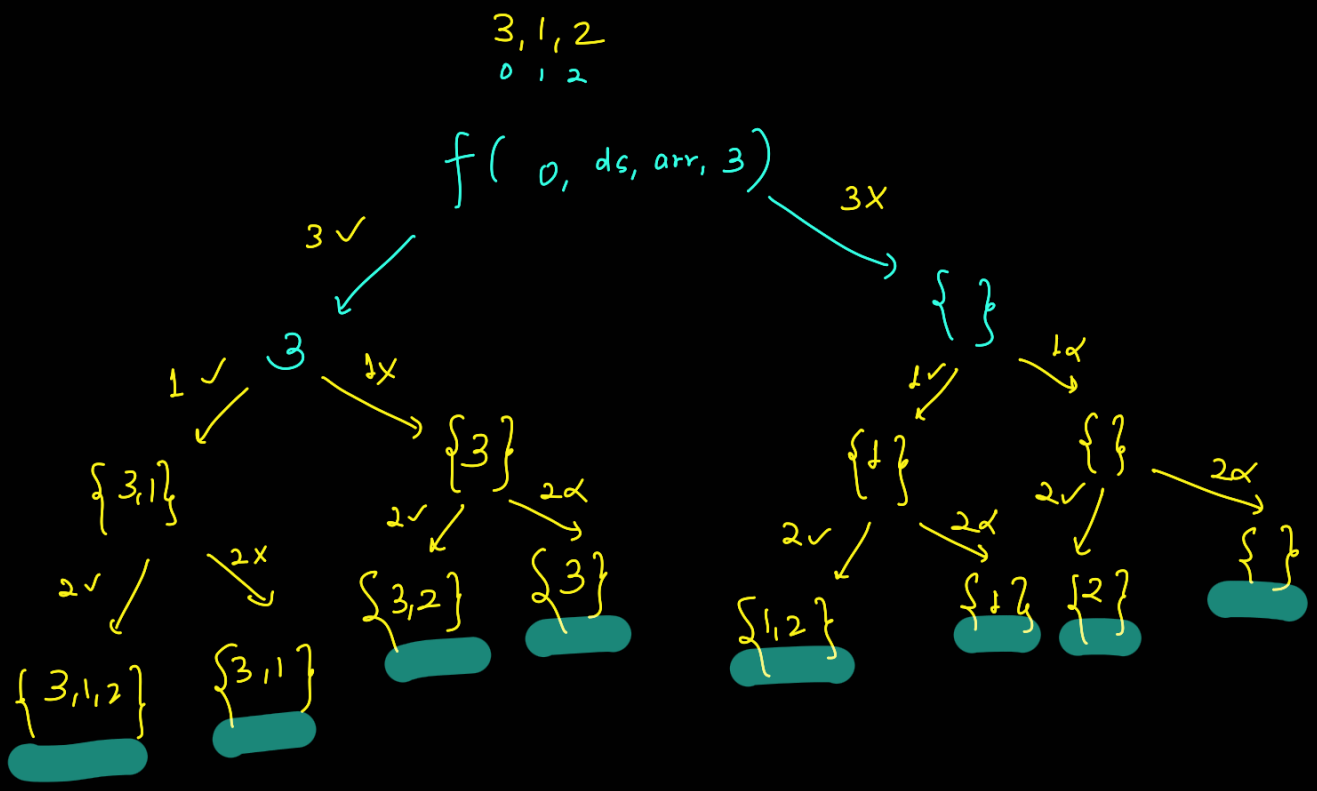
```
void printsubs ( int idx, vector<int> ds, vector<int> arr, int n ){
    if (idx == n ) {
        for (auto x : ds )  cout<<x<<" ";
        cout<<endl;
        return;
    }
    ds.push_back(arr[idx]); // pick
    printsubs (idx+1, ds, arr, n);

    ds.pop_back( ); // not pick
    printsubs (idx+1, ds, arr, n);
}
```

→ arr — this func" prints all combos from idx to n.

arr = [3, 2, 1]



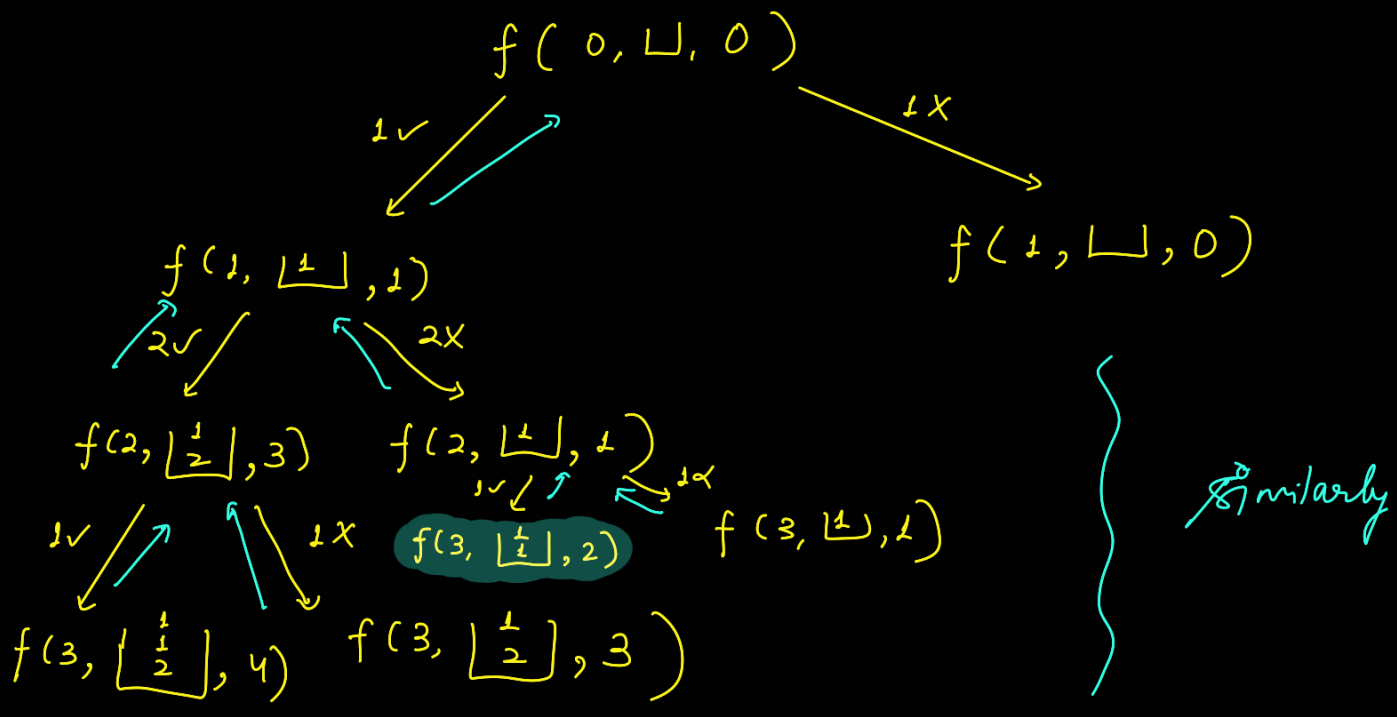


Subseq. sum k.

k = 2

[1, 2, 1]

f(i, ⌊⌋, sum)



Pattern 2

Ques → Print only one subsequence whose sum is k.

☆☆☆☆☆ technique to print only one ans → functional recursion

f(){

↳ base case:

cond → satisfied

return true;

else return false;

```
}
if (f() == true) return ;
else return false;
```

Pattern3 ⇒ Give the count of subsequences with sum k.

$[1, 2, 1]$ $k = 2$ O/P : count = 2

```
f() {
    base case
    return 1 → cond^n satisfies;
    return 0 → cond^n do not satisfies.
    left = f();
    right = f();
    return left + right;
}
```

agar n recursion calls ho toh :

```
s = 0
for(i = 1 → n) {
    s += f();
}
return s;
```

arr = [1, 2, 1] Sum = 2.

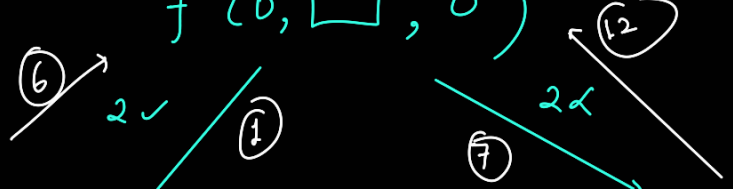
Ques ⇒ Subset Sum → {2, 3}

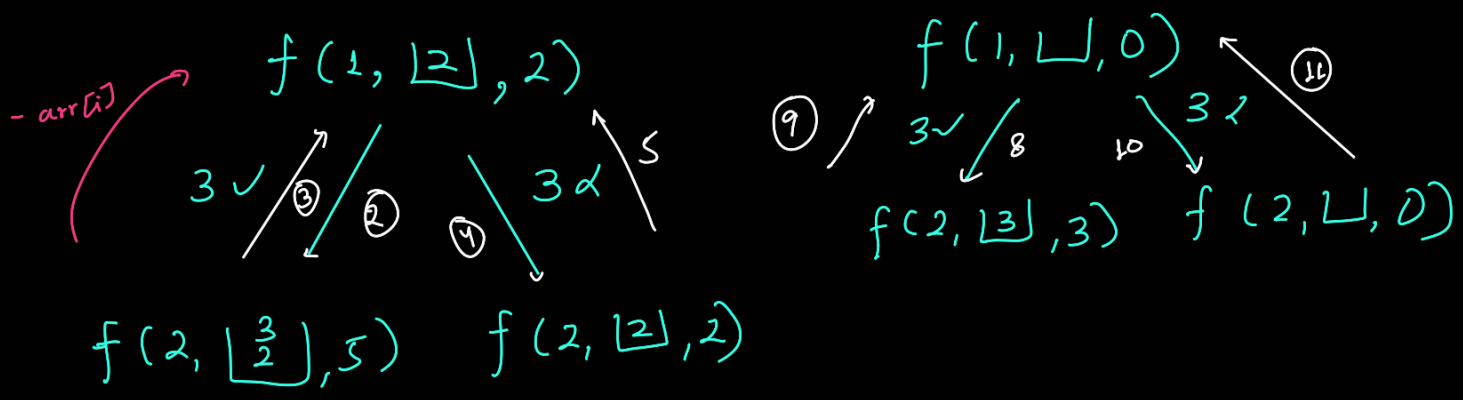
Subsets →			Sum
✓	✓	{2, 3}	5
✗	✗	{ }	0
✓	✗	{2}	2
✗	✓	{3}	3

O/P = {0, 2, 3, 5}

f(i, ds, sum) ⌊⌋ = {2, 3}

f(0, ⌊⌋, 0)



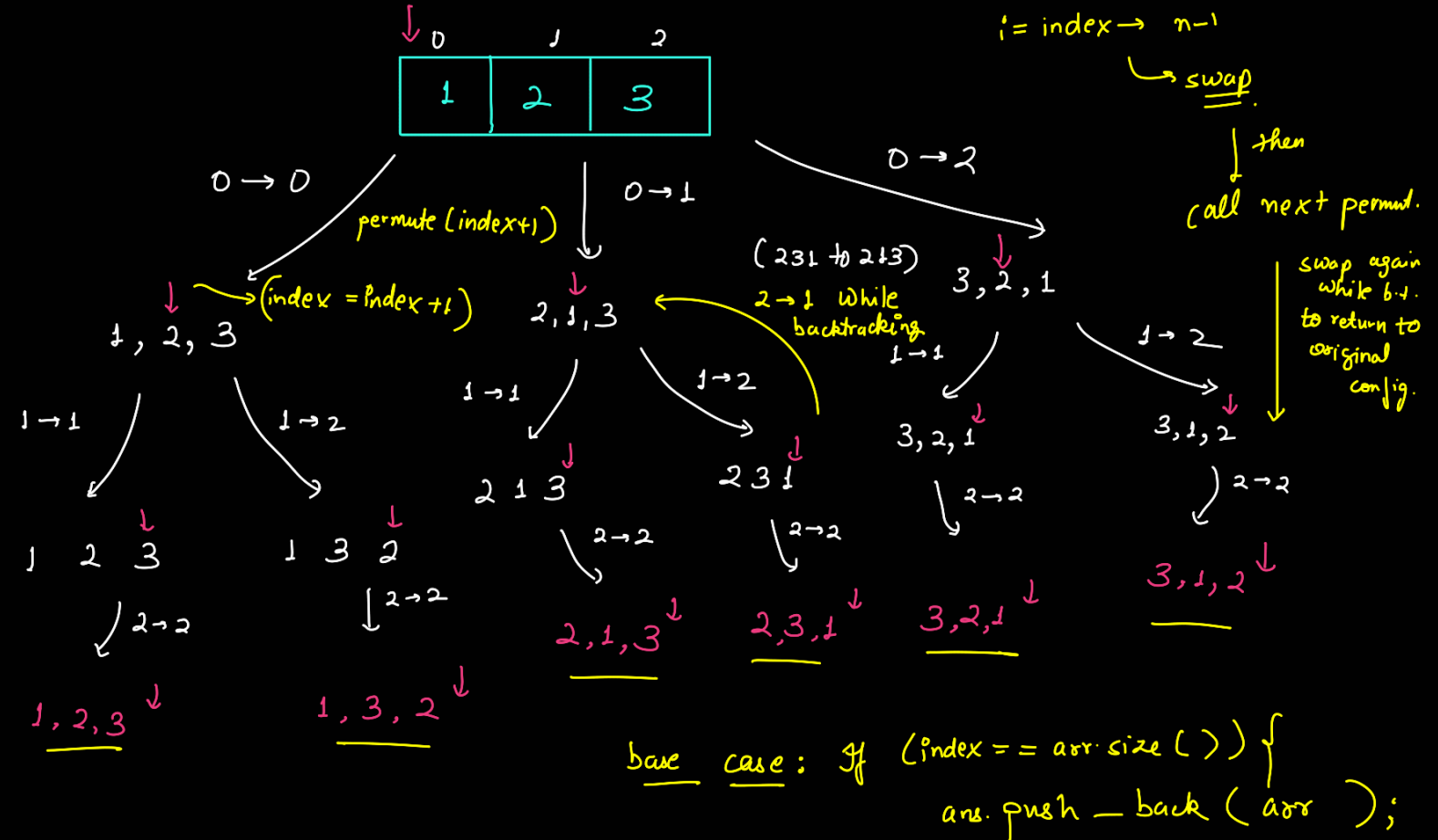



```
base if (i == n) {
    ans.push_back(sum);
```

While backtracking,
- o subtract the sum.

Generate Permutations of an array/string:

(Recursion tree): f(index, nums, ans)



base case: if (index == arr.size()) {
    ans.push_back(arr);

return;
}