Recursion & Backtracking

Print N to 1

(1) Expectation

void printDecreasing(int n)

N = 5 → 5, 4, 3, 2, 1

(2) Faith { Recursive call }

printDecreasing(n-1);

N-1 = 4 → 4, 3, 2, 1

(3) Small work (before call)

System.out.println(n);

N = 5 → 5

(4) Base case

if (n == 0) return;

{ no recursive calls }

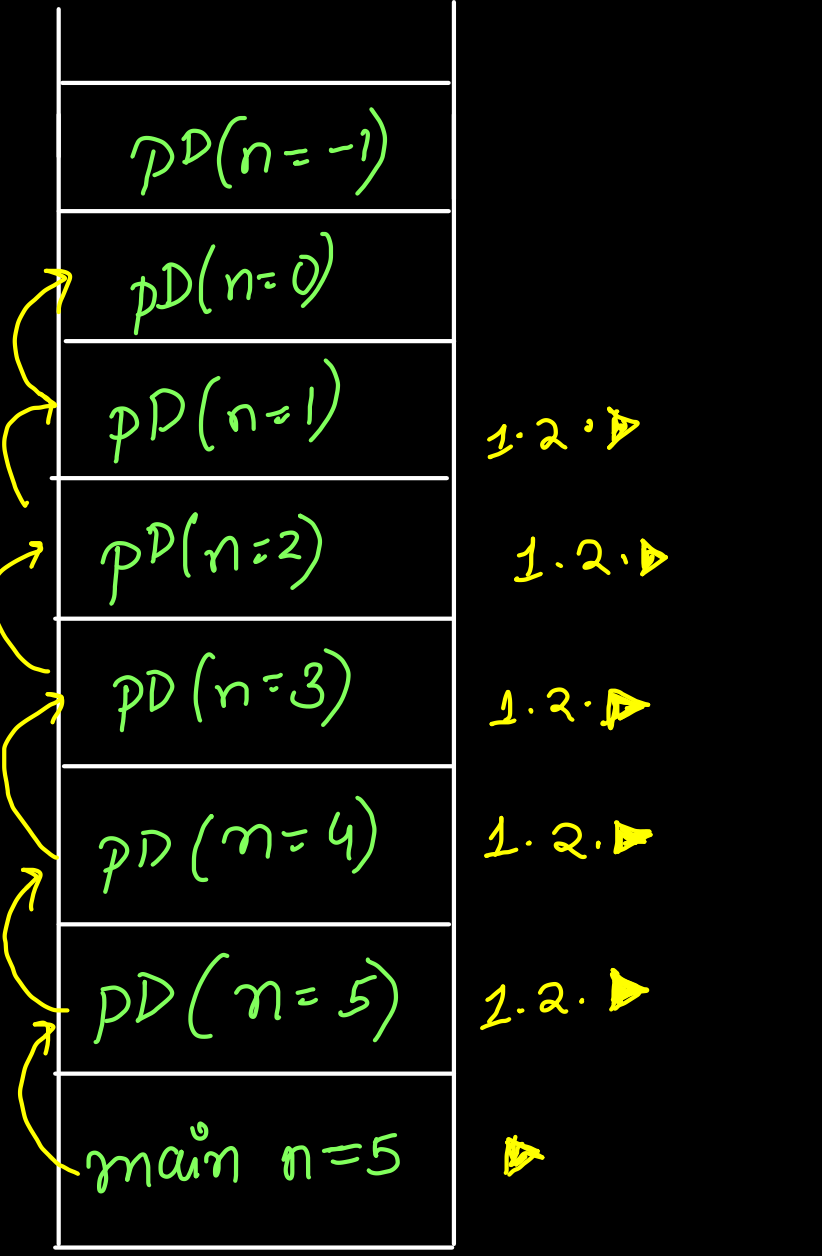
```
void printNos(int N) {
  1. System.out.println(N);
  2. printNos(N - 1);
}
```

n = 5

(runtime error)
Infinite Recursion

Java
Stack
Overflow
Except^n

C++
Segmentation
fault



funct^n callstack
or recursive call stack

output

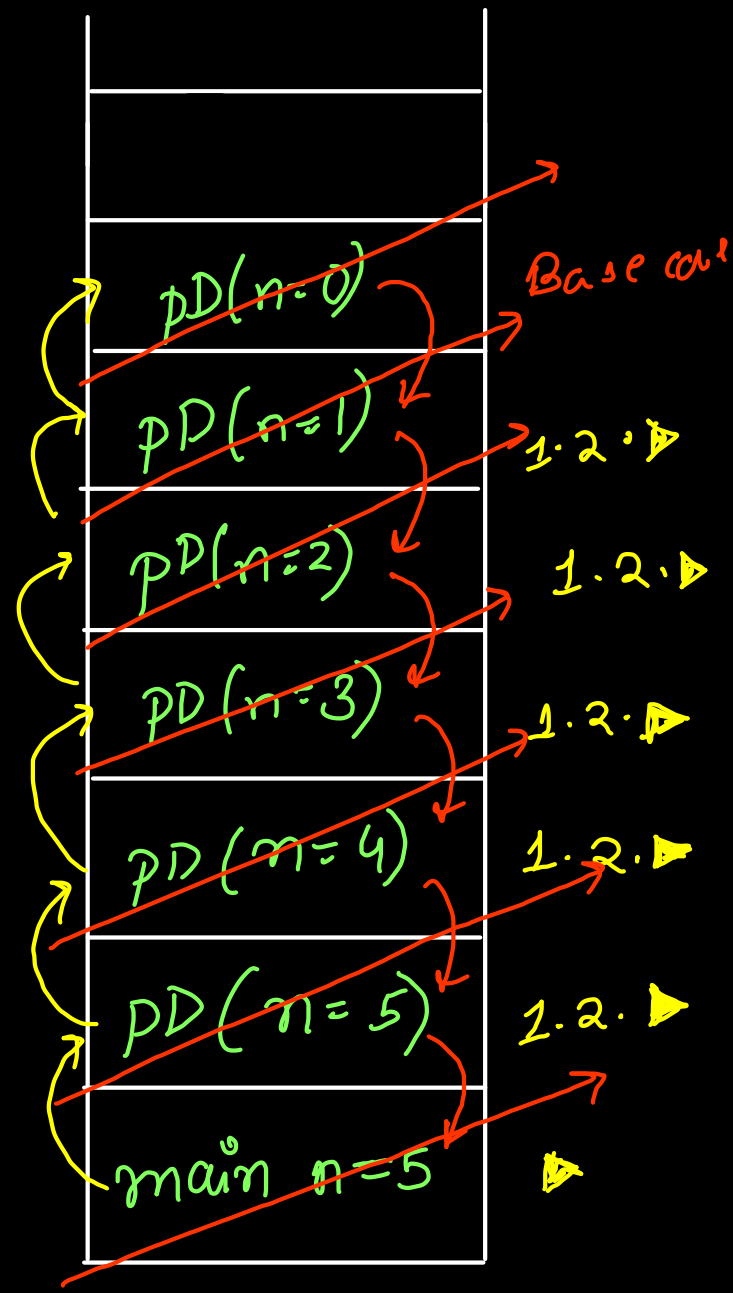
5
4
3
2
1
0
-1
-2
-3
-4

```
void printNos(int N) {
  0. if(N == 0) return;
  1. System.out.print(N + " ");
  2. printNos(N - 1);
}
```

Base case

work → fn push

{ preorder work }



Output

5
4
3
2
1
0

Time Complexity ⇒ Recurrence Relation

Expectation → $T(n) = 1 + T(n-1)$ ← Faith

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

$T(n-2) = 1 + T(n-3)$

$T(n-3) = 1 + T(n-4)$

⋮

$T(1) = 1 + T(0)$

$T(0)$ → base case

$\Rightarrow T(n) = n + T(0)$

$T(n) = n + 1$

$O(n)$

avg, best, worst

Space Complexity $\Rightarrow O(n)$

recursion call stack space

Print Increasing from 1 to N

Approach 1) Two variables

```java
class Solution
{
    public void printNos(int i, int N){
        if(i > N) return;
        System.out.print(i + " ");
        printNos(i + 1, N);
    }

    public void printNos(int N)
    {
        printNos(1, N);
    }
}
```

Time → $O(n)$

Space → $O(n)$

① Expectation

void printIncreasing(int N)

N = 5 → 1, 2, 3, 4, 5

② Faith (recursive call)

printIncreasing(n-1);

N = 4 → 1, 2, 3, 4

③ Small work

System.out.println(n)

(after call)

↳ postorder

④ Base Case

if (n == 0) return;

```java
public void printNos(int N) {
    if(N == 0) return;
    printNos(N - 1);
    System.out.print(N + " ");
}
```

base case

faith

postorder work

Output

1 2 3 4 5

work => Postorder work

or

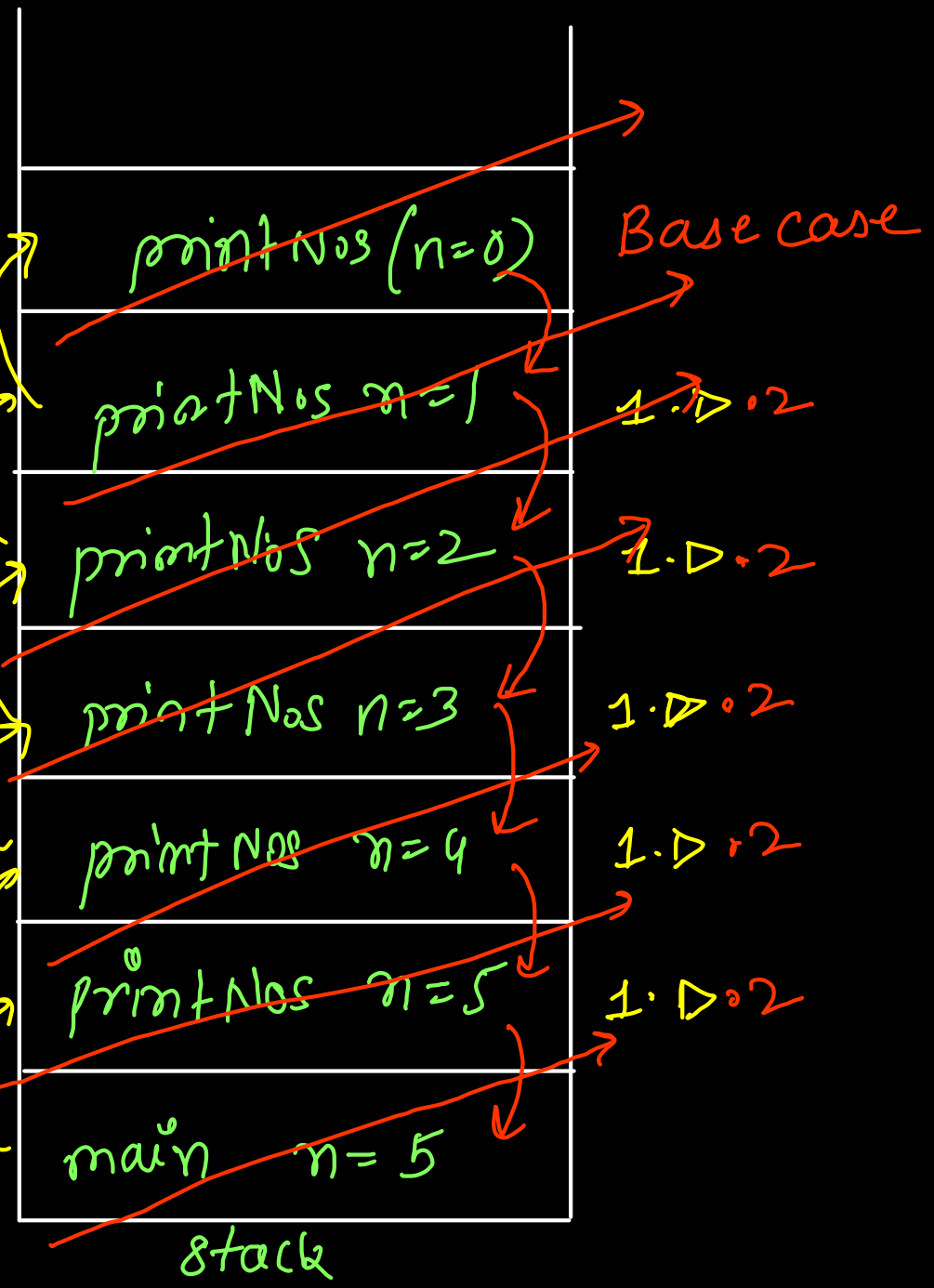
Backtracking

or

function pop

or

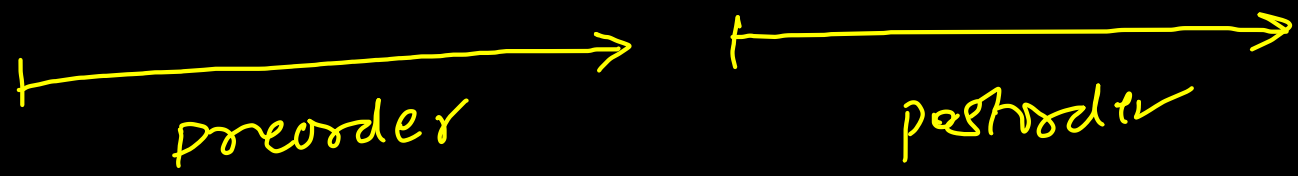
while returning



Homework : Print Decreasing Increasing

void printNos(int N)

N=5 ⇒ 5 4 3 2 1 1 2 3 4 5




```java
public void printNos(int N) {
    if(N == 0) return;
    System.out.print(N + " ");
    printNos(N - 1);
    System.out.print(N + " ");
}
```

Factorial Problem

$n! = 1 \times 2 \times 3 \times 4 \times \cdots (n-1) \times n$

$n! = (n-1)! \times n$

$0! = 1$

$1! = 1$

$2! = 1 \times 2 = 2$

$3! = 1 \times 2 \times 3 = 6$

$4! = 1 \times 2 \times 3 \times 4 = 24$

$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

$24 \times 5 = 120$

① Expectation — n!

long factorial(int N)

② faith — (n-1)!

long sans = factorial(n-1);

③ work — postorder

return (sans * n);

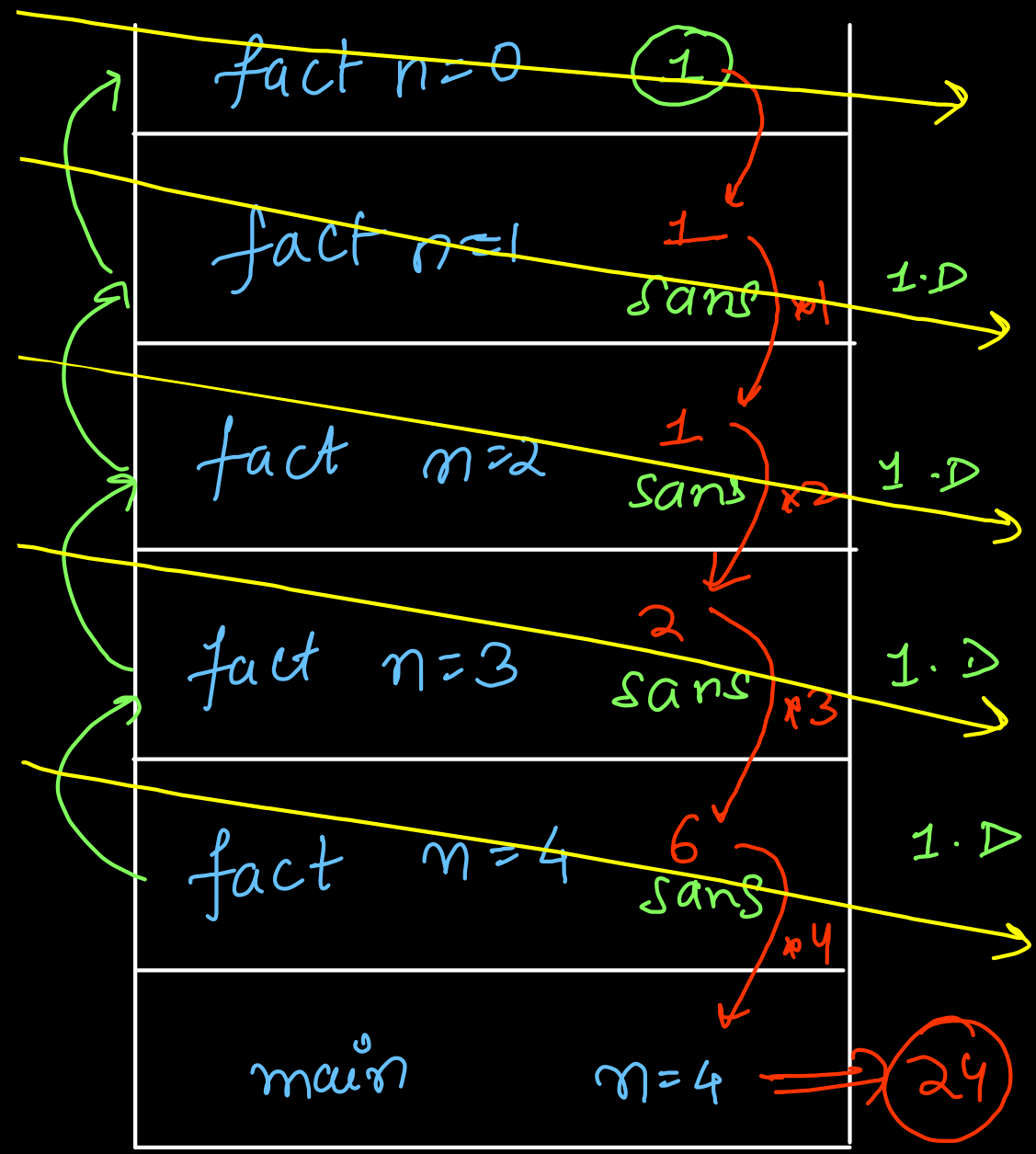
④ Base case — if(n==0) return 1;

```
static long factorial(int N){
  0. if(N == 0) return 1;
  1. long sans = factorial(N - 1);
  2. return (N * sans);
}
```

$T(n) = 1 + T(n-1)$

$\Rightarrow O(n)$ time

Space $\Rightarrow O(n)$



Leetcode 50 Pow(x, n)

Exponentiation

① Expectation

double power(double x, int n)

$x = 2, \quad n = 6 \qquad 2^6 = 2*2*2*2*2*2 = 64$

$2^6 = \underbrace{2*2*2*2*2}*2 = 2^5 * 2$

$x^n = x^{n-1} * x$

② faith

sans = power(x, n-1);

$2^5 = 32$

③ work

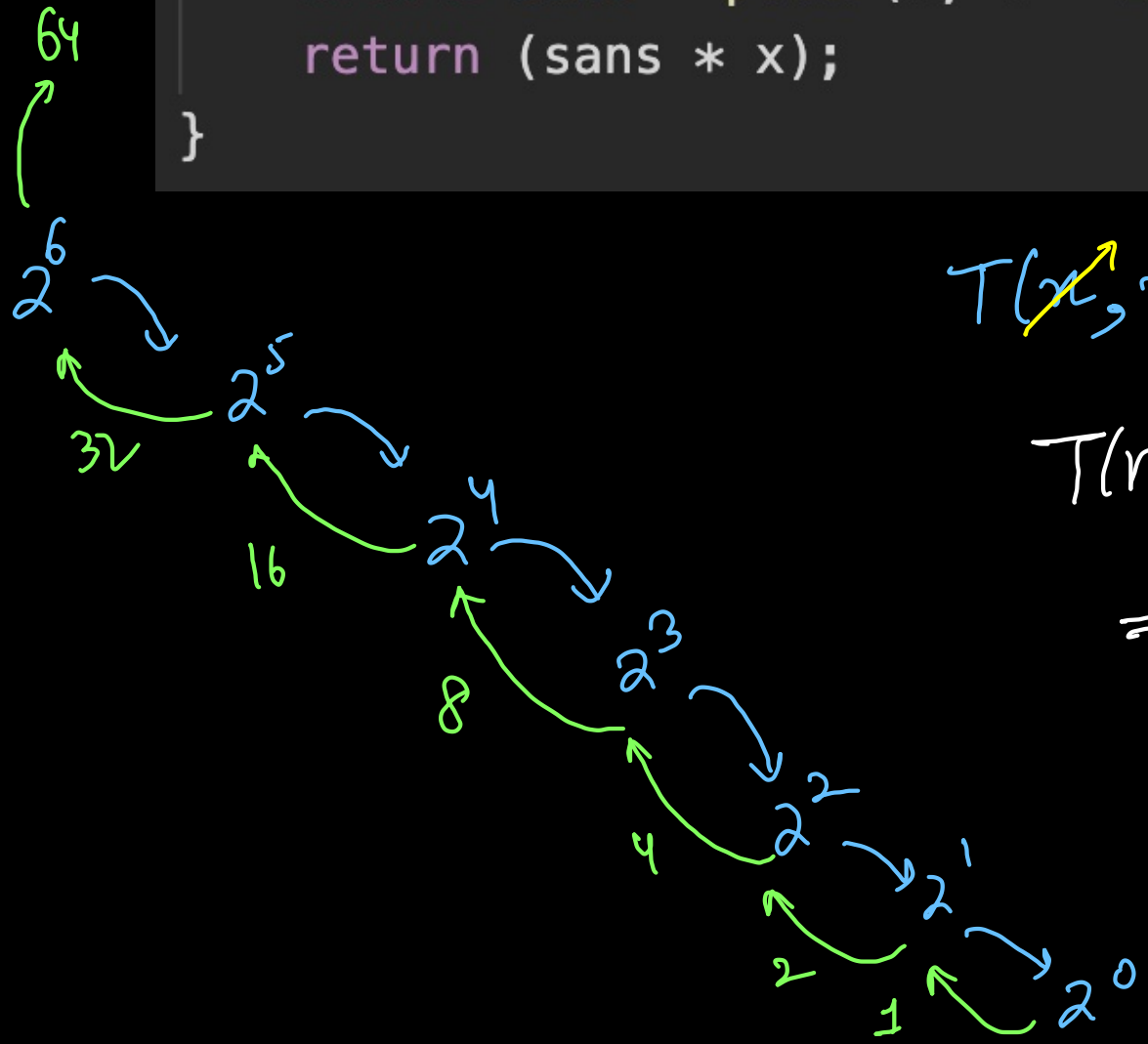
return (sans * x);

④ Base case $x^0 = 1$

if (n == 0) return 1.0;

```java
public double power(double x, int n){
    if(n == 0) return 1.0;
    double sans = power(x, n - 1);
    return (sans * x);
}
```

This soln is correct for $n = +ve$



time complexity

$T(x, n) = T(x, n-1) + 1$

$T(n) = T(n-1) + 1$

$\Rightarrow$ Time Complexity $\Rightarrow O(n)$ linear

$\Rightarrow$ Space Complexity $\Rightarrow O(n)$

Constraint $-2^{31} \leq n \leq 2^{31}-1$

$\approx -\infty \leq n \leq +\infty$

$\approx -10^9 \leq n \leq 10^9$

$O(10^9)$ in worst case $\longrightarrow$ ~~Time limit Exceeded~~

Run time Error
(Stack Overflow)

Corner case : If n is negative

$2^{-6} = (1/2)^6 = (0.5)^6 =$

$$\frac{1}{2 \times 2 \times 2 \times 2 \times 2 \times 2} = 0.5 \times 0.5 \times 0.5 \times 0.5 \times 0.5 \times 0.5$$

if (n < 0) return power (1/x, -n);

```
class Solution {
    public double power(double x, long n){
        if(n == 0) return 1.0;
        double sans = power(x, n - 1);
        return (sans * x);
    }

    public double myPow(double x, int n) {
        if(n < 0) return power(1 / x, -n);
        return power(x, n);
    }
}
```

Stack overflow (TLE)

or

$2^8 = 2*2*2*2*2*2*2*2*2$

$2^8 = (2*2*2*2*2*2*2*2)*2$

$= 2^7 * 2$

$x^n = x^{n-1} * x$

$2^8 = (2*2*2*2)*(2*2*2*2)$

$2^8 = 2^4 * 2^4$

① Expectam

double power(double x, int n)

$x = 2, \quad n = 6 \qquad 2^6 = 2*2*2*2*2*2 = 64$

② faith

if(n%2 == 0)
sans = power(x, n/2);

③ work

ans = sans * sans;
if(n%2 == 1) ans *= x;
return ans;

$2^6 = 2*2*2*2*2*2$

$= (2*2*2)*(2*2*2)$

$2^6 = 2^3 * 2^3$

$2^7 = 2*2*2*2*2*2*2$

$= (2*2*2)*(2*2*2)*2$

$= 2^3 * 2^3 * 2$

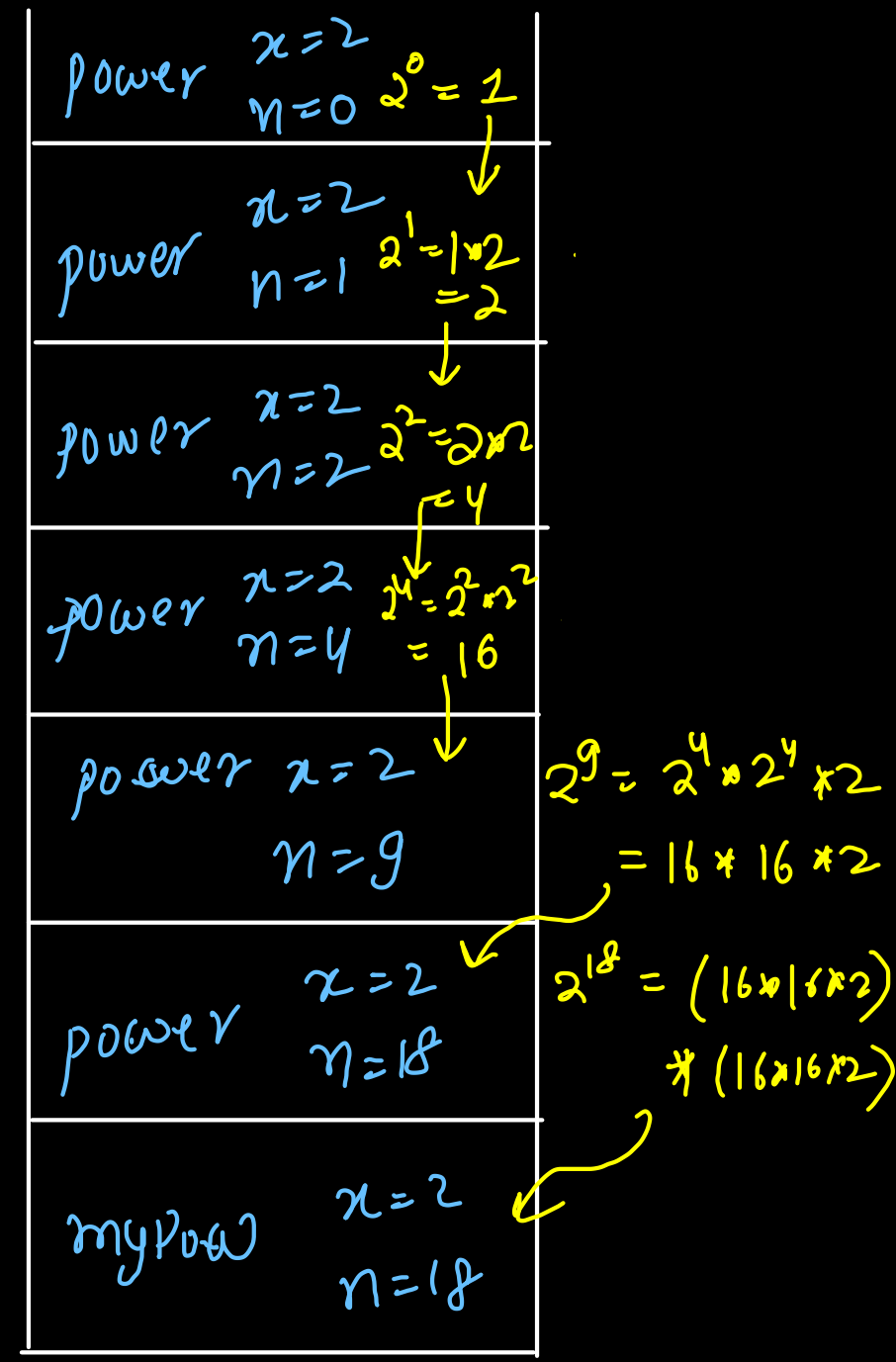
sans * sans * 2

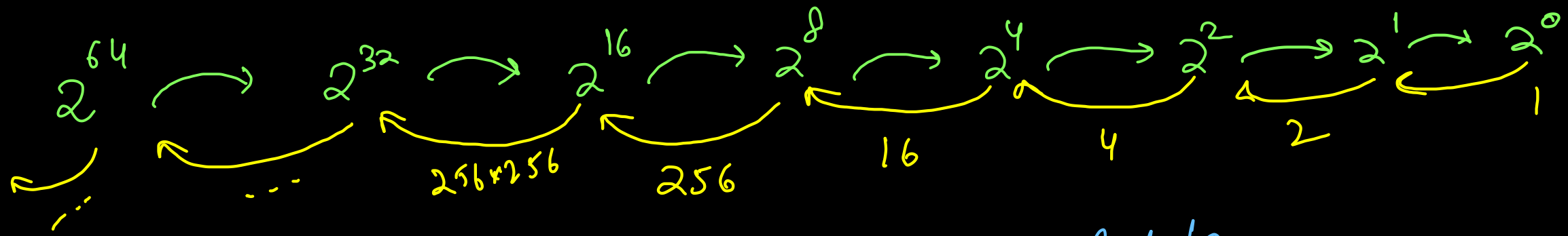
```java
public double power(double x, long n){
    if(n == 0) return 1.0;
    double smallAns = power(x, n / 2);
    double bigAns = smallAns * smallAns;
    if(n % 2 == 1) bigAns = bigAns * x;
    return bigAns;
}

public double myPow(double x, int n) {
    if(n < 0) return power(1 / x, -1l * n);
    return power(x, n);
}
```

$x = 2, \quad n = 18$

$n =$ Integer.MIN_VALUE

$-n = -(-2^{31}) = +2^{31}$ → overflow





Time $\Rightarrow O(\log_2 n)$

$T(n) = \log_2 n + T(0)$

$O(\log_2 n)$

Recursive Relation

$T(n) = T(n/2) + 1$

$T(n) = T(n/2) + 1$

$T(n/2) = T(n/4) + 1$

$T(n/4) = T(n/8) + 1$

$\vdots$

$T(1) = T(0) + 1$

$\log_2 n$ eqns

$1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8} \cdots$

LC 509) Fibonacci Number

multiple recursive calls

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

fibonacci series

0	1	1	2	3	5	8	13	21
F(6)	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	F(7)	F(8)

(1) expectation

int fibonacci (int n)

(2) faith

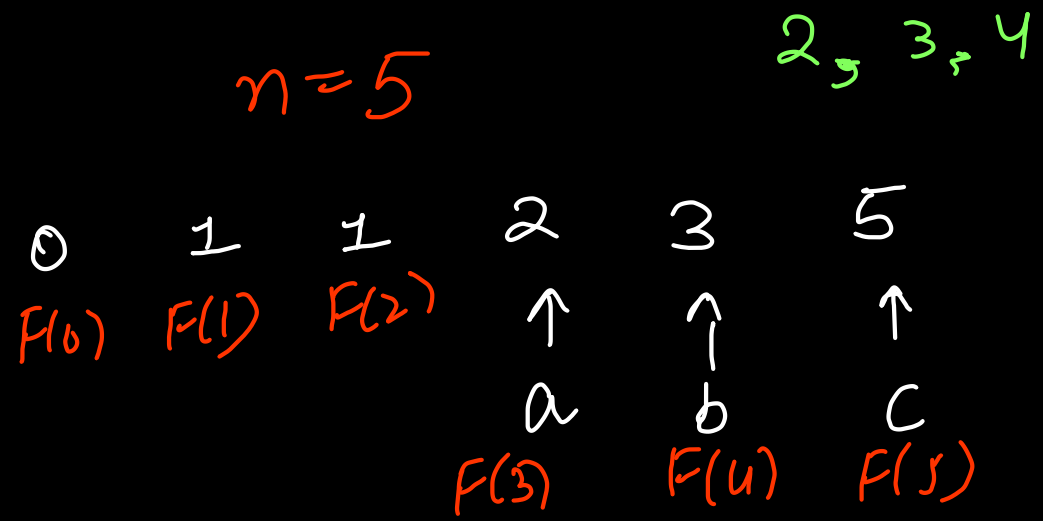
a = fib(n-1)

b = fib(n-2)

(3) work

return (a+b);

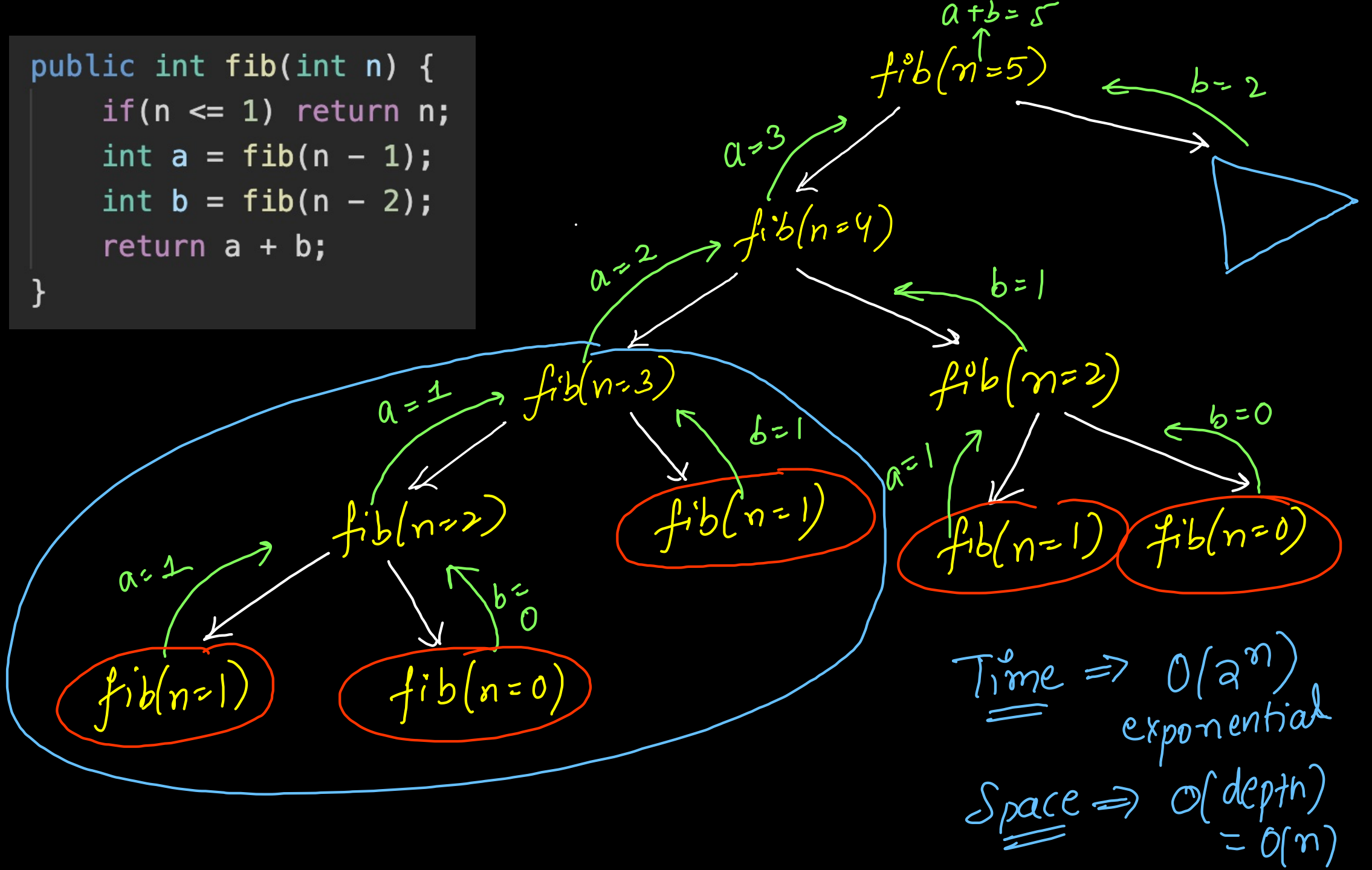
```java
public int fib(int n) {
    if(n <= 1) return n;
    int a = 0, b = 1, c = 1;
    for(int i = 2; i < n; i++){
        a = b;
        b = c;
        c = a + b;
    }
    return c;
}
```



→ Dynamic Programming
→ Tabulatn
→ w/o extra space

Time ⇒ O(n), Space ⇒ O(1)

```java
public int fib(int n) {
    if(n <= 1) return n;
    int a = fib(n - 1);
    int b = fib(n - 2);
    return a + b;
}
```



Time ⇒ $O(2^n)$
exponential

Space ⇒ $O(depth)$
$= O(n)$

Check Array is Sorted

40	30	30	60	90	true
0	1	2	3	4	

40	30	30	20	90	false
0	1	2	3	4	

```java
boolean isSorted(int[] arr, int idx){
    if(idx >= arr.length - 1) return true;
    boolean sans = isSorted(arr, idx + 1);
    return sans && (arr[idx] <= arr[idx + 1]);
}

boolean arraySortedOrNot(int[] arr, int n) {
    return isSorted(arr, 0);
}
```

① Expectation

boolean isSorted(int[] nums, int idx)

② Faith

sans = isSorted(nums, idx+1)

③ Work

return sans && (arr[idx] <= arr[idx+1])

④ Base Case

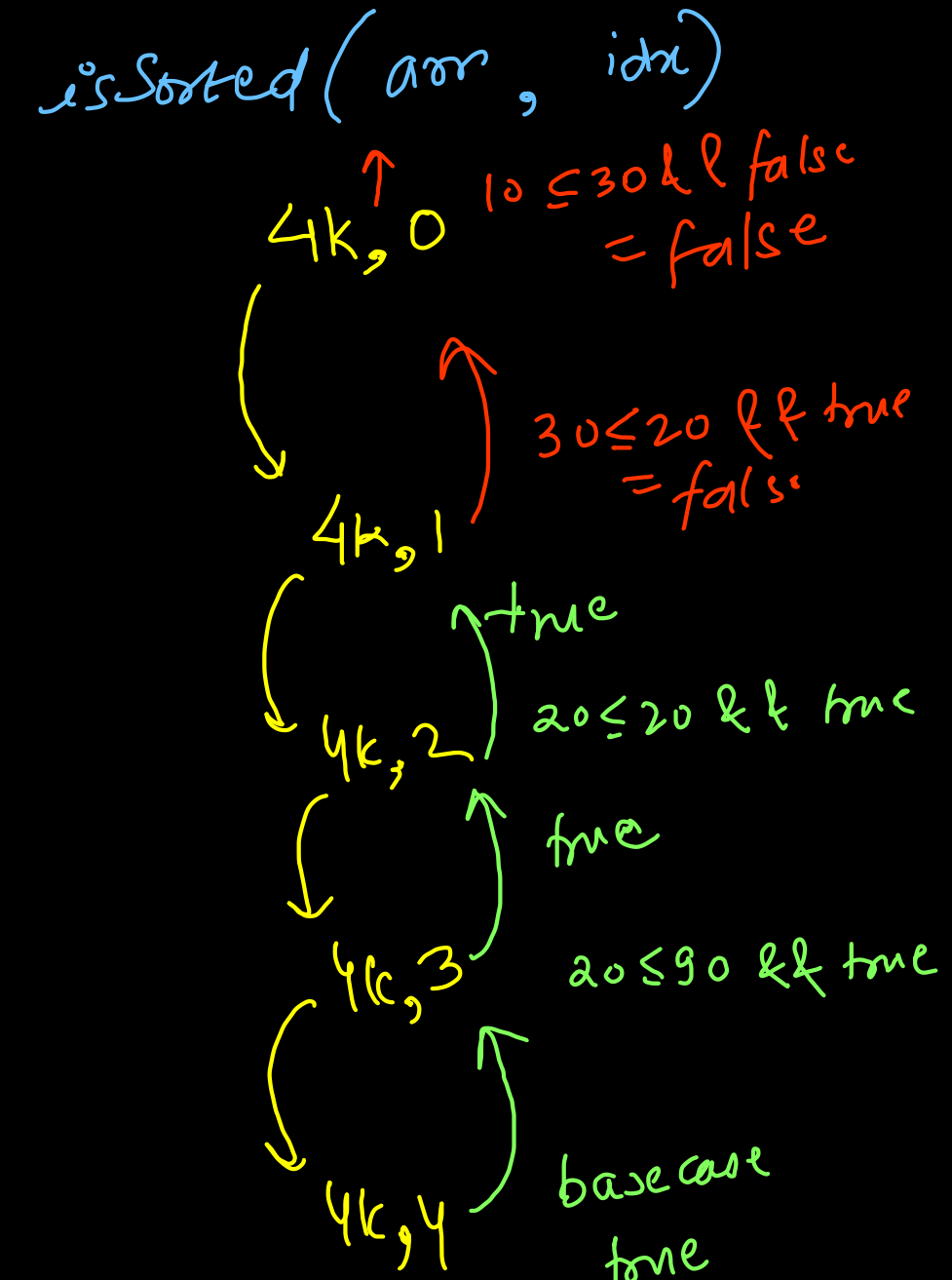
if(idx == n-1 || n) return true;

```java
boolean isSorted(int[] arr, int idx){
    if(idx >= arr.length - 1) return true;
    boolean sans = isSorted(arr, idx + 1);
    return sans && (arr[idx] <= arr[idx + 1]);
}

boolean arraySortedOrNot(int[] arr, int n) {
    return isSorted(arr, 0);
}
```

4k	40	30	20	20	90
	0	1	2	3	4

arr.length = 5

Time ⇒ O(n), Space ⇒ O(n)



Largest / Smallest Element Recursively

30 -20 40 -90 10

Homework

Time ⇒ O(n), Space ⇒ O(n)

First occurence & last occurence

Recursive Linear search on Unsorted Array

0	1	2	3	4	5
30	20	50	20	20	10

target = 20

fi = 1

li = 4

target = 50

fi = 2

li = 2

target = 40

fi = -1

li = -1

0	1	2	3	4	5
30	20	50	20	20	10

First Index

① expectation

```
int firstIndex(int[] arr, int idx, int target)
```

② faith

```
ans = firstIndex(arr, idx+1, target)
```

③ work

```
if (arr[idx] != target) return ans;
else return idx;
```

④ Base case

```
if (idx == n) return -1;
```

Last Index

① expectation

```
int lastIndex(arr, idx, target)
```

(idx: n-1)

② faith

```
ans = lastIndex(arr, idx-1, target)
```

③ work

```
if (arr[idx] != target) return ans;
else return idx;
```

④ base case

```
if (idx == -1) return -1;
```

```java
static int firstIndex(int[] arr, int idx, int key){
    if(idx == arr.length) return -1; // empty array
    if(arr[idx] == key) return idx; // leftmost
    return firstIndex(arr, idx + 1, key);
}

static int lastIndex(int[] arr, int idx, int key){
    if(idx == -1) return -1; // empty array
    if(arr[idx] == key) return idx; // rightmost
    return lastIndex(arr, idx - 1, key);
}

static int[] findIndex(int a[], int N, int key)
{
    int fi = firstIndex(a, 0, key);
    int li = lastIndex(a, N - 1, key);
    return new int[]{fi, li};
}
```

Time ⇒
O(n) worst case (unsuccessful search)

O(1) best case

Space
⇒ O(n)

→ left to right traversal

→ right to left traversal

0	1	2	3	4	5
30	20	50	20	20	10

target = 20

{ 1, 4 }
fi li
0 1

① expectation

left to right (idx = 0)

```
int[] findIndex(int[] arr, int idx, int key)
```

② faith

```
int[] ans = findIndex(arr, idx+1, key)
```

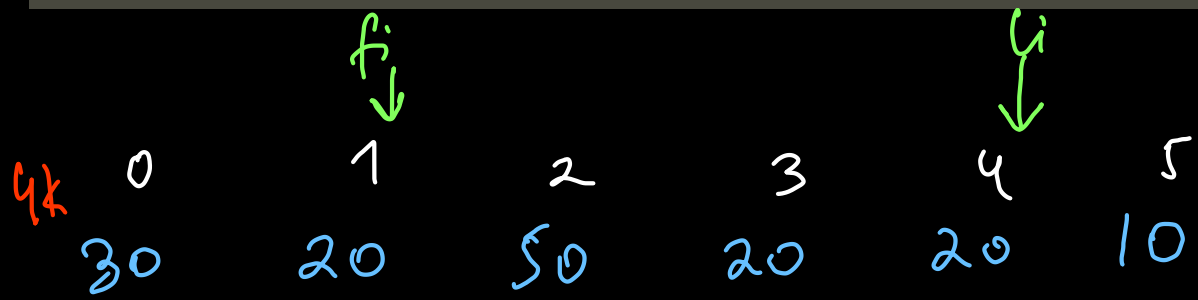
④ Base case

```
if(idx == n)
    return {-1,-1}
```

③ work

```
if (arr[idx] == key){
    ans[0] = idx;          // leftmost == current index
    ans[1] = max(ans[1], idx);   // rightmost == last index
}
```

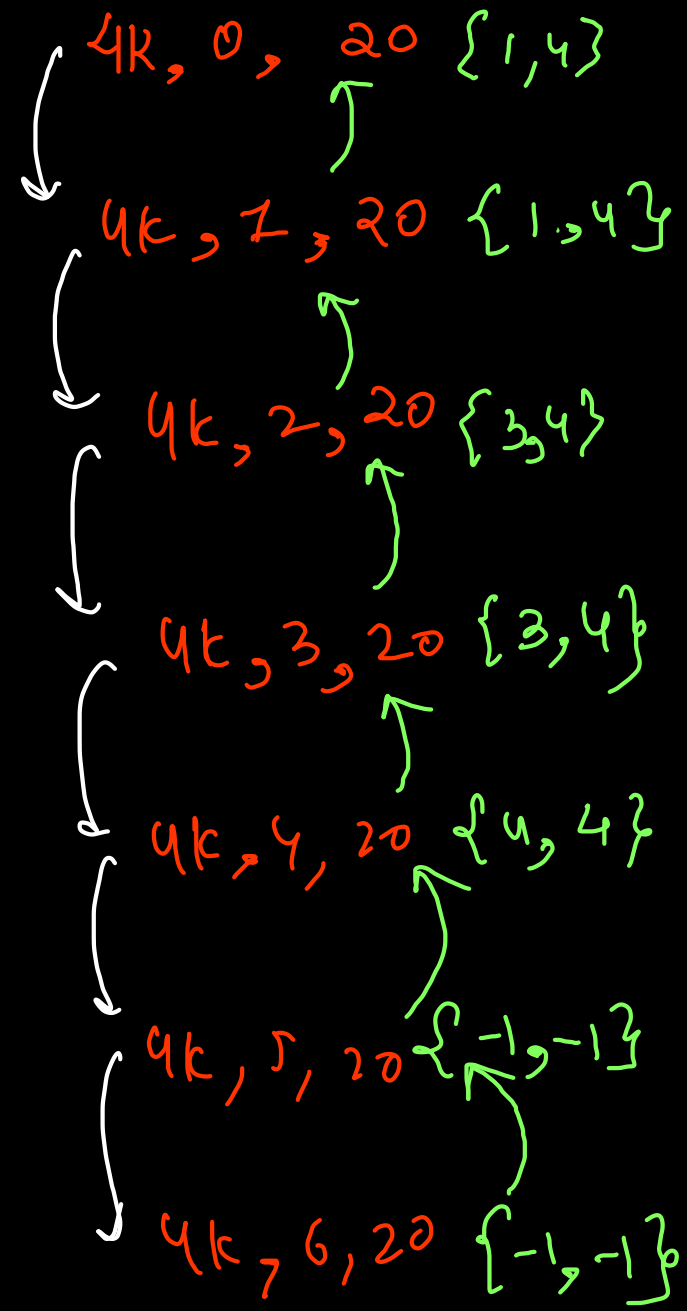
```java
static int[] search(int[] arr, int idx, int key){
    if(idx == arr.length) return new int[]{-1, -1};
    int[] ans = search(arr, idx + 1, key);
    if(arr[idx] == key){
        ans[0] = idx; // leftmost = first index
        ans[1] = Math.max(ans[1], idx);  // rightmost = last index
    }
    return ans;
}

static int[] findIndex(int a[], int N, int key) {
    return search(a, 0, key);
}
```



target = 20

Single pass Algo

Time ⇒ O(n), Space ⇒ O(n)



344. Reverse String

Hint

Easy 6.8K 1K

Companies

Write a function that reverses a string. The input string is given as an array of characters `s`.

You must do this by modifying the input array in-place with `O(1)` extra memory.

1) expectation

```
void reverse(char[] str,
             int left, int right)
```

2) faith

```
reverse(str, left+1, right-1)
```

3) work

```
Swap(str, left, right)
```

4) base case

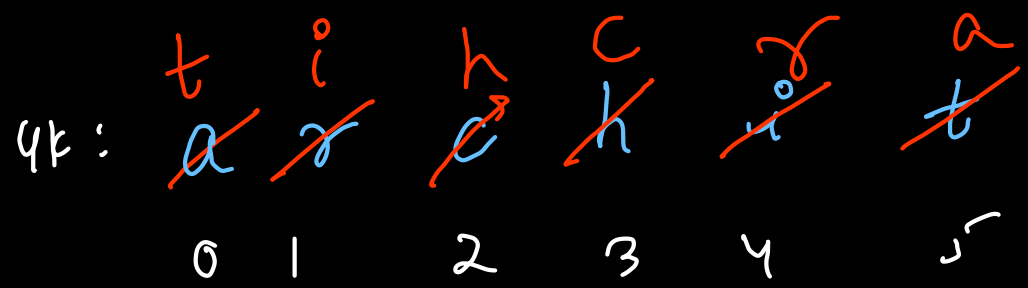
```
if (l >= r) return;
```

{'h', 'e', 'l', 'l', 'o'}

left = 0

right = n-1

```java
public void reverse(char[] s, int left, int right){
    if(left >= right) return;
    reverse(s, left + 1, right - 1);
    // swap left and right
    char temp = s[left];
    s[left] = s[right];
    s[right] = temp;
}
public void reverseString(char[] s) {
    reverse(s, 0, s.length - 1);
}
```



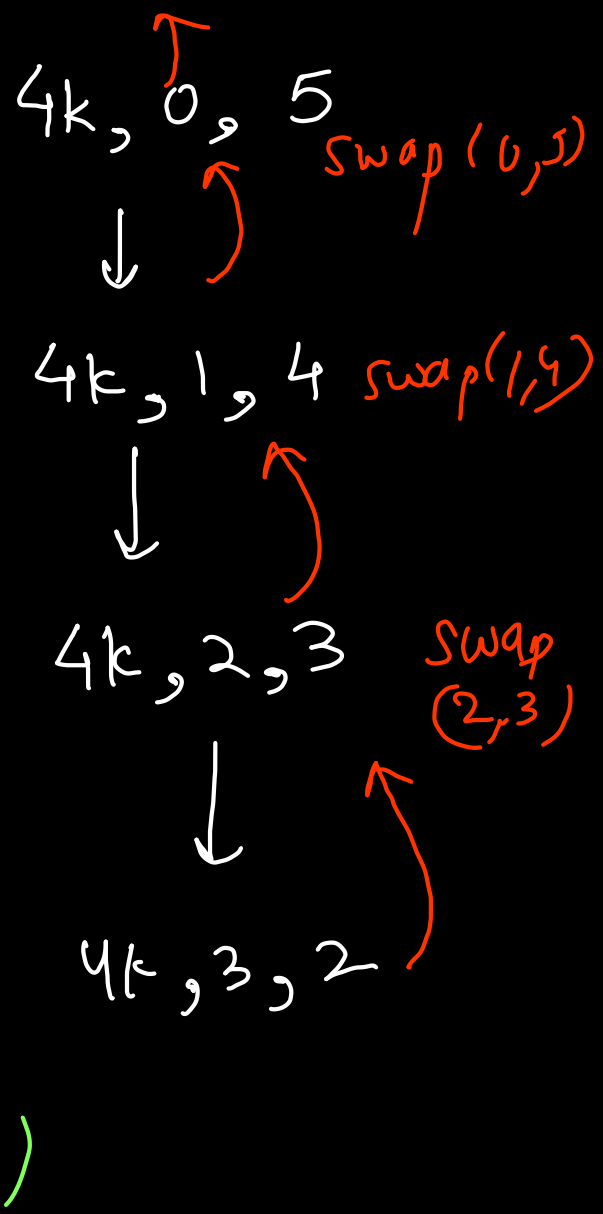
Time ↓

O(n)

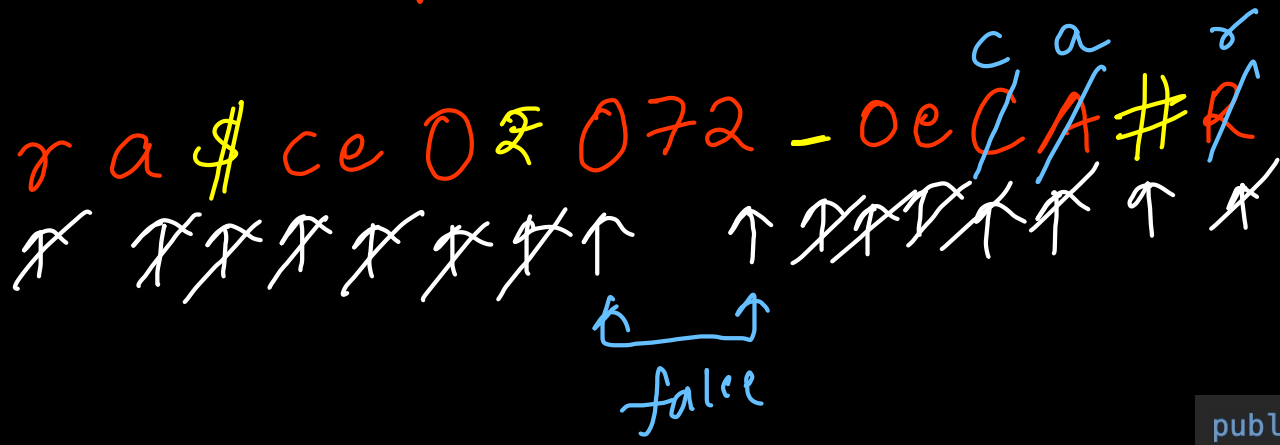
Space ↓

O(n) recursive

In-place (no auxiliary data structure)



Valid Palindrome



Time
↳ O(n)

Space
↳ O(n) Recursion

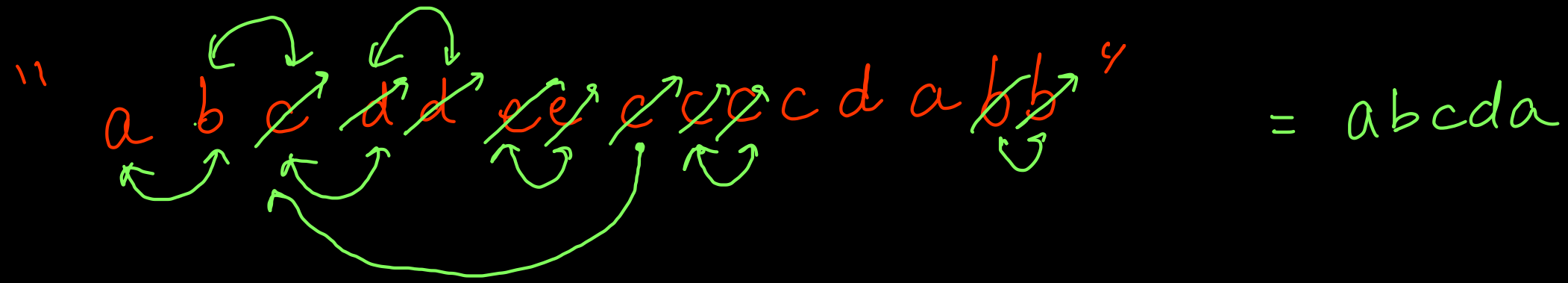
⇒ ignore non-alphanumeric

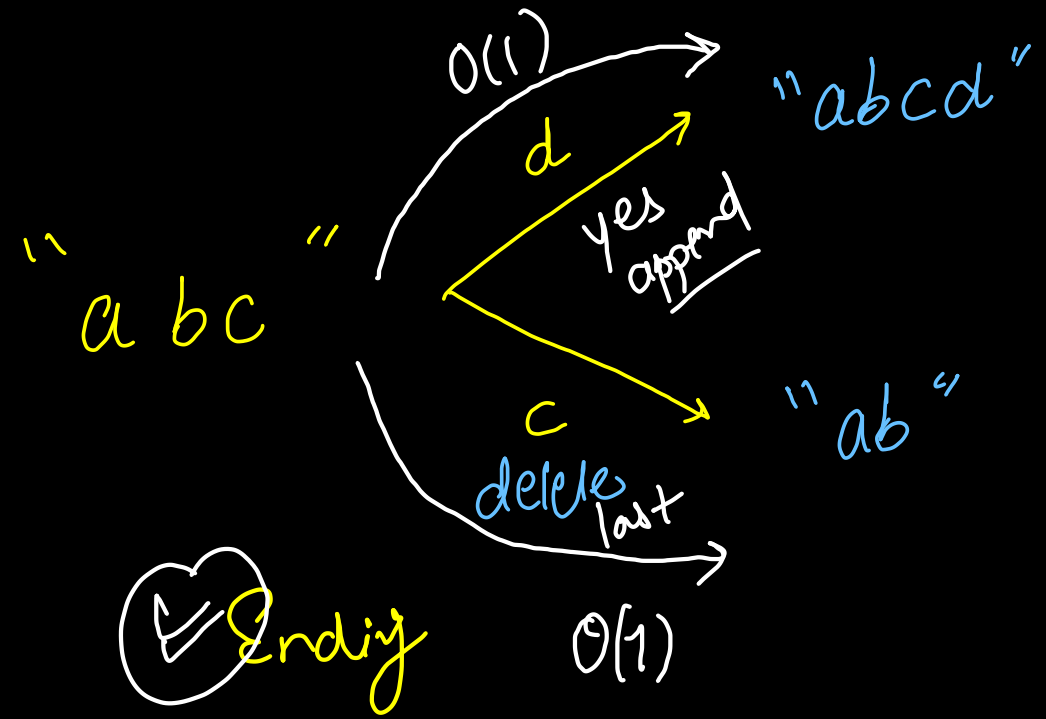
⇒ Convert uppercase to lowercase

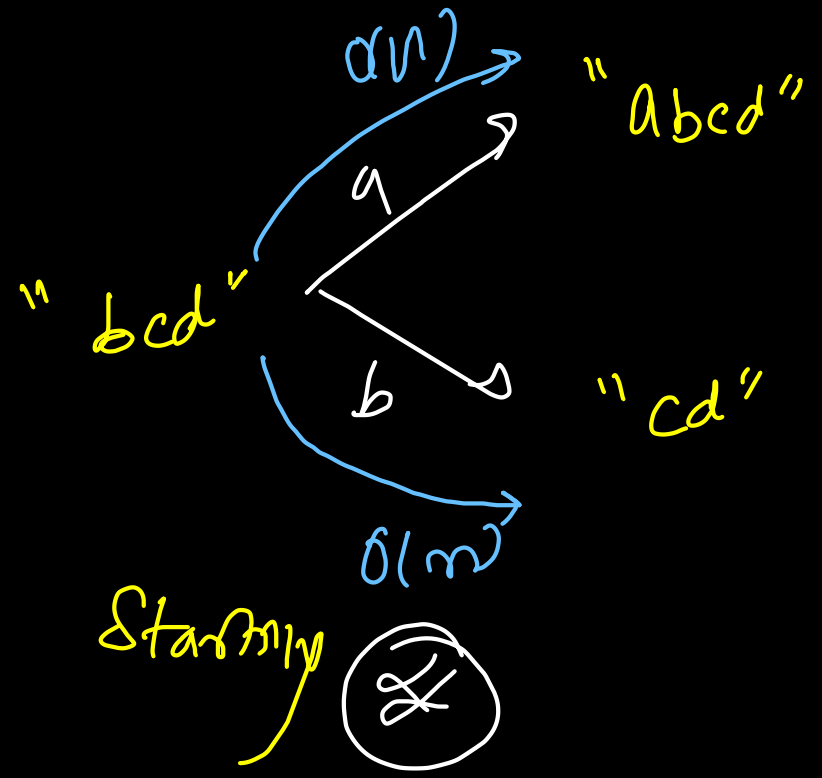
⇒ recursively check

```java
public boolean isPalindrome(String s, int left, int right){
    if(left >= right) return true;
    char l = s.charAt(left);
    char r = s.charAt(right);

    if(Character.isLetterOrDigit(l) == false)
        return isPalindrome(s, left + 1, right);

    if(Character.isLetterOrDigit(r) == false)
        return isPalindrome(s, left, right - 1);

    if(l != r) return false;
    return isPalindrome(s, left + 1, right - 1);
}

public boolean isPalindrome(String s) {
    s = s.toLowerCase();
    return isPalindrome(s, 0, s.length() - 1);
}
```

Remove All Adjacent Duplicates

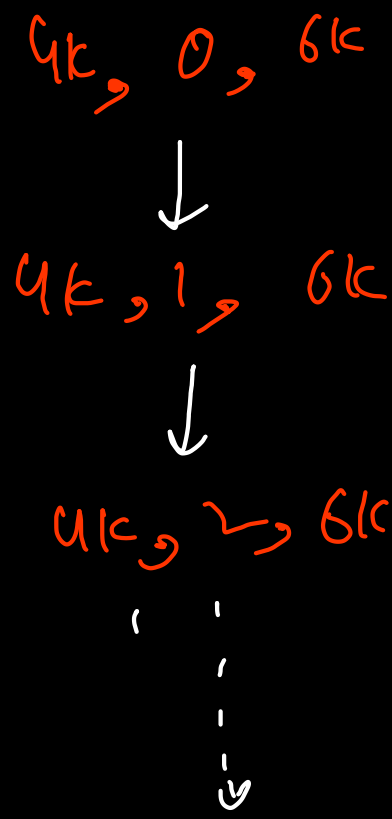




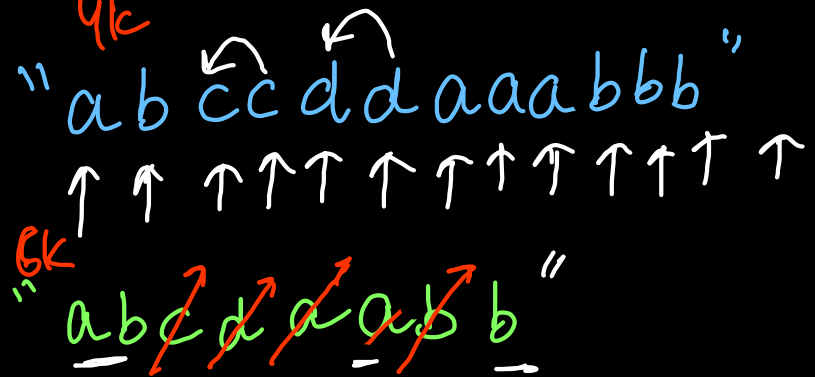


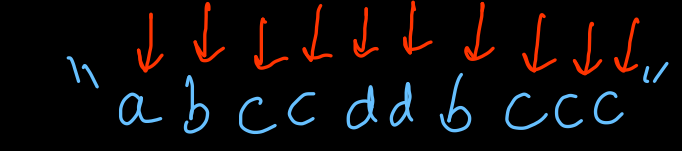
```java
public void remove(String input, int idx, StringBuilder output){
    if(idx == input.length()) return;
    char ch = input.charAt(idx);

    if(output.length() > 0 && output.charAt(output.length() - 1) == ch){
        // adjacent duplicates -> deleted
        output.deleteCharAt(output.length() - 1);
    } else output.append(ch);

    remove(input, idx + 1, output);
}

public String removeDuplicates(String s) {
    StringBuilder output = new StringBuilder();
    remove(s, 0, output);
    return output.toString();
}
```

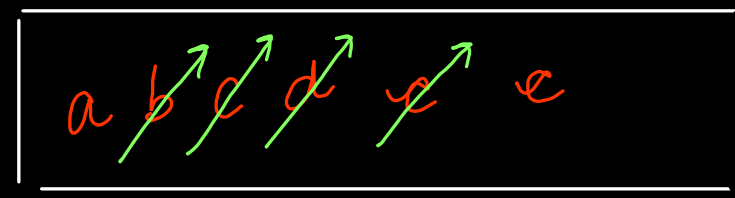
precedes



Time
↳ O(n)
Space → O(n)







Power Set

"abcd" → Subsets or Subsequence → part of a string which is not necessary to be contiguous, but order of elements should be same

"" "a" "b" "c" "d" "ab" "ac" "ad" "bc" "bd" "cd" "abc" "abd" "acd" "bcd" "abcd"

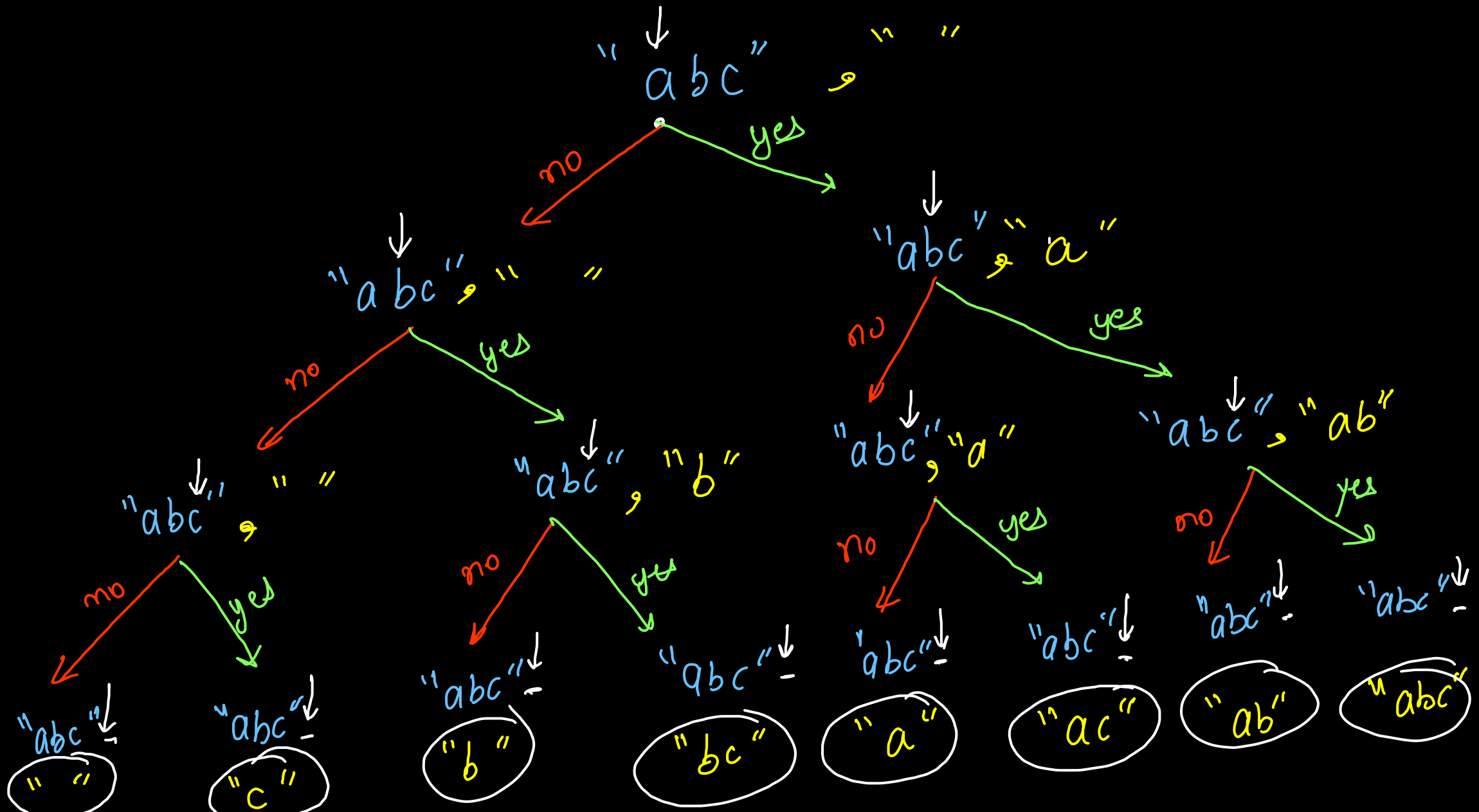
Powerset = Set of all subsets.

$= 2 \times 2 \times 2 \times 2$

$= 2^4 = 16$

⇒ Lexicographically sorted

⇒ Empty subset ignored


"abc", ""
no
yes
"abc", ""
no
yes
"abc", "a"
no
yes
"abc", "b"
no
yes
"abc", "a"
no
yes
"abc", "ab"
no
yes
"abc"
""
"abc"
"c"
"abc"
"b"
"abc"
"bc"
"abc"
"a"
"abc"
"ac"
"abc"
"ab"
"abc"
"abc"

```java
class Solution
{
    List<String> subsets = new ArrayList<>();

    public void powerset(String input, int idx, String output){
        if(idx == input.length()){
            if(output.length() > 0) subsets.add(output);
            return;
        }

        // no choice
        powerset(input, idx + 1, output);

        // yes choice
        powerset(input, idx + 1, output + input.charAt(idx));
    }

    public List<String> AllPossibleStrings(String s)
    {
        powerset(s, 0, "");
        Collections.sort(subsets); // Lexicographically Sorted
        return subsets;
    }
}
```

ignore empty subset

sort all subsets

recurrence relation

2^0 [$T(n) = 2T(n-1) + 1$]

2^1 [$T(n-1) = 2T(n-2) + 1$]

2^2 [$T(n-2) = 2T(n-3) + 1$]

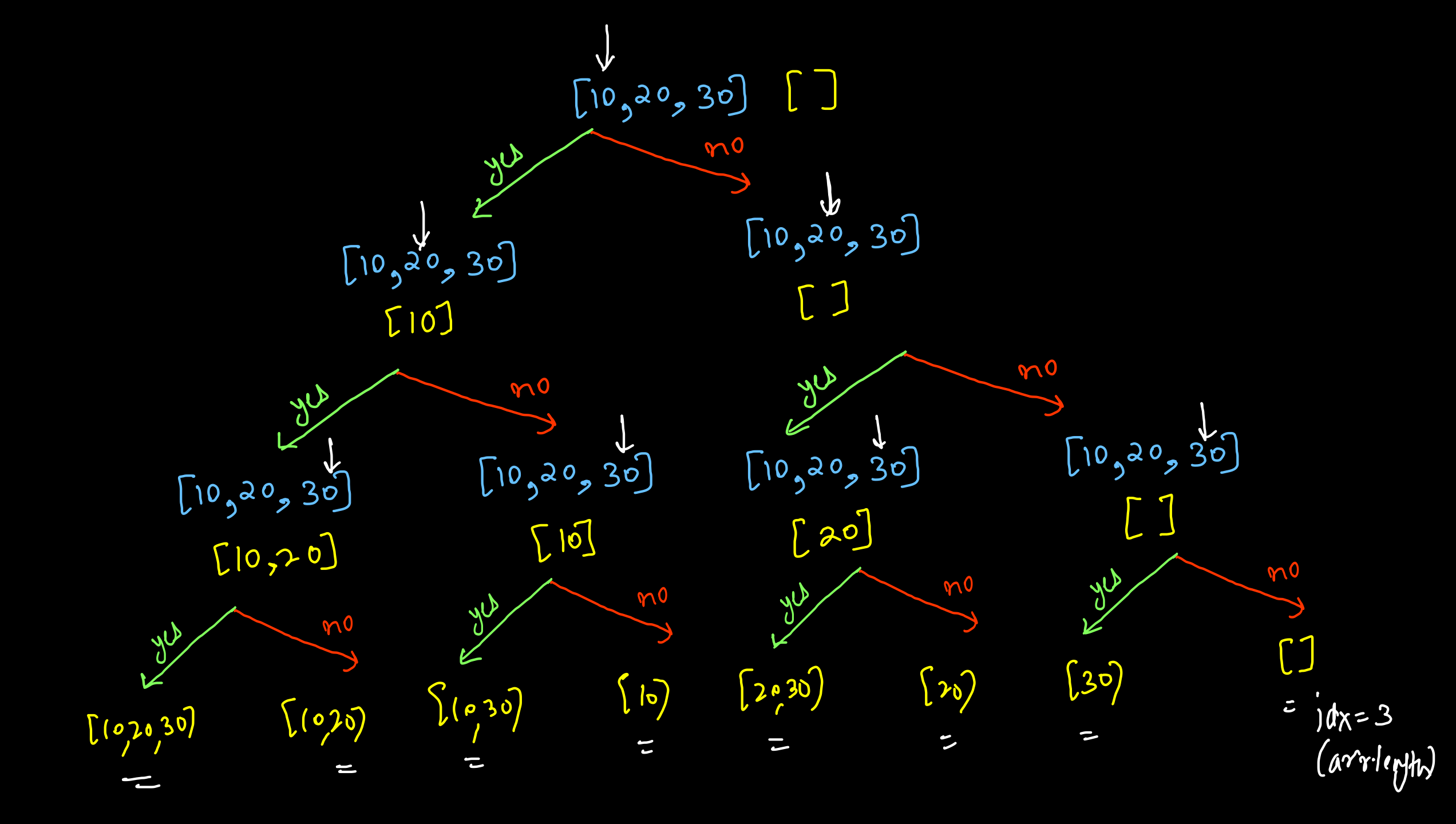
2^{n-1} [$T(1) = 2T(0) + 1$]

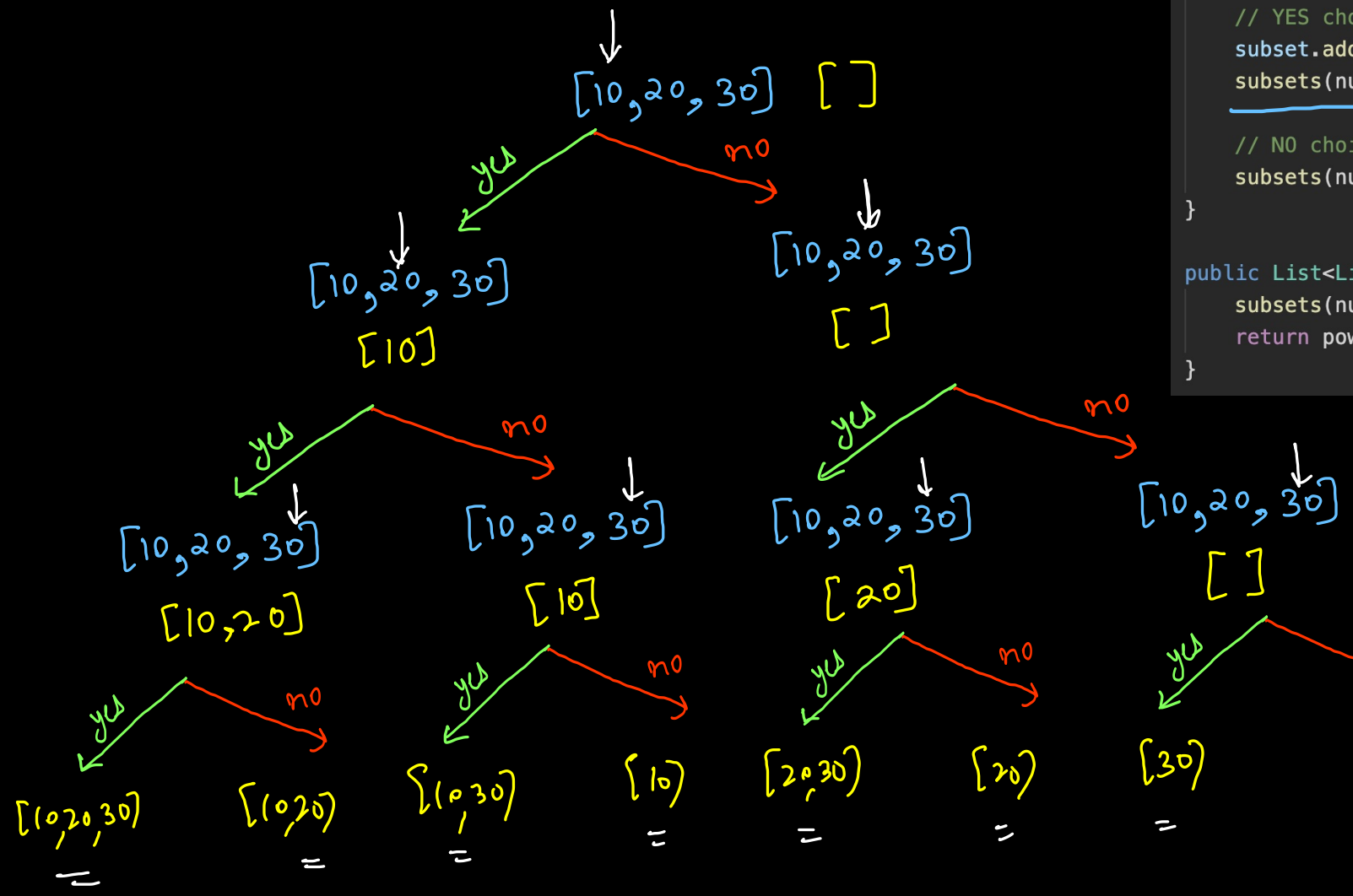
$T(n) = 2^n T(0) + 2^n$

Time ⇒ exponential

Space ⇒ $O(n)$

$O(2^n)$


[10,20,30] [ ]
yes
no
[10,20,30]
[10]
[10,20,30]
[ ]
yes
no
[10,20,30]
[10,20]
[10,20,30]
[10]
[10,20,30]
[20]
[10,20,30]
[ ]
yes
no
yes
no
yes
no
yes
no
[10,20,30]
=
[10,20]
=
[10,30]
=
[10]
=
[20,30]
=
[20]
=
[30]
=
[]
= idx=3
(arr.length)



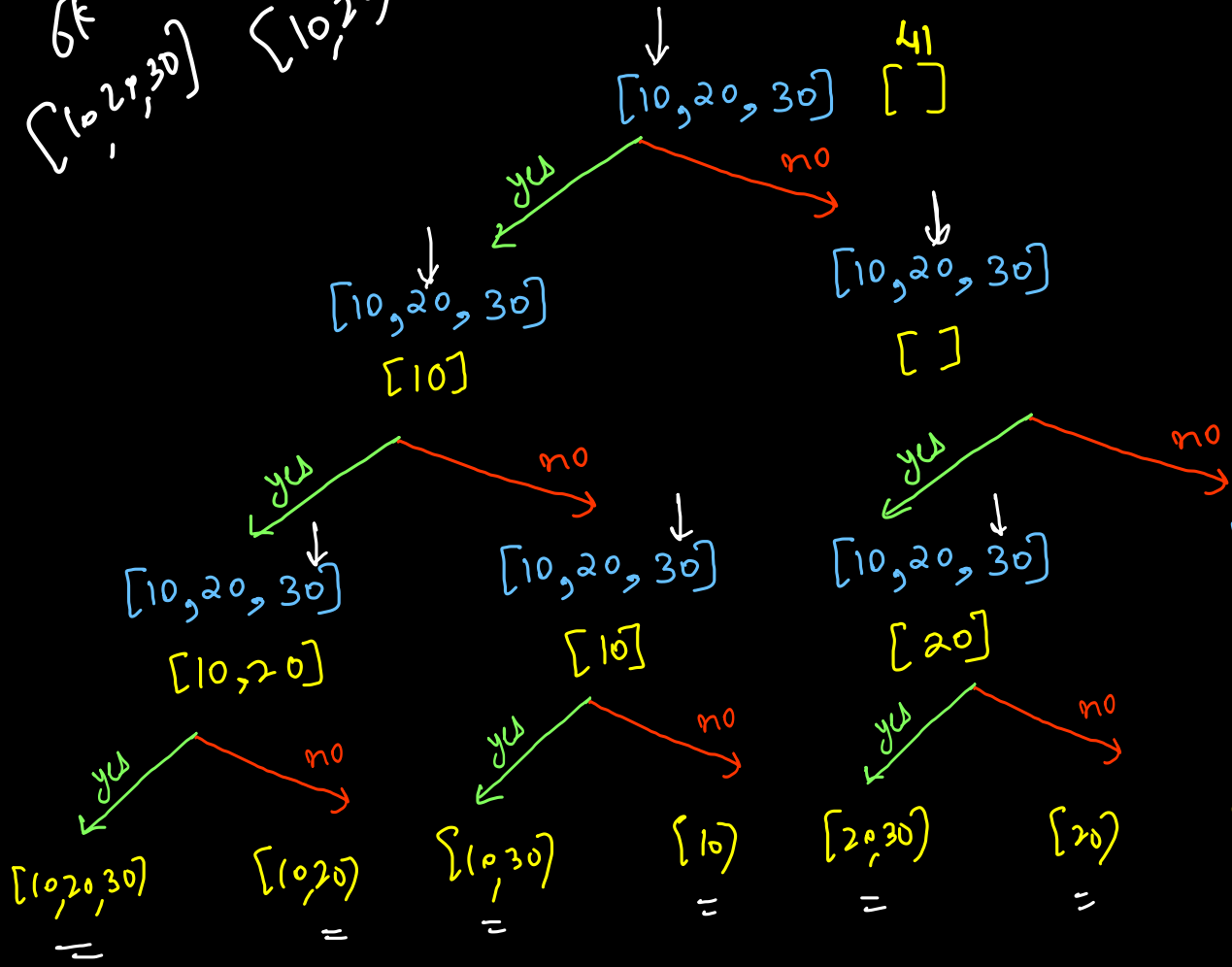

```java
List<List<Integer>> powerset = new ArrayList<>();

public void subsets(int[] nums, int idx, List<Integer> subset){
    if(idx == nums.length){
        powerset.add(subset);
        return;
    }

    // YES choice
    subset.add(nums[idx]);
    subsets(nums, idx + 1, subset);

    // NO choice
    subsets(nums, idx + 1, subset);
}

public List<List<Integer>> subsets(int[] nums) {
    subsets(nums, 0, new ArrayList<>());
    return powerset;
}
```

backtracking
subset.remove
(subset.length()-1);

2d arraylist powerset : [6k, 8k, 4k, 4k, 4k, 4k, 4k, 4k]
[] [] [] [] [] [] [] []

6k [10,20,30]

8k [10,20]



ArrayList of pointers

```java
List<List<Integer>> powerset = new ArrayList<>();

public void subsets(int[] nums, int idx, List<Integer> subset){
    if(idx == nums.length){
        powerset.add(subset);
        return;
    }

    // YES choice
    subset.add(nums[idx]);
    subsets(nums, idx + 1, subset);
    subset.remove(subset.size() - 1); // backtracking

    // NO choice
    subsets(nums, idx + 1, subset);
}

public List<List<Integer>> subsets(int[] nums) {
    subsets(nums, 0, new ArrayList<>());
    return powerset;
}
```

```java
List<List<Integer>> powerset = new ArrayList<>();

public void subsets(int[] nums, int idx, List<Integer> subset){
    if(idx == nums.length){
        powerset.add(new ArrayList<>(subset)); // deep copy
        return;
    }

    // YES choice
    subset.add(nums[idx]);
    subsets(nums, idx + 1, subset);
    subset.remove(subset.size() - 1); // backtracking

    // NO choice
    subsets(nums, idx + 1, subset);
}

public List<List<Integer>> subsets(int[] nums) {
    subsets(nums, 0, new ArrayList<>());
    return powerset;
}
```

① Backtracking

↳ Yes choice ⇒ will never be removed

② deep copy

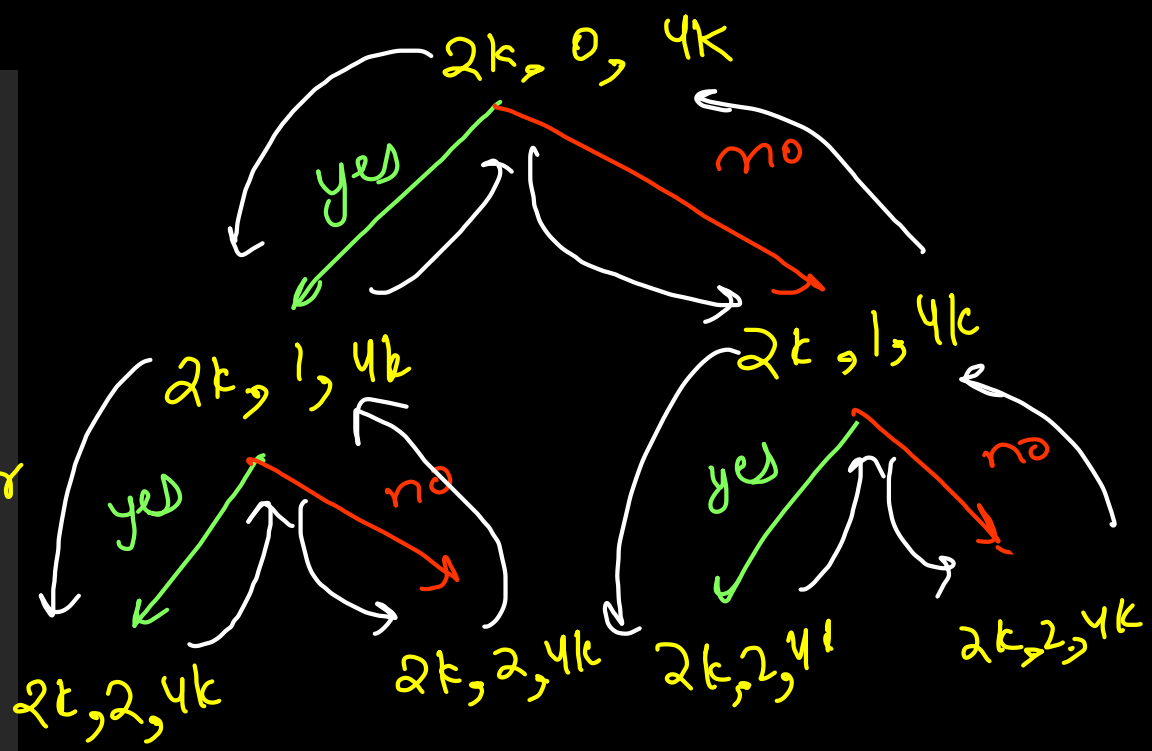
↳ 2D arraylist will contain same empty subsets

Time ⇒ $O(2^N)$

Space ⇒ $O(N)$ extra

```java
List<List<Integer>> powerset = new ArrayList<>();

public void subsets(int[] nums, int idx, List<Integer> subset){
    if(idx == nums.length){
        powerset.add(new ArrayList<>(subset)); // deep copy
        return;
    }

    // YES choice
    subset.add(nums[idx]);
    subsets(nums, idx + 1, subset);
    subset.remove(subset.size() - 1); // backtracking

    // NO choice
    subsets(nums, idx + 1, subset);
}

public List<List<Integer>> subsets(int[] nums) {
    subsets(nums, 0, new ArrayList<>());
    return powerset;
}
```

in postorder, undo the changes which were done in preorder



2k nums = [10, 20] (0, 1)

4k subset = [ ]

6k powerset = [8k, 18k, 23k, 25k]

8k [10,20]
18k [10]
23k [20]
25k [ ]

primitive variable (stack) & string (immutable) need not be backtracked or deep copied!

Combinatn (Selectn)

$n = 3, k = 0$

$3C_0 = 1 \Rightarrow [[]]$

$n = 3, k = 1$

$3C_1 = 3 \quad [[1], [2], [3]]$

$n = 3, k = 2$

$3C_2 = 3 \quad [[1,2], [1,3], [2,3]]$

$n = 3, k = 3$

$3C_3 = 1 \quad [[1,2,3]]$

Permutation (Arrangement)

$[1,2,3] = 3! = 6$

$[[1,2,3]$

$[1,3,2]$

$[2,1,3]$

$[2,3,1]$

$[3,1,2]$

$[3,2,1]]$

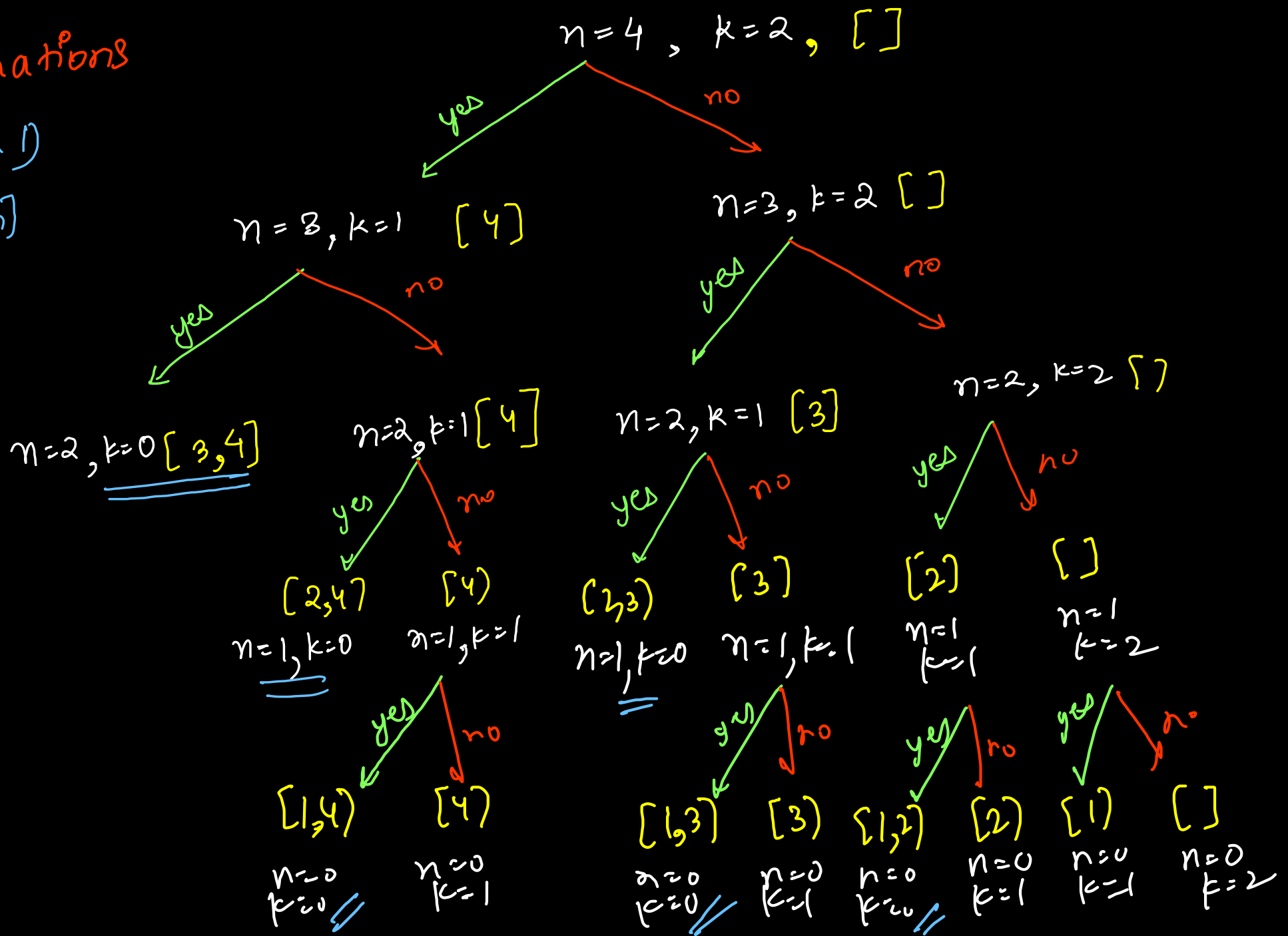
Combinations

Approach 1)

Item [1, n]

has choice

yes/no



```java
List<List<Integer>> res = new ArrayList<>();

public void combine(int n, int k, List<Integer> output){
    if(k == 0){
        res.add(new ArrayList<>(output)); // deep copy
        return;
    }
    if(n == 0) return;

    // yes choice
    output.add(n);
    combine(n - 1, k - 1, output);
    output.remove(output.size() - 1); // backtracking

    // no choice
    combine(n - 1, k, output);
}

public List<List<Integer>> combine(int n, int k) {
    combine(n, k, new ArrayList<>());
    return res;
}
```

Choices = 2, depth = N

Time ⇒ $O(2^N)$

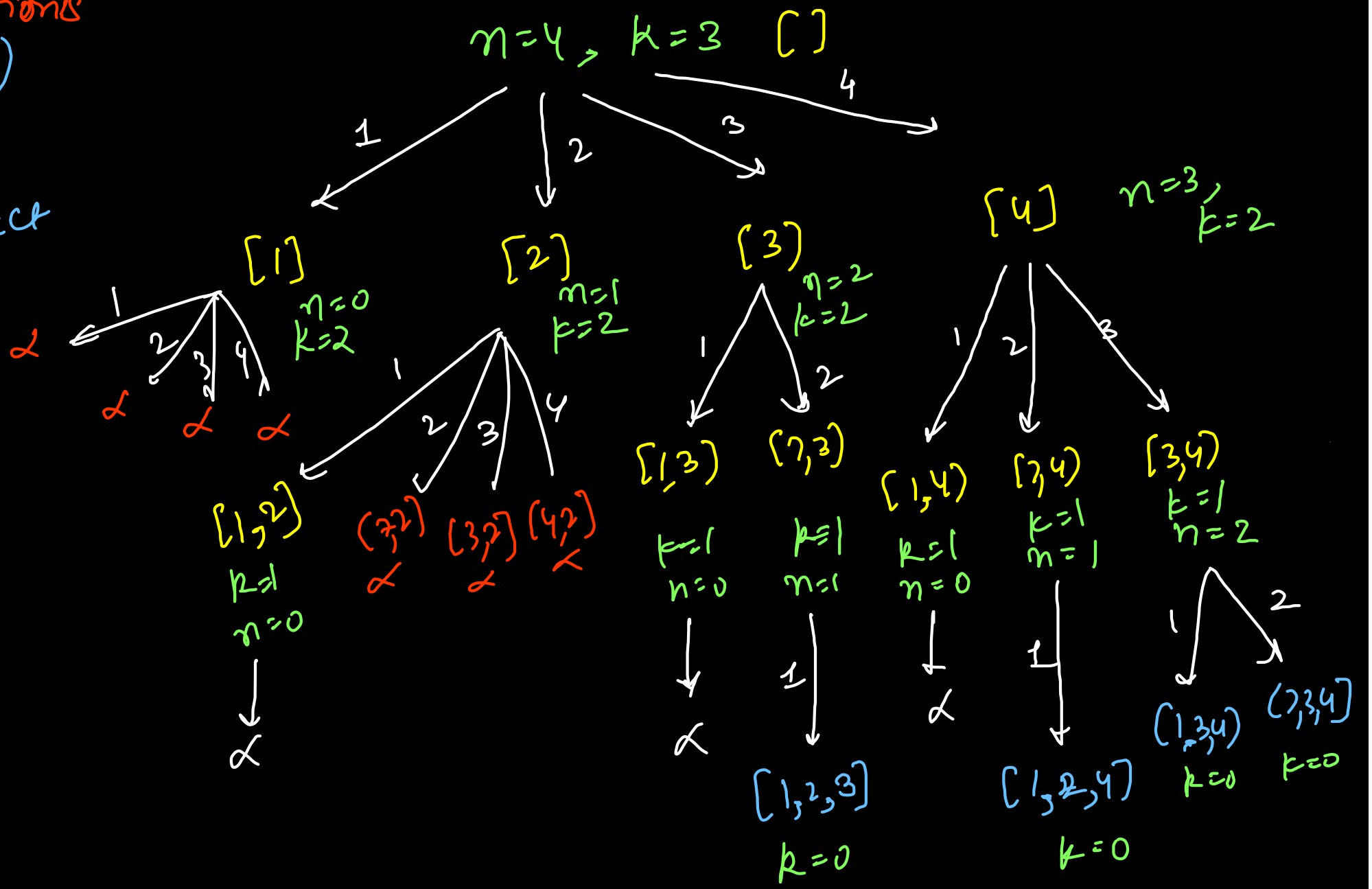
Space ⇒ $O(N)$
depth (Rec)

Approach 1)
Items have choice
(yes/no)

Combinations

Approach 2)

Which item to choose/select



```java
class Solution {
    List<List<Integer>> res = new ArrayList<>();

    public void combine(int n, int k, List<Integer> output){
        if(k == 0){
            res.add(new ArrayList<>(output)); // deep copy
            return;
        }
        if(n == 0) return;

        for(int item = n; item >= 1; item--){
            output.add(item);
            combine(item - 1, k - 1, output);
            output.remove(output.size() - 1);
        }
    }

    public List<List<Integer>> combine(int n, int k) {
        combine(n, k, new ArrayList<>());
        return res;
    }
}
```

Time → O(N*N-1*...1)

= O(N!)

exponential

Space → O(k)

depth

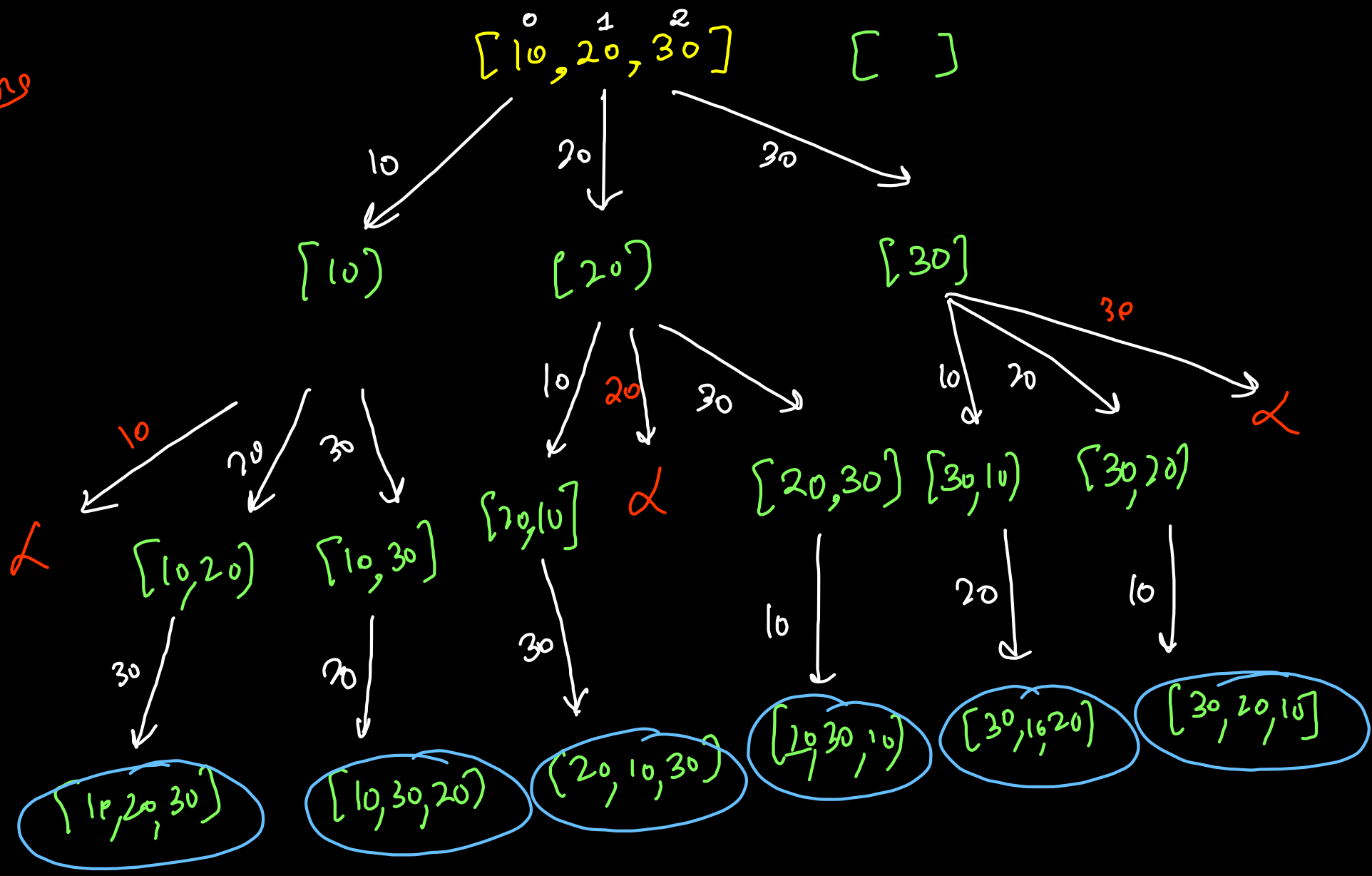
Approach 2)

Which item to select!

Permutations

Approach 1)

which item to pick first



```java
class Solution {
    List<List<Integer>> res = new ArrayList<>();

    public void permute(int[] nums, List<Integer> output){
        if(output.size() == nums.length){
            res.add(new ArrayList<>(output));
            return;
        }

        for(int item: nums){
            if(output.contains(item) == true) continue;
            output.add(item);
            permute(nums, output);
            output.remove(output.size() - 1);
        }
    }

    public List<List<Integer>> permute(int[] nums) {
        permute(nums, new ArrayList<>());
        return res;
    }
}
```

to avoid duplicate items

exponential

Time ⇒ O(N!)

Space ⇒ O(N)

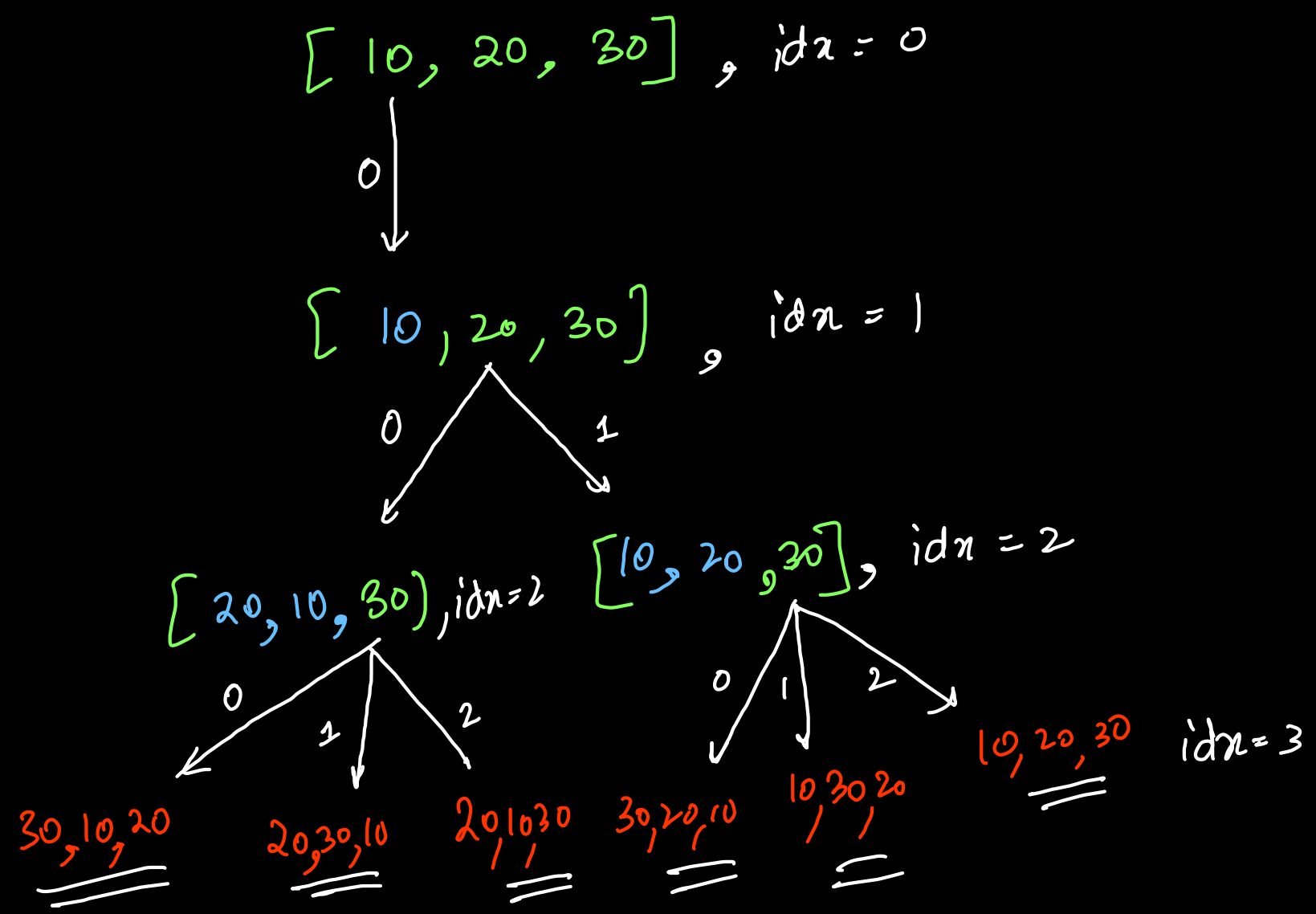
linear

Approach 1)

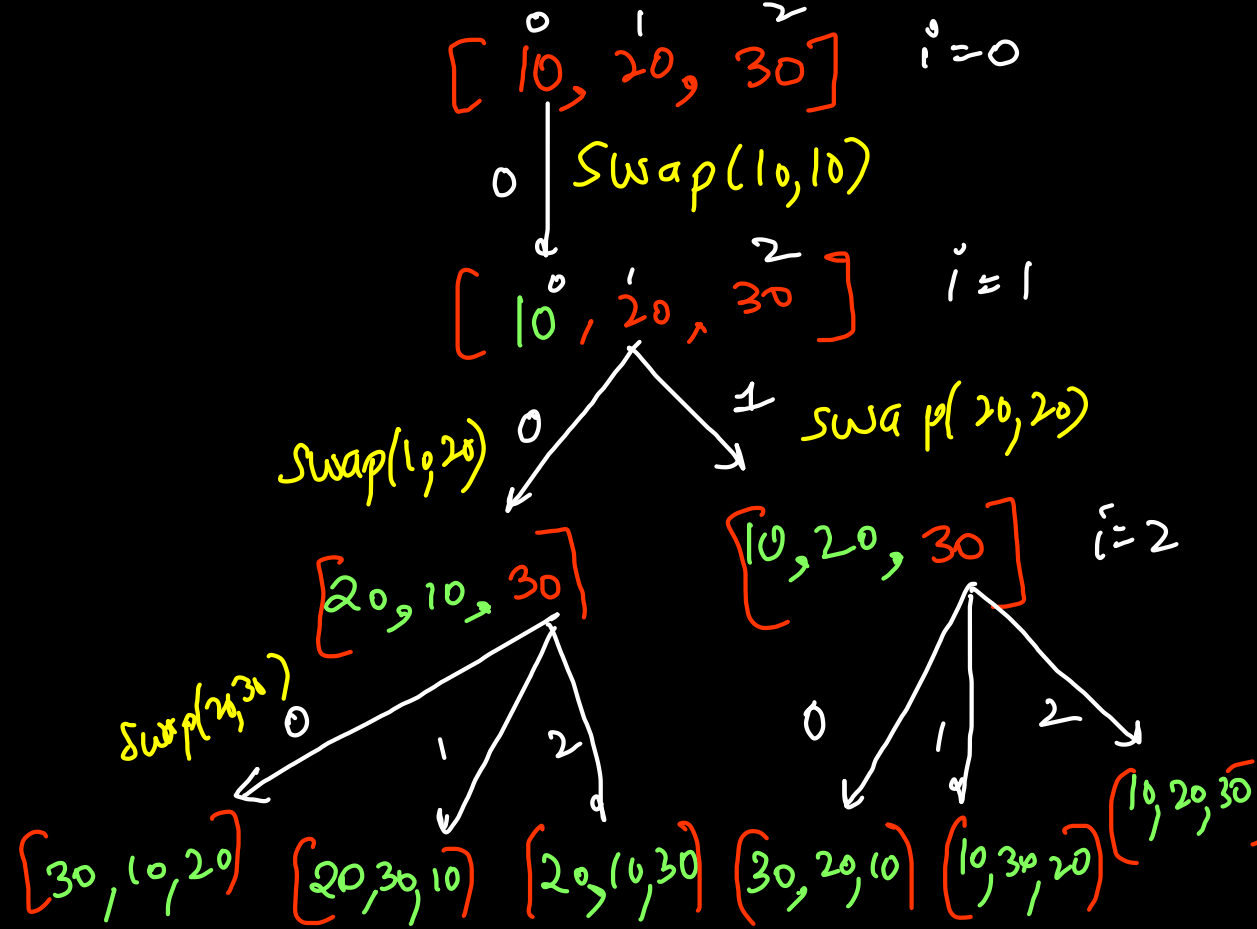
which item to choose!

Approach 2)

Item should go at which position



```java
class Solution {
    List<List<Integer>> res = new ArrayList<>();

    public void permute(int i, List<Integer> output){
        if(i == output.size()){
            res.add(new ArrayList<>(output));
            return;
        }

        for(int j = 0; j <= i; j++){
            Collections.swap(output, i, j);
            permute(i + 1, output);
            Collections.swap(output, i, j); // backtracking
        }
    }

    public List<List<Integer>> permute(int[] nums) {
        List<Integer> output = new ArrayList<>();
        for(int val: nums) output.add(val);
        permute(0, output);
        return res;
    }
}
```



Approach 2) Item → choice, index

Time = O(N!) Exponential

Space = O(N) depth

LC17 Letter Combinations of Phone No (print keypad)

0 → " "

1 → " "

2 → "abc"

3 → "def"

4 → "ghi"

5 → "jkl"

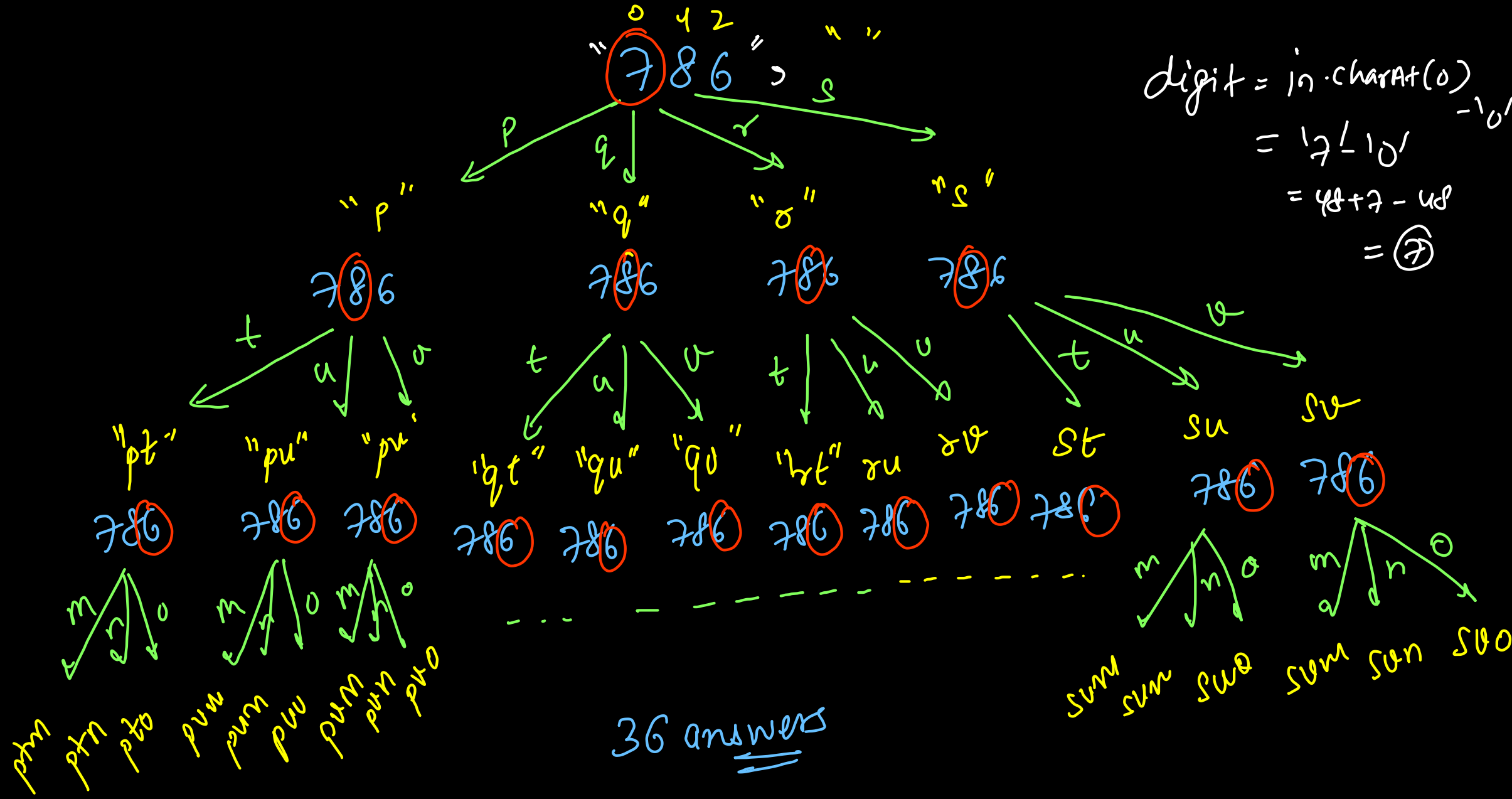
6 → "mno"

7 → "pqrs"

8 → "tuv"

9 → "wxyz"

"23" → ["ad", "ae", "af"
"bd", "be", "bf"
"cd", "ce", "cf"]

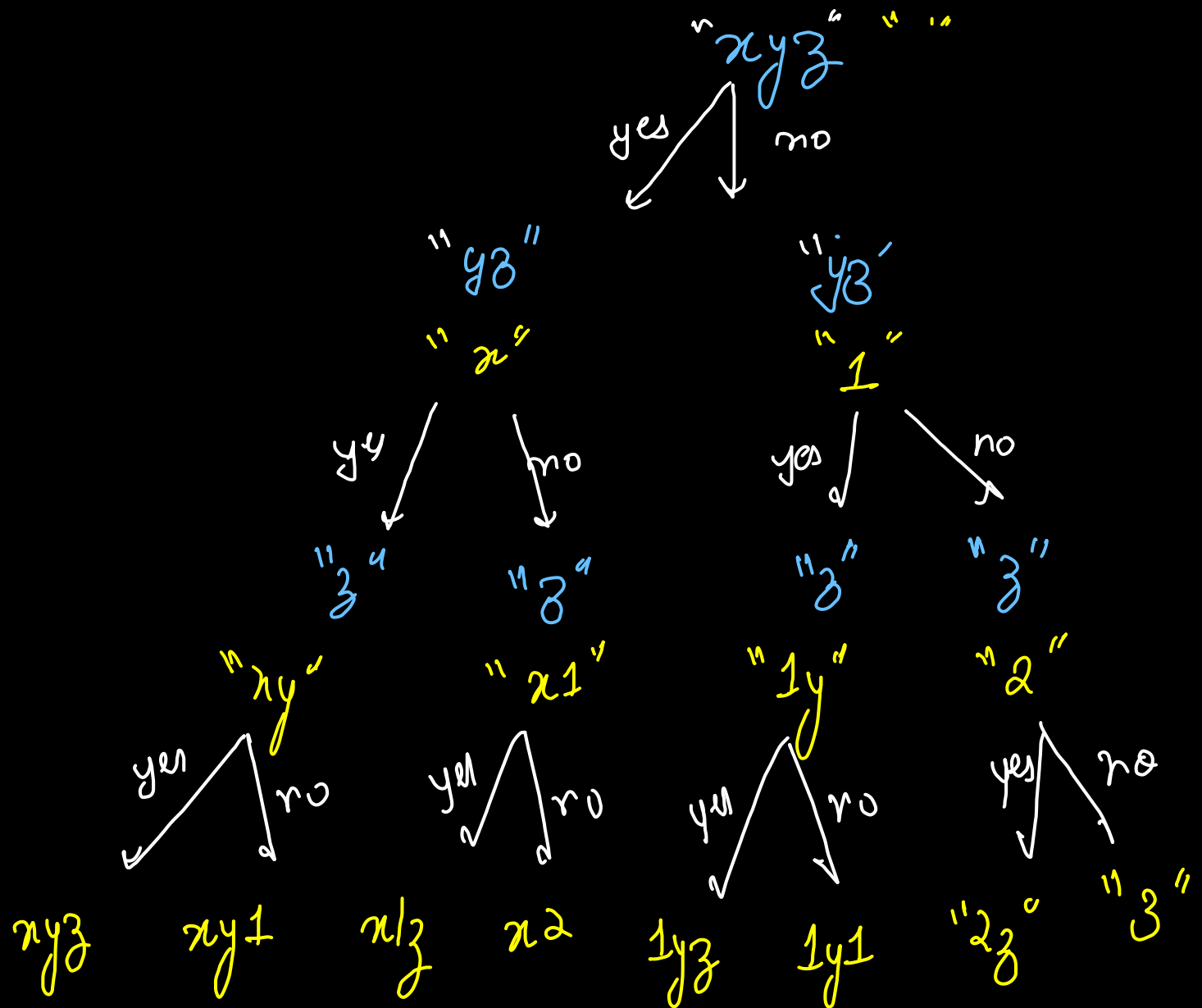

digit = in.charAt(0) - '0'
= '7' - '0'
= 55 - 48
= 7
36 answers

```java
class Solution {
    String[] map = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
    List<String> res = new ArrayList<>();

    public void helper(String input, int idx, String output){
        if(idx == input.length()) {
            res.add(output);
            return;
        }

        int digit = input.charAt(idx) - '0';
        for(char letter: map[digit].toCharArray()){
            helper(input, idx + 1, output + letter);
        }
    }

    public List<String> letterCombinations(String input) {
        if(input.length() == 0) return res;
        helper(input, 0, "");
        return res;
    }
}
```

Time
↳ $O(3^N)$
avg case
exponential

Space
↳ $O(N)$
depth
N = str.length

Generalized Abbreviations



"xyz"

xyz

1yz

x1z

xy1

2z

1y1

x2

3

"pqrs"

pqrs

1qrs

p1rs

pq1s

pqr1

2rs

1q1s

1qr1

p2s

p1r1

pq2

3s

2r1

1q2

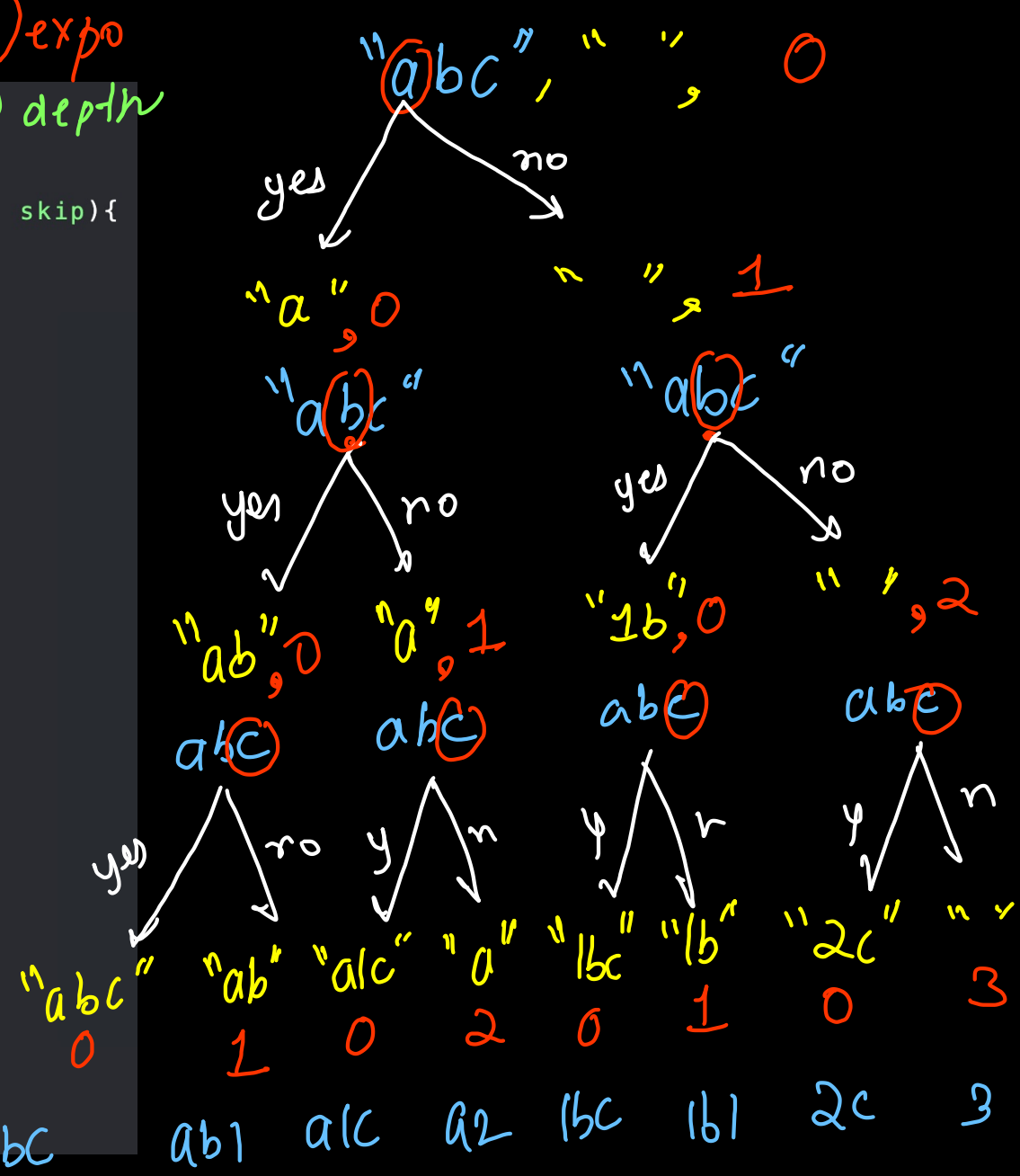
p3

4

Time ⇒ $O(2^N)$ expo

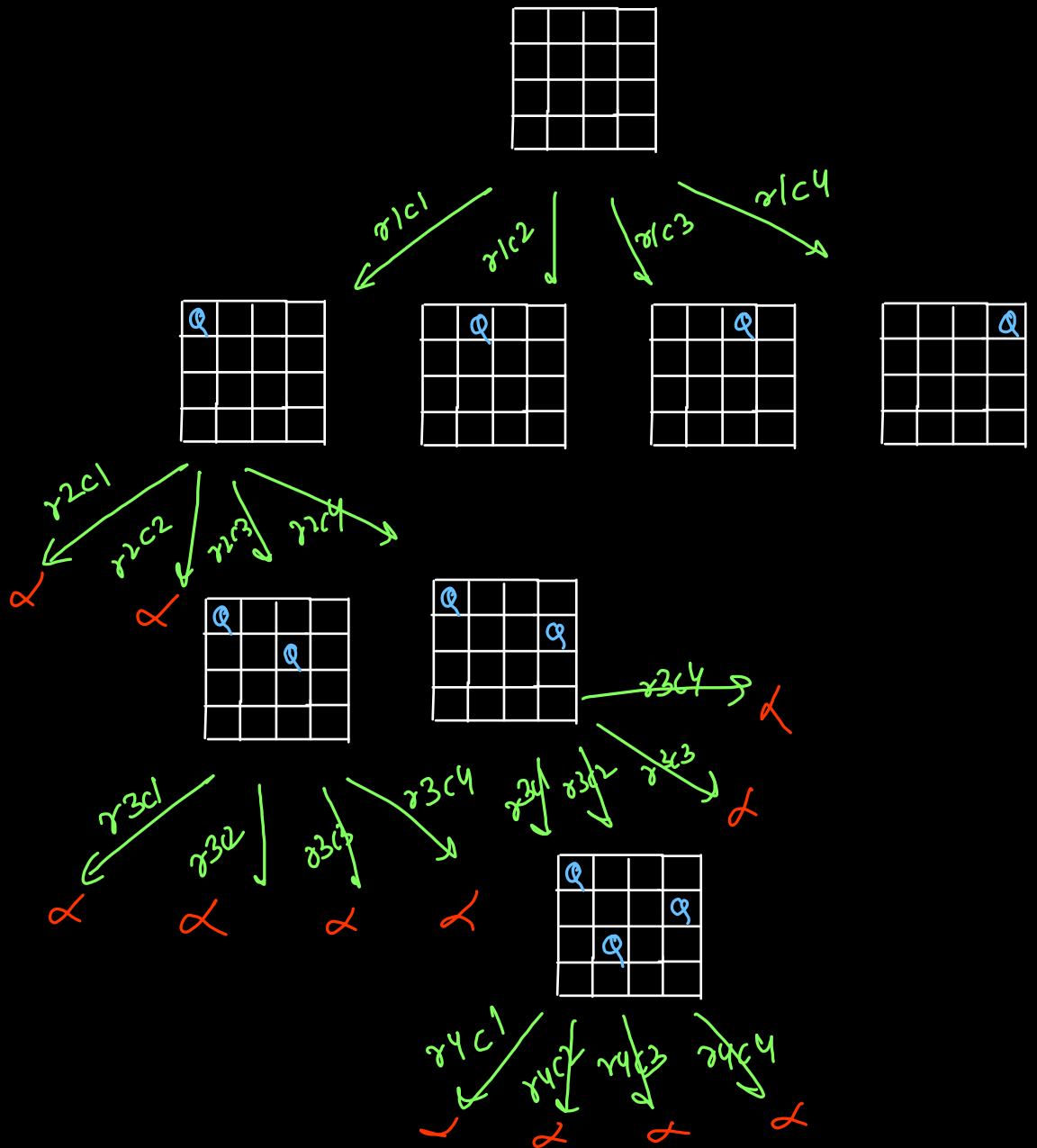
Space ⇒ $O(N)$ depth

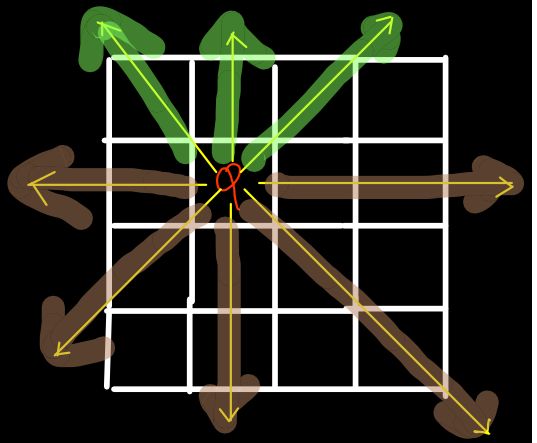
```java
static ArrayList<String> res;

0 references
static void helper(String input, int idx, String output, int skip){
    if(idx == input.length()){
        if(skip > 0) res.add(output + skip);
        else res.add(output);
        return;
    }

    char ch = input.charAt(idx);

    // yes (letter)
    if(skip > 0){
        helper(input, idx + 1, output + skip + ch, 0);
    } else {
        helper(input, idx + 1, output + ch, 0);
    }

    // no (digit)
    helper(input, idx + 1, output, skip + 1);
}

0 references
public static ArrayList < String > findAbbr(String str) {
    res = new ArrayList<>();
    helper(str, 0, "", 0);
    Collections.sort(res);
    return res;
}
```

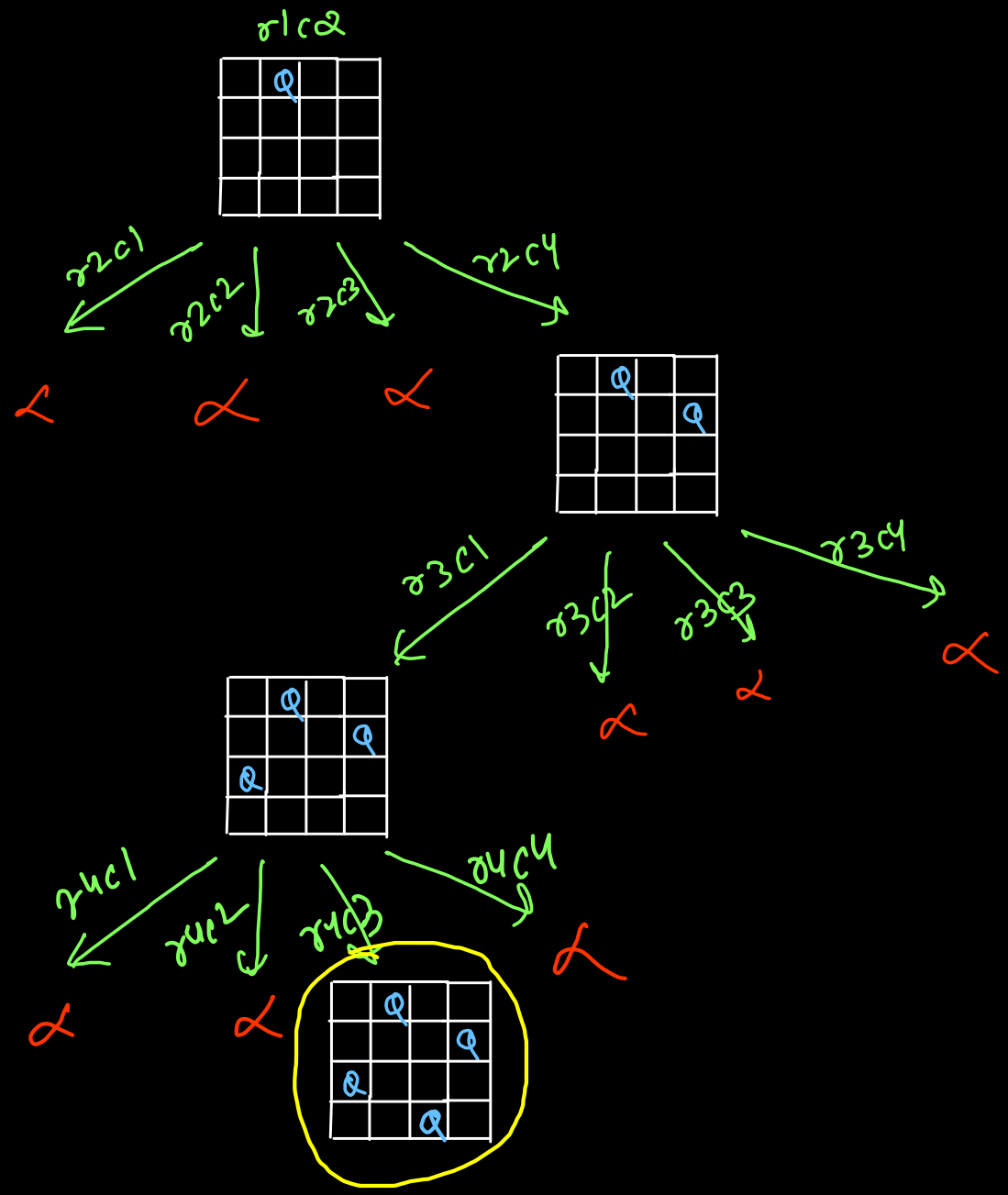


N Queen

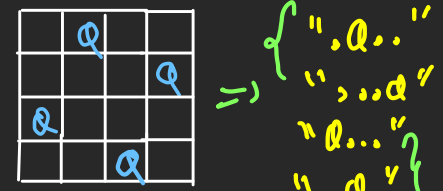
n = 4


r1c1
r1c2
r1c3
r1c4
r2c1
r2c2
r2c3
r2c4
r3c4
r3c1
r3c2
r3c3
r3c4
r4c1
r4c2
r4c3
r4c4




r1c2
r2c1
r2c2
r2c3
r2c4
r3c1
r3c2
r3c3
r3c4
r4c1
r4c2
r4c3
r4c4

```java
public void addOutput(boolean[][] chess, int n){
    List<String> config = new ArrayList<>();
    for(int row = 0; row < n; row++){
        String curr = "";
        for(int col = 0; col < n; col++){
            if(chess[row][col] == true)
                curr = curr + "Q";
            else curr = curr + ".";
        }
        config.add(curr);
    }
    configs.add(config);
}
```



=> { ".Q.." "...Q" "Q..." "..Q." }

```java
public void solve(int n, int row, boolean[][] chess){
    if(row == n){
        addOutput(chess, n);
        return;
    }
    for(int col = 0; col < n; col++){
        if(isAttacking(chess, row, col) == false){
            chess[row][col] = true;
            solve(n, row + 1, chess);
            chess[row][col] = false; // backtracking
        }
    }
}
```

fixed (N)

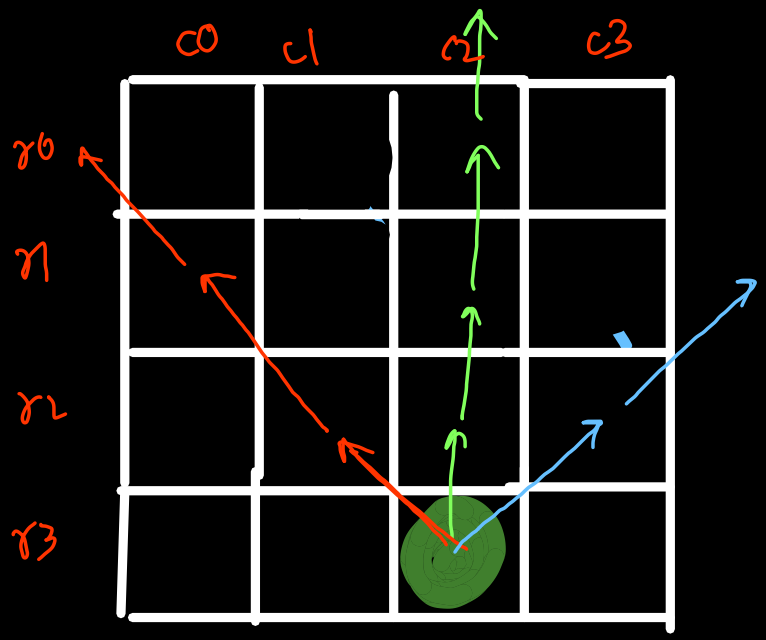
choice (N)

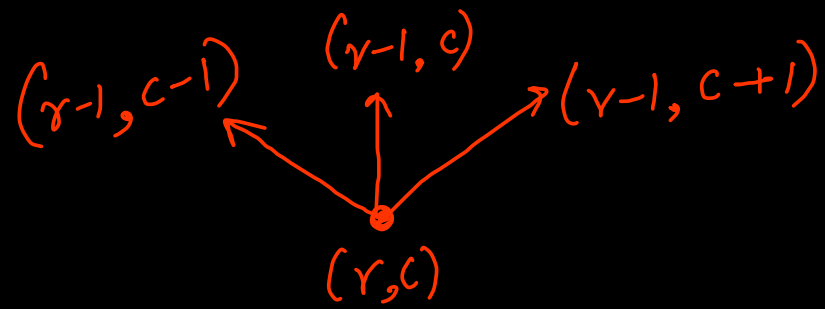
```java
public List<List<String>> solveNQueens(int n) {
    boolean[][] chess = new boolean[n][n];
    solve(n, 0, chess);
    return configs;
}
```

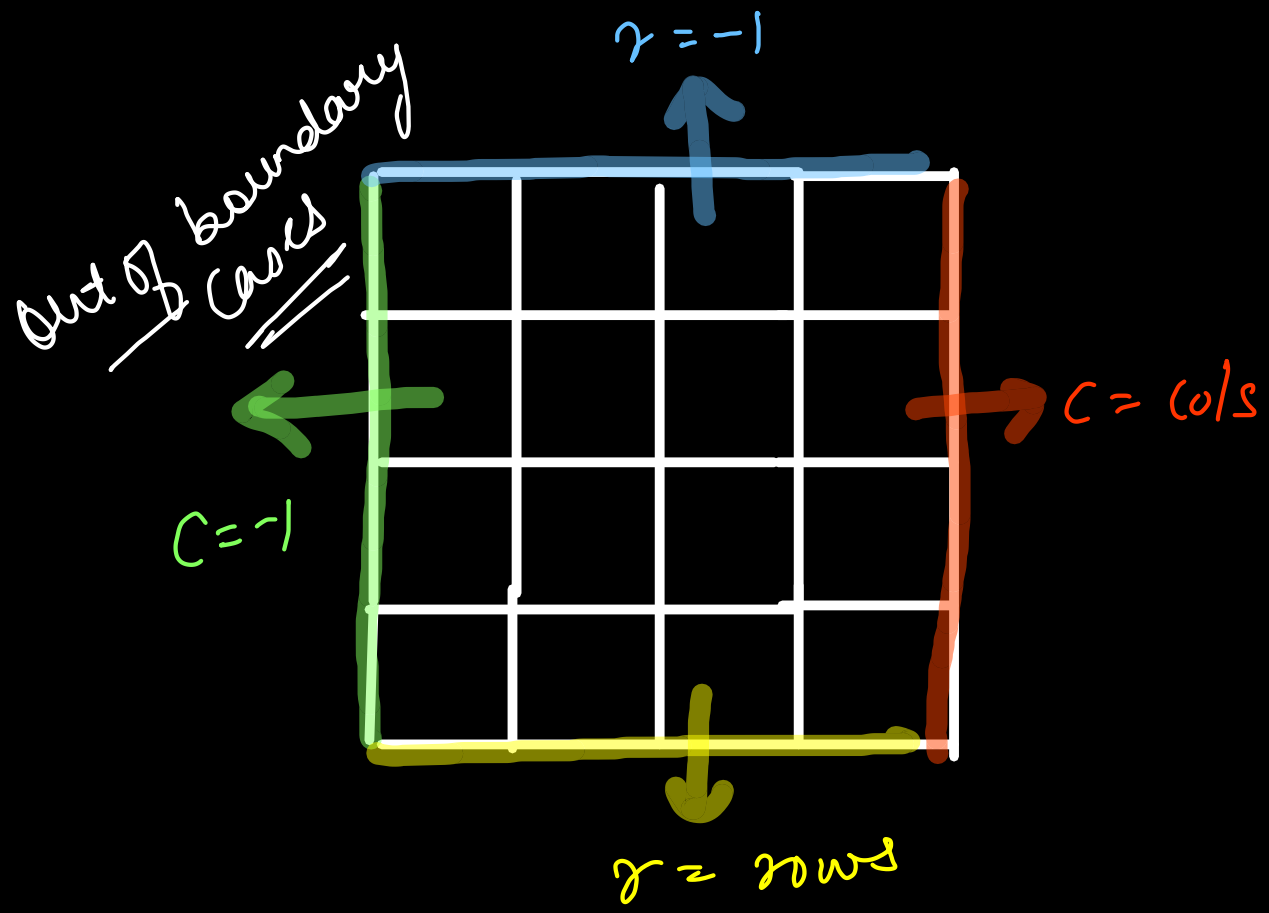
Time ~> $O(N^N)$

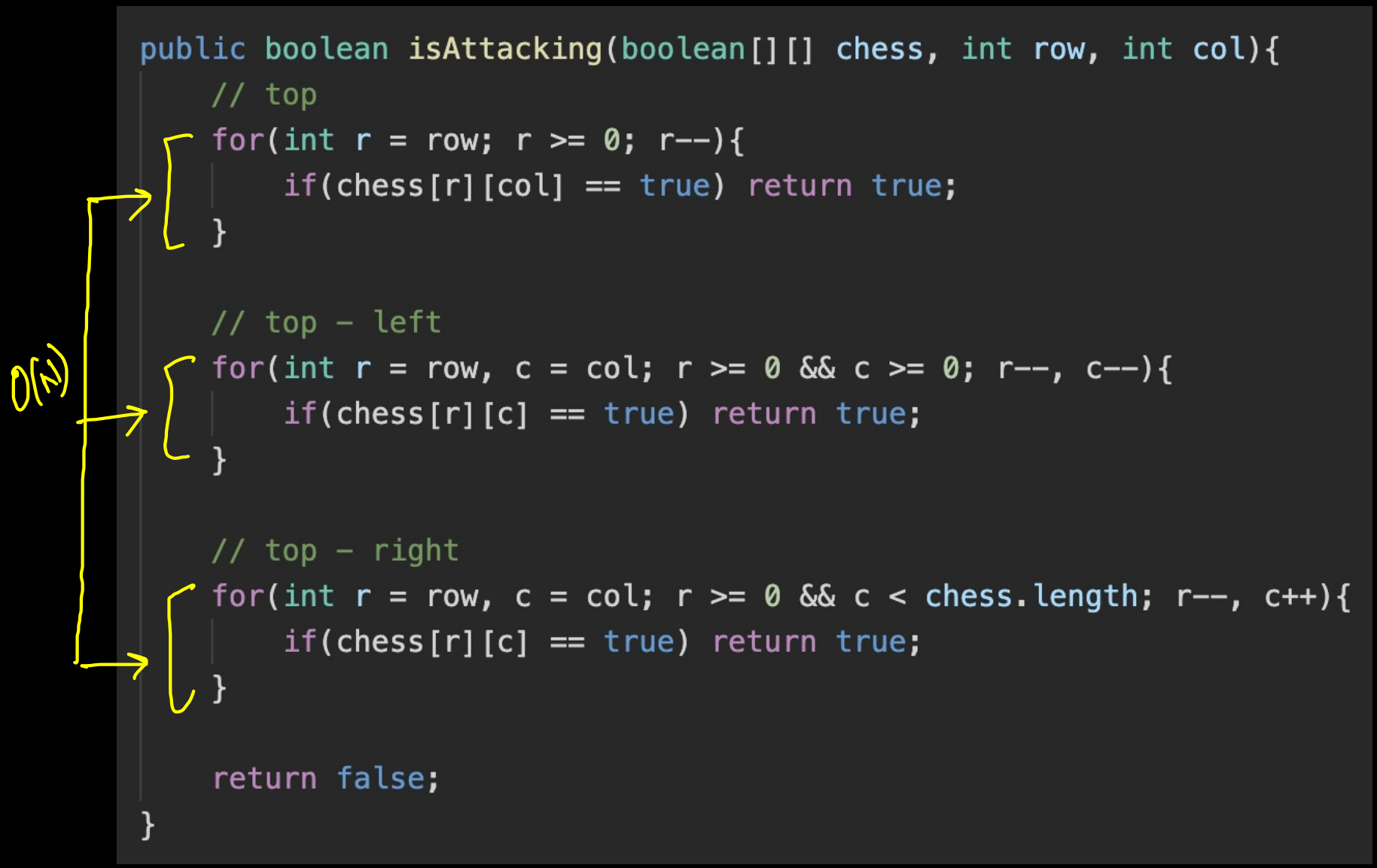
exponential

Space ~> $O(N)$


c0
c1
c2
c3
r0
r1
r2
r3


(r-1, c-1)
(r-1, c)
(r-1, c+1)
(r, c)


Out of boundary cases
r = -1
c = -1
c = cols
r = rows




```java
public boolean isAttacking(boolean[][] chess, int row, int col){
    // top
    for(int r = row; r >= 0; r--){
        if(chess[r][col] == true) return true;
    }

    // top - left
    for(int r = row, c = col; r >= 0 && c >= 0; r--, c--){
        if(chess[r][c] == true) return true;
    }

    // top - right
    for(int r = row, c = col; r >= 0 && c < chess.length; r--, c++){
        if(chess[r][c] == true) return true;
    }

    return false;
}
```

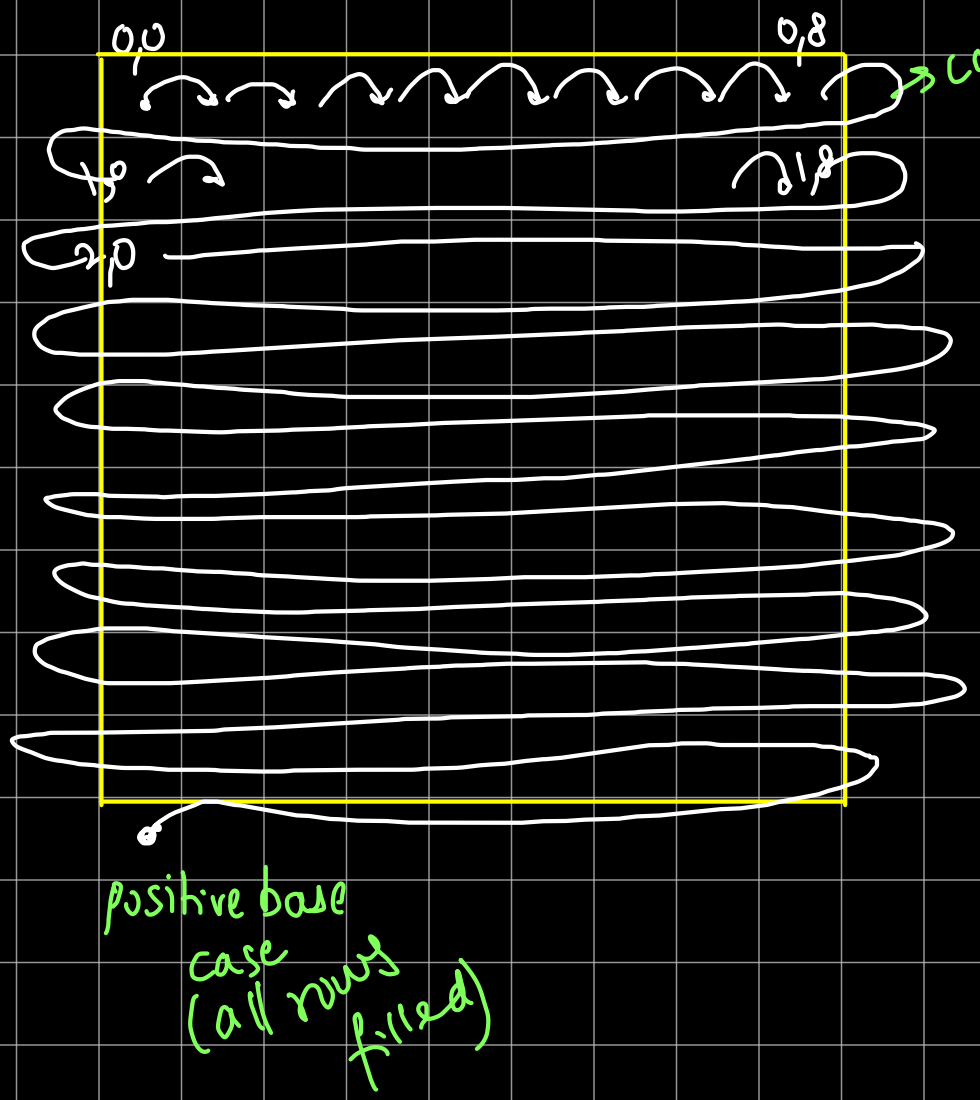
LC 37 Sudoku Solver Constraint

- Each row : $1 \leftrightarrow 9$
- Each col : $1 \leftrightarrow 9$
- 3x3 subgrid : $1 \leftrightarrow 9$

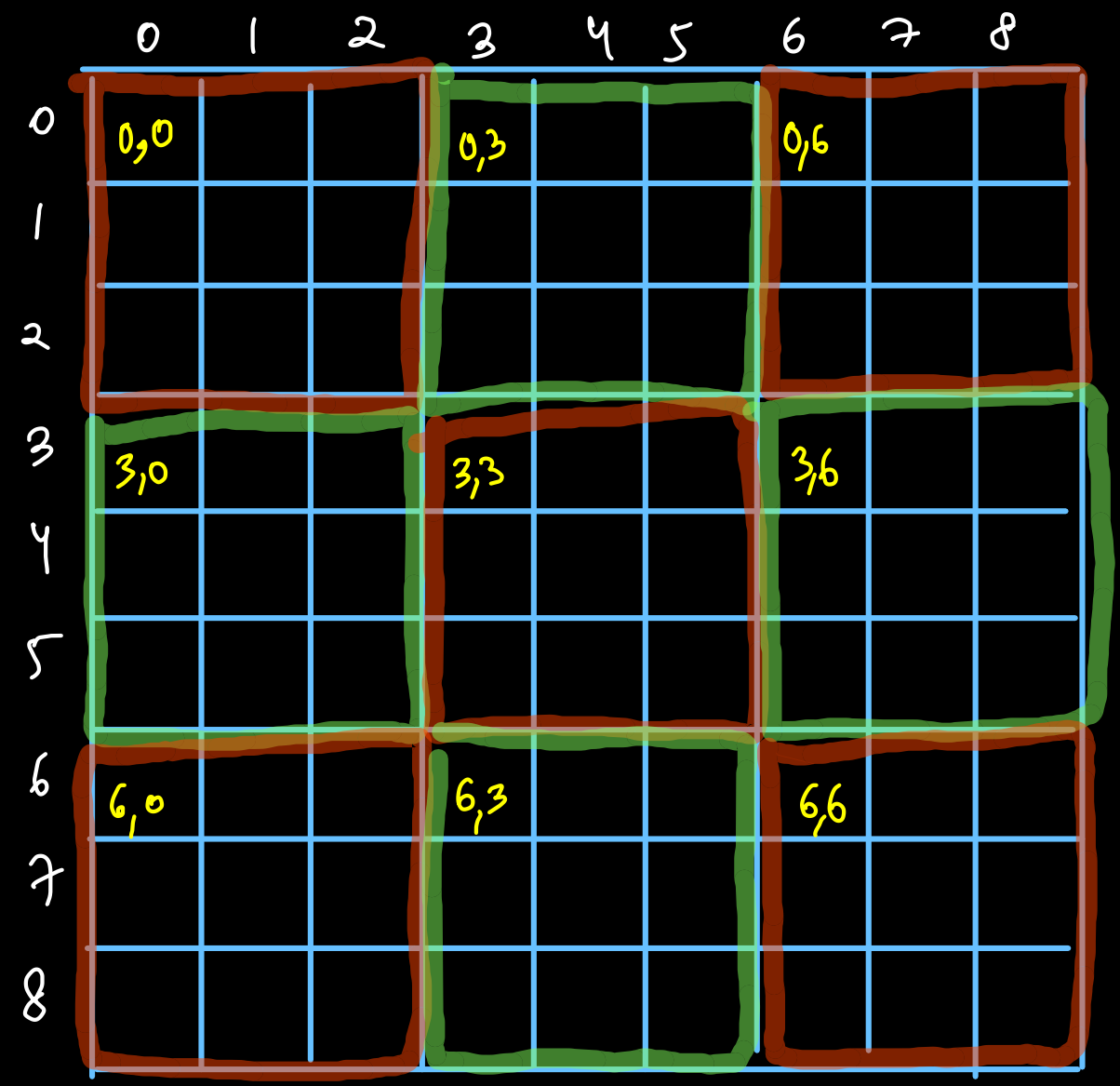
5	3	1	2	7	6			
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

9 x 9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9




```java
public boolean solve(char[][] board, int row, int col){
    if(col == 9){ row++; col = 0; }
    if(row == 9) return true;
    if(board[row][col] != '.')
        return solve(board, row, col + 1);

    for(char ch = '1'; ch <= '9'; ch++){
        if(isSafe(board, row, col, ch) == false)
            continue;
        board[row][col] = ch;
        if(solve(board, row, col + 1) == true)
            return true;
        board[row][col] = '.'; //backtracking
    }

    return false;
}

public void solveSudoku(char[][] board) {
    solve(board, 0, 0);
}
```



$0 \leftarrow 0, 1, 2$ $0/3 = 1/3 = 2/3 = 0$

$3 \leftarrow 3, 4, 5$ $3/3 = 4/3 = 5/3 = 1$

$6 \leftarrow 6, 7, 8$ $6/3 = 7/3 = 8/3 = 2$

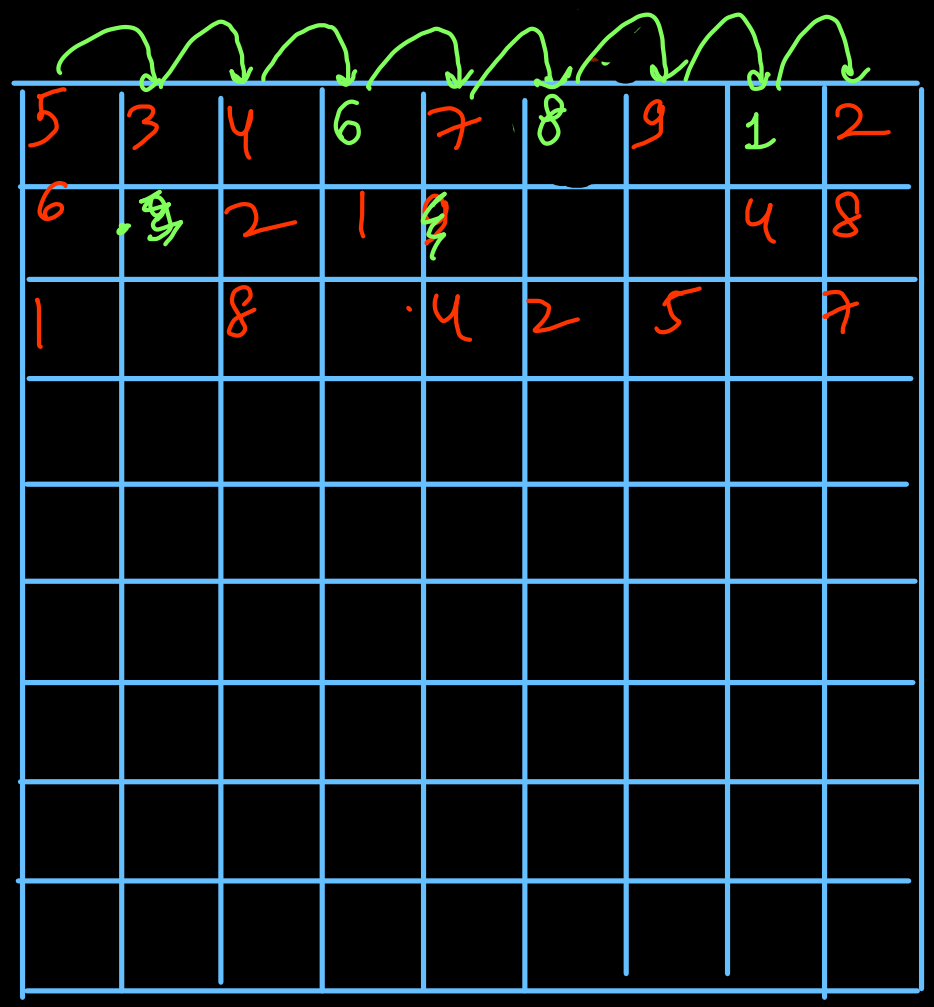
$\text{toprow} = (\text{row}/3) * 3$

$\text{topcol} = (\text{col}/3) * 3$

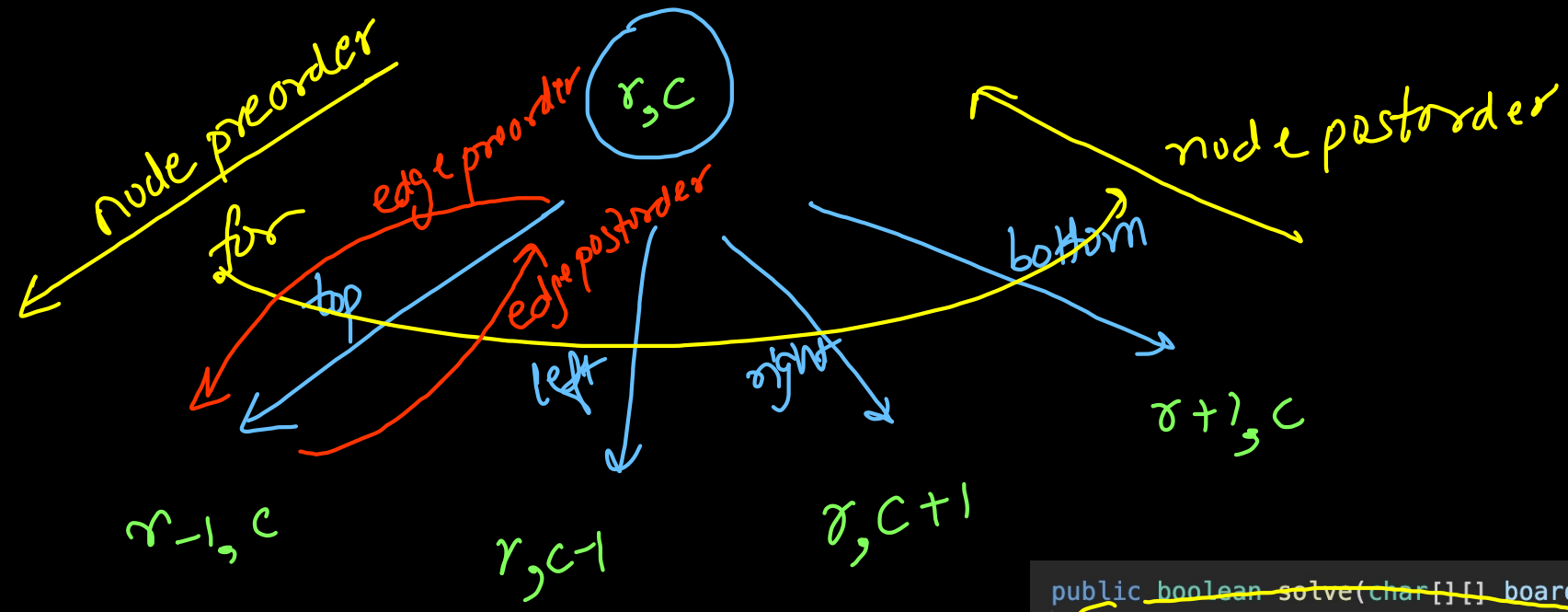
```java
public boolean isSafe(char[][] board, int row, int col, char ch){
    // row
    for(int c = 0; c < 9; c++){
        if(board[row][c] == ch) return false;
    }

    // col
    for(int r = 0; r < 9; r++){
        if(board[r][col] == ch) return false;
    }

    // subgrid
    int tr = (row / 3) * 3, tc = (col / 3) * 3;
    for(int r = 0; r < 3; r++){
        for(int c = 0; c < 3; c++){
            if(board[r + tr][c + tc] == ch) return false;
        }
    }

    return true;
}
```

Time → $(9 \times 9)^9$ exponential

```java
public boolean solve(char[][] board, int row, int col){
    if(col == 9){ row++; col = 0; }
    if(row == 9) return true;
    if(board[row][col] != '.')
        return solve(board, row, col + 1);

    for(char ch = '1'; ch <= '9'; ch++){
        if(isSafe(board, row, col, ch) == false)
            continue;
        board[row][col] = ch;
        if(solve(board, row, col + 1) == true)
            return true;
        board[row][col] = '.'; // backtracking
    }

    return false;
}

public void solveSudoku(char[][] board) {
    solve(board, 0, 0);
}
```



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9




```java
public boolean solve(char[][] board, int row, int col){
    if(col == 9){ row++; col = 0; }
    if(row == 9) return true;
    if(board[row][col] != '.')
        return solve(board, row, col + 1);

    for(char ch = '1'; ch <= '9'; ch++){
        if(isSafe(board, row, col, ch) == false)
            continue;
        board[row][col] = ch;
        if(solve(board, row, col + 1) == true)
            return true;
        board[row][col] = '.';
    }

    return false;
}
```

node preorder

edge preorder

edge postorder

node postorder

LC 36

```java
public boolean isSafe(char[][] board, int row, int col, char ch){
    // row
    for(int c = 0; c < 9; c++){
        if(board[row][c] == ch) return false;
    }

    // col
    for(int r = 0; r < 9; r++){
        if(board[r][col] == ch) return false;
    }

    // subgrid
    int tr = (row / 3) * 3, tc = (col / 3) * 3;
    for(int r = 0; r < 3; r++){
        for(int c = 0; c < 3; c++){
            if(board[r + tr][c + tc] == ch) return false;
        }
    }

    return true;
}
```

```java
public boolean solve(char[][] board, int row, int col){
    if(col == 9){ row++; col = 0; }
    if(row == 9) return true;
    if(board[row][col] == '.')
        return solve(board, row, col + 1);

    char ch = board[row][col];
    board[row][col] = '.';

    if(isSafe(board, row, col, ch) == false)
        return false;

    board[row][col] = ch;
    return solve(board, row, col + 1);
}

public boolean isValidSudoku(char[][] board) {
    return solve(board, 0, 0);
}
```

empty → skip

filled → duplicate

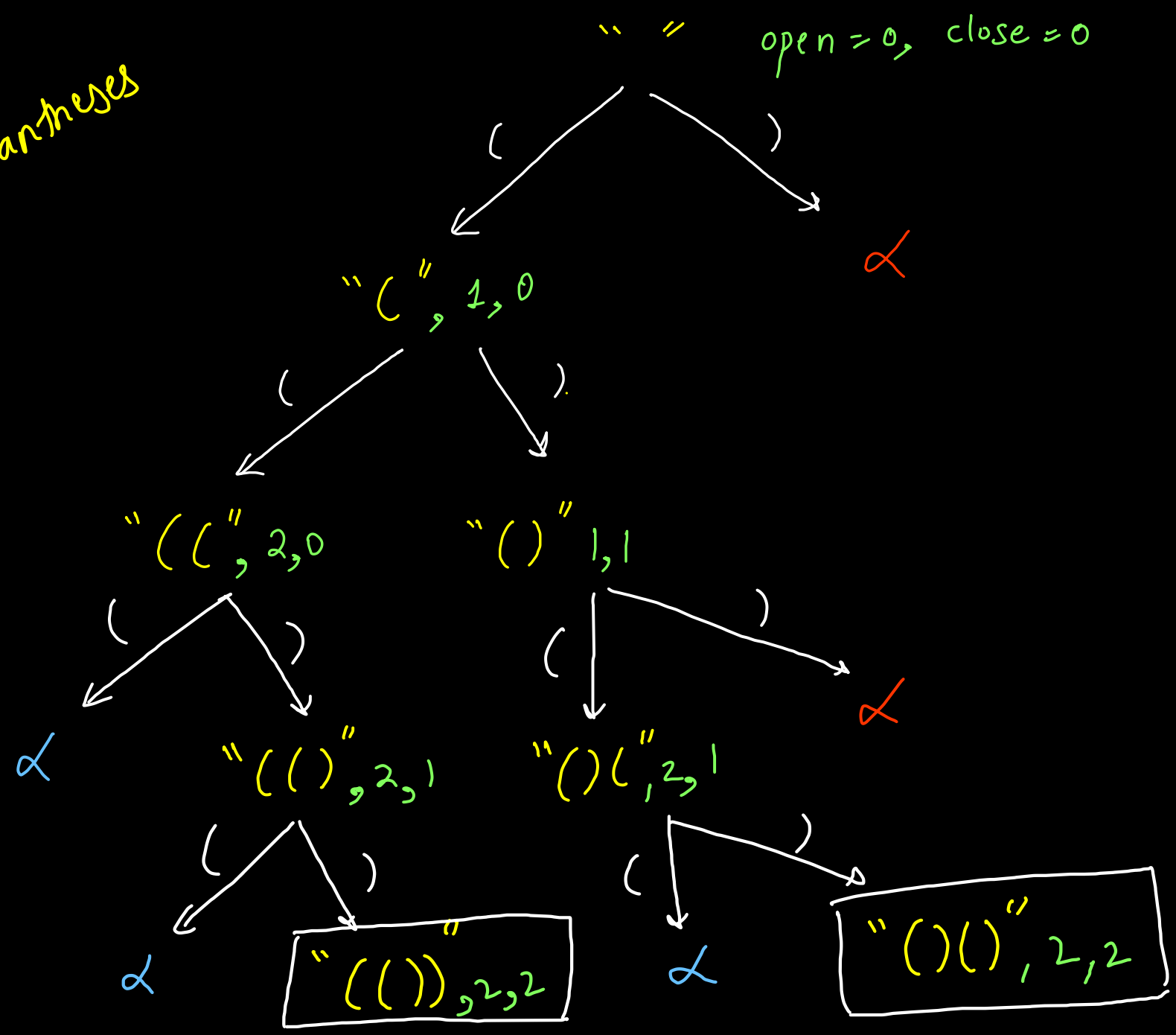
linear recursion → $O(81*9)$

$O(n^3)$

$O(81*9)$ → depth, issafe

n=2

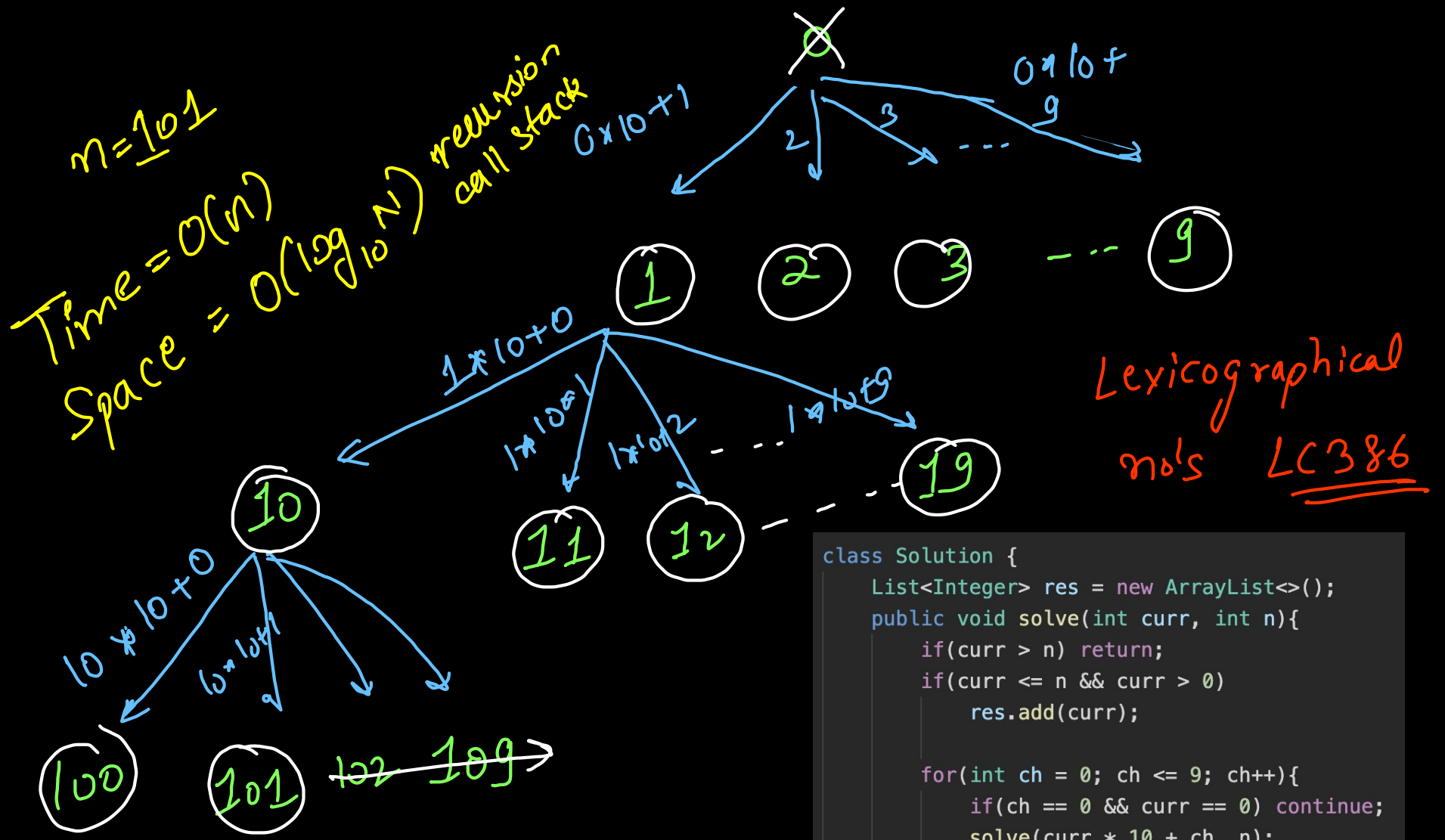
LC22
Generate Parantheses



```java
class Solution {
    List<String> res = new ArrayList<>();

    public void solve(String output, int open, int close, int n){
        if(output.length() == 2 * n){
            res.add(output);
            return;
        }

        if(open < n) solve(output + "(", open + 1, close, n);
        if(close < open) solve(output + ")", open, close + 1, n);
    }

    public List<String> generateParenthesis(int n) {
        solve("", 0, 0, n);
        return res;
    }
}
```

Time = $O(2^N)$ Expo

Space = $O(N)$ depth



Lexicographical nos LC386

```
class Solution {
    List<Integer> res = new ArrayList<>();
    public void solve(int curr, int n){
        if(curr > n) return;
        if(curr <= n && curr > 0)
            res.add(curr);

        for(int ch = 0; ch <= 9; ch++){
            if(ch == 0 && curr == 0) continue;
            solve(curr * 10 + ch, n);
        }
    }
    public List<Integer> lexicalOrder(int n) {
        solve(0, n);
        return res;
    }
}
```

1
10
100
101
11
12
19
2
20 ... 29
3
30 ... 39
4
40 ... 49

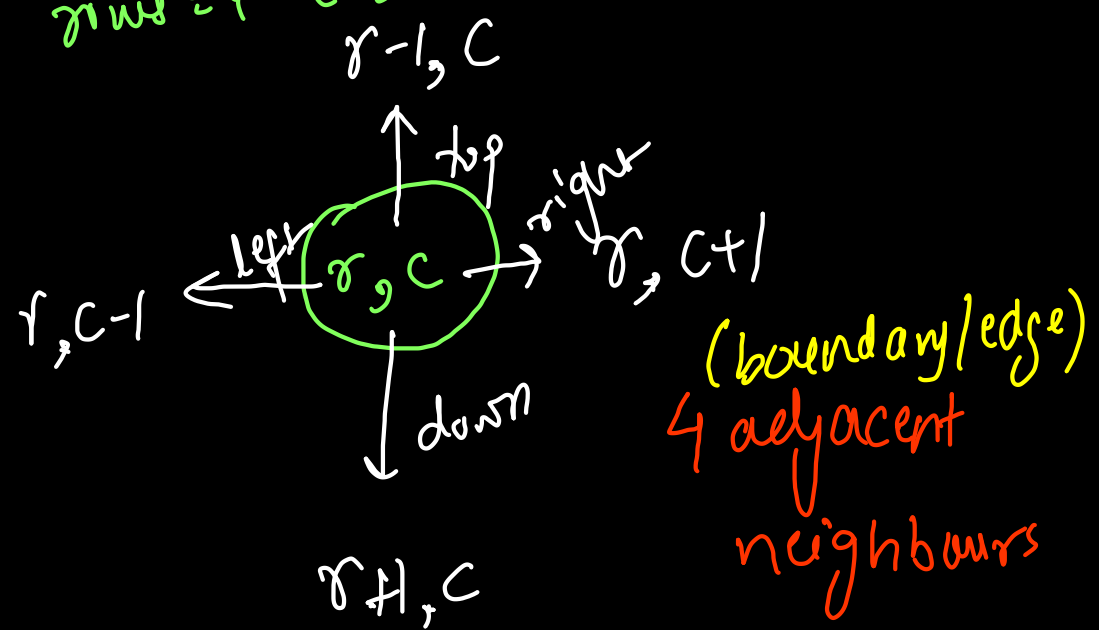
	0	1	2	3	4	5	6	7	8
0	a	r	c	h	i	t	a	g	g
1	a	g	w	a	l	t	e	r	s
2	c	o	a	t	c-1	o-1	d-1	e-1	s
3	a	h	r	o	h	i	t	-1 d	d

"coded" → true

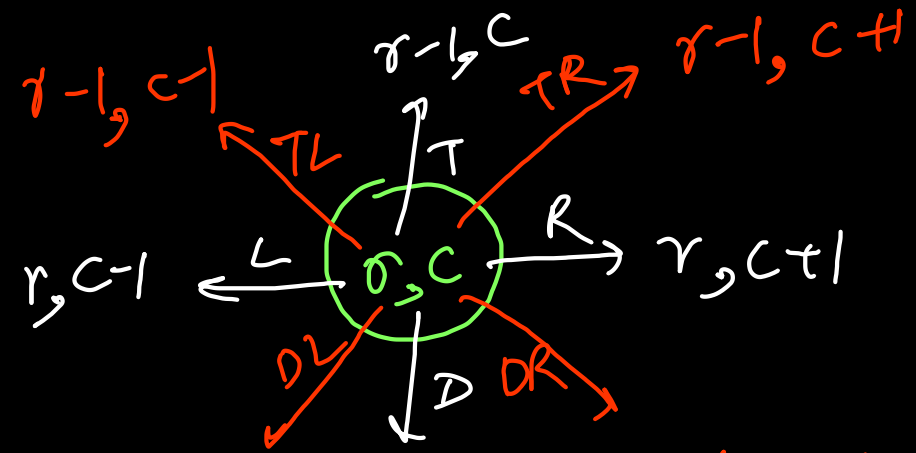
"aggarwal" → false

```
// Top, Left, Down, Right
int[] dx = {-1, 0, +1, 0};
int[] dy = {0, -1, 0, +1};
```

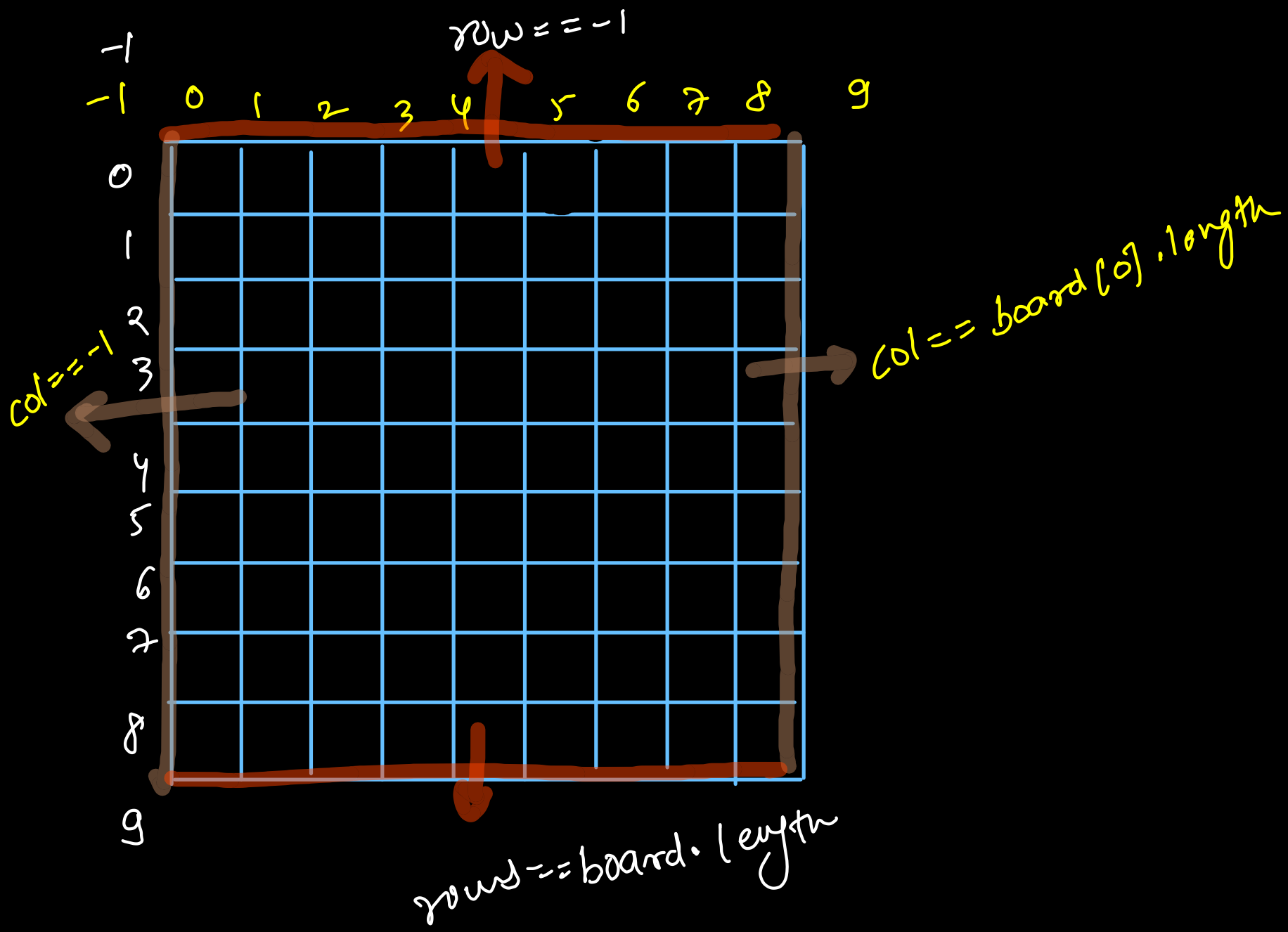
rows = 4 cols = 9



(boundary/edge)
4 adjacent neighbours



(vertex sharing)
8 adjacent neighbours


row == -1
-1
-1 0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9
col == -1
col == board[0].length
rows == board.length

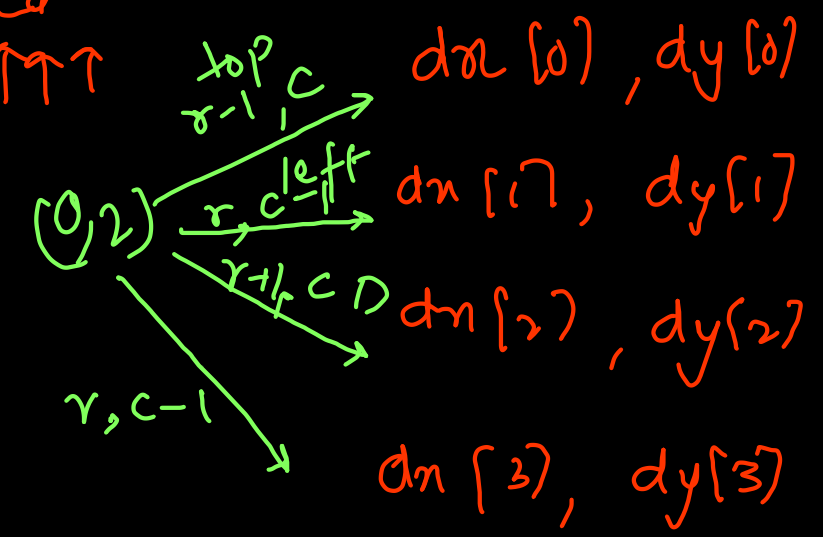
LC79 Word Search - I

```java
// Top, Left, Down, Right
int[] dx = {-1, 0, +1, 0};
int[] dy = {0, -1, 0, +1};

public boolean dfs(char[][] board, String word, int row, int col, int idx){
    if(idx == word.length())
        return true; // Entire Word Search Successfully
    if(row >= board.length || col >= board[0].length || row < 0 || col < 0)
        return false; // Out of Matrix
    if(board[row][col] != word.charAt(idx))
        return false; // Not Matching

    for(int i = 0; i < 4; i++){
        char ch = board[row][col];
        board[row][col] = '#'; //visited
        if(dfs(board, word, row + dx[i], col + dy[i], idx + 1) == true)
            return true;
        board[row][col] = ch; // backtracking
    }

    return false;
}
```

no of calls = 4
depth = L
$O(4^L)$

	0	1	2	3	4	5	6	7	8
0	a	r	c	h	i	t	a	g	g
1	a	g	w	a	l	t	e	r	s
2	c	o	a	t	c	o	d	e	s
3	a	h	r	o	h	i	t	d	d

"Coded"




```java
public boolean exist(char[][] board, String word) {
    for(int r = 0; r < board.length; r++){
        for(int c = 0; c < board[0].length; c++){
            if(dfs(board, word, r, c, 0) == true)
                return true;
        }
    }
    return false;
}
```

A knight is a chess piece that can move from cell (x1, y1) to the cell (x2, y2) if one of the following conditions is met:

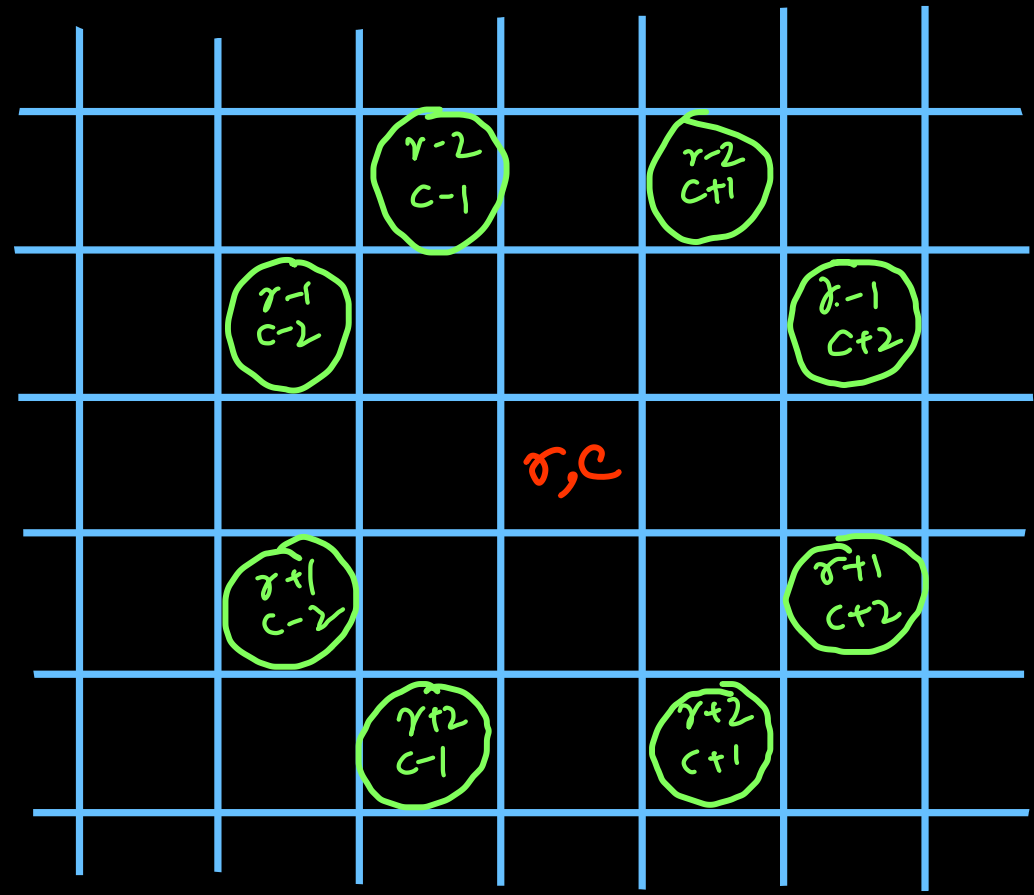
$|x1-x2| = 2$ and $|y1-y2| = 1$, or

$|x1-x2| = 1$ and $|y1-y2| = 2$.

A knight cannot move outside the chessboard.

Knight's Tour

0	59	38	33	30	17	8	63
37	34	31	60	9	62	29	16
58	1	36	39	32	27	18	7
35	48	41	26	61	10	15	28
42	57	2	49	40	23	6	19
47	50	45	54	25	20	11	14
56	43	52	3	22	13	24	5
51	46	55	44	53	4	21	12



```java
static ArrayList<ArrayList<Integer>> chess;
static int rows, cols;

static int[] dx = {-1, -1, +1, +1, +2, +2, -2, -2};
static int[] dy = {+2, -2, -2, +2, -1, +1, -1, +1};

0 references
public static boolean dfs(int r, int c, int count){
    if(count == rows * cols) return true;
    if(r < 0 || r >= rows || c < 0 || c >= cols) return false;
    if(chess.get(r).get(c) != -1) return false;

    for(int i = 0; i < 8; i++){
        chess.get(r).set(c, count); // visited mark
        if(dfs(r + dx[i], c + dy[i], count + 1) == true)
            return true;
        chess.get(r).set(c, -1); // backtracking
    }

    return false;
}
```

Calls = 8

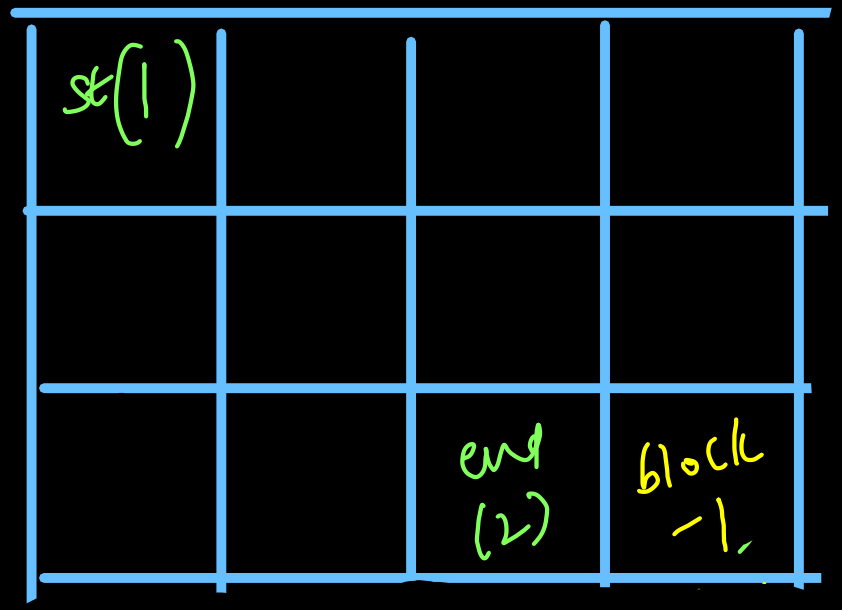
Depth = m x n

Time = O(8^mn)

exponential

```java
public static ArrayList<ArrayList<Integer>> knightTour(int m, int n) {
    rows = m; cols = n;
    chess = new ArrayList<>();
    for(int i=0; i < rows; i++){
        ArrayList<Integer> row = new ArrayList<>();
        for(int j = 0; j < cols; j++)
            row.add(-1);
        chess.add(row);
    }

    dfs(0, 0, 0);
    return chess;
}
```

Unique Paths - III



① don't walk over -1

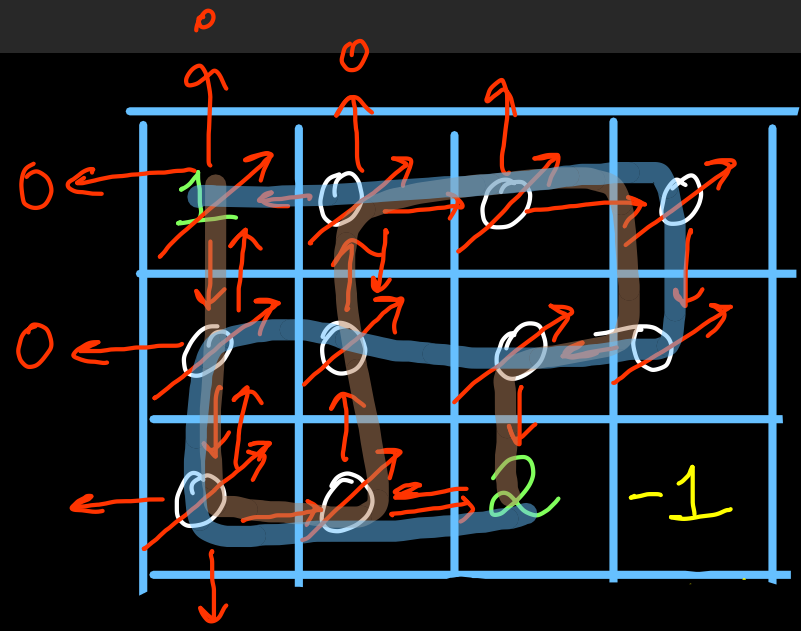
② vist every cell exactly once

Count = 11

(Rat in a Maze)

```java
int[] dx = {-1, +1, 0, 0}; int[] dy = {0, 0, -1, +1};

public int dfs(int[][] grid, int r, int c, int count){
    if(r < 0 || c < 0 || r >= grid.length || c >= grid[0].length)
        return 0;    // out of matrix
    if(grid[r][c] == -1) return 0; // blocked or visited
    if(grid[r][c] == 2) return (count == 1) ? 1 : 0; // target

    int paths = 0;
    for(int i = 0; i < 4; i++){
        grid[r][c] = -1; // visited
        paths += dfs(grid, r + dx[i], c + dy[i], count - 1);
        grid[r][c] = 0; // backtracking
    }

    return paths;
}
```

```java
public int uniquePathsIII(int[][] grid) {
    int r = 0, c = 0;
    int count = 0;  → number of non-obstacle path
    for(int i = 0; i < grid.length; i++){
        for(int j = 0; j < grid[0].length; j++){
            if(grid[i][j] == 1){
                r = i; c = j;   // Starting pt
            }
            if(grid[i][j] != -1)
                count++;        // non-obstacle
        }
    }

    return dfs(grid, r, c, count);
}
```

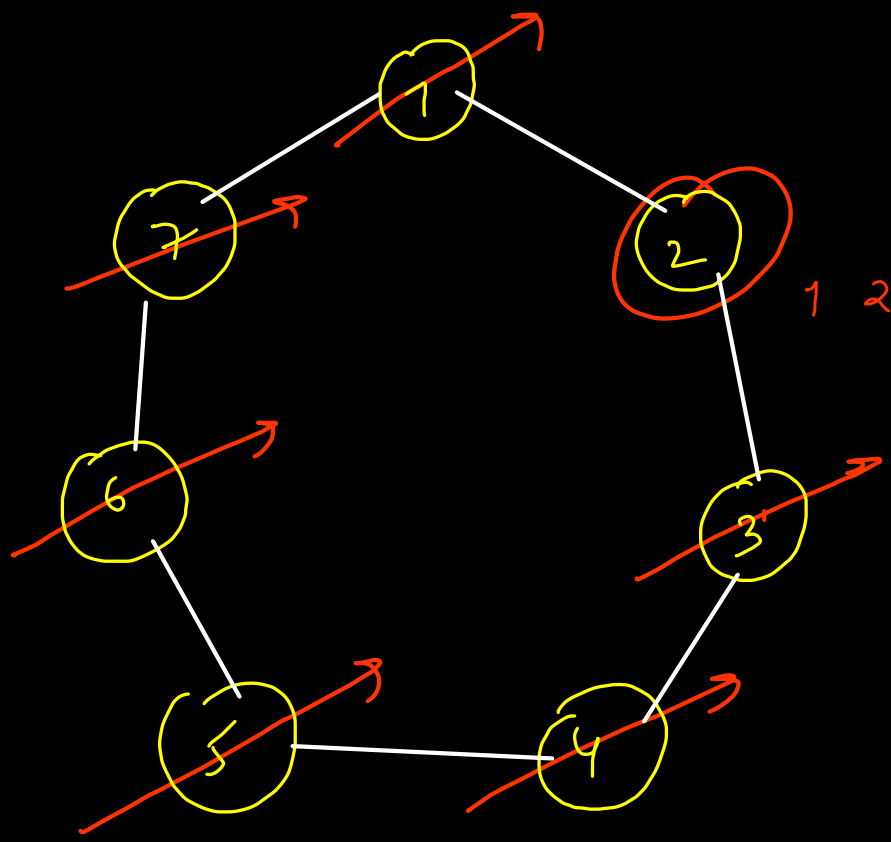


count = 11 10 9 8 7 6 5 4 3 2 1

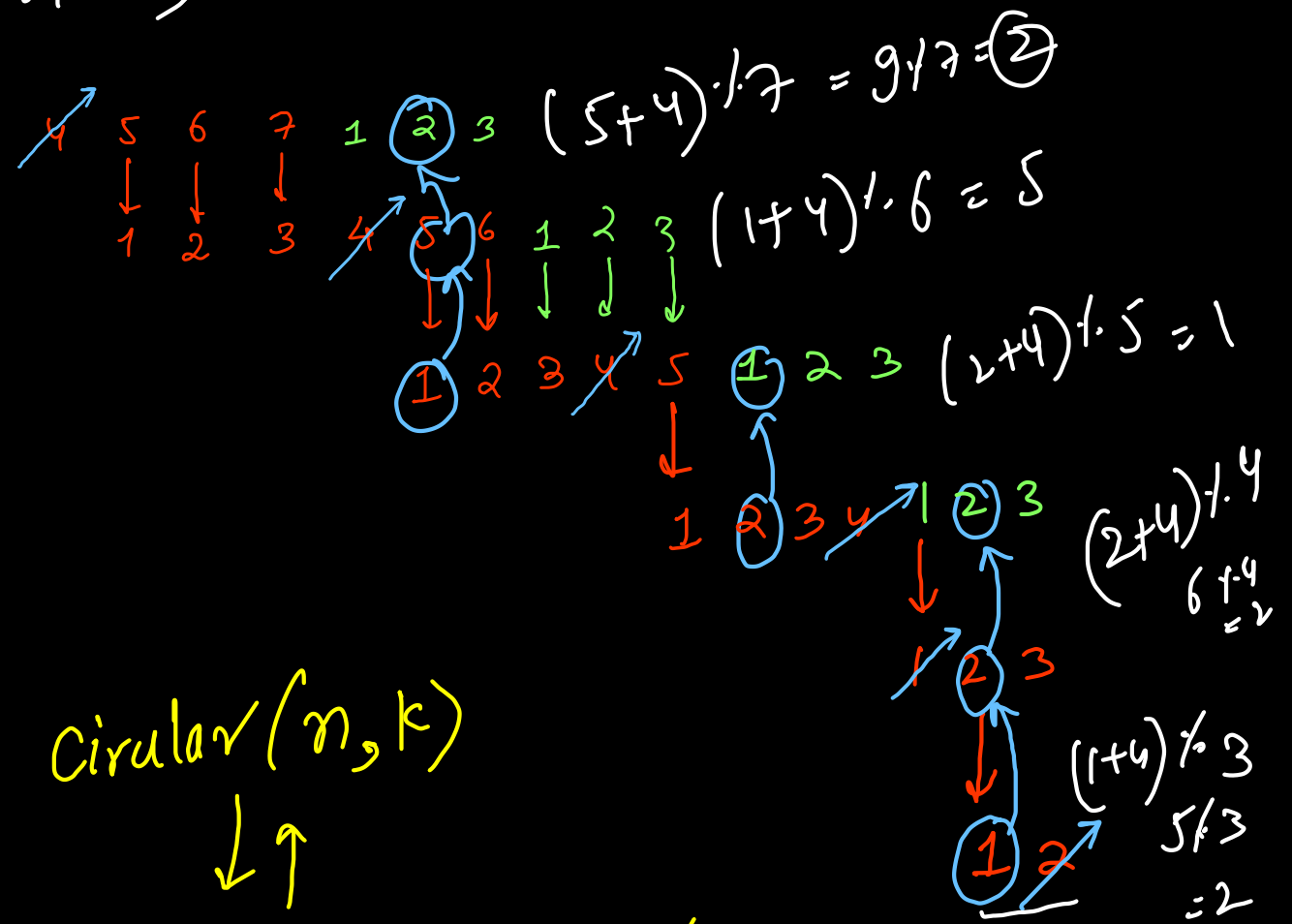
Calls = 4 depth = m×n

$O(4^{mn})$

Find winner of circular game



$n = 7,\ k = 4$




```java
public int circular(int n, int k){
    if(n == 1) return 0;
    return (circular(n - 1, k) + k) % n;
}
public int findTheWinner(int n, int k) {
    return circular(n, k) + 1;
}
```

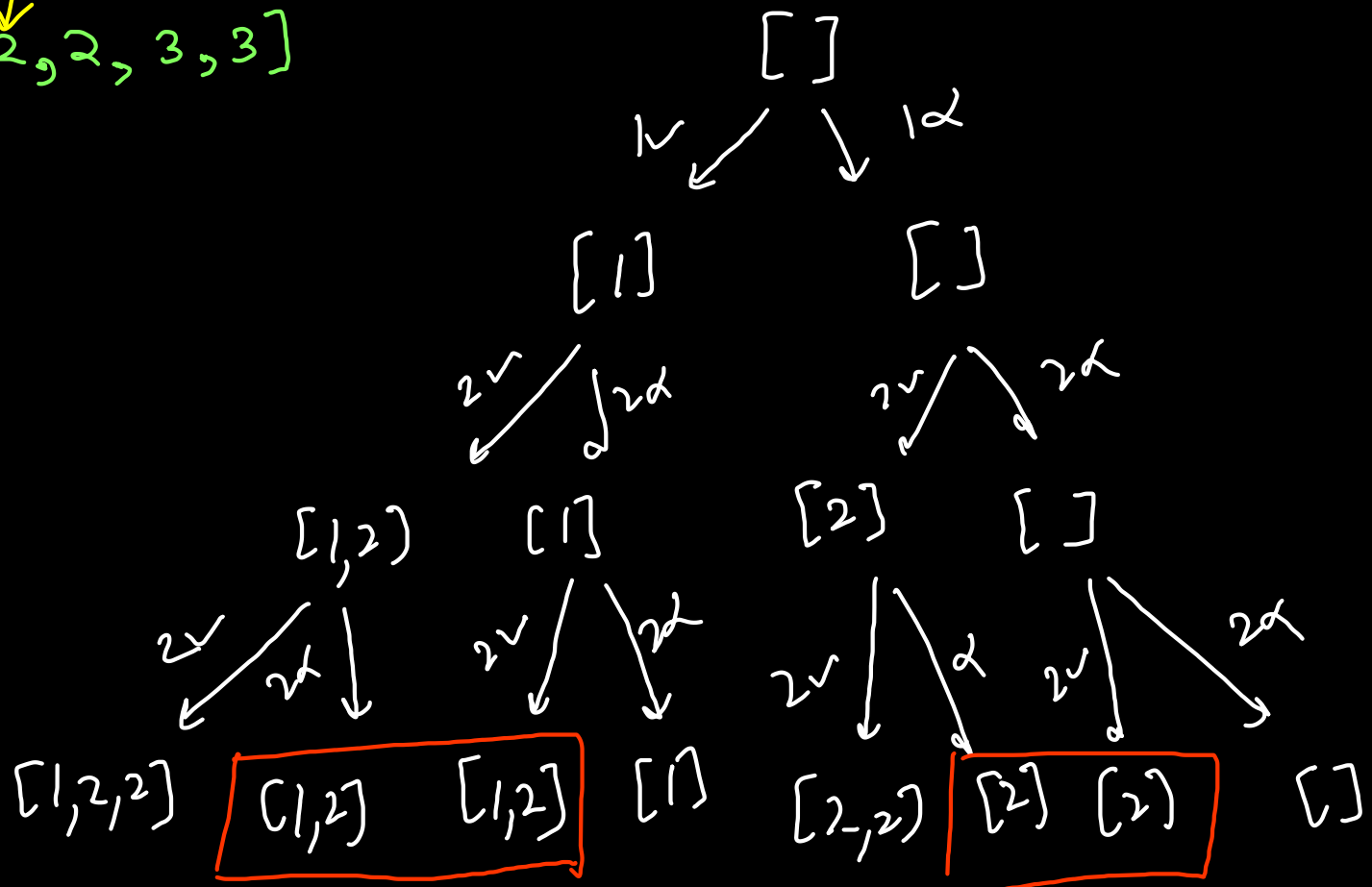
circular(n, k)

↓ ↑

(circular(n-1, k) + k) % n

Subsets - II 2c 90

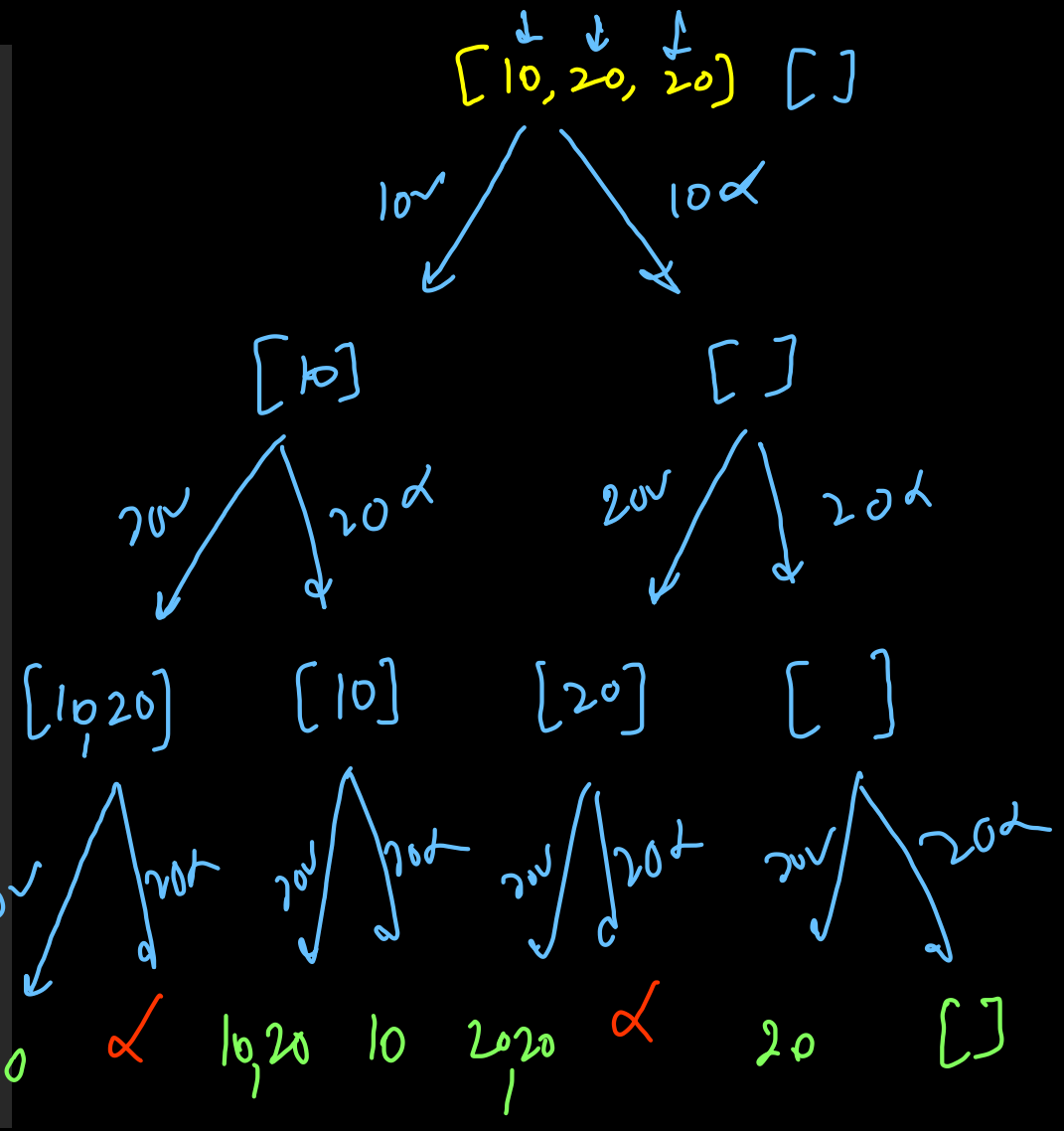
[1, 2, 2, 3, 3]



n('idx-1') == n('dw')

```java
class Solution {
    List<List<Integer>> subsets = new ArrayList<>();

    public void helper(int[] nums, int idx, List<Integer> subset){
        if(idx == nums.length){
            subsets.add(new ArrayList<>(subset)); // deep copy
            return;
        }

        // yes choice
        subset.add(nums[idx]);
        helper(nums, idx + 1, subset);
        subset.remove(subset.size() - 1); // backtracking

        // no choice
        if(subset.size() > 0 && subset.get(subset.size() - 1) == nums[idx])
            return;
        helper(nums, idx + 1, subset);
    }

    public List<List<Integer>> subsetsWithDup(int[] nums) {
        Arrays.sort(nums);
        helper(nums, 0, new ArrayList<>());
        return subsets;
    }
}
```

↳ duplicates ignore



```java
List<List<Integer>> ans = new ArrayList<>();

public void subsets(int lastItem, ArrayList<Integer> unique, HashMap<Integer, Integer> freq,
    ans.add(new ArrayList<>(res));

    for(int i=lastItem; i<unique.size(); i++){
        int val = unique.get(i);
        int oldFreq = freq.get(val);

        if(oldFreq > 0){
            freq.put(val, oldFreq - 1);
            res.add(val);

            subsets(i, unique, freq, res);

            res.remove(res.size() - 1);
            freq.put(val, oldFreq);
        }
    }
}
```

Approach 2)
Using hashmap
(without sorting)

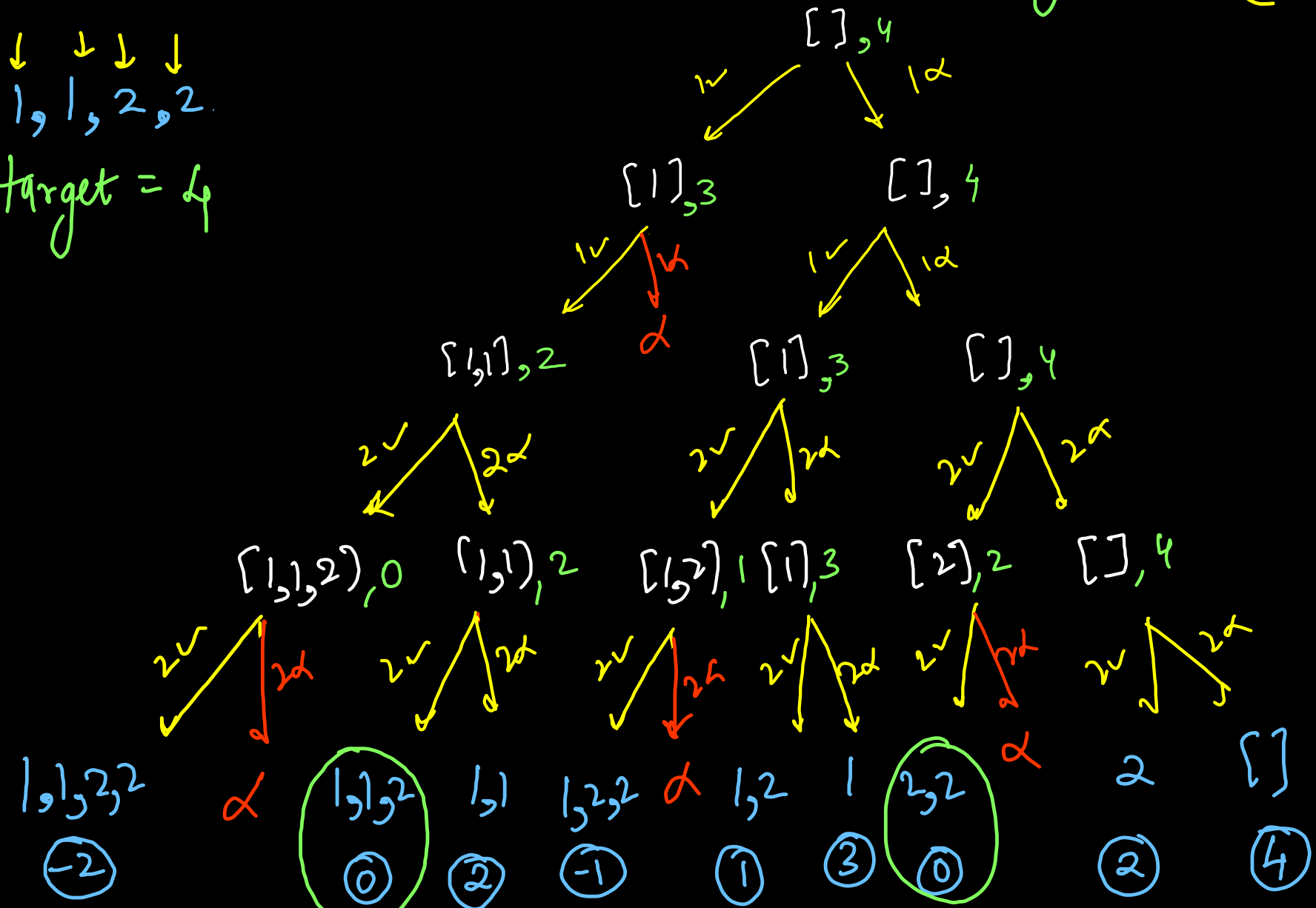
```java
public List<List<Integer>> subsetsWithDup(int[] nums) {
    ArrayList<Integer> unique = new ArrayList<>();
    HashMap<Integer, Integer> freq = new HashMap<>();

    for(int val: nums){
        if(freq.containsKey(val) == true){
            freq.put(val, freq.get(val) + 1);
        } else {
            freq.put(val, 1);
            unique.add(val);
        }
    }

    subsets(0, unique, freq, new ArrayList<>());
    return ans;
}
```

Combination Sum - II

Coin change (LC 40)

1, 1, 2, 2

target = 4

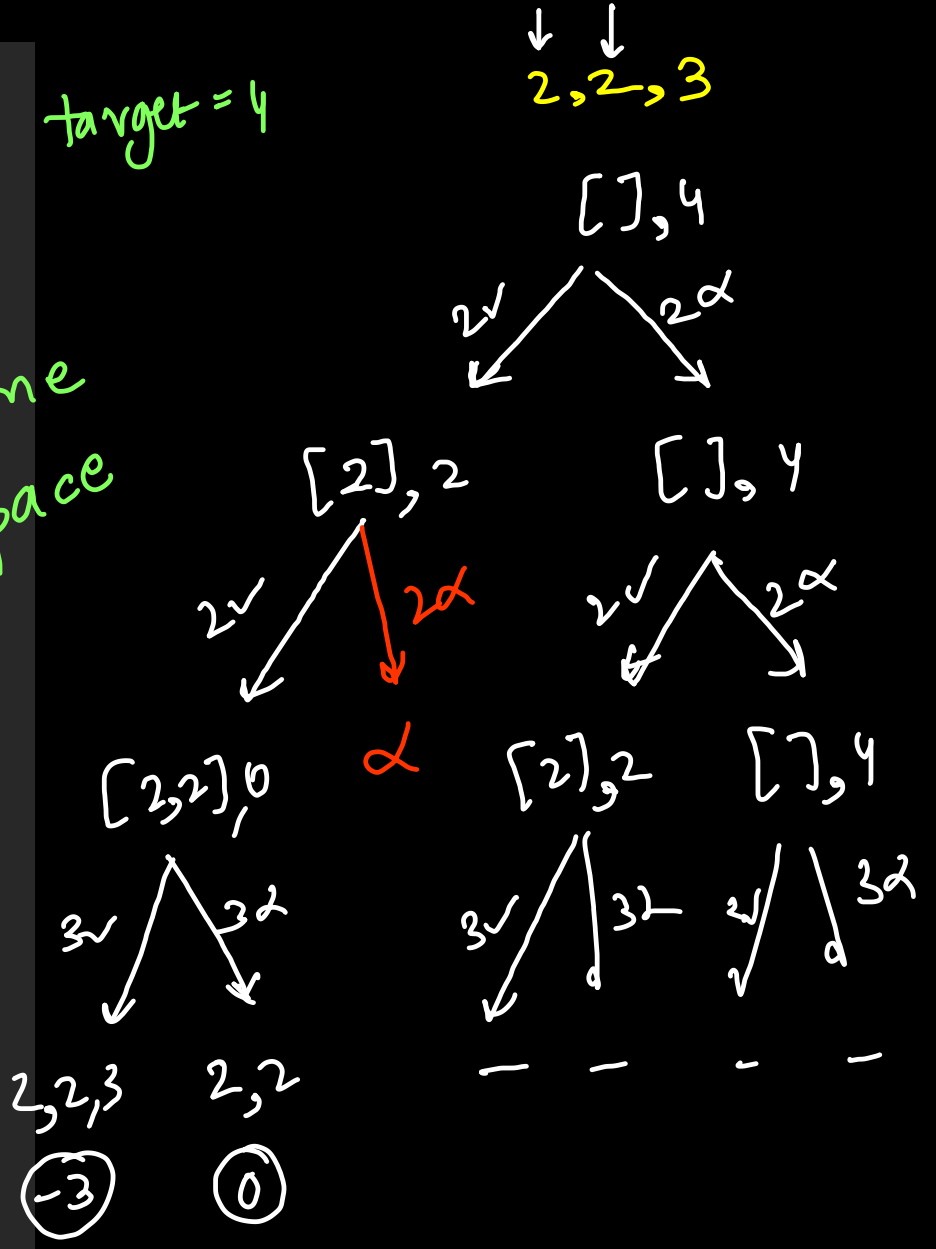


```java
class Solution {
    List<List<Integer>> subsets = new ArrayList<>();

    public void helper(int[] coins, int idx, int target, List<Integer> subset){
        if(target < 0) return; // pruning
        if(idx == coins.length){
            if(target == 0) subsets.add(new ArrayList<>(subset));
            return;
        }

        // yes choice
        subset.add(coins[idx]);
        helper(coins, idx + 1, target - coins[idx], subset);
        subset.remove(subset.size() - 1); // backtracking

        // no choice
        if(subset.size() > 0 && subset.get(subset.size() - 1) == coins[idx])
            return;
        helper(coins, idx + 1, target, subset);
    }

    public List<List<Integer>> combinationSum2(int[] coins, int target) {
        Arrays.sort(coins);
        helper(coins, 0, target, new ArrayList<>());
        return subsets;
    }
}
```

to avoid TLE

$O(2^N)$ time

$O(N)$ space



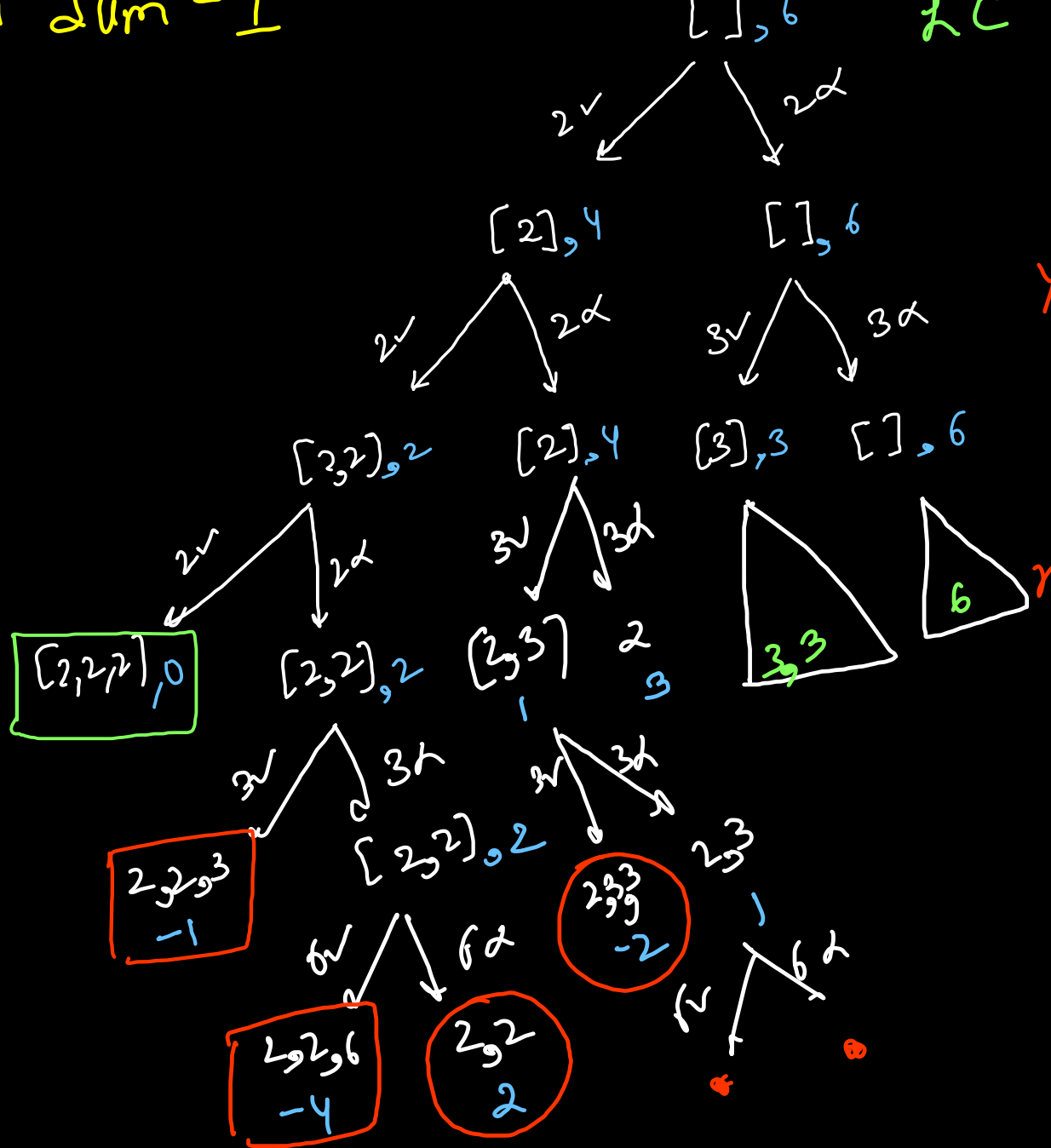
Combination Sum - I

LC 39

target = 6

{2, 3, 6}

infinite supply



yes → same index (multiple occurrences)

no → next element index choice

```java
class Solution {
    List<List<Integer>> subsets = new ArrayList<>();

    public void helper(int[] coins, int idx, int target, List<Integer> subset){
        if(target < 0) return; // pruning
        if(idx == coins.length){
            if(target == 0) subsets.add(new ArrayList<>(subset));
            return;
        }

        // yes choice
        subset.add(coins[idx]);
        helper(coins, idx, target - coins[idx], subset);
        subset.remove(subset.size() - 1); // backtracking

        // no choice
        helper(coins, idx + 1, target, subset);
    }
    public List<List<Integer>> combinationSum(int[] coins, int target) {
        helper(coins, 0, target, new ArrayList<>());
        return subsets;
    }
}
```

LC 39

Time
↳ Exponential

Space
↳ Linear

Leetcode 47 Permutations - II

{1, 2, 3}

{1, 2, 3}

{1, 3, 2}

{2, 1, 3}

{2, 3, 1}

{3, 1, 2}

{3, 2, 1}

{1, 2, 2}

{1, 2, 2}

{2, 1, 2}

{2, 2, 1}

{1, 1, 2, 2}

{1, 1, 2, 2}

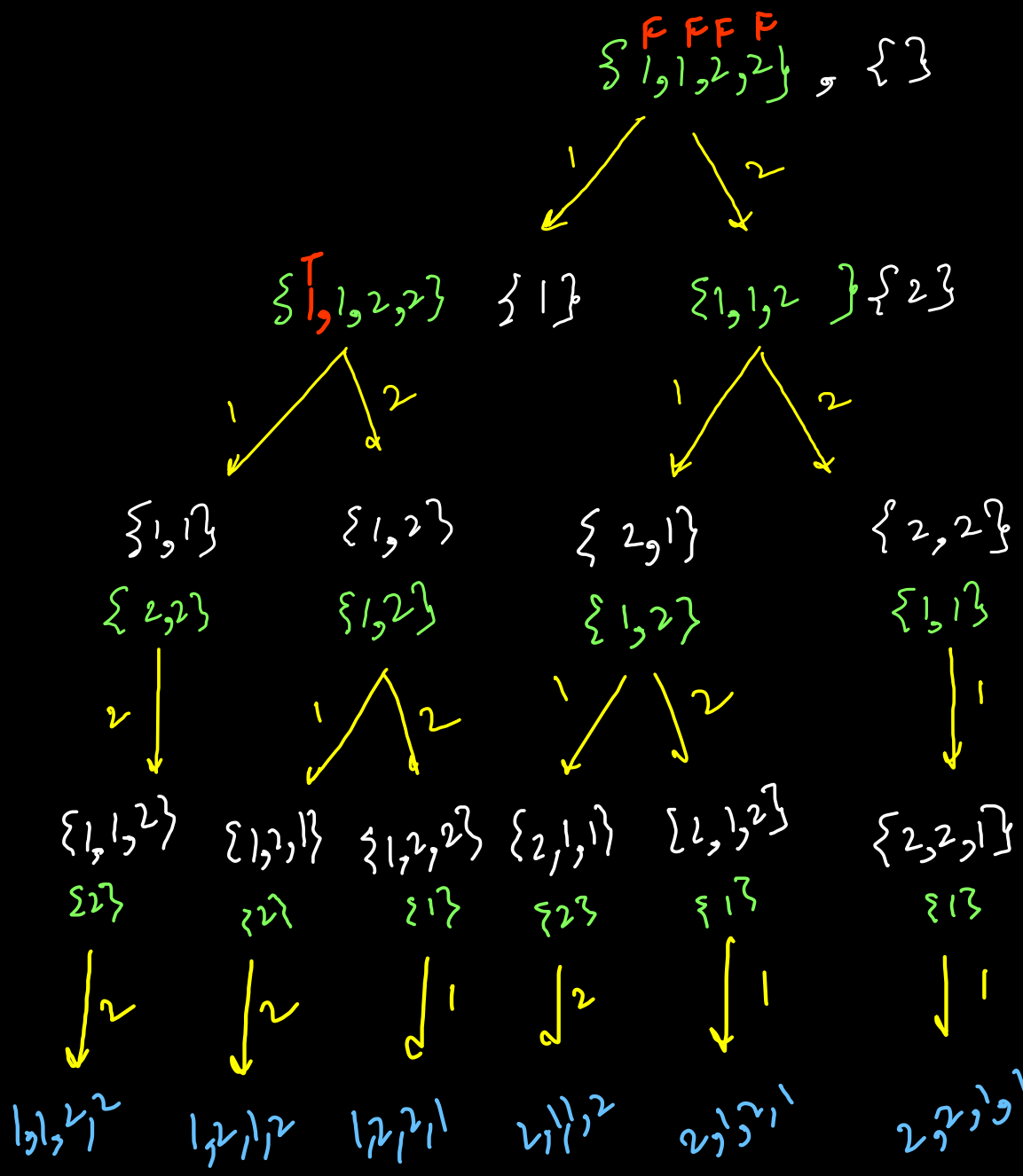
{1, 2, 1, 2}

{1, 2, 2, 1}

{2, 1, 1, 2}

{2, 1, 2, 1}

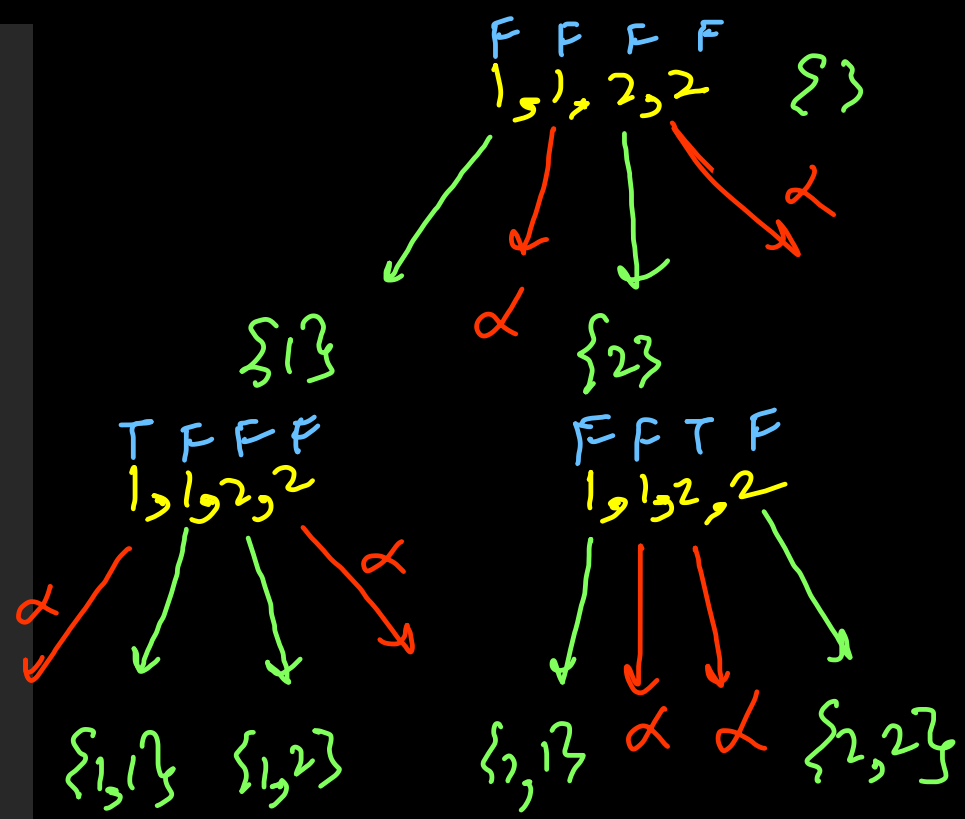
{2, 2, 1, 1}


F FF F
{1,1,2,2} , {}
1
2
T
{1,1,2,2} {1}
{1,1,2} {2}
1
2
1
2
{1,1} {1,2} {2,1} {2,2}
{2,2} {1,2} {1,2} {1,1}
2
1
2
1
2
{1,1,2} {1,2,1} {1,2,2} {2,1,1} {2,1,2} {2,2,1}
{2} {2} {1} {2} {1} {1}
2
2
1
2
1
1

```java
List<List<Integer>> subsets = new ArrayList<>();

public void helper(int[] nums, boolean[] vis, List<Integer> output){
    if(output.size() == nums.length){
        subsets.add(new ArrayList<>(output));
        return;
    }

    for(int idx = 0; idx < nums.length; idx++){
        if(vis[idx] == true) continue;
        if(idx > 0 && vis[idx - 1] == false && nums[idx] == nums[idx - 1])
            continue;

        vis[idx] = true;
        output.add(nums[idx]);

        helper(nums, vis, output);

        vis[idx] = false;
        output.remove(output.size() - 1);
    }
}
```

→ to avoid duplicacy

→ backtracking




```java
public List<List<Integer>> permuteUnique(int[] nums) {
    Arrays.sort(nums);
    boolean[] vis = new boolean[nums.length];
    helper(nums, vis, new ArrayList<>());
    return subsets;
}
```

Time ⇒ $O(N!)$

Space ⇒ $O(N)$

Letter Tile Possibilities

LC 1079

Subsets + permutations

"AAB"

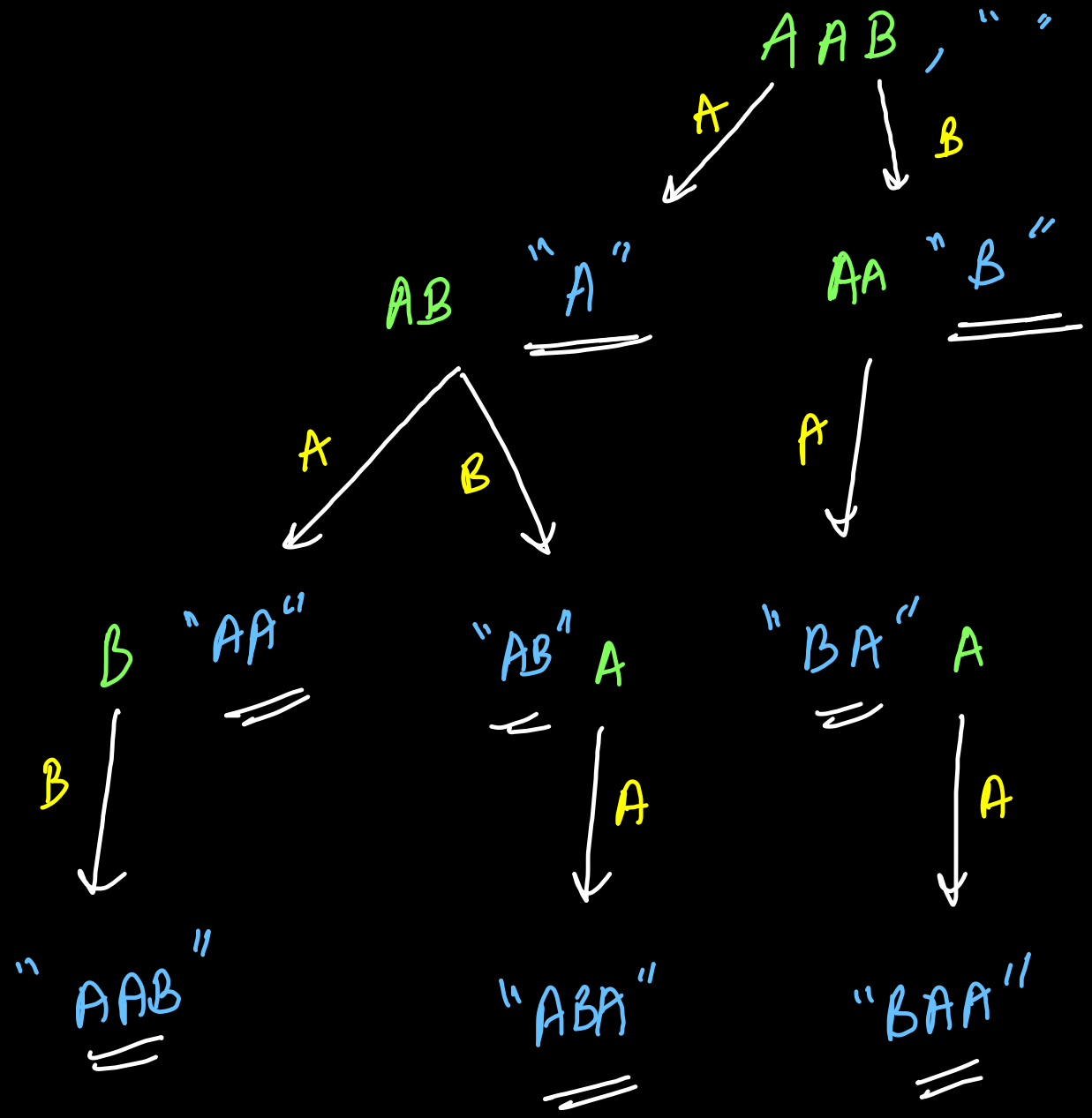
A → A 1

B → B 1

AA → AA 1

AB → AB, BA 2

AAB → AAB, ABA, BAA 3


AAB , " "
A
B
AB "A"
AA "B"
A
B
A
B
B "AA"
"AB" A
"BA" A
B
A
A
"AAB"
"ABA"
"BAA"

```java
List<String> subsets = new ArrayList<>();

public void helper(String nums, boolean[] vis, String output){
    subsets.add(output);
    if(output.length() == nums.length())
        return;

    for(int idx = 0; idx < nums.length(); idx++){
        if(vis[idx] == true) continue;
        if(idx > 0 && vis[idx - 1] == false && nums.charAt(idx) == nums.charAt(idx - 1))
            continue;

        vis[idx] = true;
        helper(nums, vis, output + nums.charAt(idx));
        vis[idx] = false;
    }
}
```

Subsets + permutations

2 → Permutations I & II

```java
public int numTilePossibilities(String str) {
    char[] chars = str.toCharArray();
    Arrays.sort(chars);
    str = new String(chars);

    helper(str, new boolean[str.length()], "");
    System.out.println(subsets);
    return subsets.size() - 1;
}
```

Sort & String

to ignore empty subset