"The key of persistence opens
all doors closed by resistance."

— John Di Lemme

SCALER

Backtracking

TABLE OF CONTENTS

Advanced Contest Syllabus

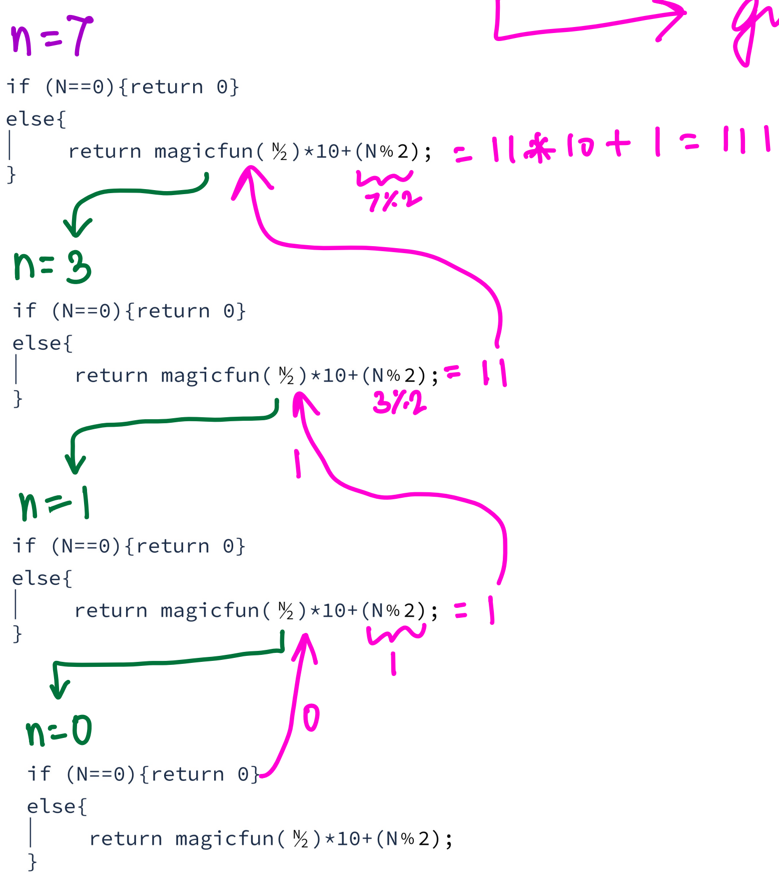
→ Arrays

→ Bit Manipulation

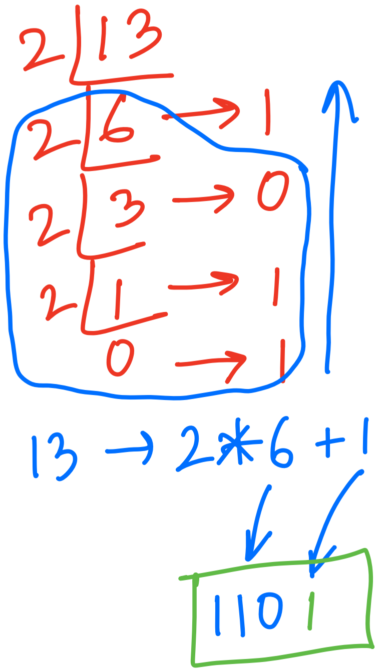
Output based question →

1.
```
int magicfun (int N){
    if (N==0){return 0}
    else{
        return magicfun( N/2 )*10+(N%2);
    }
}
```

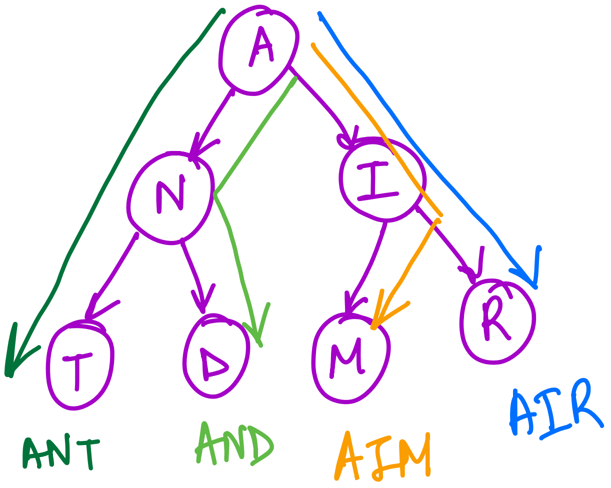
$f(n) = f(n/2) * 10 + (n\%2)$ → gives the binary of $(n)_{10}$.

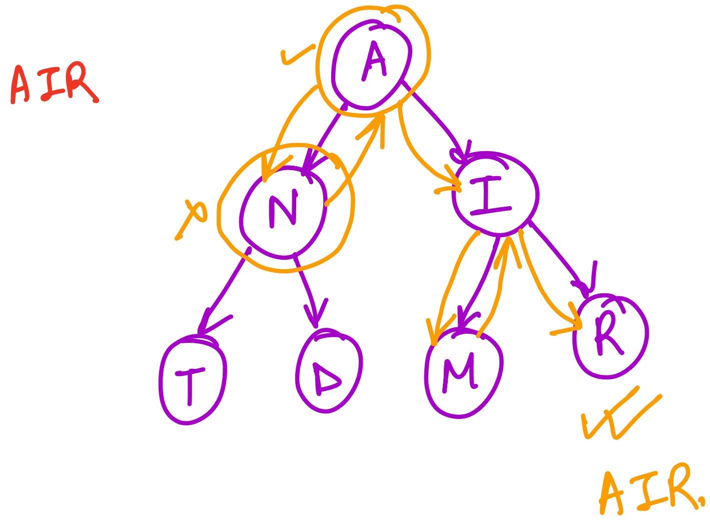


```
if (N==0){return 0}
else{
    return magicfun( N/2 )*10+(N%2);
}
```



Backtracking





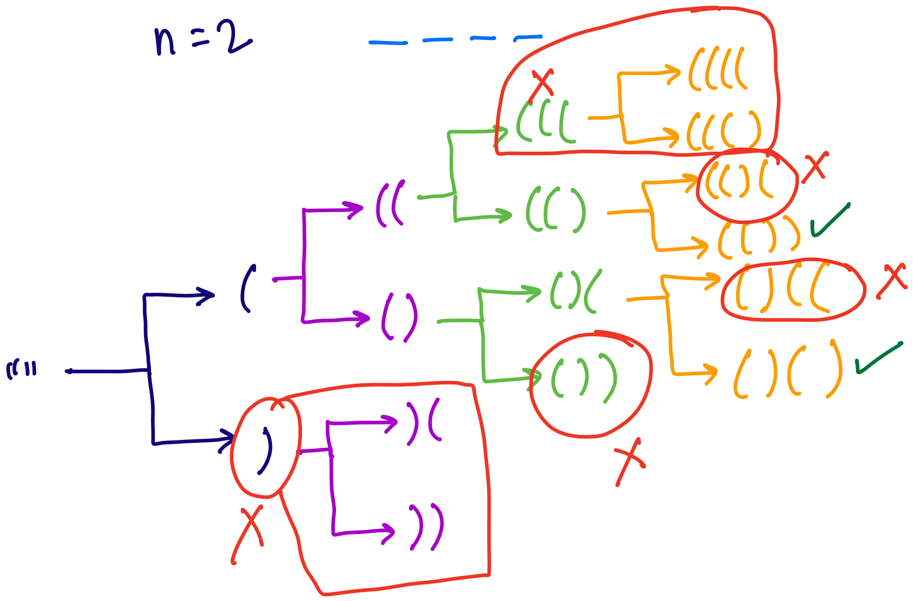
Valid Parenthesis

Generate all valid combinations of parenthesis of length 2N.

N=3

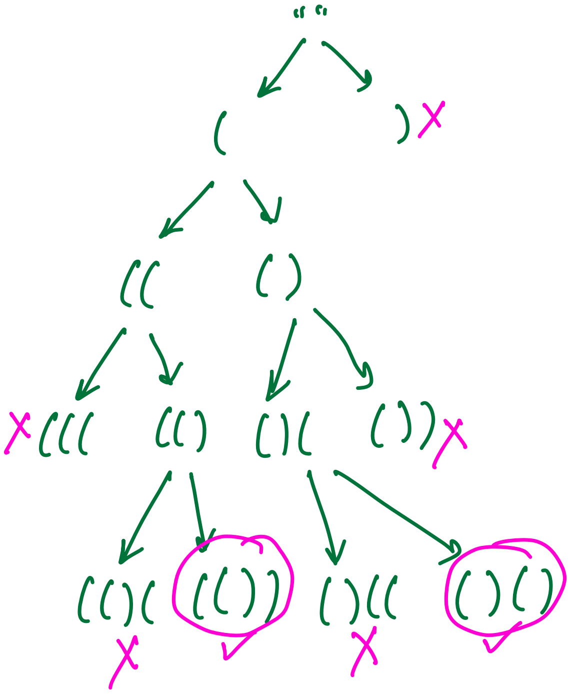
((()))
()(())
(())()
()()()
(()())

n=2



```
PrintValidParenthesis(n, 0, 0, new char[2*n])

void PrintValidParenthesis(int n, int opening, int closing,
                                                  char[] str){
        idx = opening + closing
        if(idx == 2*n)
            //Print str
        if(opening > closing){
            str[idx] = ')'
            PrintValidParenthesis(n, opening, closing+1, str)
        }
        if(opening < N){
            str[idx] = '('
            PrintValidParenthesis(n, opening+1, closing, str)
        }
}
```



→ level 1 $\leq 2^1$

→ level 2 $\leq 2^2$

→ level 3 $\leq 2^3$

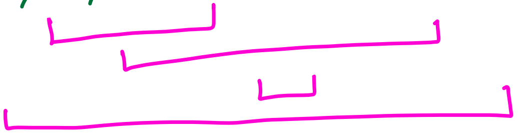
→ level 4 $\leq 2^4$

$O(2^n)$ T.C

$O(n)$ S.C

Each subsequence of an array ⟺ A subset of the array.

{3, 5, 2, 9, 6, 4, 10, 15}



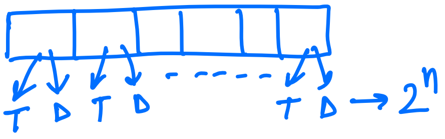
subarrays $\frac{n(n+1)}{2}$

{3, 5, 2, 9, 6, 4, 10, 15}

5 6 4 15

2 9 6 4 15

subsets (2^n)



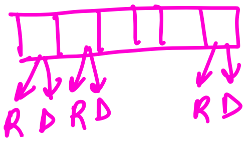
{3, 5, 2, 9, 6, 4, 10, 15}

~~3~~, 5, ~~2~~, 9, ~~6~~, ~~4~~, 10, 15 → 5 9 10 15

~~3~~, ~~5~~, 2, 9, 6, 4, ~~10~~, 15 → 2 9 6 4 15

~~3~~, ~~5~~, ~~2~~, ~~9~~, ~~6~~, 4, ~~10~~, ~~15~~ → 4

subsequences



↓

2^n

② Generate all subsets of the given arr[]

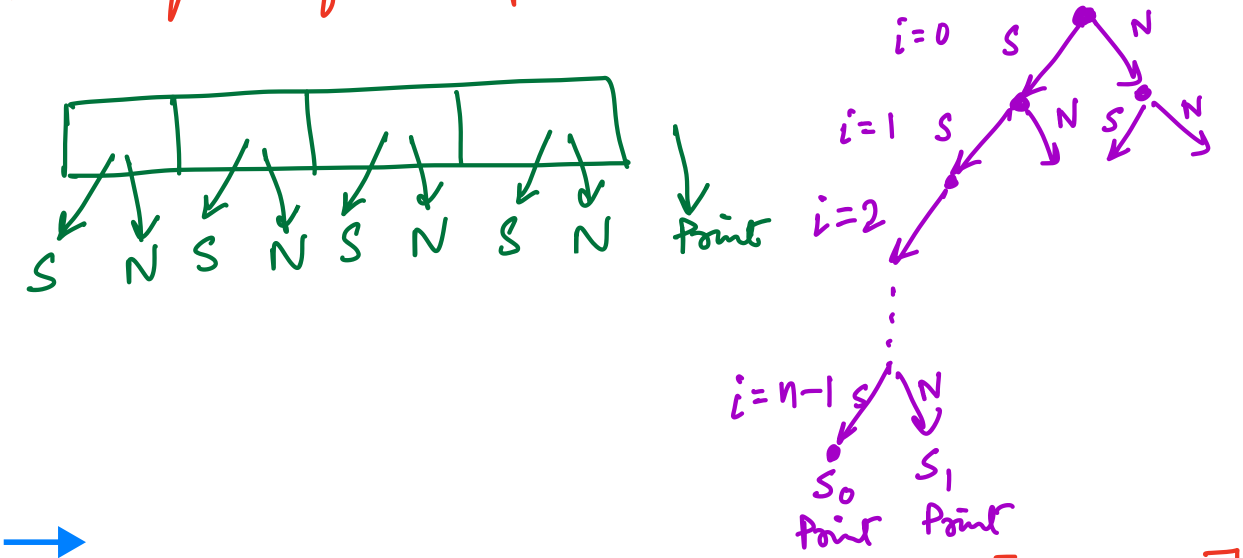
$2 * 2 * 2 = 2^3 = 8$.

arr → [10 , 20 ,30]
0 1 2

Total no. of subsets = { } {10} {20} {30} {10, 20} {10, 30} {20, 30} {10, 20, 30}

arr[n] → 2^n subsets .

Each subsequence of an array ⟺ A subset of the array .

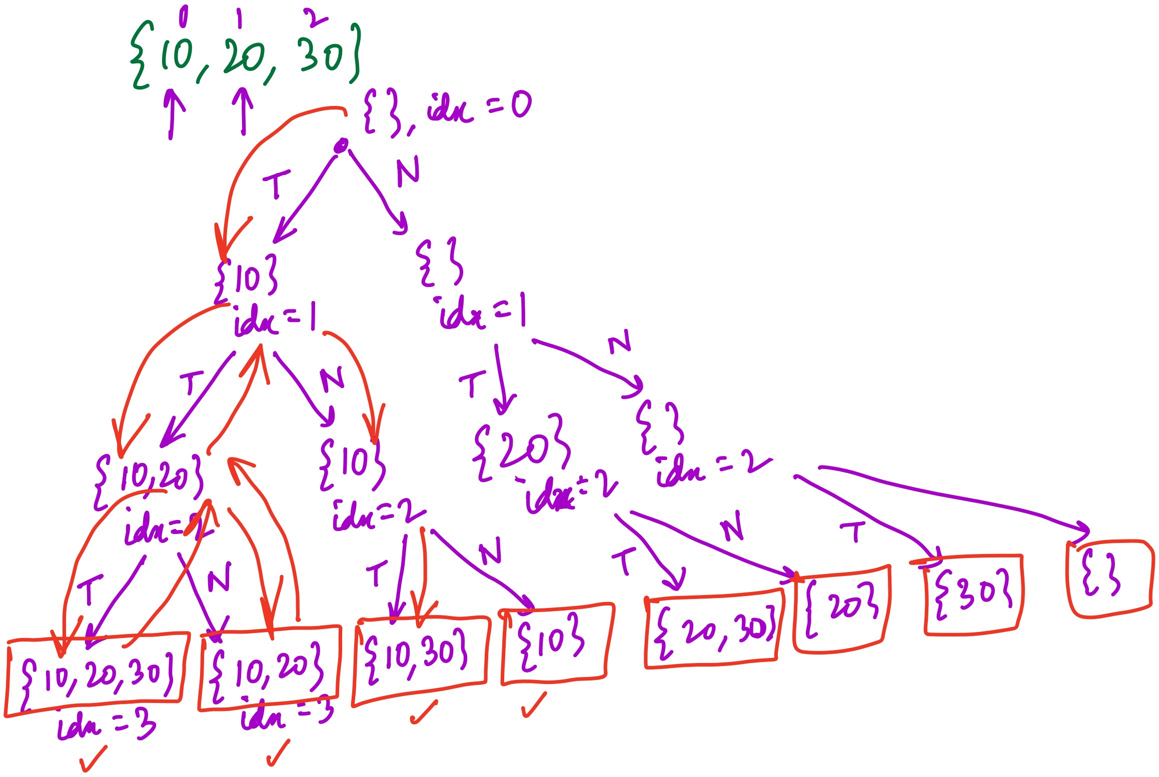


code →

// Print all subsets of arr by selecting all combinations in [idx, n-1]

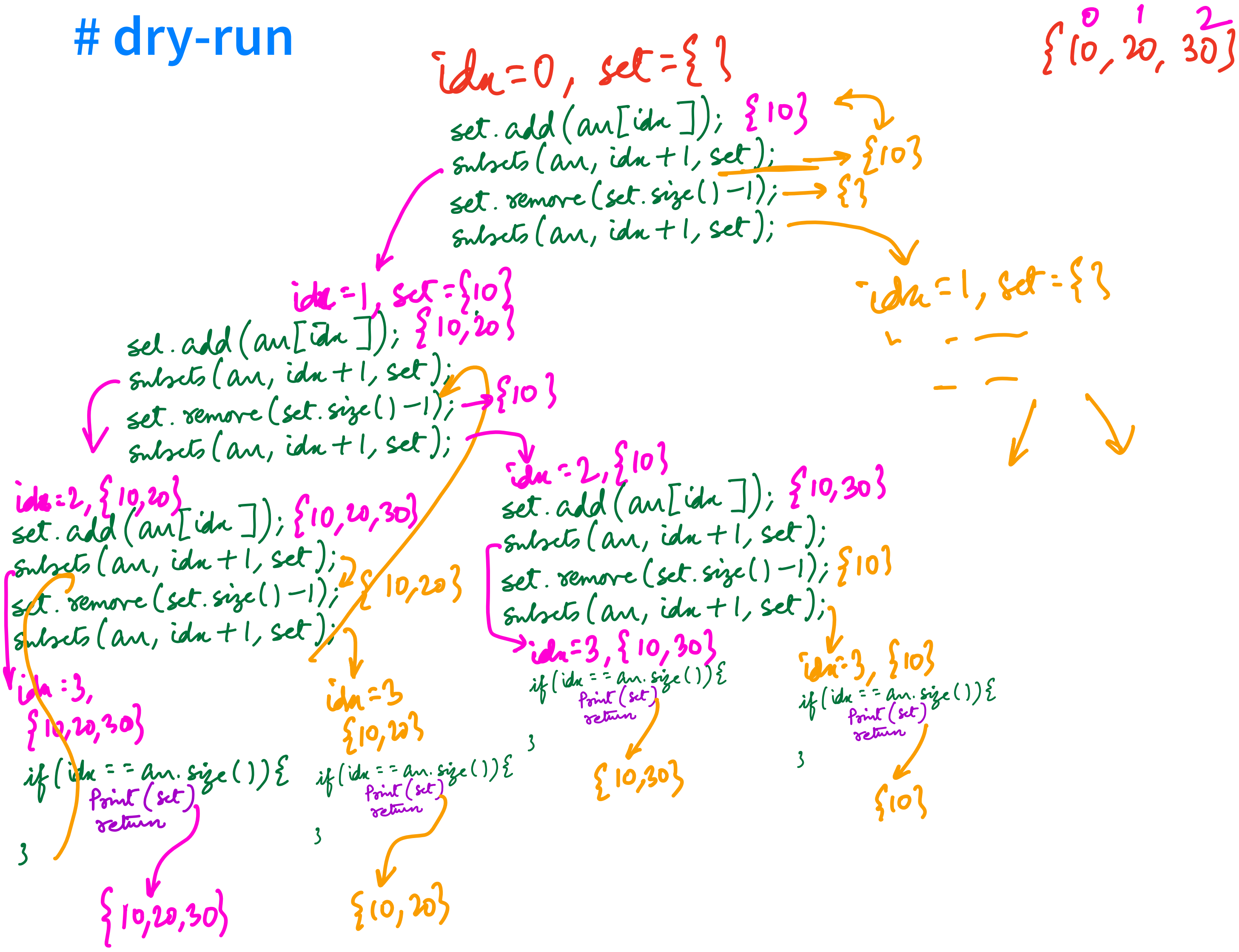
```
void subsets (int() arr,int idx,list<int>set){
    if(idx == arr.size()){
        Print(set)
        return
    }
    set.add(arr[idx]);
    subsets(arr, idx+1, set);
    set.remove(set.size()-1);
    subsets(arr, idx+1, set);
}
```

[Break till 10:42PM]



```
if(idx == arr.size()){
        Print(set)
        return
}
set.add(arr[idx]);
subsets(arr, idx+1, set);
set.remove(set.size()-1);
subsets(arr, idx+1, set);
```

dry-run



0 1 2
{10, 20, 30}
idx=0, set={ }
set.add(arr[idx]); {10}
subsets(arr, idx+1, set); {10}
set.remove(set.size()-1); { }
subsets(arr, idx+1, set);
idx=1, set={ }
idx=1, set={10}
set.add(arr[idx]); {10,20}
subsets(arr, idx+1, set);
set.remove(set.size()-1); {10}
subsets(arr, idx+1, set);
idx=2, {10,20}
set.add(arr[idx]); {10,20,30}
subsets(arr, idx+1, set);
set.remove(set.size()-1); {10,20}
subsets(arr, idx+1, set);
idx:3, {10,20,30}
if(idx == arr.size()){
print(set)
return
}
{10,20,30}
idx=3
{10,20}
if(idx == arr.size()){
print(set)
return
}
{10,20}
idx=2, {10}
set.add(arr[idx]); {10,30}
subsets(arr, idx+1, set);
set.remove(set.size()-1); {10}
subsets(arr, idx+1, set);
idx=3, {10,30}
if(idx == arr.size()){
print(set)
return
}
{10,30}
idx=3, {10}
if(idx == arr.size()){
print(set)
return
}
{10}

Permutations

Given a character array with distinct elements, print all permutations of it without modifying it.

A → [a b c]

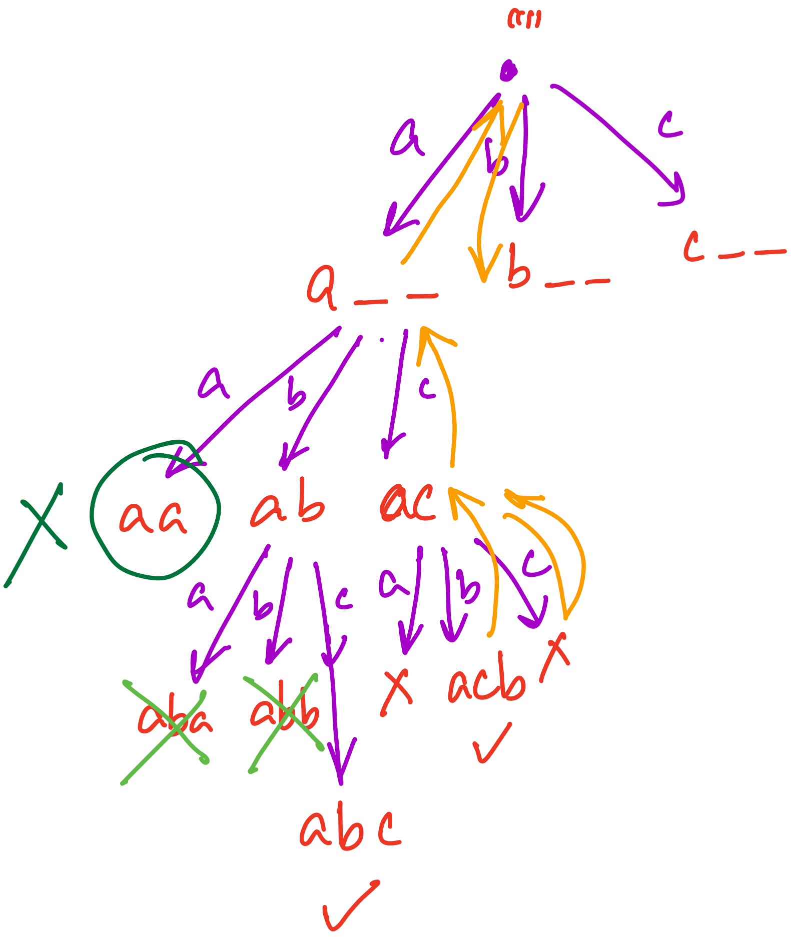
abc
acb
bac
bca
cab
cba

_ _ _

$3*2*1 = 3! = 6.$

_ _ _ _ ... _ _ _

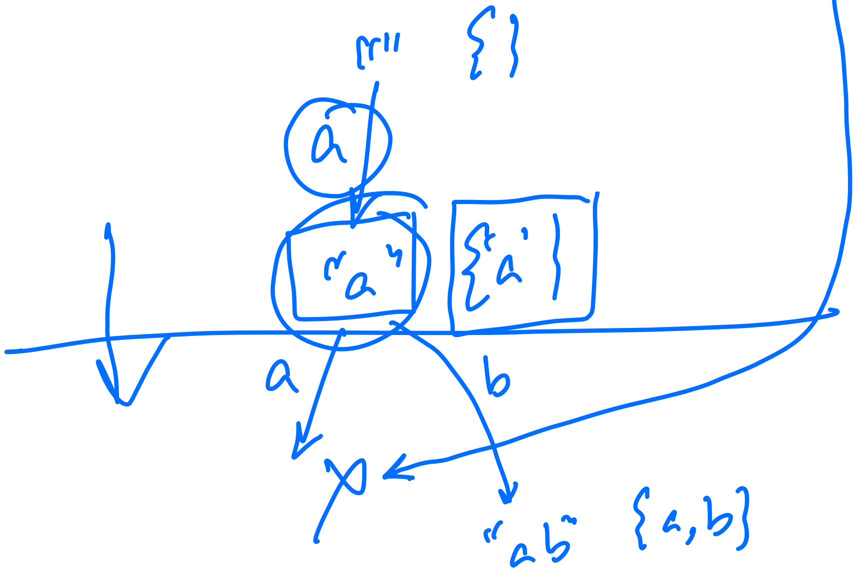
$n \quad n-1 \quad n-2 \quad n-3 \quad 3 \quad 2 \quad 1 \longrightarrow n!$



set (HashSet)
↓
to check if an element has been taken or not.

pseudo-code →

```
permutations(str, n, "", new HashSet<x>);
void permutations( string str, int n, String perm, HashSet set ){
    if (perm.size()==n){
        print(perm)
        return
    }
    for(i → 0 to n-1){
        if (!set.contains(str[i])){
            set.add(str[i])
            permutations(str, n, perm+str[i], set)
            set.remove(str[i])
        }
    }
}
```

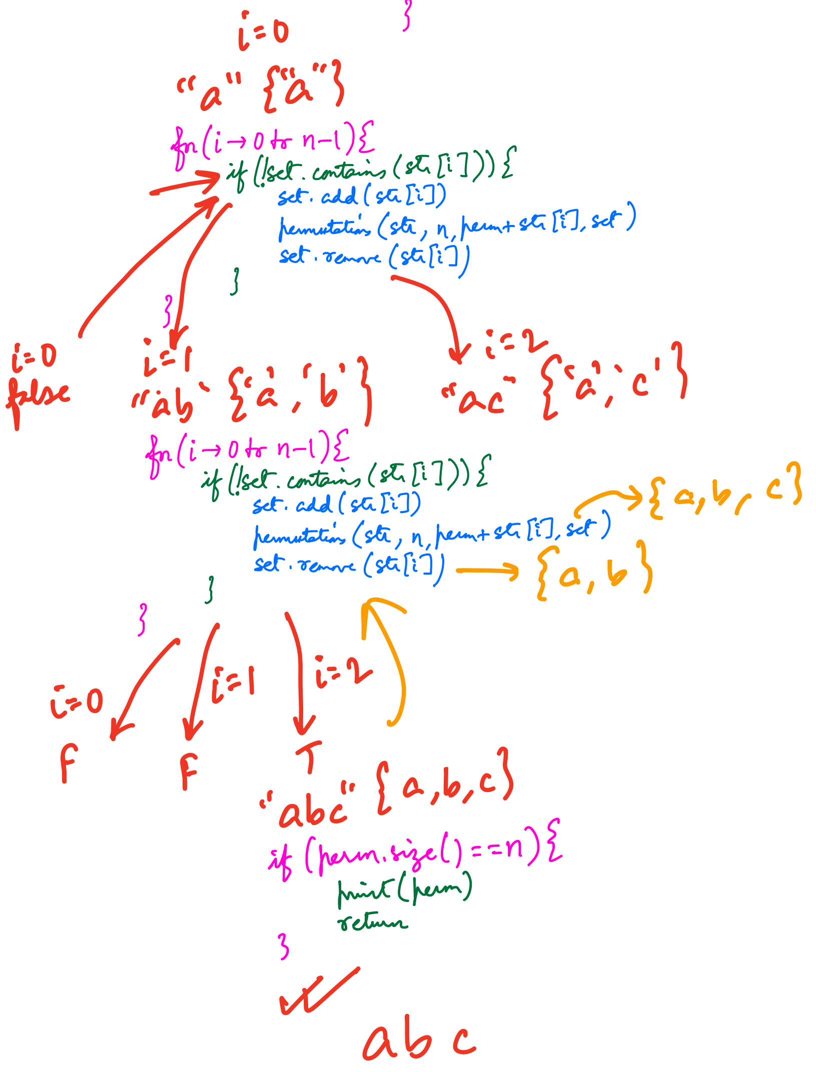


Time & Space Complexity Analysis →

$\overset{0}{a}\,\overset{1}{b}\,\overset{2}{c}$ ↑

"" n=3 "$\overset{0}{a}\overset{1}{b}\overset{2}{c}$" {}

```
for(i→0 to n-1){
    if(!set.contains(str[i])){
        set.add(str[i])
        permutations(str, n, perm+str[i], set)
        set.remove(str[i])
    }
}
```



i=0

"a" {"a"}

```
for(i→0 to n-1){
    if(!set.contains(str[i])){
        set.add(str[i])
        permutations(str, n, perm+str[i], set)
        set.remove(str[i])
    }
}
```

"ab" {'a','b'}

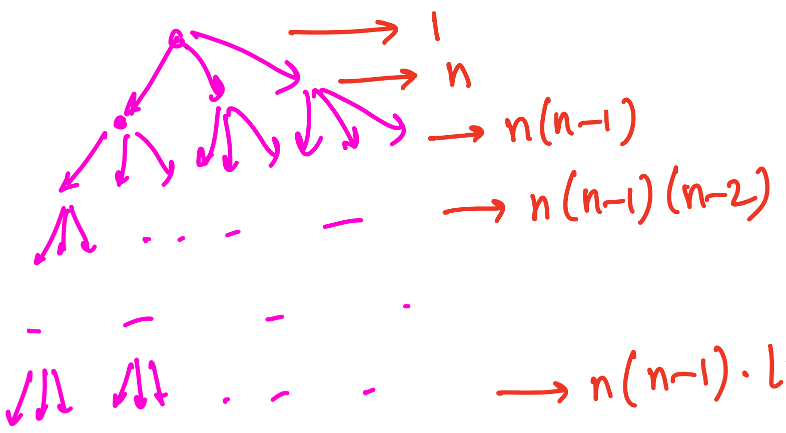
"ac" {'a','c'}

```
for(i→0 to n-1){
    if(!set.contains(str[i])){
        set.add(str[i])
        permutations(str, n, perm+str[i], set) → {a,b,c}
        set.remove(str[i]) → {a,b}
    }
}
```

"abc" {a,b,c}

```
if(perm.size()==n){
    print(perm)
    return
}
```

✓ abc



$$n! + \frac{n!}{2!} + \frac{n!}{3!} + \frac{n!}{4!} + \frac{n!}{5!} + \dots \frac{n!}{n!}$$

$$= n!\underbrace{\left(1 + \frac{1}{2!} + \frac{1}{3!} + \dots \frac{1}{n!}\right)}_{(\leq e)}$$

$$= n! \cdot e$$

$\leq e \cdot n!$

T.C. → O(n!)

S.C → O(n)

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \infty$$

$$1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} \leqslant e^x$$

$$x = 1 \Rightarrow 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!} \leqslant e^1$$