Backtracking →

Algorithmic technique used to find solution by exploring all the possible candidate for your ans.

① Print all N-digit numbers which can be formed by {1, 2}

N = 3.

1 1 1
1 1 2
1 2 1
1 2 2
2 1 1
2 1 2
2 2 1
2 2 2

1/2 1/2 1/2

1 [2 digit (1/2)]

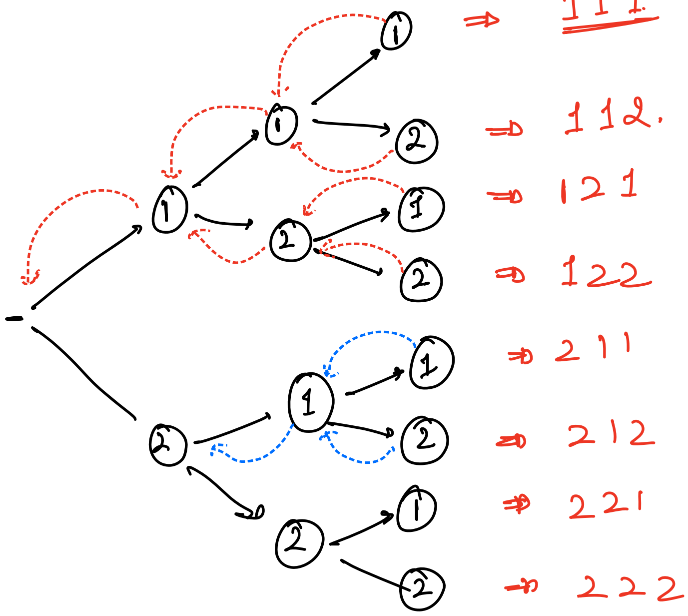
2 [2 digit no. {1,2}]

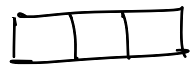
} **Recursion**

idea 1. Using Bit Manipulation

⇒ 0 to $2^N - 1$.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

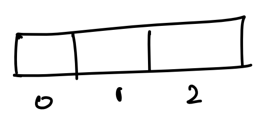




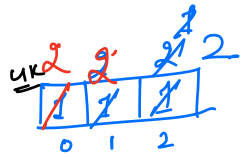
parameters → i , n , temp[n]

i ⇓ current position to be fixed

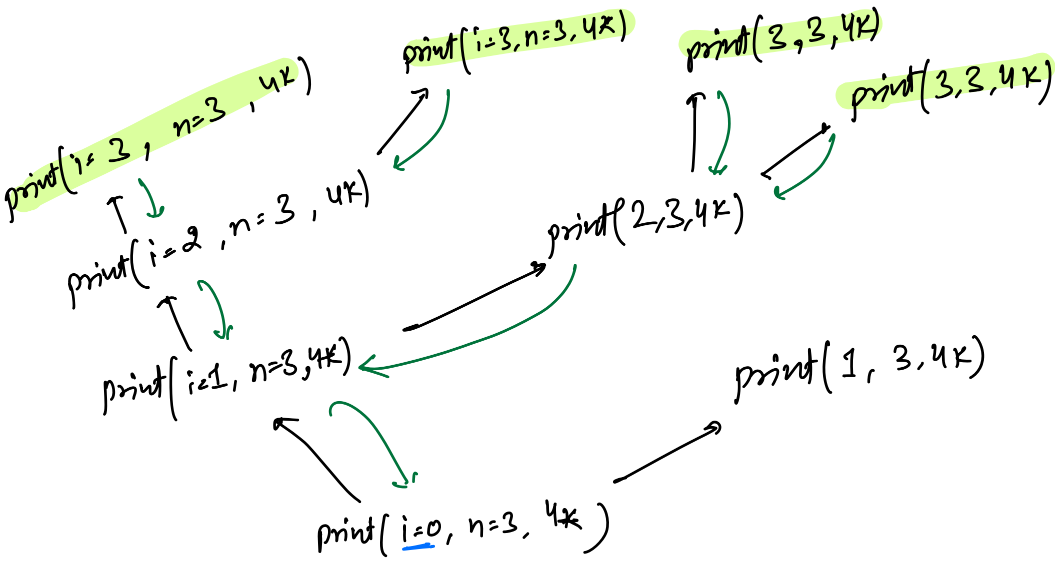
```
void print( int i, int n, int temp[n] ){
    if( i == n ){ for( j=0; j<n; j++ ){ print( temp[j] ) }
                                        return }

    temp[i] = 1 ;
    print( i+1, n, temp );

    temp[i] = 2 ;
    print( i+1, n, temp );
}
```



```
void print(int i, int n, int temp[n]){
    if(i == n){ for(j=0; j<n; j++){ print(temp[j]) } }
                                 return

    temp[i] = 1 ;
    print(i+1, n, temp);

    temp(i) = 2 ;
    print(i+1, n, temp);
}
```

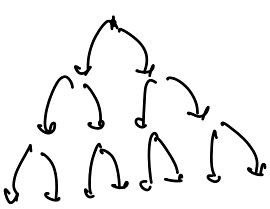


1 1 1
1 1 2
1 2 1
1 2 2



no. of recursive calls * time taken

N=3.



$T.C \rightarrow O(2^N * N)$

function calls

time to print the temp[]

$S.C \rightarrow O(N)$

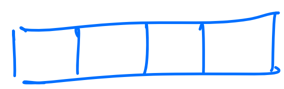
1
2
4
8
16
32

digits → { 1, 2, 3, 4, 5 }

```
void print ( int i, int n, int temp(n] ){
   void print ( int i, int n, int temp(n] ){
      if( i == n){ for( j=0; j < n; j++) { print (temp (j]) } }
      if( i == n)  for( j=0; j < n; j++){ print (temp (j]) } }
                                        return

         for( j=1 ; j <= 5 ; j++){
                                        i++){
         {
               temp (i] = j
               print( i+1, n, temp);
                                   n , temp)
         }

         }
}
}
```

{ use loop when options are very large in no.

options → { 2, 3, 7, 9 }



```
void print( int i, int n, temp(n] , options[] ){
{
        // base condition

        for( j = 0 ; j < options.size(); j++){
        {
                temp (i] = options [j]
                print( i+1, n, temp, options)
        }
}
```

Q:) Find the count of subsets with sum = K.

Given N distinct elements.

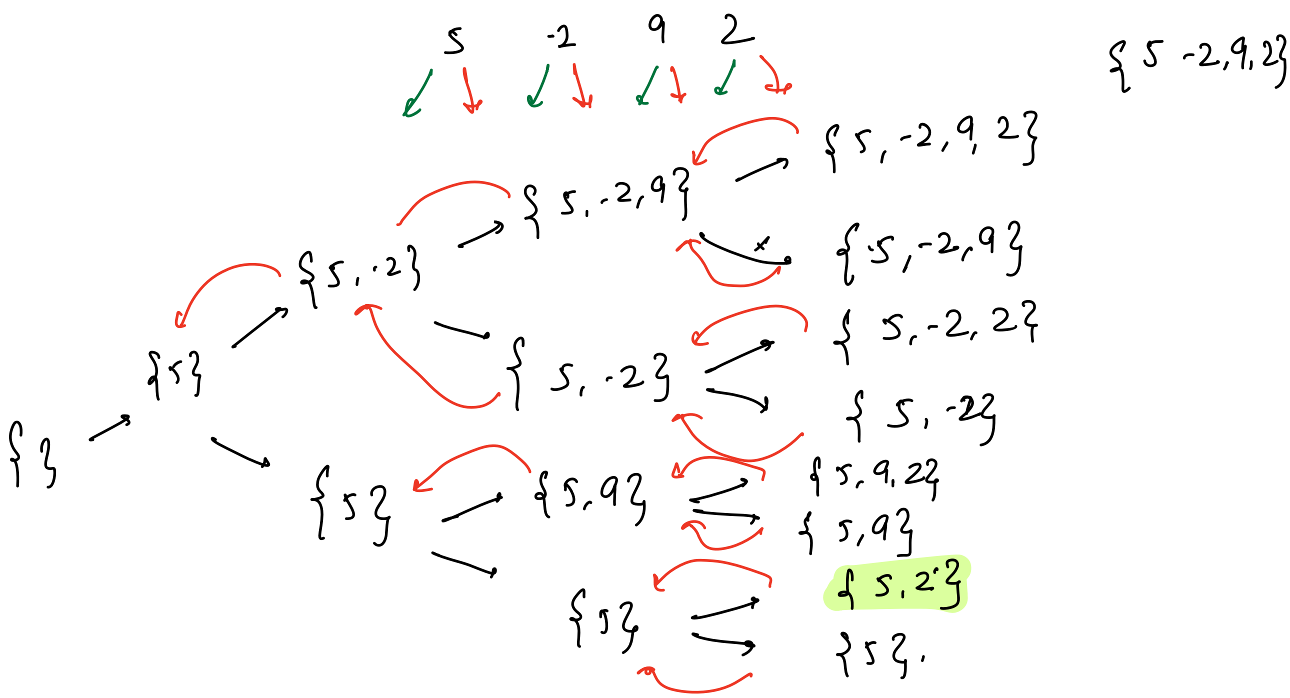
arr[] = [5   -2   9   2], K = 7

(indices: 0   1   2   3)

[5, 2]
[-2, 9]

ans = 2.

B.F. Consider all the subsets.



parameters

i, n, currsum, arr[], K

```
int countSubsetsWithSumK ( i, n, currsum, arr[N] , k ){
    if ( i == n) {  if ( currSum == k) return 1 }
                    else return 0

    ans = 0
    // option no.1 → if current element is included
        currsum += arr[i]
    ans += countSubsetsWithSumK (i+1 , n, currsum, arr, k)

    //option no.2 → if current element is not included
        currSum -= arr[i]
    ans += countSubsetsWithSumK ( i+1, n, currsum, arr, k)

    return ans;
}
```

$$\begin{bmatrix} T.C \rightarrow O(2^n) \\ S.C \rightarrow O(N) \end{bmatrix}$$

→ [Generate all the subsets.]

H.W.

Break → 10:38 → 10:45

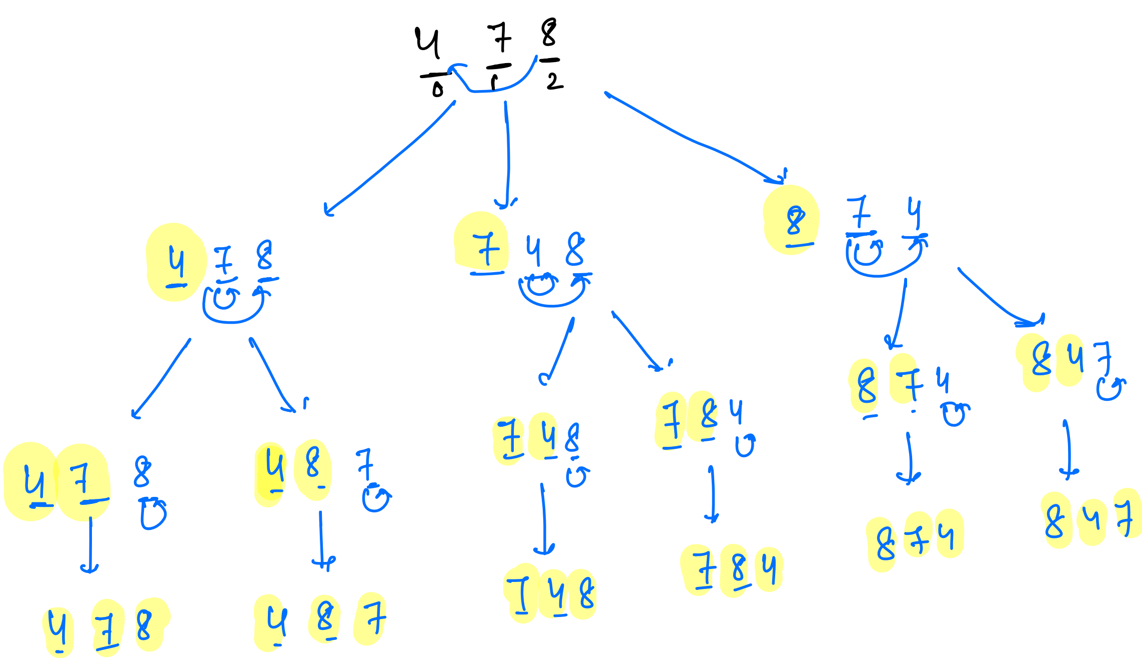
Q. Generate All Permutations (unique elements)

4 7 9

4 7 9
4 9 7
7 4 9
7 9 4
9 4 7
9 7 4

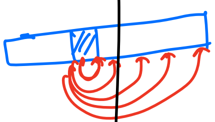
n elements
↓
n!

$n * (n-1) \cdot (n-2) \_\_\_\_ 1$
⇓
$n!$



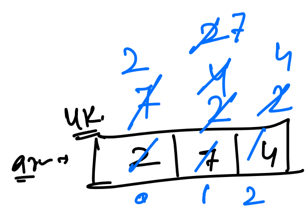
parameters → i, n, arr[]

```
void permutations (int i, int n, int arr[n]) {
    if(i == n) { print arr[] & return }

        for( j = i ; j < n ; j++) {
            swap ( arr[i], arr[j])
            permutations( i+1 , n , arr)
            swap ( arr[i], arr[j] )
        }
}
```

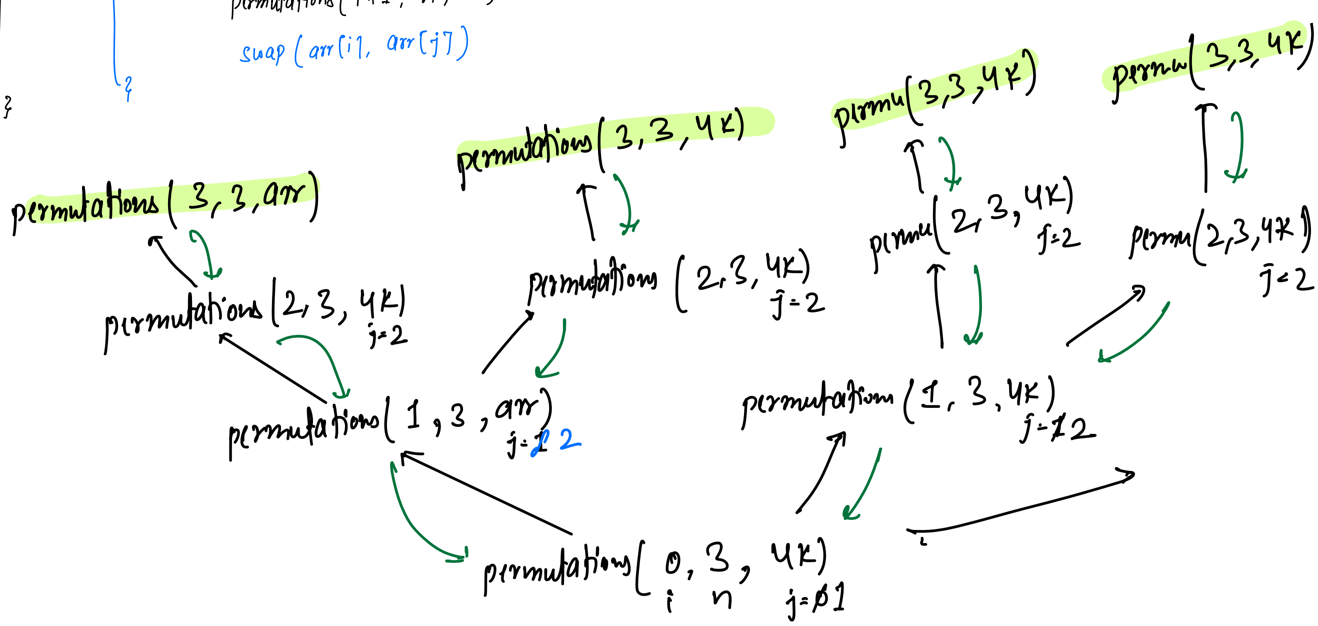


$\left\{ \begin{array}{l} T.C \rightarrow O(n! * n) \\ S.C \rightarrow O(n) \end{array} \right\}$


```
void permutations (int i, int n, int arr[n]) {
    if(i == n) { print arr[] & return }

        for( j = i ; j < n ; j++) {
            swap ( arr[i], arr[j])
            permutations( i+1 , n , arr)
            swap ( arr[i], arr[j] )
        }
}
```

2, 7, 4
2, 4, 7
7, 2, 4
7, 4, 2



Duplicates?

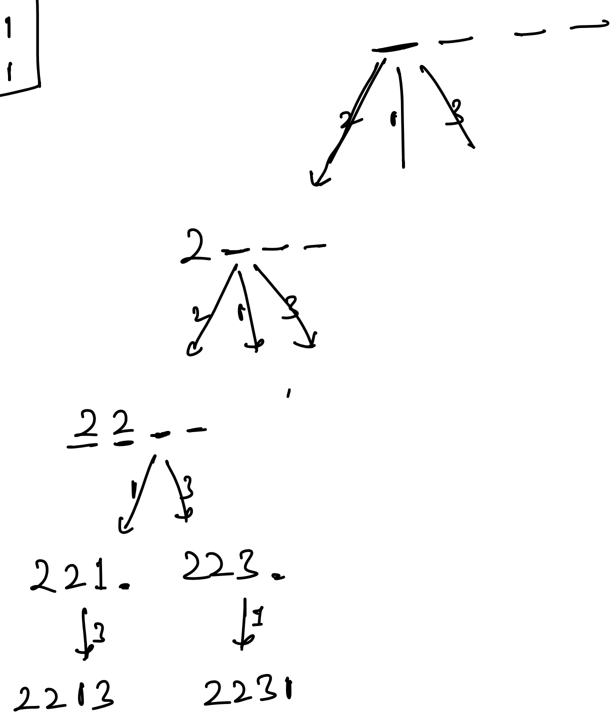
swap ✗

2 1 2 3

2123 1223 2123 3122

2	$\cancel{2}$ 1 0
1	1
3	1

$[2,1,2,3] \Rightarrow \frac{4!}{2!} = \frac{4\times3\times\cancel{2}}{\cancel{2}} = 12$



1, 2, 2, 3
1, 2, 3, 2
1, 3, 2, 2
2, 1, 2, 3
2, 1, 3, 2
2, 2, 3, 1
2, 2, 1, 3
2, 3, 1, 2
2, 3, 2, 1
3 1 2 2
3 2 1 2
3 2 2 1

Hm → freq.

```
void permutations( i , n, arr, freq ){
    if(i == n) { ——— }

    for( traverse in the hashmap){
        if( hm[x] > 0 ){
            arr[i] = x
            hm[x] --
            permutations( i+1, n, arr, freq)
            hm[x] ++
        }
    }
}
```

T.C → $O(n! * n)$
S.C → $O(n)$

Structure

func( ) {

// base-case.

try all the options {

$\boxed{}$ do.

recursive call

$\boxed{}$ undo

}

}

→ Backtracking.

- N Queens.
- Rat in maze.
- Sudoku.

- <u>Word Break</u>