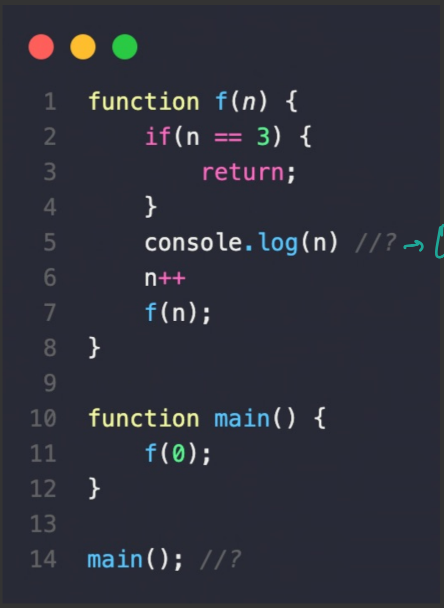
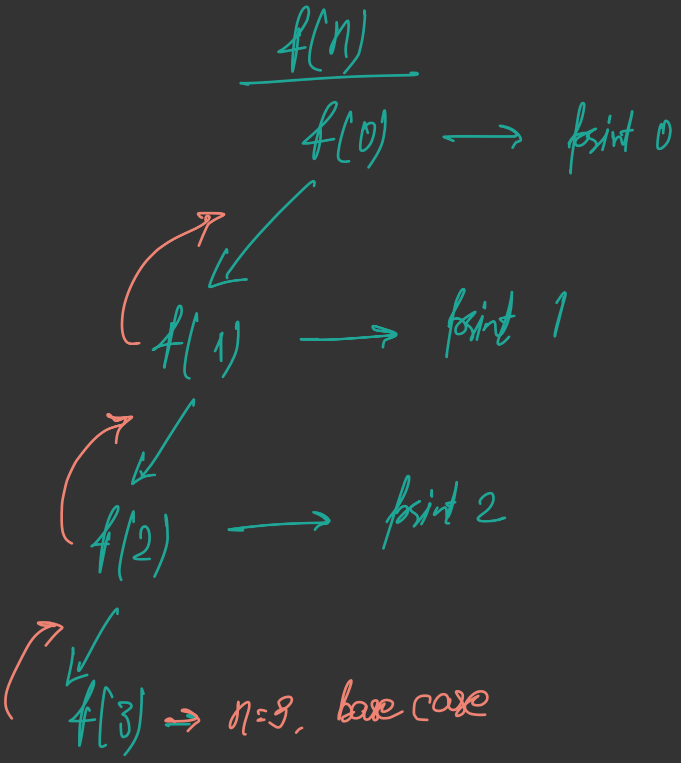
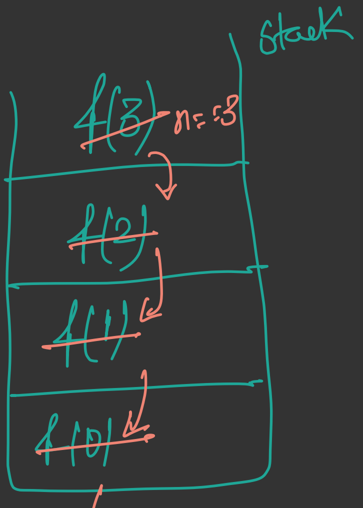
Recursion → when a function calls itself until a specified cond is met



```
function f(n) {
    if(n == 3) {
        return;
    }
    console.log(n) //? → 0,1,2
    n++
    f(n);
}

function main() {
    f(0);
}

main(); //?
```





control goes back to main f^n

lec2 Problems on recursion

→ print name 5times

→ print linearly from 1 to N

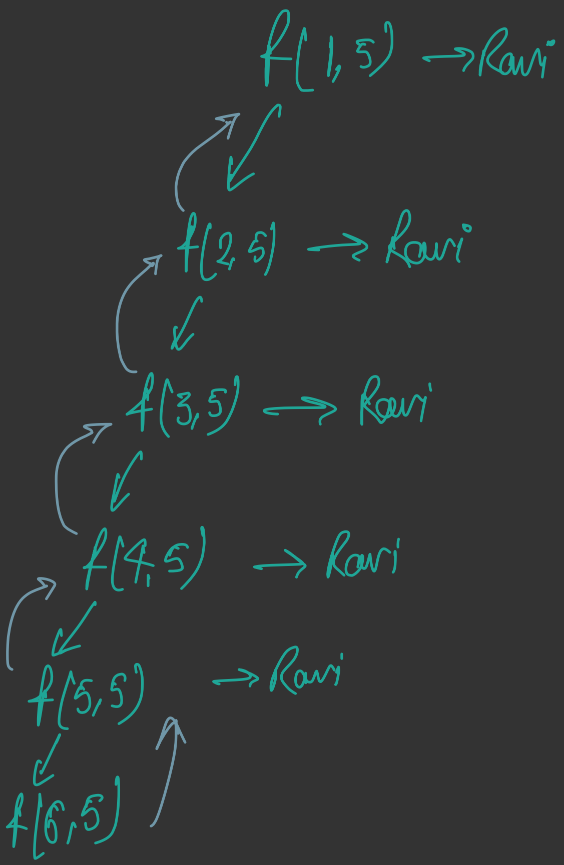
→ print from N to 1

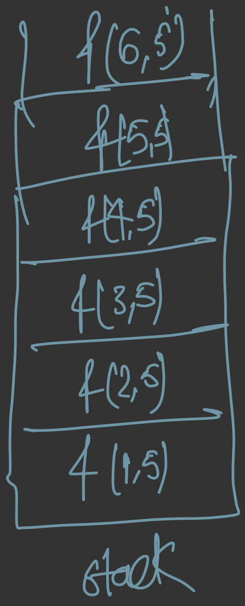
→ print linearly from 1 to N (but by backtracking)

→ print from N to 1 (by backtrack)

print name 5 times.

```
function main(){
    n=5
    f(1,n)
  }
function f(i,n){
    if (i>n)
        return;
    printf("Ravi");
    f(i+1, n);
 }
```



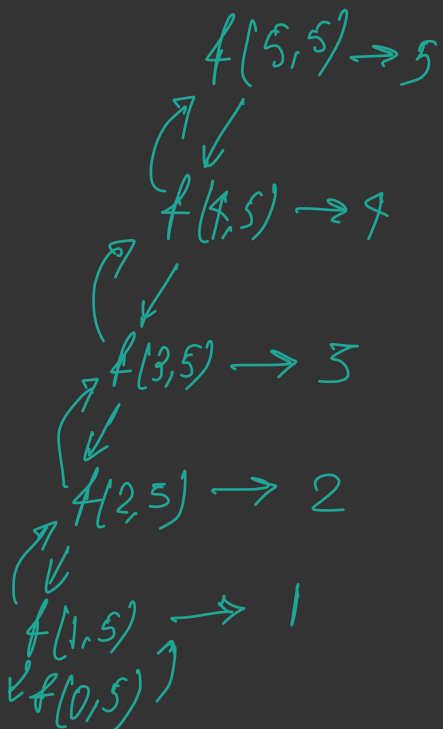


Print 1 to N

```
main(){
        n=5
   f(1,n)

  }
f(i,n){
   if(i>n) return;
   print(i)
   f(i+1,n)
 }
```

print N to1

```
main(){
   n=5
   f(n,n)
 }
f(i,n){
   if(i<1) return
   print(i)
   f(i-1, 5)}
```

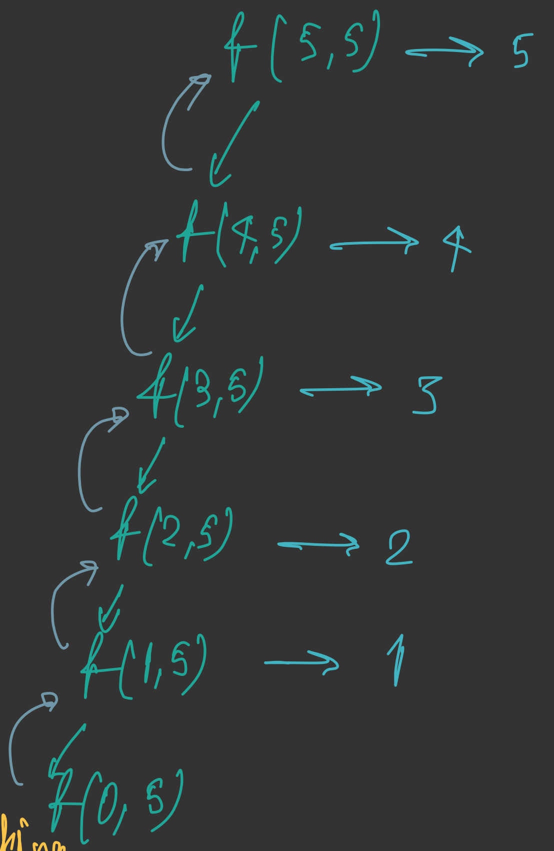


print linearly from 1 to N backtracking:

do not use i+1,
but can use i-1

recursion first then action

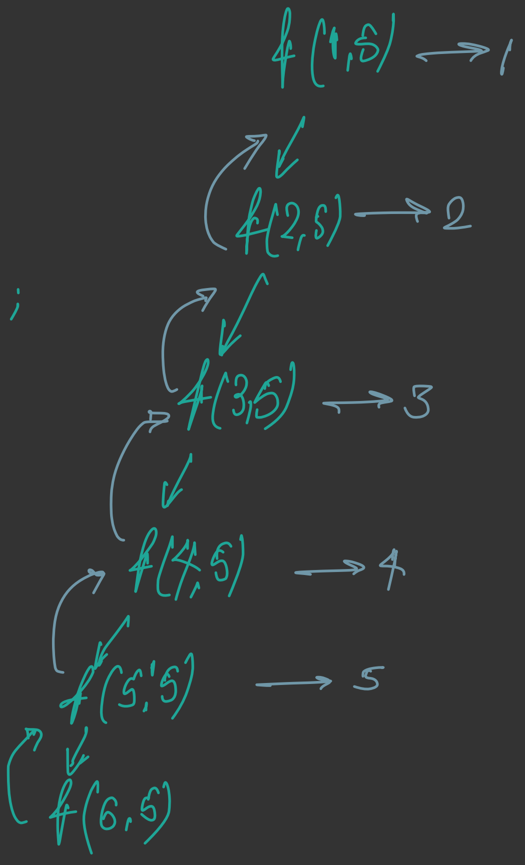
```
main() {
   n=5
   f(n,n)
}
f(i,n){
  if(i<1) return;
  f(i-1,n);
   print(i)
}
```



1, 2, 3, 4, 5

print N to 1 using backtracking:

```
main(){
  n=5
  f(1,n)
}
f(i,n){
   if(i>n) return;
  f(i+1,n)
   print(i)
}
```



5 4 3 2 1

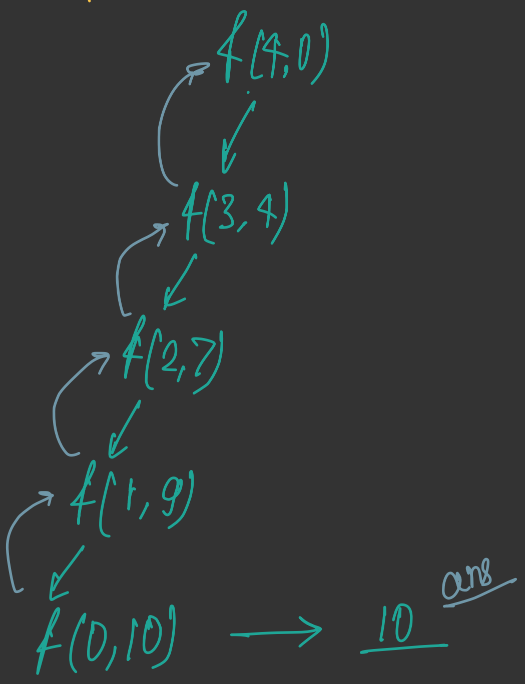
lec3 Parameterised and functional recursion

Q. sum of first N numbers.

parameter

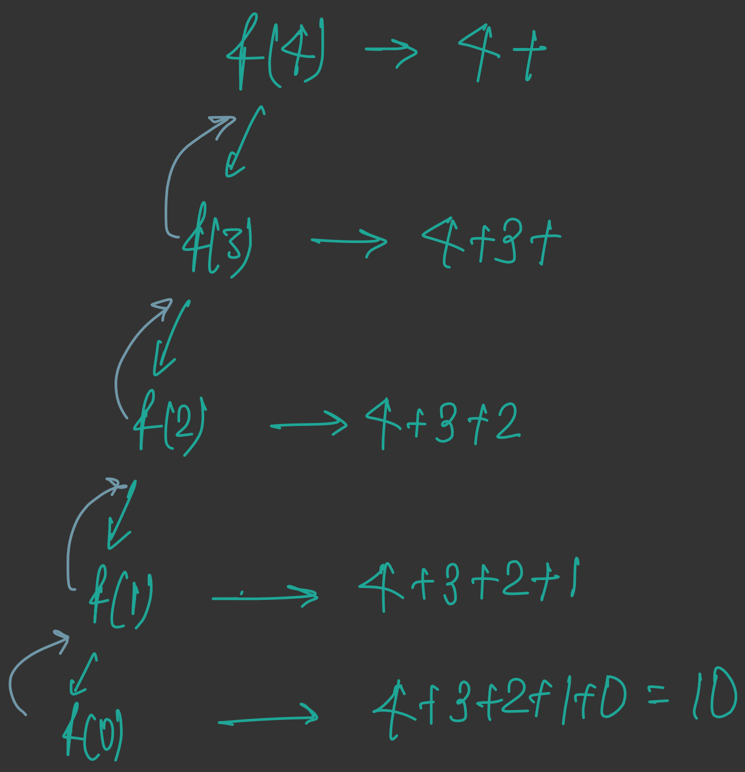
functional

```
main() {
  n = 4
  f(n, 0)
}
f(i, sum) {
   if( i<1 ){
    print(sum)
     return;
   }
  f(i-1, sum+i)
}
```



functional

```
  main() {
    n=4
    f(n)
   }
f(i) {
  if (i==0) return 0;
 return  i + f(i-1)
 }
```



TC → O(N)

SC → O(N)

↑ stack

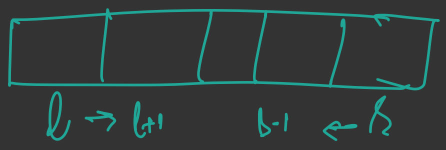
lec4. problems on functional recursion

Q. reverse an array

$a = 1, 2, 3, 4, 2$ (i under 1, j under the last 2)   swap (i, j) till $n/2$

$a = 2, 4, 3, 2, 1$

```
f(0, 4) → swap (a[0], a[4])
f(1, 3) → swap (a[1], a[3])
f(2, 2) → return
```



```
f(l, r) {
    if( l>=r) return;
    swap (a[l], a[r])
    f(l+1, r-1)
}
main() {
  l=0
  r= n-1
  f(l,r)
}
```

using one parameter

l , $r = n-1-l$

if ($l \geq n/2$) return;

```
f(i) {
  if (i ≥ n/2) return;
 swap (a[i], a[n-i-1])
  f(i+1);
}
main() {
  arr;
  f(0);
}
```

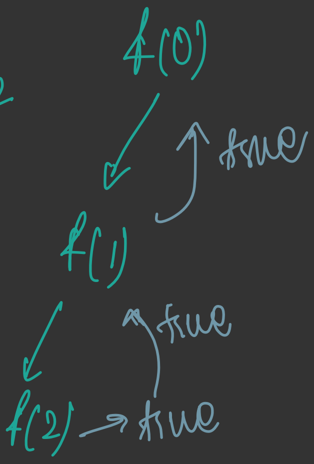
```
f(0) → swap (a[0], a[4])
f(1) → swap (a[1], a[3])
f(2) → return
```

$n/2 = 5/2 = 2$

Q check if given string is palindrome

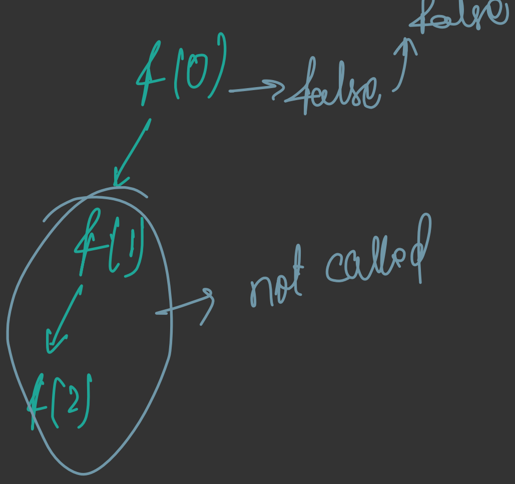
```
main() {
   s = " cavac"
   f(0)
}
f(i) {
   if (i ≥ n/2) return true;
   if(s[i] != s[n-1-i])
          return false
   return f(i+1)
}
```

$n/2 = 5/2 = 2$



cool

$4/2 = 2$



TC -> $O(N/2)$

SC -> $O(N/2)$

lec5 -> multiple recursion calls.

Q. fibonacci numbers

0 1 1 2 3 5 8 13 - - -

0th 1st 2nd 3 4 5

N -> f(n) -> nth fibonacci num

f(3) -> 3rd -> 2

f(2) -> 2nd -> 1

f(4) -> 4th -> 3

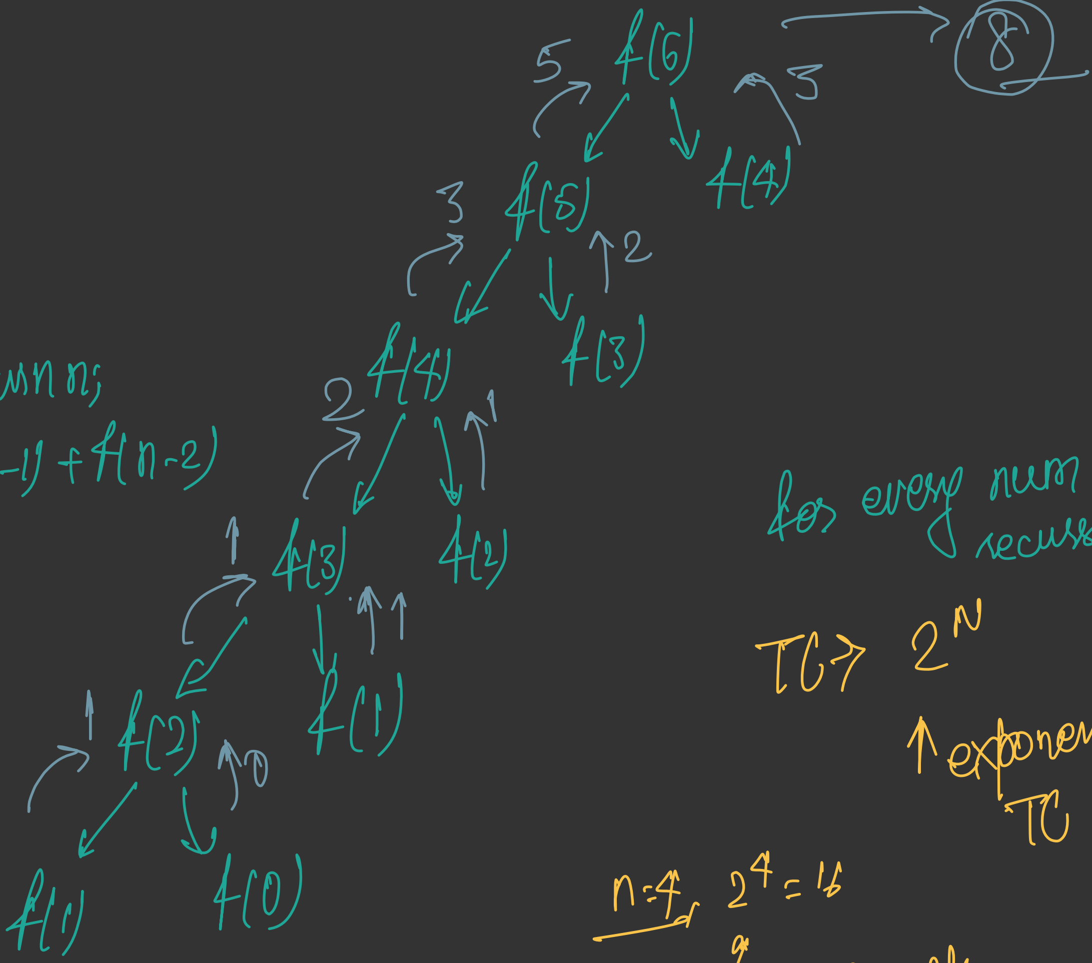
$f(5) \rightarrow f(4) + f(3) = 3+2 = 5$

$f(0) = 0$

$f(1) = 0$

$f(2) = f(0) + f(1) = 0+1 = 1$

```
main() {
  f(6)
}

f(n) {
  if(n≤1) return n;
  return f(n-1)+f(n-2)
}
```



for every num 2 recursion call

TC → 2^N

↑ exponential TC

n=4, $2^4=16$

actual 9 calls

so it's exactly 2^N but it's exponential

---

lec6. Recursion on subsequences

→ a contigous / non-contigous sequence, which follows the order of array

[3, 1, 2]

→ [3, 2]

[1, 2]

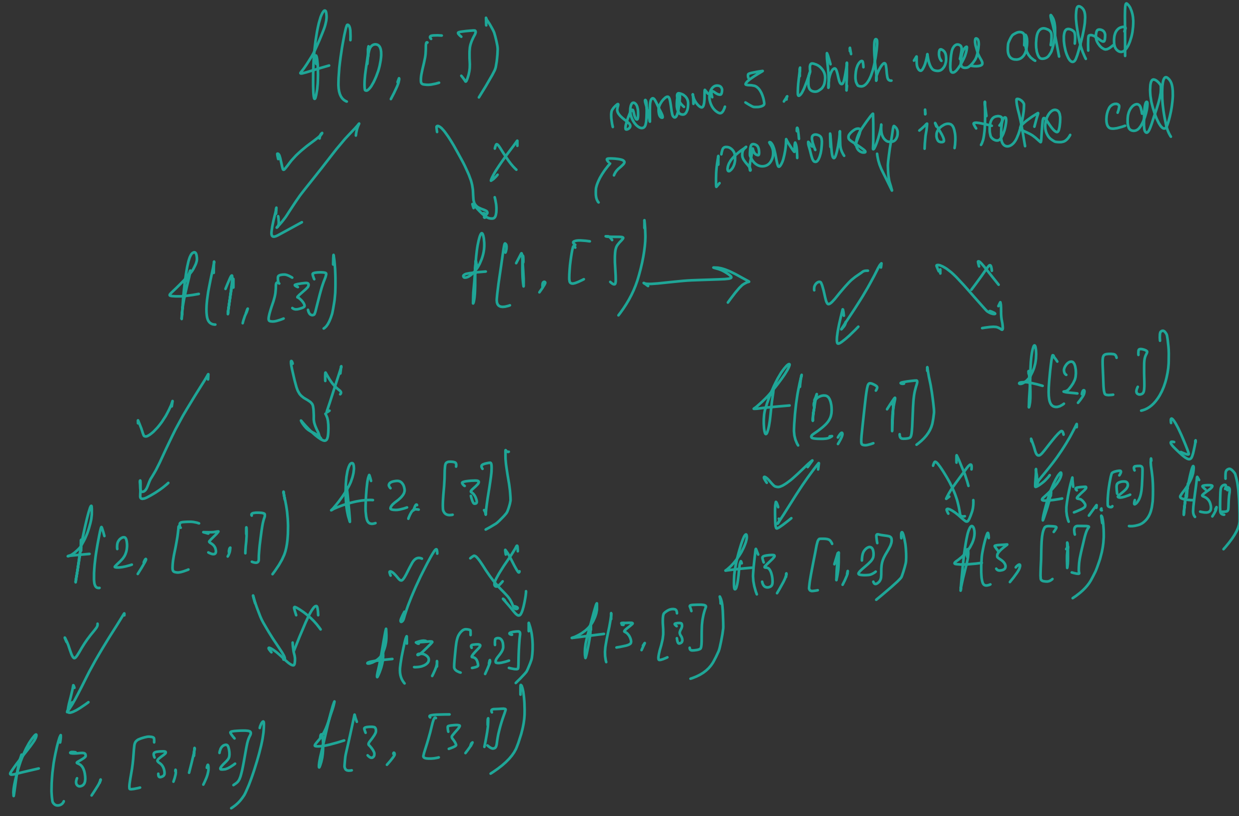
[ ]

Q. print all subsequences.

take / not Take

```
f(ind, [ ])
{  if(ind ≥ n)
        print([ ])
        return;
   [ ].add(arr[i]);
   f(ind+1, [ ]);  → take
   [ ].remove(arr[i]);
   f(ind+1, [ ])   → not take
}
```

$\boxed{[3,1,2]}$

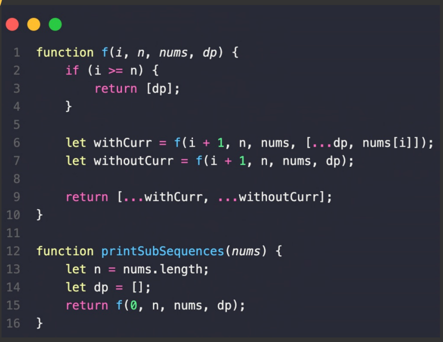


remove 3, which was added previously in take call

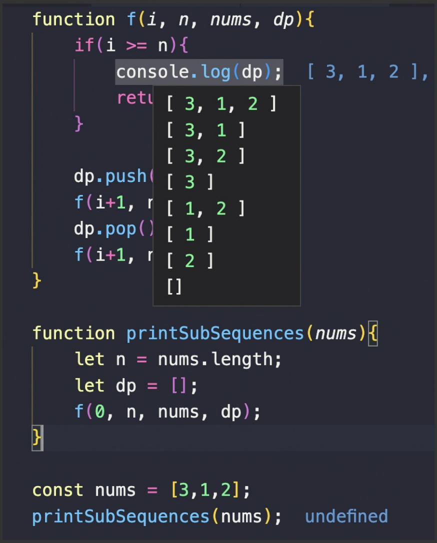
[3,1,2], [3,1], [3,2], [3], [1,2], [1], [2], [ ]

TC → $O(2^N \times N)$

SC → $O(N)$

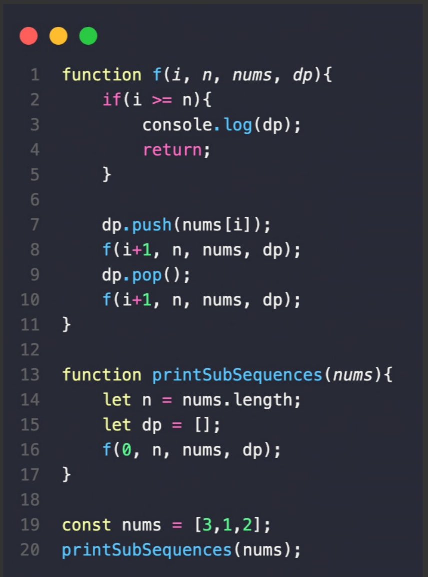


```
function f(i, n, nums, dp) {
    if (i >= n) {
        return [dp];
    }

    let withCurr = f(i + 1, n, nums, [...dp, nums[i]]);
    let withoutCurr = f(i + 1, n, nums, dp);

    return [...withCurr, ...withoutCurr];
}

function printSubSequences(nums) {
    let n = nums.length;
    let dp = [];
    return f(0, n, nums, dp);
}
```



```
function f(i, n, nums, dp){
    if(i >= n){
        console.log(dp);  [ 3, 1, 2 ],
        retu
    }

    dp.push(
    f(i+1, r
    dp.pop()
    f(i+1, r
}

function printSubSequences(nums){
    let n = nums.length;
    let dp = [];
    f(0, n, nums, dp);
}

const nums = [3,1,2];
printSubSequences(nums);  undefined
```

```
[ 3, 1, 2 ]
[ 3, 1 ]
[ 3, 2 ]
[ 3 ]
[ 1, 2 ]
[ 1 ]
[ 2 ]
[]
```



```
function f(i, n, nums, dp){
    if(i >= n){
        console.log(dp);
        return;
    }

    dp.push(nums[i]);
    f(i+1, n, nums, dp);
    dp.pop();
    f(i+1, n, nums, dp);
}

function printSubSequences(nums){
    let n = nums.length;
    let dp = [];
    f(0, n, nums, dp);
}

const nums = [3,1,2];
printSubSequences(nums);
```

lec 7 → all kind of pattesns in secussion

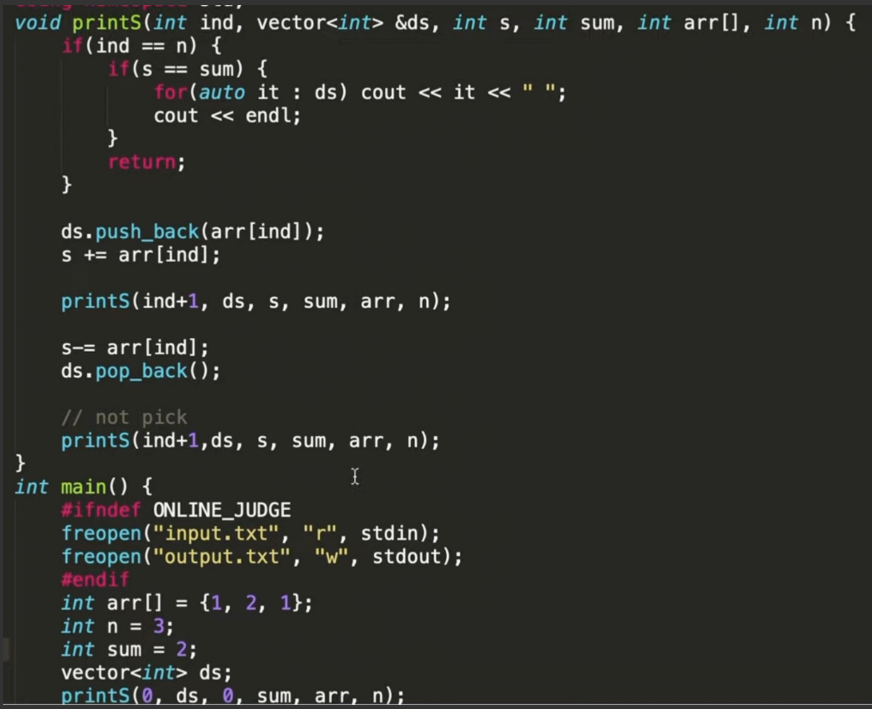
→ take & not-take

→ pick & not-pick

Q. find subassay which sum is k

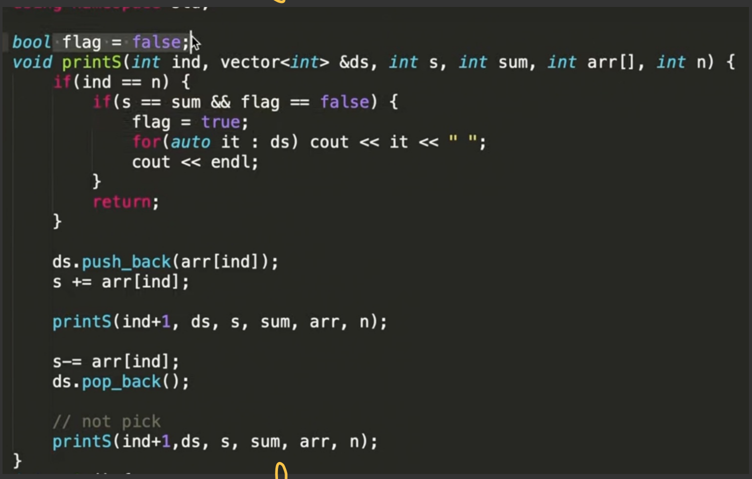
[1, 2, 1] , k = 3

```
f(i, ds, s ){
    if(i==n){
        if(s==sum)
            print(ds)
        setusn
    }
    ds.add(ass[i])
    s+=ass[i]
    f(i+1, ds, s)  → pick
    ds.semove(ass[i])
    s -= ass[i]
    f(i+1,ds,s)  → not pick
}
```



```
void printS(int ind, vector<int> &ds, int s, int sum, int arr[], int n) {
    if(ind == n) {
        if(s == sum) {
            for(auto it : ds) cout << it << " ";
            cout << endl;
        }
        return;
    }

    ds.push_back(arr[ind]);
    s += arr[ind];

    printS(ind+1, ds, s, sum, arr, n);

    s-= arr[ind];
    ds.pop_back();

    // not pick
    printS(ind+1,ds, s, sum, arr, n);
}
int main() {
    #ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    #endif
    int arr[] = {1, 2, 1};
    int n = 3;
    int sum = 2;
    vector<int> ds;
    printS(0, ds, 0, sum, arr, n);
```

Q. print any one sub-sequence whose sum is k



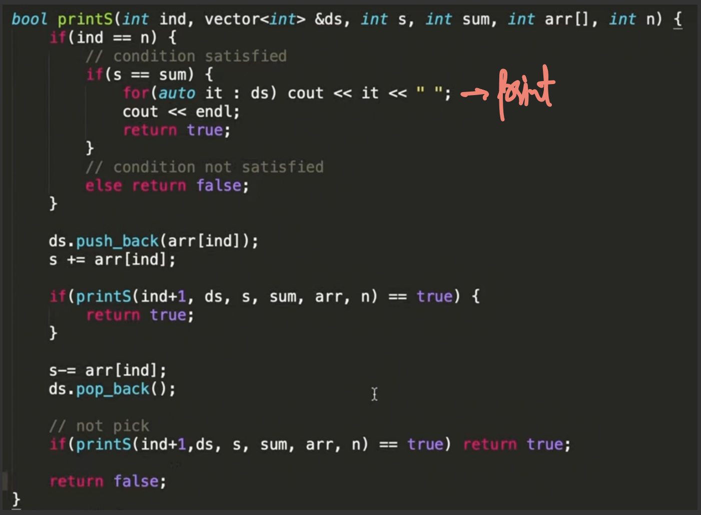
```
bool flag = false;
void printS(int ind, vector<int> &ds, int s, int sum, int arr[], int n) {
    if(ind == n) {
        if(s == sum && flag == false) {
            flag = true;
            for(auto it : ds) cout << it << " ";
            cout << endl;
        }
        return;
    }

    ds.push_back(arr[ind]);
    s += arr[ind];

    printS(ind+1, ds, s, sum, arr, n);

    s-= arr[ind];
    ds.pop_back();

    // not pick
    printS(ind+1,ds, s, sum, arr, n);
}
```

not-prefersed

⇒ technique to print only one answes

in base case if condition satisfied setusn tsue

```
base case
    cond → satisfied
        setusn tsue
    else
        setusn false

if( f() == tsue)
    setusn tsue
f(n)
    → setusn false
```



```cpp
bool printS(int ind, vector<int> &ds, int s, int sum, int arr[], int n) {
    if(ind == n) {
        // condition satisfied
        if(s == sum) {
            for(auto it : ds) cout << it << " ";
            cout << endl;
            return true;
        }
        // condition not satisfied
        else return false;
    }

    ds.push_back(arr[ind]);
    s += arr[ind];

    if(printS(ind+1, ds, s, sum, arr, n) == true) {
        return true;
    }

    s-= arr[ind];
    ds.pop_back();

    // not pick
    if(printS(ind+1,ds, s, sum, arr, n) == true) return true;

    return false;
}
```

→ print

✓ TC →

Q. count subsequences with sum k

[1, 2, 1]   k = 2

count

base case

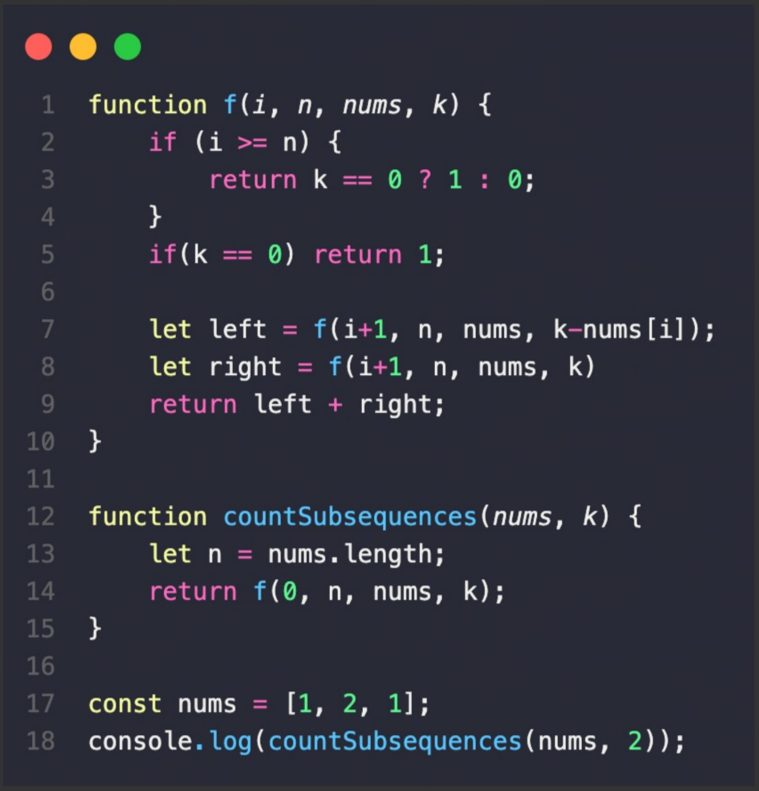
return 1 → condⁿ satisfied

return 0 → condⁿ not satisfied

left = f()

right = f()

return (left + right)



```javascript
function f(i, n, nums, k) {
    if (i >= n) {
        return k == 0 ? 1 : 0;
    }
    if(k == 0) return 1;

    let left = f(i+1, n, nums, k-nums[i]);
    let right = f(i+1, n, nums, k)
    return left + right;
}

function countSubsequences(nums, k) {
    let n = nums.length;
    return f(0, n, nums, k);
}

const nums = [1, 2, 1];
console.log(countSubsequences(nums, 2));
```

TC → $O(2^N)$

SC → O

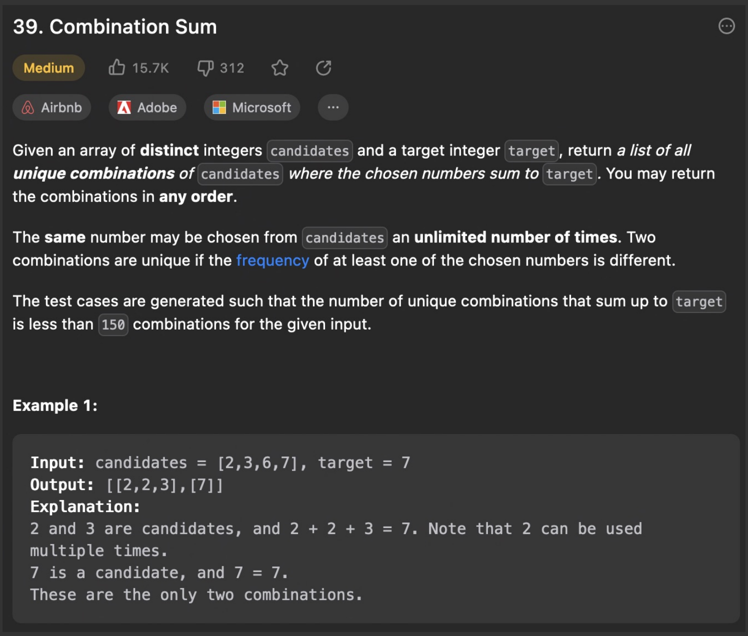
lec8. merge sort → already completed in merge sort notes

lec9. quick sort → already completed in sorting notes

lec10. combination sum

pick, not pick
& infinite supplies

$a = [2, 3, 6, 7]$, $k = 7$



39. Combination Sum

Medium 15.7K 312

Airbnb Adobe Microsoft ···

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `candidates` *where the chosen numbers sum to* `target`*.* You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used
multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

f(ind, target, ds)

✓ ds.add(a[ind])
f(ind, target - a[ind], ds)

✗ ds.pop(a[i])
f(ind+1, target, ds)

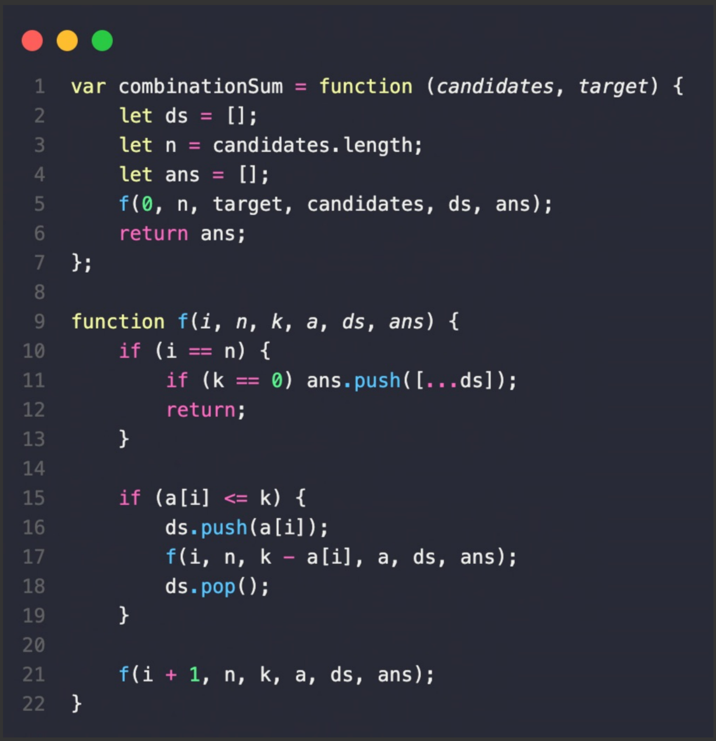
if(a[ind] <= target)

TC → $2^t \times k$ exponential

SC → $k \times x$

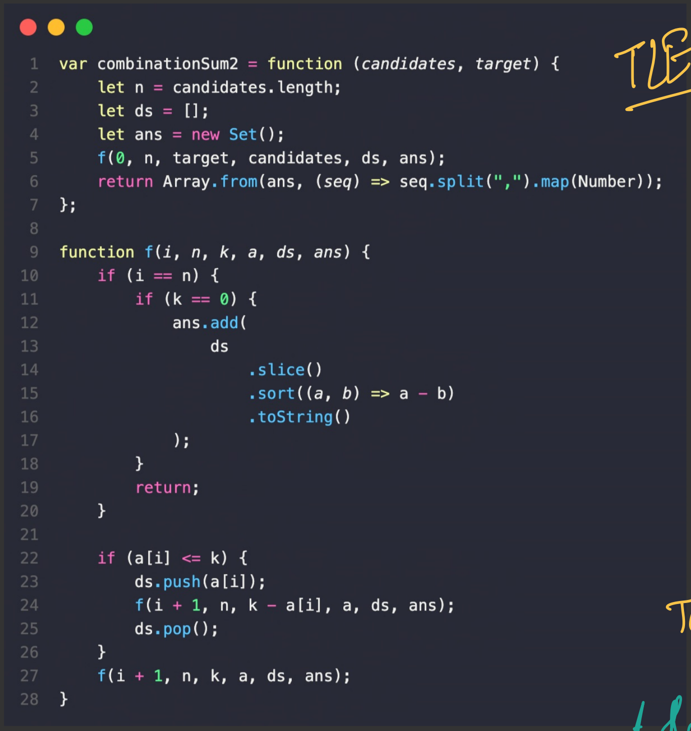
base case

```
if (ind == n)
    if (target == 0)  log(ds)
    else return
```



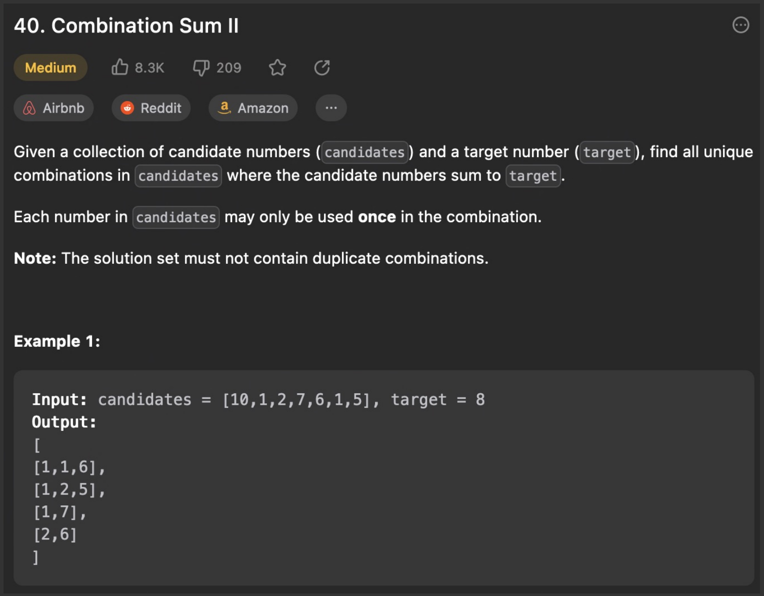
```
var combinationSum = function (candidates, target) {
    let ds = [];
    let n = candidates.length;
    let ans = [];
    f(0, n, target, candidates, ds, ans);
    return ans;
};

function f(i, n, k, a, ds, ans) {
    if (i == n) {
        if (k == 0) ans.push([...ds]);
        return;
    }

    if (a[i] <= k) {
        ds.push(a[i]);
        f(i, n, k - a[i], a, ds, ans);
        ds.pop();
    }

    f(i + 1, n, k, a, ds, ans);
}
```

lec 11   combination sum 2



```javascript
var combinationSum2 = function (candidates, target) {
    let n = candidates.length;
    let ds = [];
    let ans = new Set();
    f(0, n, target, candidates, ds, ans);
    return Array.from(ans, (seq) => seq.split(",").map(Number));
};

function f(i, n, k, a, ds, ans) {
    if (i == n) {
        if (k == 0) {
            ans.add(
                ds
                    .slice()
                    .sort((a, b) => a - b)
                    .toString()
            );
        }
        return;
    }

    if (a[i] <= k) {
        ds.push(a[i]);
        f(i + 1, n, k - a[i], a, ds, ans);
        ds.pop();
    }
    f(i + 1, n, k, a, ds, ans);
}
```

TLE



40. Combination Sum II

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sum to `target`.

Each number in `candidates` may only be used **once** in the combination.

Note: The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]
```

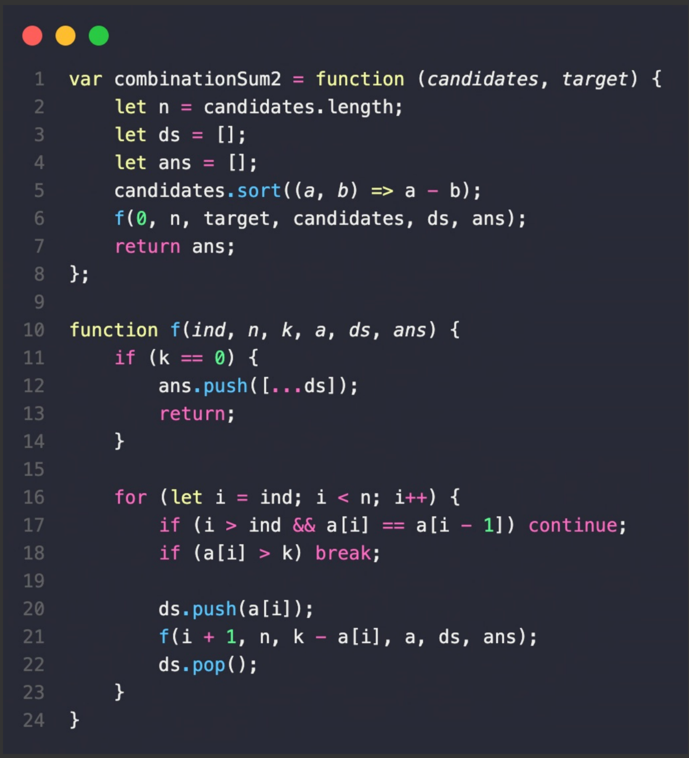
TC => $O(2^t \times k \log n)$ → set

sort the given array

avg length of subsequence

TC → $2^N \times k$

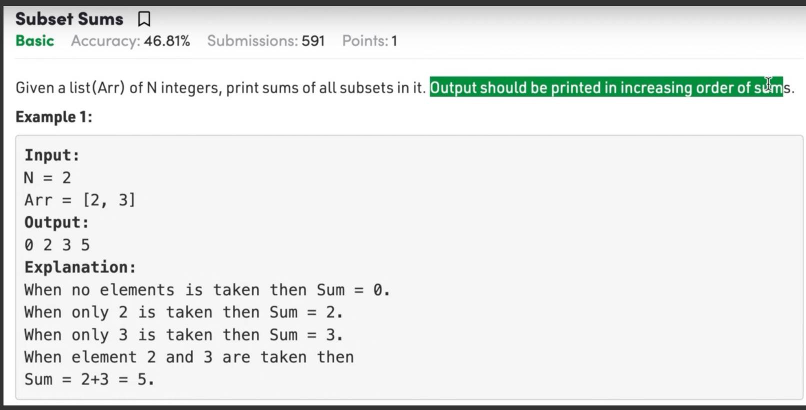
```
f(ind, target, ds, ans) {
    if(target = =0)
        ans.push(ds)
        return
    for ( let i = ind; i<n ; i++) {
        if( i >ind && arr[i] = = arr[i-1]) continue
        if (arr[i] >target) break;
        ds. push(arr[i])
        f(i+1, arr, target - arr[i], ans, ds);
        ds. pop()
    }
}
```



```javascript
var combinationSum2 = function (candidates, target) {
    let n = candidates.length;
    let ds = [];
    let ans = [];
    candidates.sort((a, b) => a - b);
    f(0, n, target, candidates, ds, ans);
    return ans;
};

function f(ind, n, k, a, ds, ans) {
    if (k == 0) {
        ans.push([...ds]);
        return;
    }

    for (let i = ind; i < n; i++) {
        if (i > ind && a[i] == a[i - 1]) continue;
        if (a[i] > k) break;

        ds.push(a[i]);
        f(i + 1, n, k - a[i], a, ds, ans);
        ds.pop();
    }
}
```

lec12. subset sum. n=3

num of subsets = $2^n \Rightarrow 2^3 = 8$



Subset Sums

Basic Accuracy: 46.81% Submissions: 591 Points: 1

Given a list(Arr) of N integers, print sums of all subsets in it. Output should be printed in increasing order of sums.

Example 1:

```
Input:
N = 2
Arr = [2, 3]
Output:
0 2 3 5
Explanation:
When no elements is taken then Sum = 0.
When only 2 is taken then Sum = 2.
When only 3 is taken then Sum = 3.
When element 2 and 3 are taken then
Sum = 2+3 = 5.
```

3, 1, 2

[ ]
[3]
[1]
[2]
[3, 1]
[3, 2]
[1, 2]
[3, 1, 2]

M1 brute force → power set

generate all subset

& traverse all subsets & return sum

TC → $2^N \times N$

f(0,0)

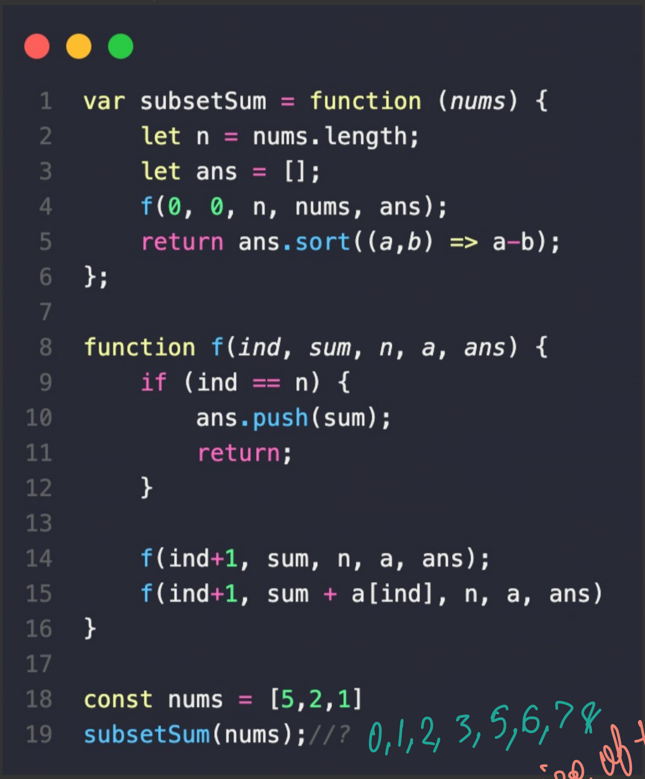
[3, 1, 2]

take, not take

ans

```
f(i, sum){
  if (i==n){
    ans.push(sum)
    return}
  f(ind+1, sum)
  f(ind+1, sum+a[i])
}
```

M2 optimal



```
var subsetSum = function (nums) {
    let n = nums.length;
    let ans = [];
    f(0, 0, n, nums, ans);
    return ans.sort((a,b) => a-b);
};

function f(ind, sum, n, a, ans) {
    if (ind == n) {
        ans.push(sum);
        return;
    }

    f(ind+1, sum, n, a, ans);
    f(ind+1, sum + a[ind], n, a, ans)
}

const nums = [5,2,1]
subsetSum(nums);//?
```

0, 1, 2, 3, 5, 6, 7, 8

TC → $2^N + 2^N \log(\underbrace{2^N}_{\text{sorting}})$ → size of the output

SC →

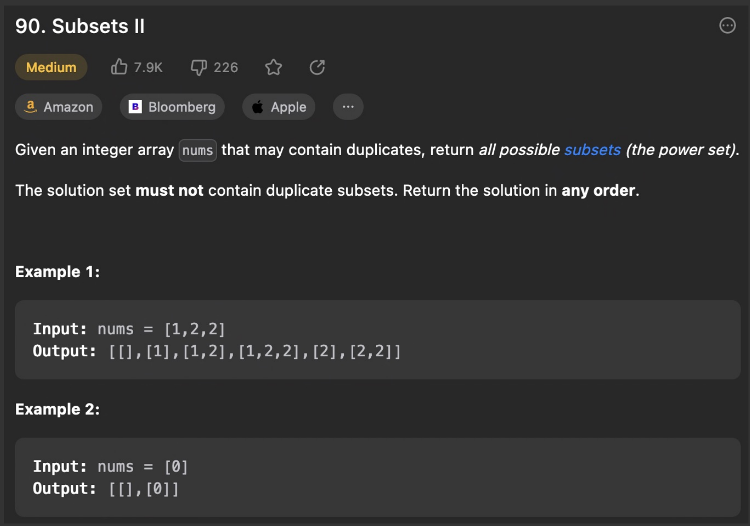
lec13 → subset sum 2

M1 brute force

generate all subsets → 2^N

put these (2^N) into a set

convert set to array of array



90. Subsets II

Medium 7.9K 226

Amazon Bloomberg Apple ···

Given an integer array `nums` that may contain duplicates, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

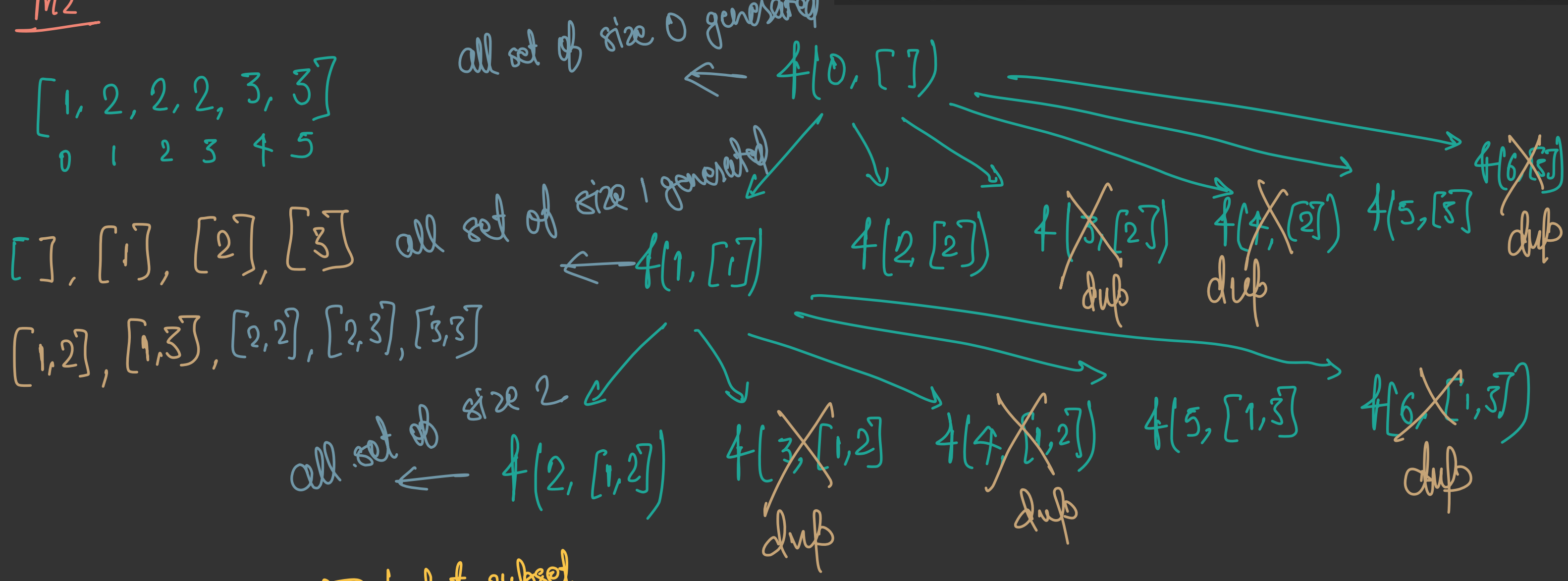
Example 1:

```
Input: nums = [1,2,2]
Output: [[],[1],[1,2],[1,2,2],[2],[2,2]]
```

Example 2:

```
Input: nums = [0]
Output: [[],[0]]
```

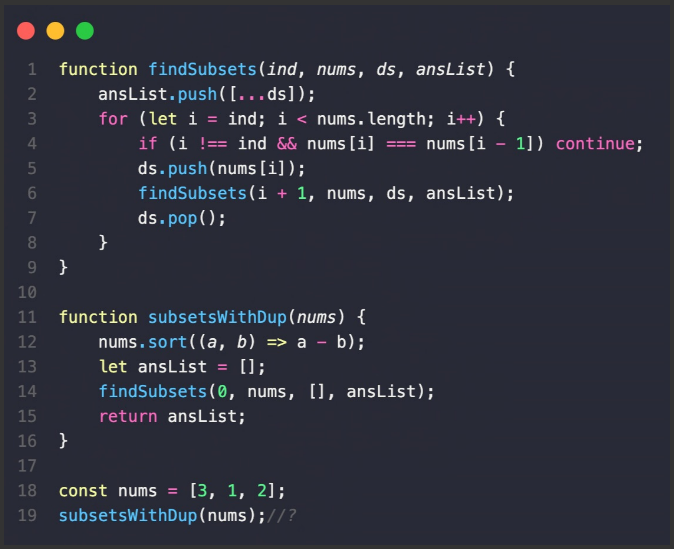
M2



TC → $2^N \times N$ ← to put subset into ans array

SC → $O(2^N) \times O(K) + O(N)$

↑ recursion



```
function findSubsets(ind, nums, ds, ansList) {
    ansList.push([...ds]);
    for (let i = ind; i < nums.length; i++) {
        if (i !== ind && nums[i] === nums[i - 1]) continue;
        ds.push(nums[i]);
        findSubsets(i + 1, nums, ds, ansList);
        ds.pop();
    }
}

function subsetsWithDup(nums) {
    nums.sort((a, b) => a - b);
    let ansList = [];
    findSubsets(0, nums, [], ansList);
    return ansList;
}

const nums = [3, 1, 2];
subsetsWithDup(nums);//?
```

```
f(0)
f(ind){
    ans.push(ds)
    for(i = ind to n-1){
        if ( i>ind && nums[i] == nums[i-1])
            continue;
        ds.push(nums[i])
        f(i+1)
        ds.pop()
    }
}
```

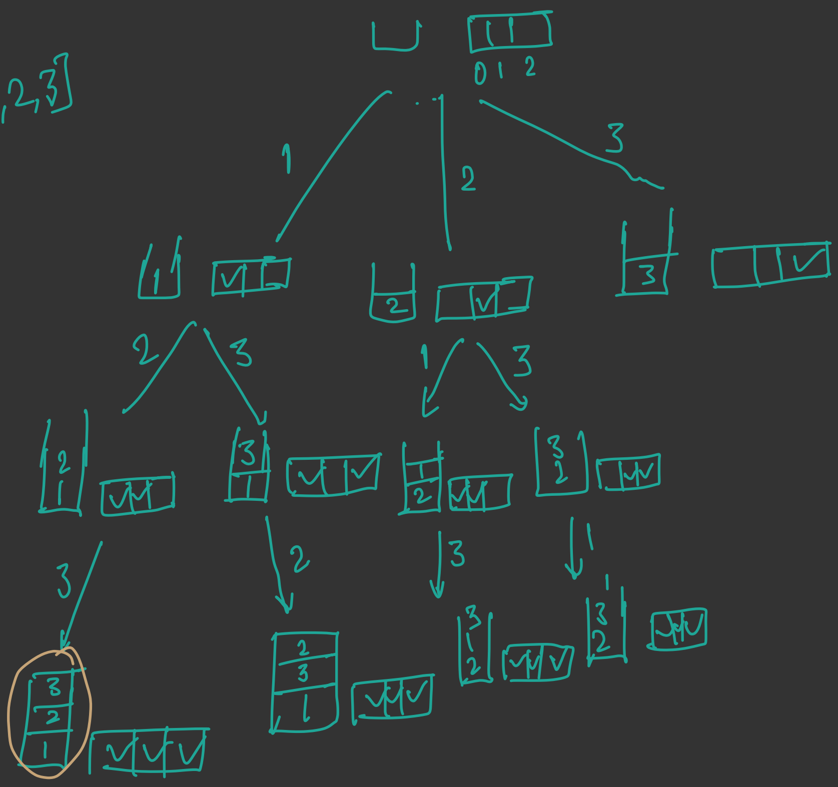
lec 14 print all permutations of a string/array

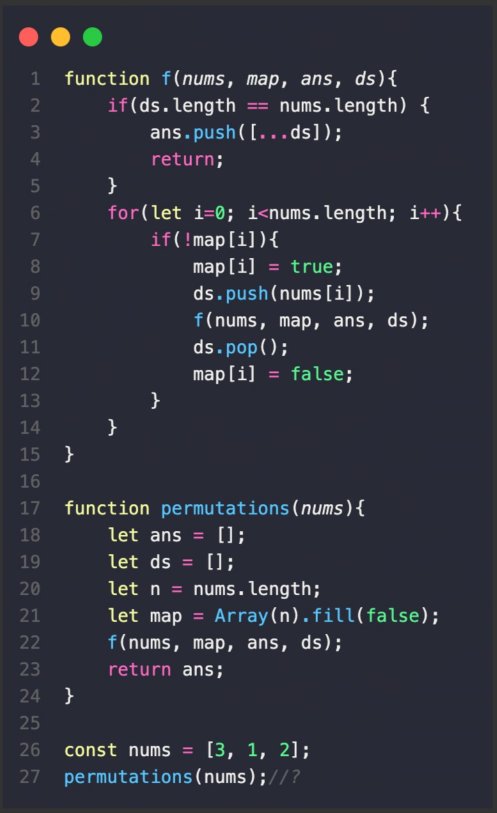
nums = [1, 2, 3] n = 3

total permutation = 3!
= 3×2×1
= 6

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

[1,2,3]

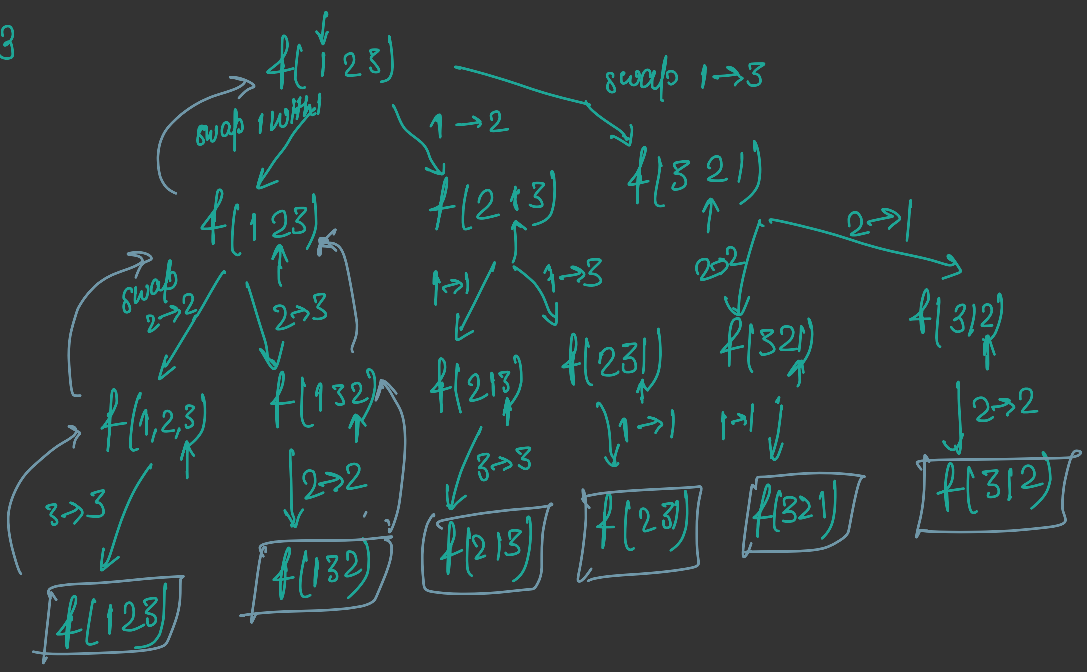




```
function f(nums, map, ans, ds){
    if(ds.length == nums.length) {
        ans.push([...ds]);
        return;
    }
    for(let i=0; i<nums.length; i++){
        if(!map[i]){
            map[i] = true;
            ds.push(nums[i]);
            f(nums, map, ans, ds);
            ds.pop();
            map[i] = false;
        }
    }
}

function permutations(nums){
    let ans = [];
    let ds = [];
    let n = nums.length;
    let map = Array(n).fill(false);
    f(nums, map, ans, ds);
    return ans;
}

const nums = [3, 1, 2];
permutations(nums);//?
```

lec15 print all permutations of a string

[1, 2, 3] n=3

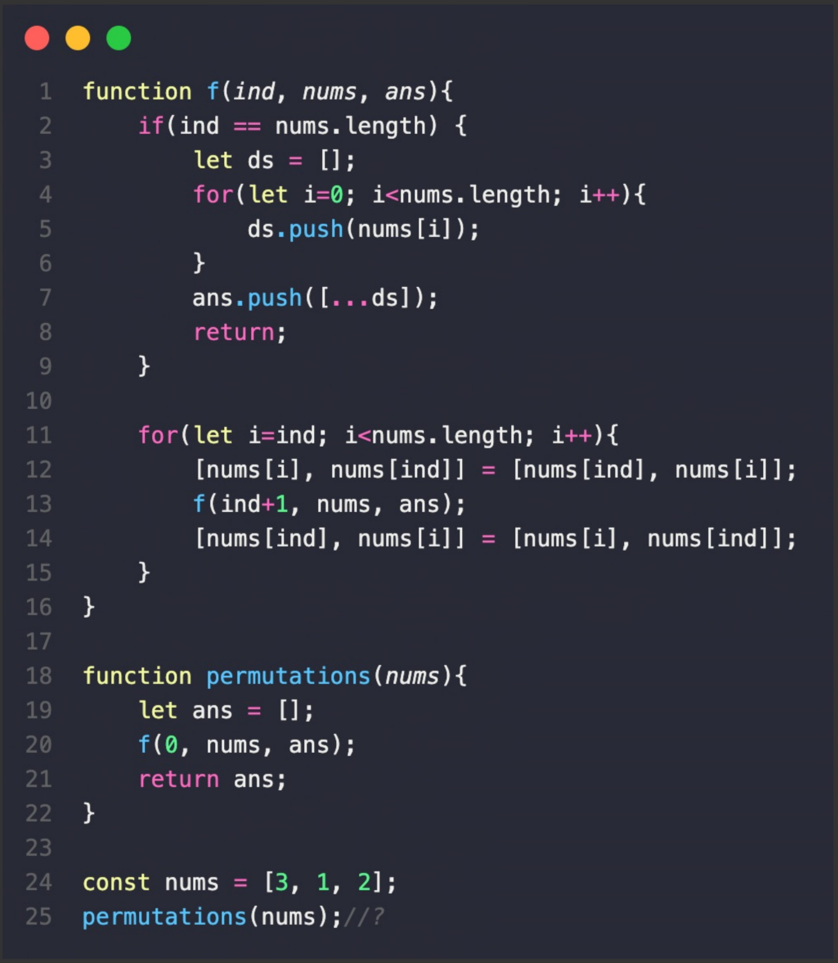


123
132
213
231
321
312

TC → N! × N

SC → O(N) + O(N!)

↑ recursion — ↑ ans

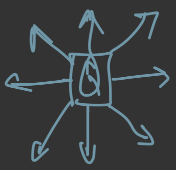


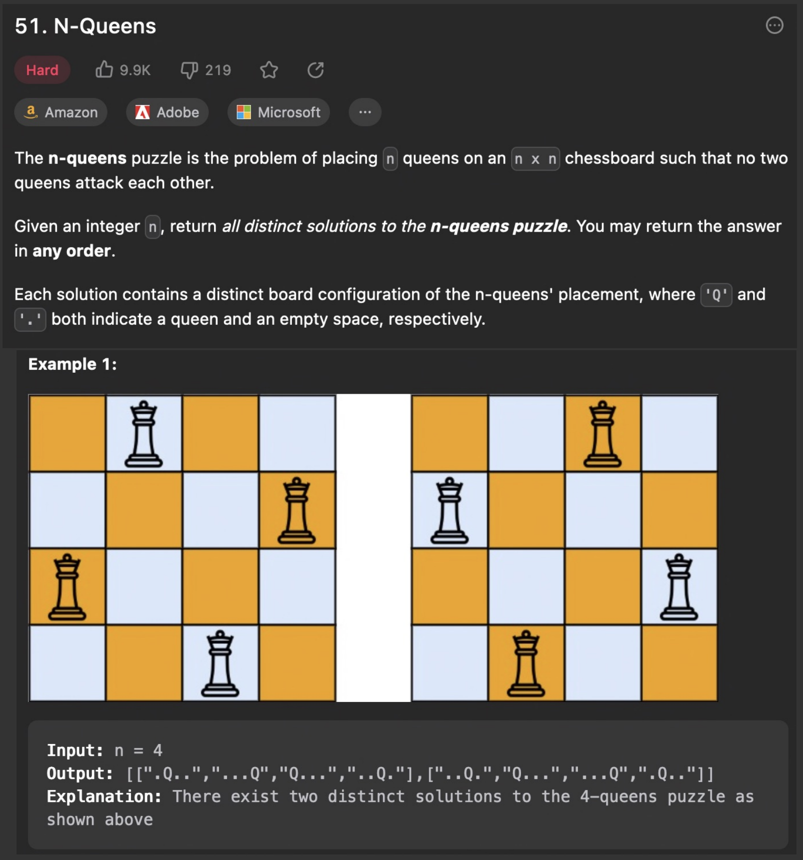
```
function f(ind, nums, ans){
    if(ind == nums.length) {
        let ds = [];
        for(let i=0; i<nums.length; i++){
            ds.push(nums[i]);
        }
        ans.push([...ds]);
        return;
    }

    for(let i=ind; i<nums.length; i++){
        [nums[i], nums[ind]] = [nums[ind], nums[i]];
        f(ind+1, nums, ans);
        [nums[ind], nums[i]] = [nums[i], nums[ind]];
    }
}

function permutations(nums){
    let ans = [];
    f(0, nums, ans);
    return ans;
}

const nums = [3, 1, 2];
permutations(nums);//?
```

lec16. N-Queens

- → every row has 1 queen
- → every col has 1 queen
- → none of the Queen should attack each other

attack





51. N-Queens

Hard 9.9K 219

Amazon Adobe Microsoft ···

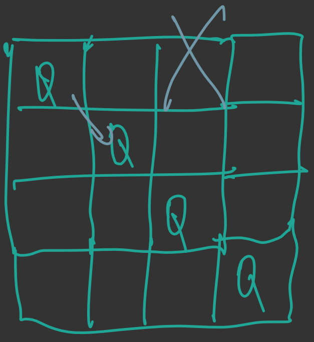
The **n-queens** puzzle is the problem of placing `n` queens on an `n x n` chessboard such that no two queens attack each other.

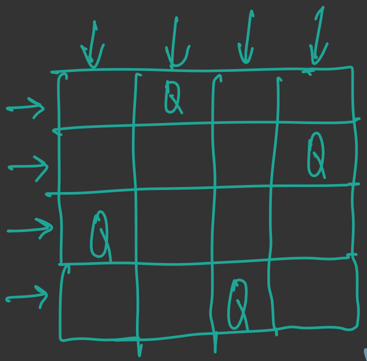
Given an integer `n`, return *all distinct solutions to the **n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

Example 1:

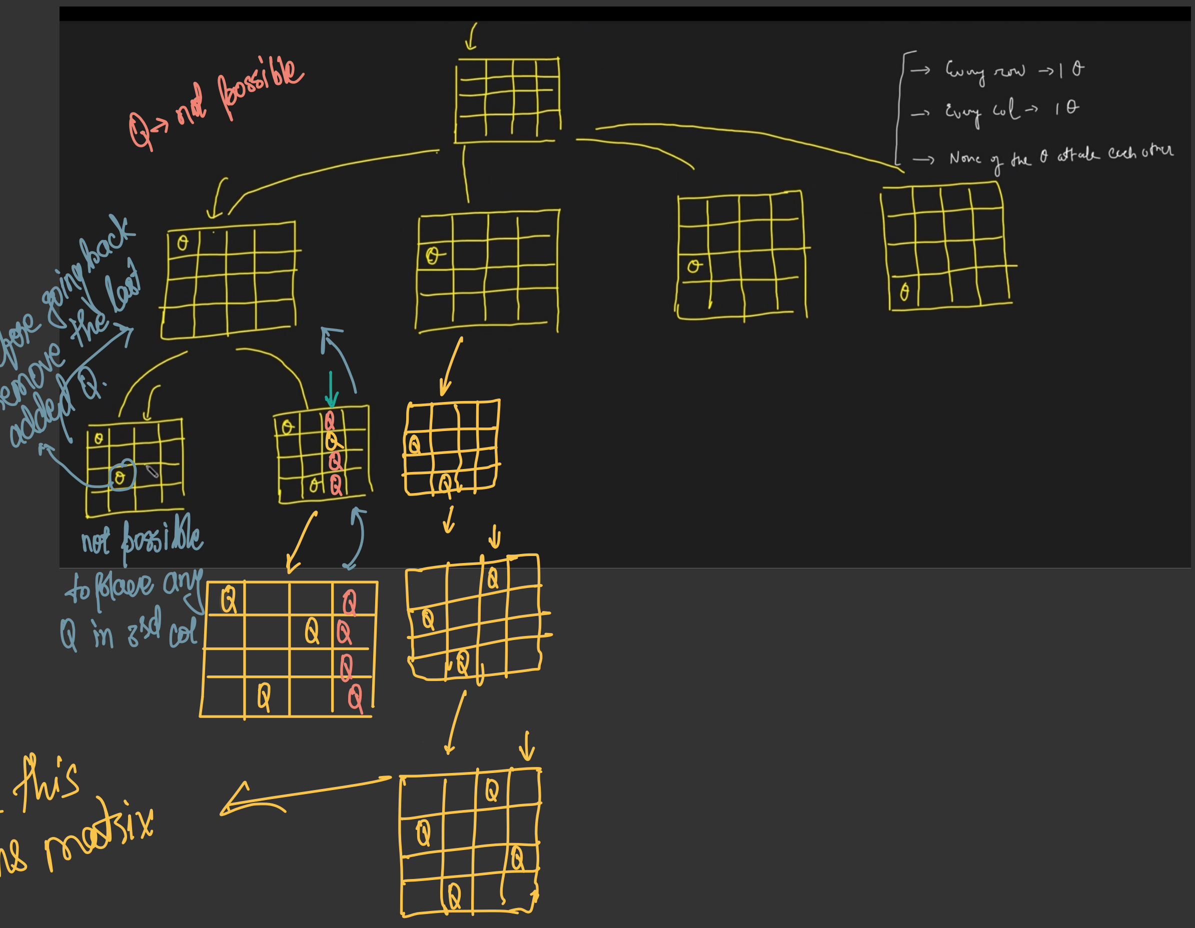
```
Input: n = 4
Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
Explanation: There exist two distinct solutions to the 4-queens puzzle as
shown above
```

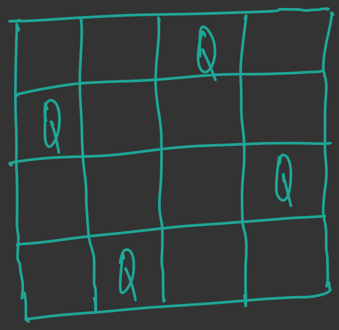




N given

count num of ways to place N Queens in NXN matrix?


Q→ not possible
→ Every row → 1 Q
→ Every col → 1 Q
→ None of the Q attack each other
before going back remove the last added Q.
not possible
to place any Q in 3rd col



store this in ans matrix

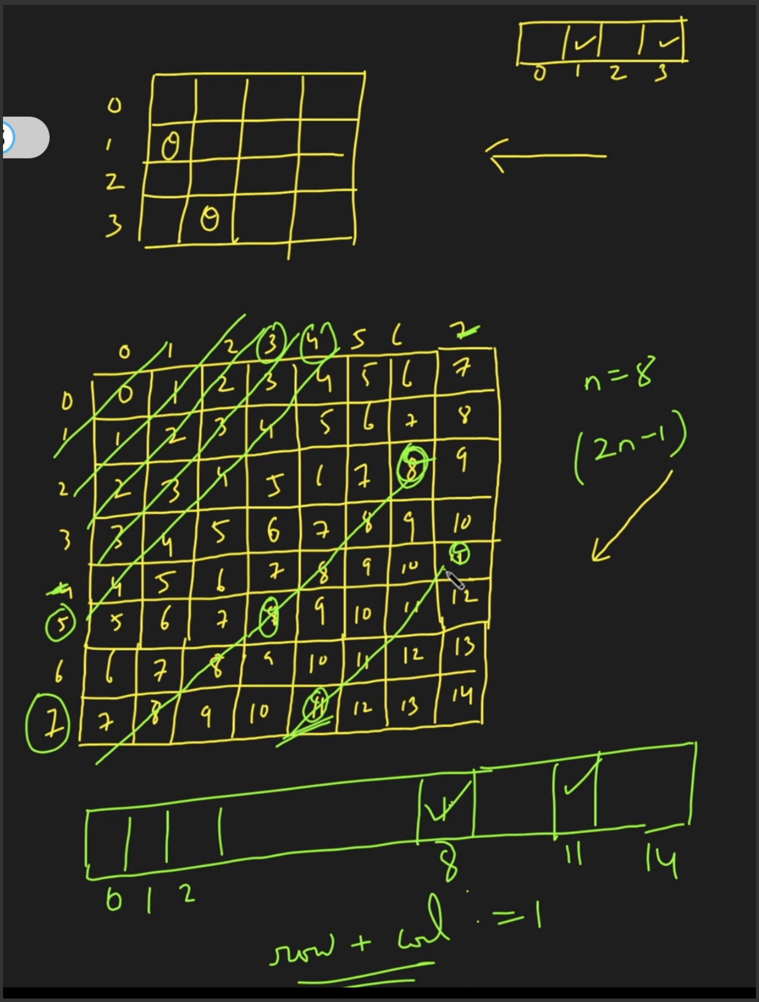
f(col){

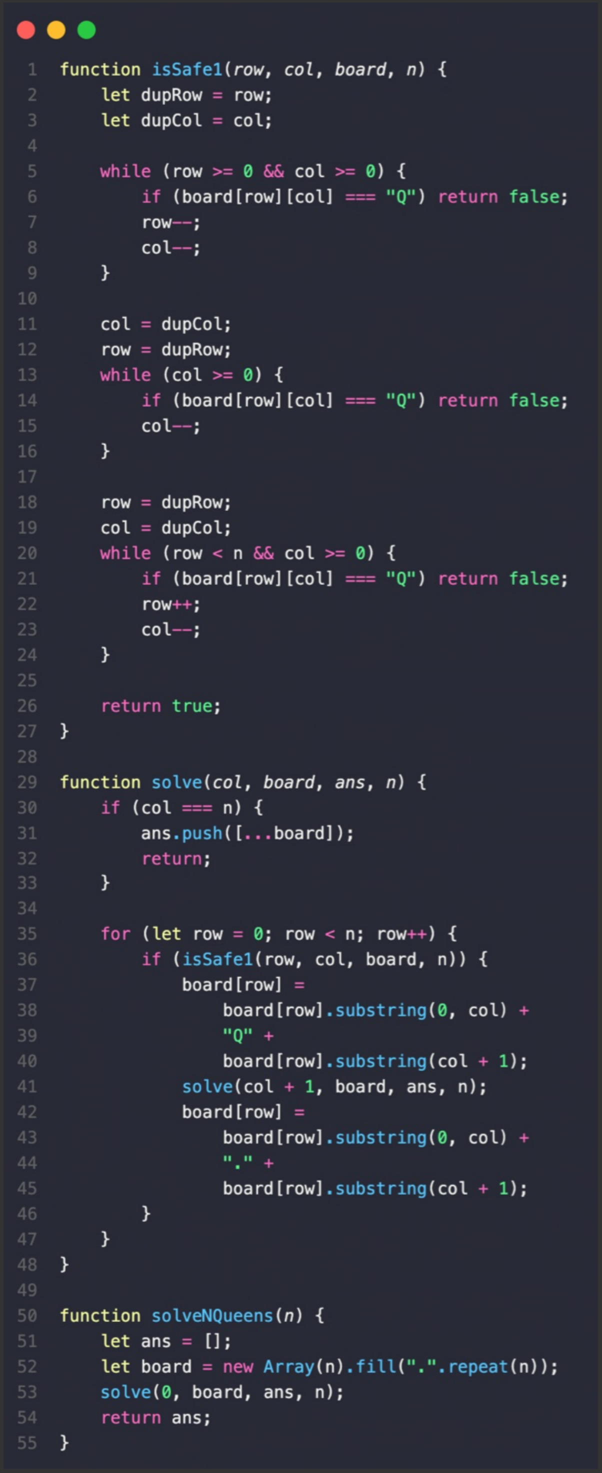
for(i=0 to n-1)
{ if(fill → ✓)
matrix[row][col] = 0
f(col+1)

}

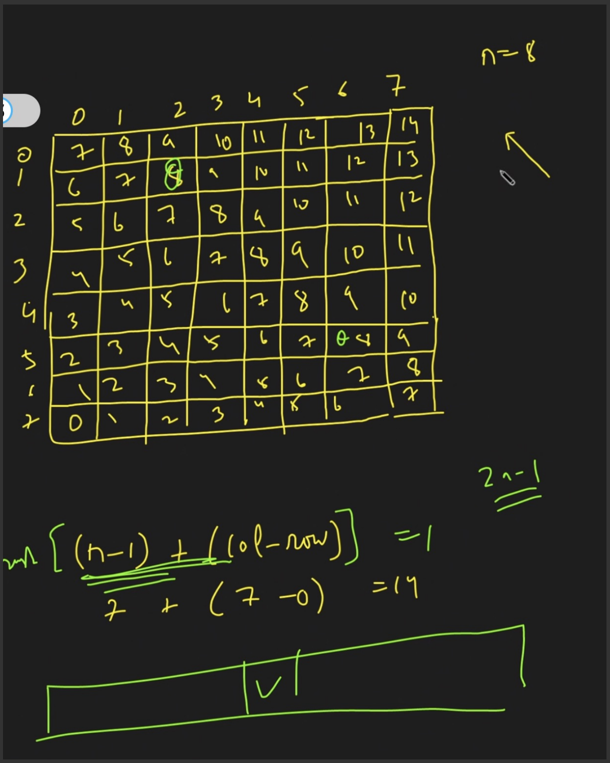
}

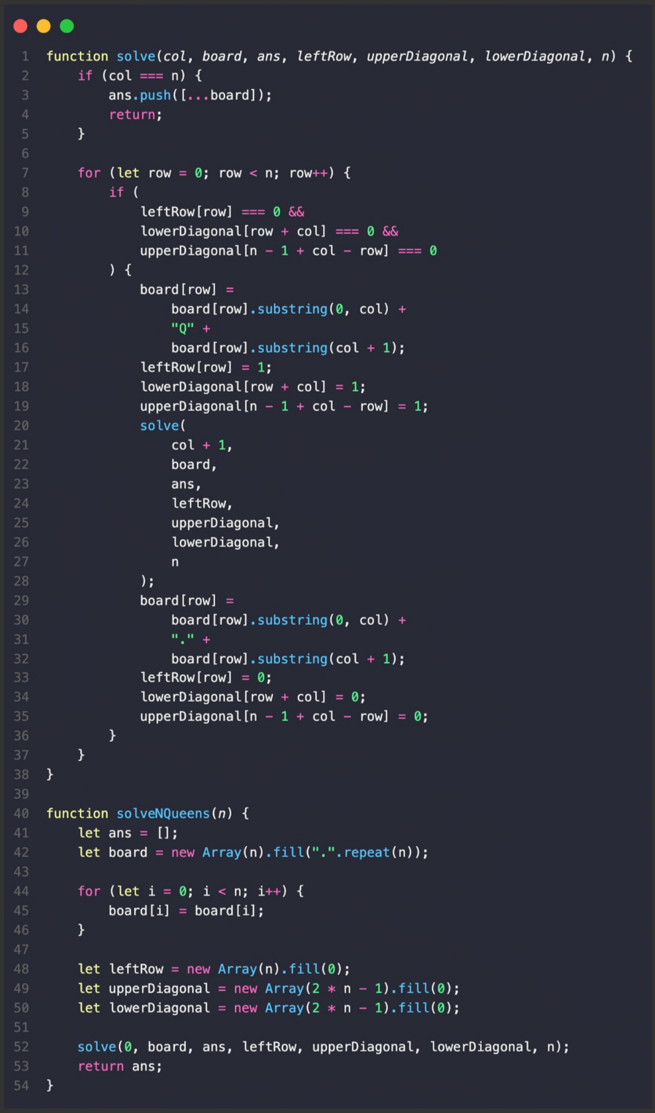
M2 → optimal → hashing





```
function isSafe1(row, col, board, n) {
    let dupRow = row;
    let dupCol = col;

    while (row >= 0 && col >= 0) {
        if (board[row][col] === "Q") return false;
        row--;
        col--;
    }

    col = dupCol;
    row = dupRow;
    while (col >= 0) {
        if (board[row][col] === "Q") return false;
        col--;
    }

    row = dupRow;
    col = dupCol;
    while (row < n && col >= 0) {
        if (board[row][col] === "Q") return false;
        row++;
        col--;
    }

    return true;
}

function solve(col, board, ans, n) {
    if (col === n) {
        ans.push([...board]);
        return;
    }

    for (let row = 0; row < n; row++) {
        if (isSafe1(row, col, board, n)) {
            board[row] =
                board[row].substring(0, col) +
                "Q" +
                board[row].substring(col + 1);
            solve(col + 1, board, ans, n);
            board[row] =
                board[row].substring(0, col) +
                "." +
                board[row].substring(col + 1);
        }
    }
}

function solveNQueens(n) {
    let ans = [];
    let board = new Array(n).fill(".".repeat(n));
    solve(0, board, ans, n);
    return ans;
}
```





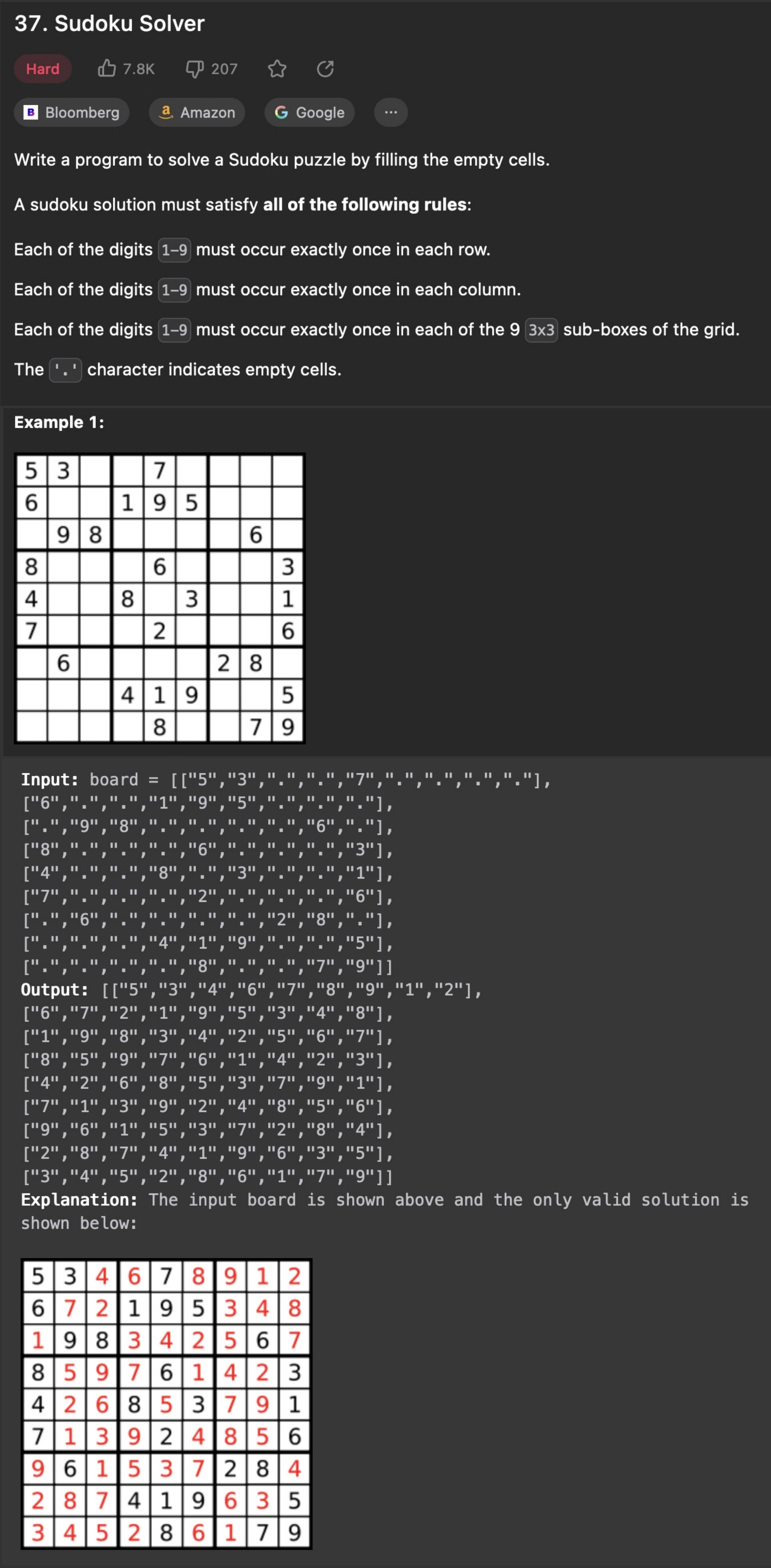
```
function solve(col, board, ans, leftRow, upperDiagonal, lowerDiagonal, n) {
    if (col === n) {
        ans.push([...board]);
        return;
    }

    for (let row = 0; row < n; row++) {
        if (
            leftRow[row] === 0 &&
            lowerDiagonal[row + col] === 0 &&
            upperDiagonal[n - 1 + col - row] === 0
        ) {
            board[row] =
                board[row].substring(0, col) +
                "Q" +
                board[row].substring(col + 1);
            leftRow[row] = 1;
            lowerDiagonal[row + col] = 1;
            upperDiagonal[n - 1 + col - row] = 1;
            solve(
                col + 1,
                board,
                ans,
                leftRow,
                upperDiagonal,
                lowerDiagonal,
                n
            );
            board[row] =
                board[row].substring(0, col) +
                "." +
                board[row].substring(col + 1);
            leftRow[row] = 0;
            lowerDiagonal[row + col] = 0;
            upperDiagonal[n - 1 + col - row] = 0;
        }
    }
}

function solveNQueens(n) {
    let ans = [];
    let board = new Array(n).fill(".".repeat(n));

    for (let i = 0; i < n; i++) {
        board[i] = board[i];
    }

    let leftRow = new Array(n).fill(0);
    let upperDiagonal = new Array(2 * n - 1).fill(0);
    let lowerDiagonal = new Array(2 * n - 1).fill(0);

    solve(0, board, ans, leftRow, upperDiagonal, lowerDiagonal, n);
    return ans;
}
```

lec17 sudoko solves

9x9 board → 3x3 boards

Rules.

1. the digit 1–9 → once in any row
2. " " 1–9 → " " " col
3. " " 1–9 → " " " cell



37. Sudoku Solver

Hard 7.8K 207

Bloomberg Amazon Google ···

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

Each of the digits `1-9` must occur exactly once in each row.

Each of the digits `1-9` must occur exactly once in each column.

Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

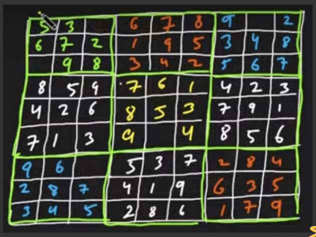
The `'.'` character indicates empty cells.

Example 1:

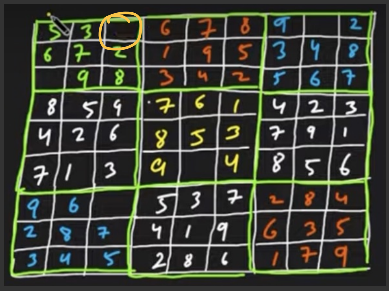
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```
Input: board = [["5","3",".",".","7",".",".",".","."],
["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"]]
Output: [["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"]]
Explanation: The input board is shown above and the only valid solution is
shown below:
```

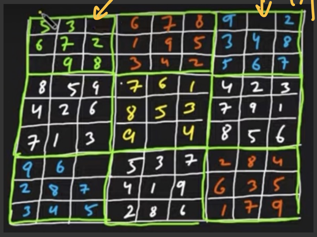
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9



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



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



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

1

try to fill up here
=> not possible
so return false

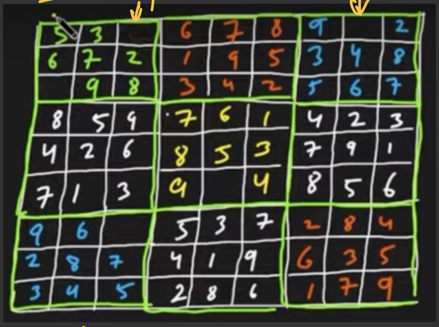
after filling out
try out next
empty

go right then bottom

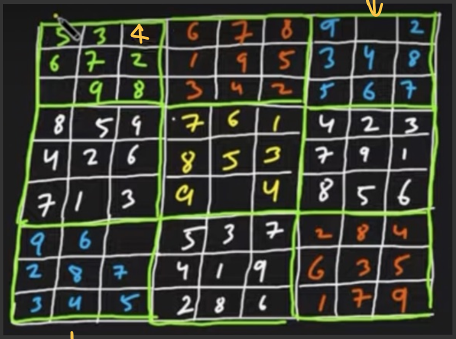
Check to fill empty by trying
1 to 9

return 1 valid
sudoko

4

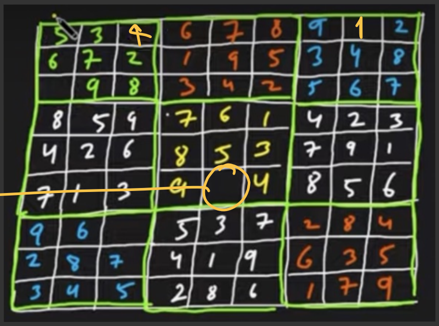


5	3		6	7	8	9		2
6	7	2	1	9	5	3	4	8
	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9		4	8	5	6
9	6		5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9



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

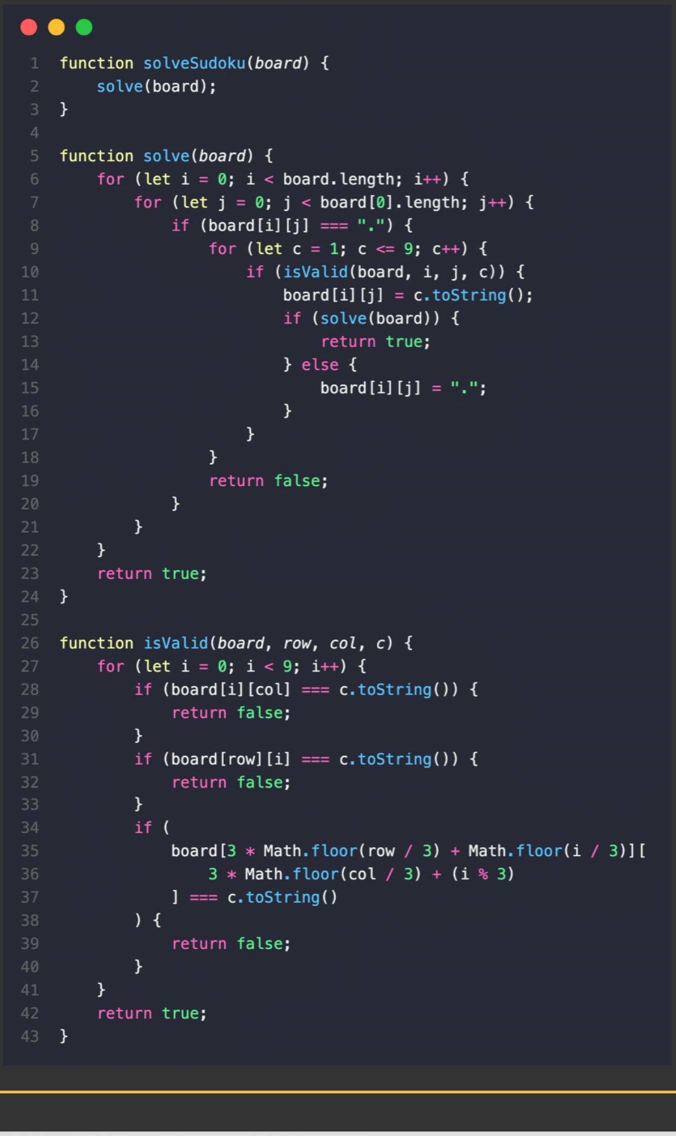
1



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

M2 → optimal

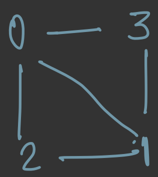
Check for row, col & cell in a single iteration



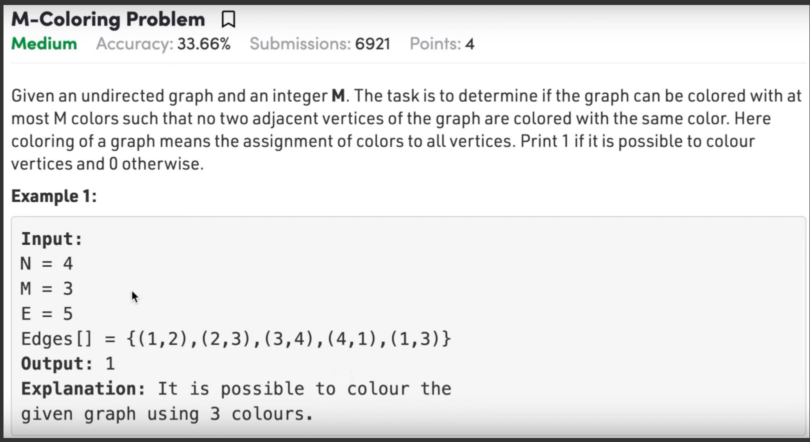
```
function solveSudoku(board) {
    solve(board);
}

function solve(board) {
    for (let i = 0; i < board.length; i++) {
        for (let j = 0; j < board[0].length; j++) {
            if (board[i][j] === ".") {
                for (let c = 1; c <= 9; c++) {
                    if (isValid(board, i, j, c)) {
                        board[i][j] = c.toString();
                        if (solve(board)) {
                            return true;
                        } else {
                            board[i][j] = ".";
                        }
                    }
                }
                return false;
            }
        }
    }
    return true;
}

function isValid(board, row, col, c) {
    for (let i = 0; i < 9; i++) {
        if (board[i][col] === c.toString()) {
            return false;
        }
        if (board[row][i] === c.toString()) {
            return false;
        }
        if (
            board[3 * Math.floor(row / 3) + Math.floor(i / 3)][
                3 * Math.floor(col / 3) + (i % 3)
            ] === c.toString()
        ) {
            return false;
        }
    }
    return true;
}
```

lec 18. M-coloring problem

at most M colors   M=3



{(0,3), (3,1), (1,2), (2,0), (0,1)}



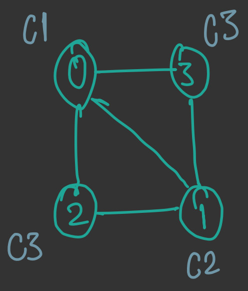
M-Coloring Problem

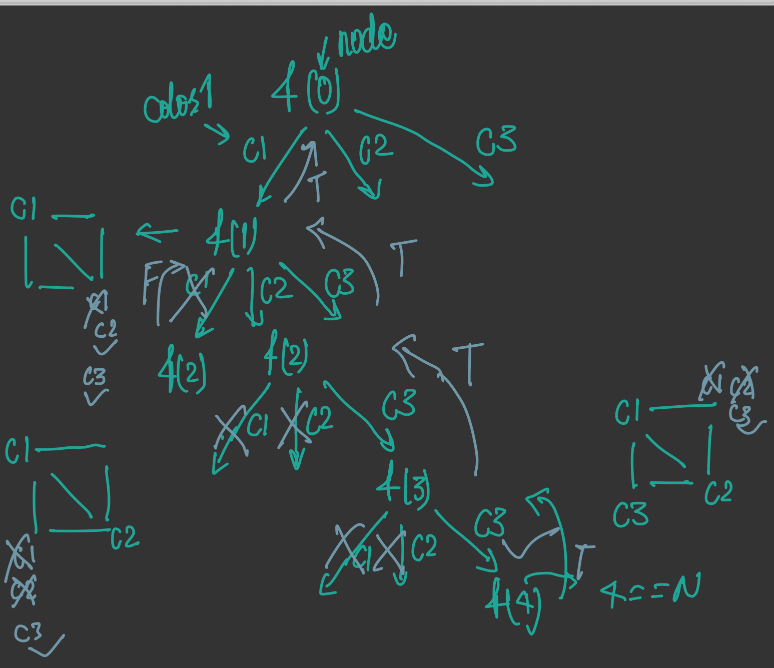
Medium Accuracy: 33.66% Submissions: 6921 Points: 4

Given an undirected graph and an integer **M**. The task is to determine if the graph can be colored with at most M colors such that no two adjacent vertices of the graph are colored with the same color. Here coloring of a graph means the assignment of colors to all vertices. Print 1 if it is possible to colour vertices and 0 otherwise.

Example 1:

```
Input:
N = 4
M = 3
E = 5
Edges[] = {(1,2),(2,3),(3,4),(4,1),(1,3)}
Output: 1
Explanation: It is possible to colour the
given graph using 3 colours.
```

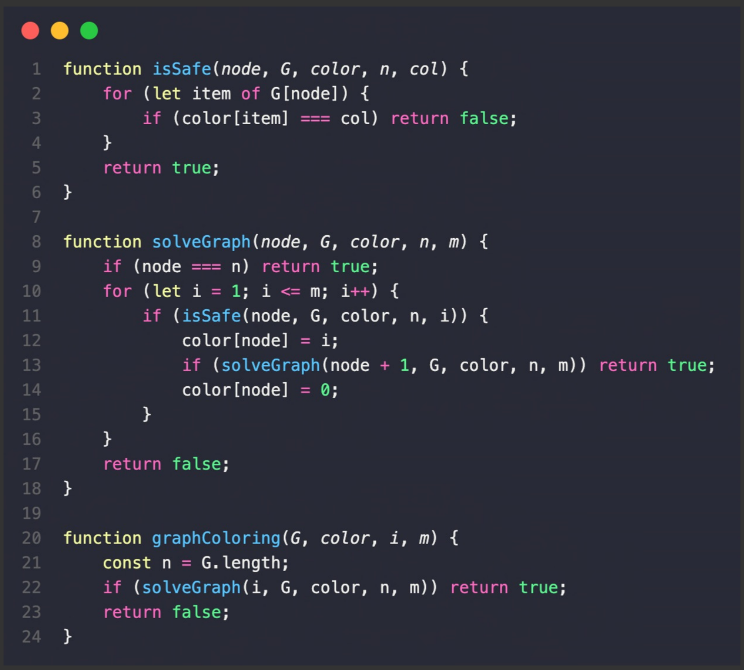




```
f(node){
    if(node == N) return T;
    for (col = 1 to m)
          ↑ color
    { if (possible ✓)
        {
          colors[node] = col;
          if( f(node+1) == T)
               return T;
          colors[node] = 0;
        }
    }
  return F;
}
```

TC → O(N^M) → try m colors at each node

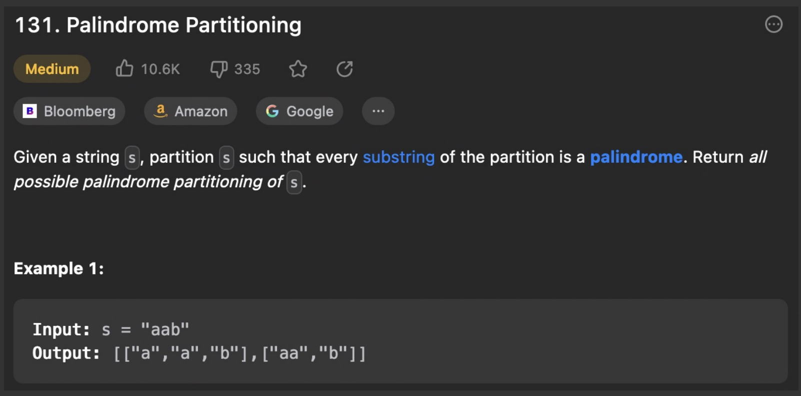
SC → O(N) + O(N)
recursion



```
function isSafe(node, G, color, n, col) {
    for (let item of G[node]) {
        if (color[item] === col) return false;
    }
    return true;
}

function solveGraph(node, G, color, n, m) {
    if (node === n) return true;
    for (let i = 1; i <= m; i++) {
        if (isSafe(node, G, color, n, i)) {
            color[node] = i;
            if (solveGraph(node + 1, G, color, n, m)) return true;
            color[node] = 0;
        }
    }
    return false;
}

function graphColoring(G, color, i, m) {
    const n = G.length;
    if (solveGraph(i, G, color, n, m)) return true;
    return false;
}
```

lec19. Palindrome Partitioning.

aabb → {a, a, b, b}
{a, a, bb}
{aa, b, b}
{aa, bb}



131. Palindrome Partitioning

Medium 10.6K 335

Bloomberg Amazon Google ···

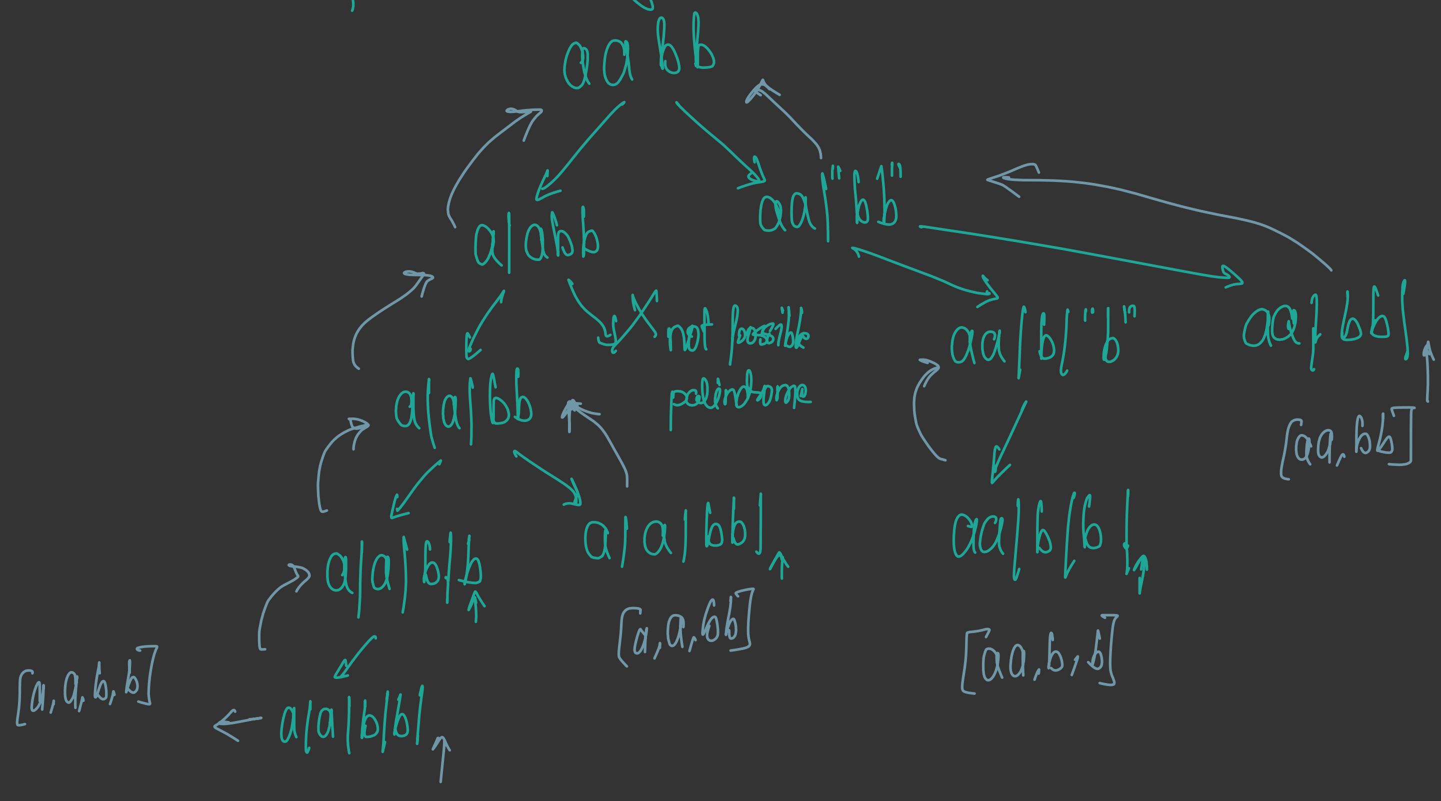
Given a string s, partition s such that every substring of the partition is a **palindrome**. Return *all possible palindrome partitioning of* s.

Example 1:

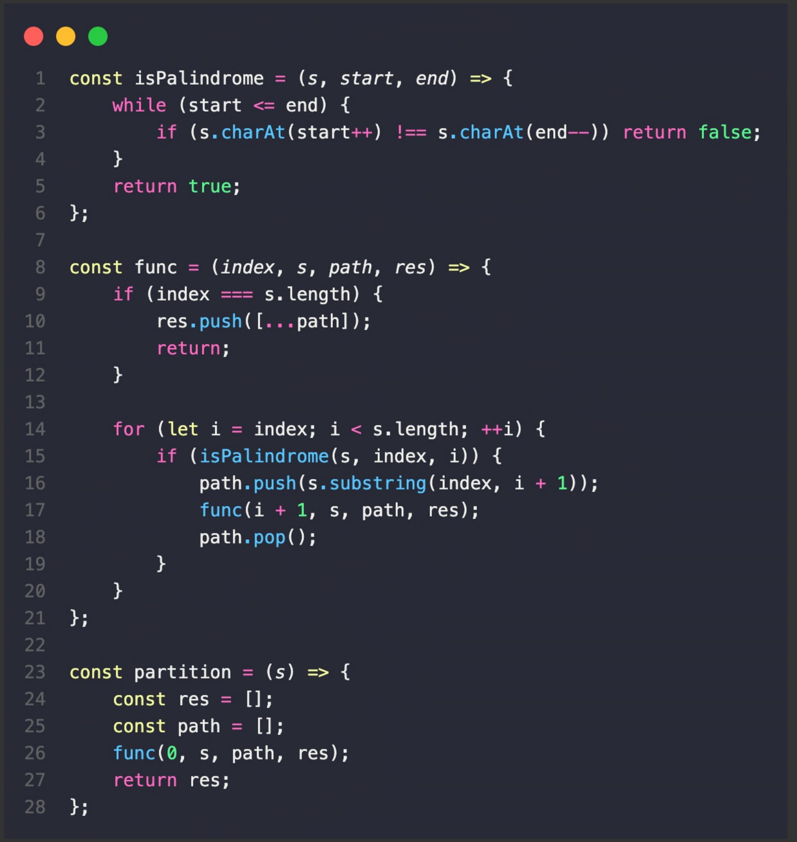
```
Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]
```

M1 brute force.

we can do partition only where left part is a palindrome

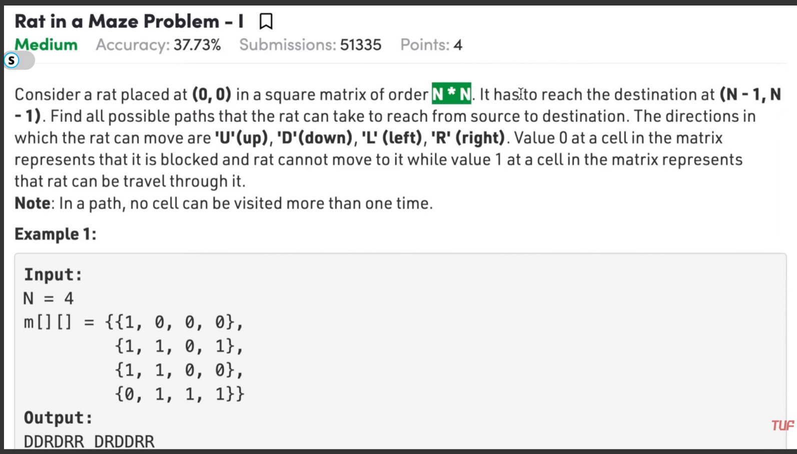


```
f(ind) {
    if(ind == n) {
        ans.push(path)
        return;
    }

    for( i = ind; i<n; i++) {
        if( isPalindrome(s, ind, i) {
            path.push( s.substring(ind, i+1))
            f(i+1)
            path.pop()
        }
}
```



```
const isPalindrome = (s, start, end) => {
    while (start <= end) {
        if (s.charAt(start++) !== s.charAt(end--)) return false;
    }
    return true;
};

const func = (index, s, path, res) => {
    if (index === s.length) {
        res.push([...path]);
        return;
    }

    for (let i = index; i < s.length; ++i) {
        if (isPalindrome(s, index, i)) {
            path.push(s.substring(index, i + 1));
            func(i + 1, s, path, res);
            path.pop();
        }
    }
};

const partition = (s) => {
    const res = [];
    const path = [];
    func(0, s, path, res);
    return res;
};
```

lec 20 → Rat in a maze



Rat in a Maze Problem - I

Medium Accuracy: 37.73% Submissions: 51335 Points: 4

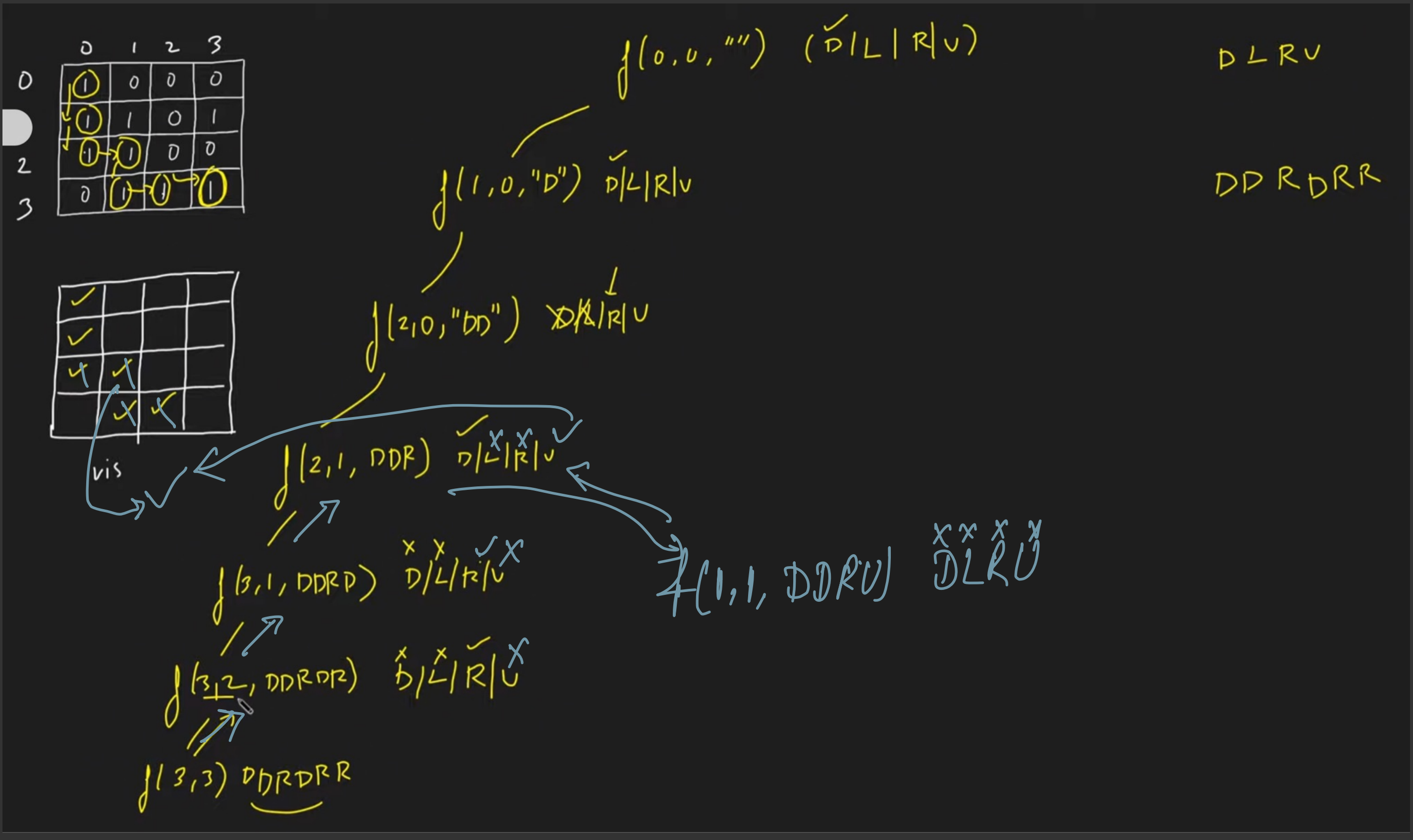
Consider a rat placed at **(0, 0)** in a square matrix of order **N * N**. It has to reach the destination at **(N - 1, N - 1)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are **'U'(up)**, **'D'(down)**, **'L' (left)**, **'R' (right)**. Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it.

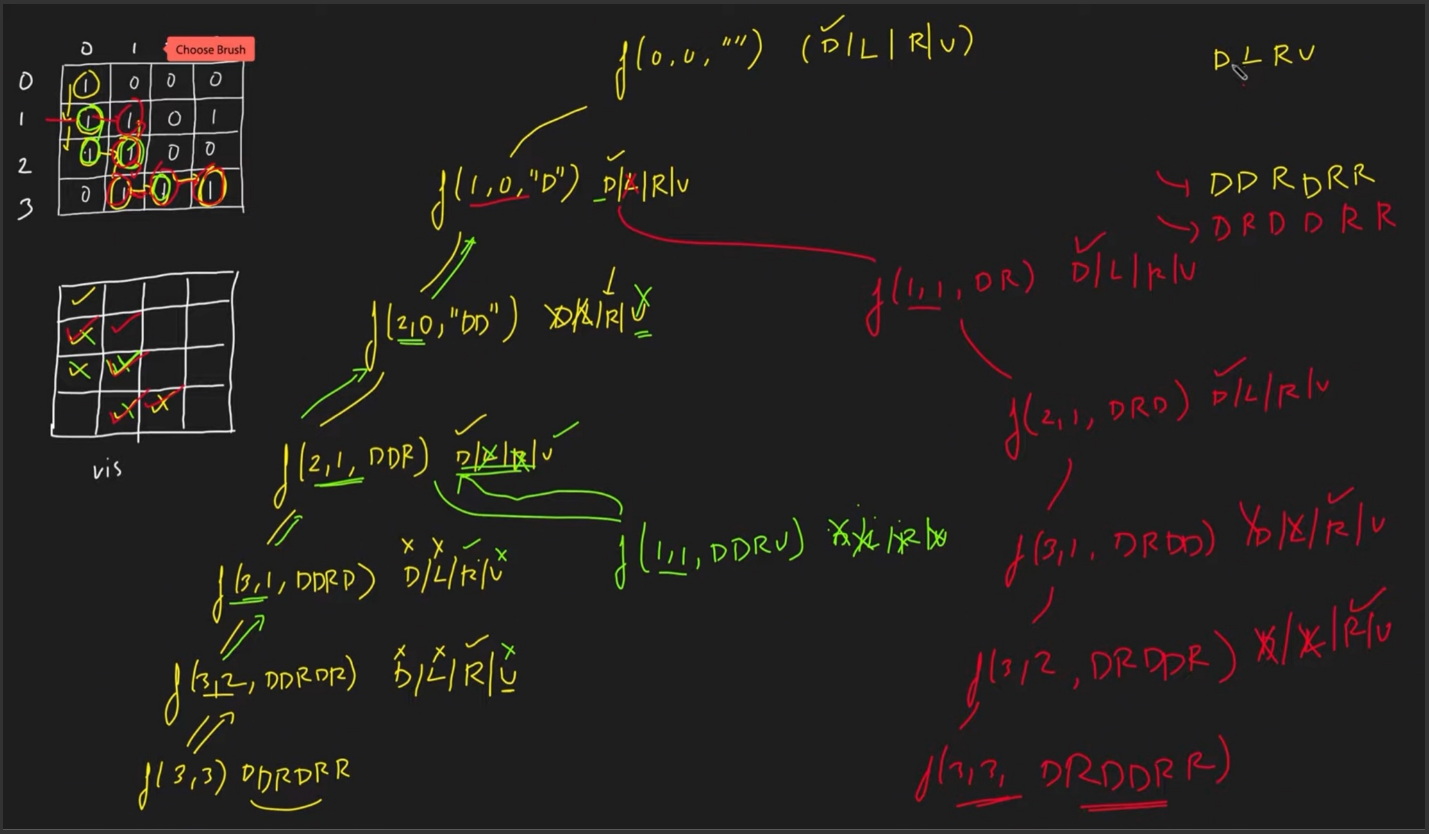
Note: In a path, no cell can be visited more than one time.

Example 1:

```
Input:
N = 4
m[][] = {{1, 0, 0, 0},
         {1, 1, 0, 1},
         {1, 1, 0, 0},
         {0, 1, 1, 1}}
Output:
DDRDRR DRDDRR
```

this color → backtrack





f(){

down

f()

left

f()

right

f()

up

f()

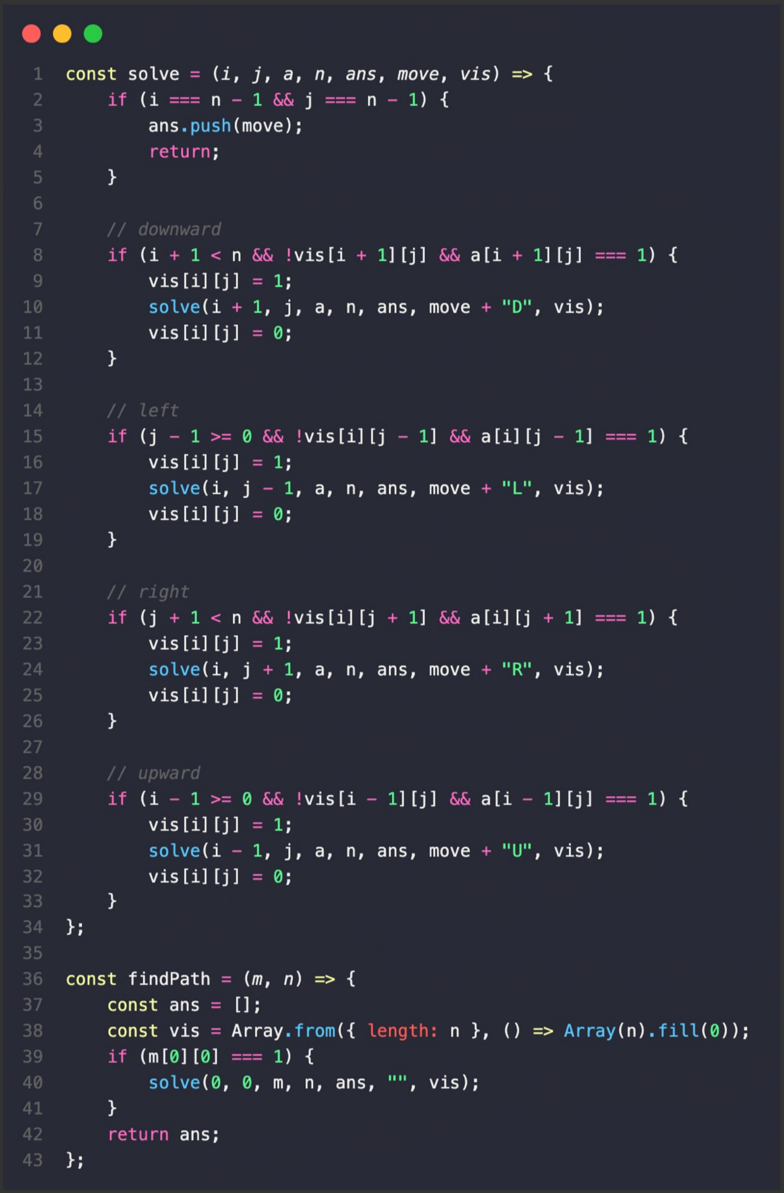
}

TC → $4^{N \times M}$

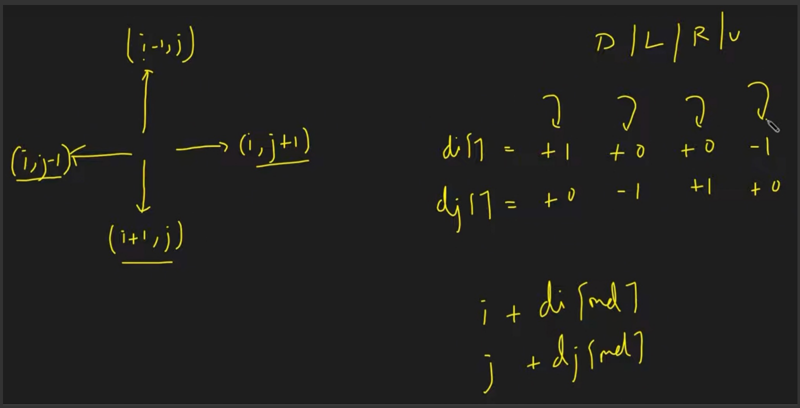
SC → O(M×N)

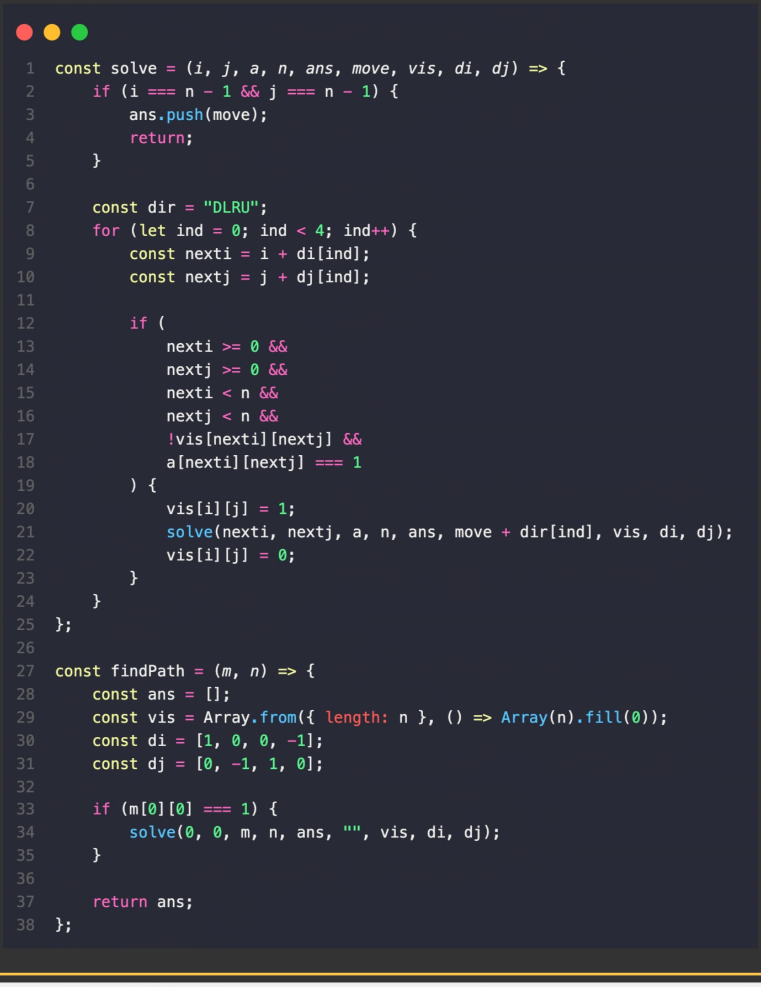
↑

recursion



```javascript
const solve = (i, j, a, n, ans, move, vis) => {
    if (i === n - 1 && j === n - 1) {
        ans.push(move);
        return;
    }

    // downward
    if (i + 1 < n && !vis[i + 1][j] && a[i + 1][j] === 1) {
        vis[i][j] = 1;
        solve(i + 1, j, a, n, ans, move + "D", vis);
        vis[i][j] = 0;
    }

    // left
    if (j - 1 >= 0 && !vis[i][j - 1] && a[i][j - 1] === 1) {
        vis[i][j] = 1;
        solve(i, j - 1, a, n, ans, move + "L", vis);
        vis[i][j] = 0;
    }

    // right
    if (j + 1 < n && !vis[i][j + 1] && a[i][j + 1] === 1) {
        vis[i][j] = 1;
        solve(i, j + 1, a, n, ans, move + "R", vis);
        vis[i][j] = 0;
    }

    // upward
    if (i - 1 >= 0 && !vis[i - 1][j] && a[i - 1][j] === 1) {
        vis[i][j] = 1;
        solve(i - 1, j, a, n, ans, move + "U", vis);
        vis[i][j] = 0;
    }
};

const findPath = (m, n) => {
    const ans = [];
    const vis = Array.from({ length: n }, () => Array(n).fill(0));
    if (m[0][0] === 1) {
        solve(0, 0, m, n, ans, "", vis);
    }
    return ans;
};
```





```
const solve = (i, j, a, n, ans, move, vis, di, dj) => {
    if (i === n - 1 && j === n - 1) {
        ans.push(move);
        return;
    }

    const dir = "DLRU";
    for (let ind = 0; ind < 4; ind++) {
        const nexti = i + di[ind];
        const nextj = j + dj[ind];

        if (
            nexti >= 0 &&
            nextj >= 0 &&
            nexti < n &&
            nextj < n &&
            !vis[nexti][nextj] &&
            a[nexti][nextj] === 1
        ) {
            vis[i][j] = 1;
            solve(nexti, nextj, a, n, ans, move + dir[ind], vis, di, dj);
            vis[i][j] = 0;
        }
    }
};

const findPath = (m, n) => {
    const ans = [];
    const vis = Array.from({ length: n }, () => Array(n).fill(0));
    const di = [1, 0, 0, -1];
    const dj = [0, -1, 1, 0];

    if (m[0][0] === 1) {
        solve(0, 0, m, n, ans, "", vis, di, dj);
    }

    return ans;
};
```

lec 21   kth permutation sequence

M1 bruteforce

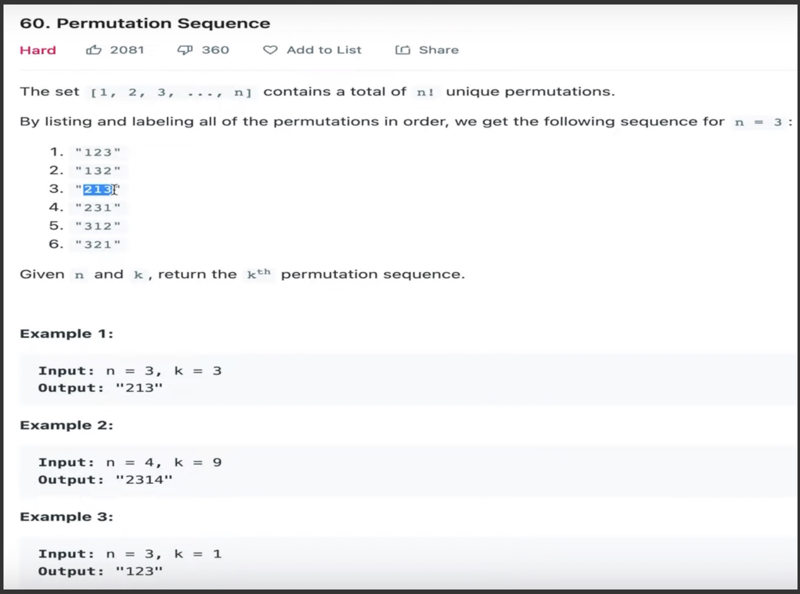
→ generate all permutations and store it in a data structure ans

→ sort ans

→ and return (k-1) index item

TC → N! × N + O(N! × NlogN)

↑ deep copy of ds



60. Permutation Sequence

Hard   2081   360   Add to List   Share

The set `[1, 2, 3, ..., n]` contains a total of `n!` unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for `n = 3`:

1. `"123"`
2. `"132"`
3. `"213"`
4. `"231"`
5. `"312"`
6. `"321"`

Given `n` and `k`, return the `kth` permutation sequence.

Example 1:

```
Input: n = 3, k = 3
Output: "213"
```

Example 2:

```
Input: n = 4, k = 9
Output: "2314"
```

Example 3:

```
Input: n = 3, k = 1
Output: "123"
```

M2 optimal solⁿ

n = 4   k = 17

tota permutations = 4! = 4×3×2 = 24

[1, 2, 3, 4]
 0 1 2 3

1 2 3 4 → 0th permutation

4 3 2 1 → 23th

if starts with

1 + (2,3,4)] 6   (0-5)

2 + (1,3,4)] 6   (6-11)

3 + (1,2,4)] 6   (12-17)

4 + (1,2,3)] 6   (18-23)

24

3 _ _ _

return k-1 = 16th ⇒ $16/6 = 2$

in each set

at index 2 ⇒ 3

1	2	3	4
0	1	2	3

$16 \% 6 = 4$ ⇒ 4th sequence in (12—17)

$[1, 2, 4]$, $k = 4$

0 1 2

$4/2 ⇒ 2$

$4 \% 2 ⇒ 0$

$1 + \{2, 4\}]\ 2 \quad (0-1)$

$2 + (1, 4)]\ 2 \quad (2-3)$

$4 + (1, 2)]\ 2 \quad (4-5)$

6

3 4 1 2

remaining

0th sequence generated by $\{1, 2\}$

$\{1, 2\}$

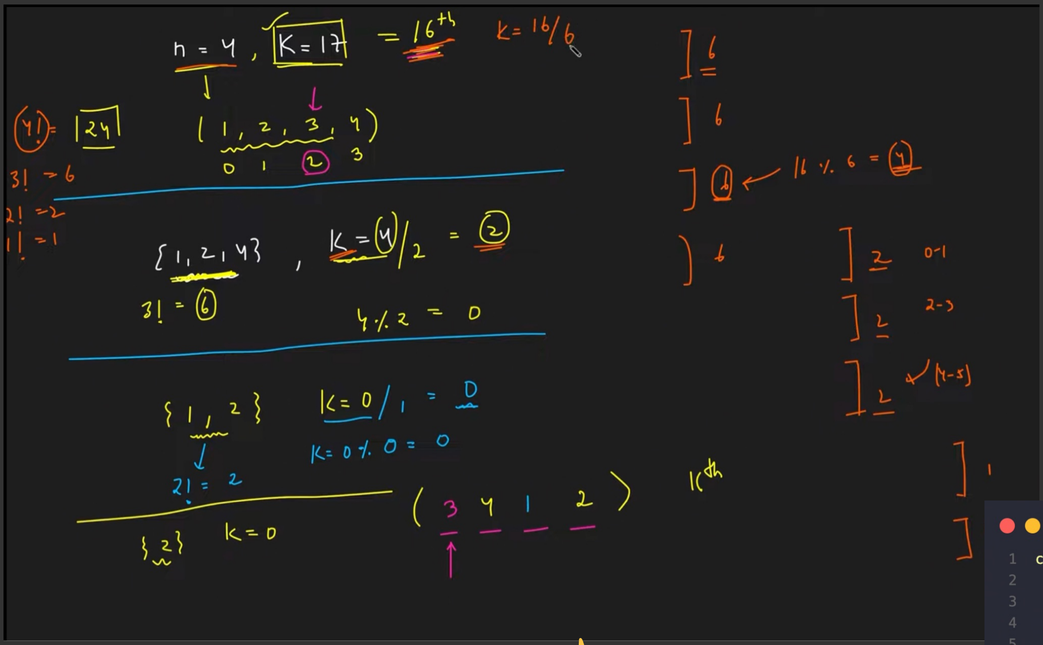
0 1

$0/1 = 0$

$1 + \{2\}]\ 1 \quad (0-0)$

$2 + \{1\}]\ 1 \quad (1-1)$

2



$n = 4$, $\boxed{K = 17}$ = 16th $\quad k = 16/6$

$4! = \boxed{24}$

$(1, 2, 3, 4)$

0 1 2 3

$3! \to 6$

$2! = 2$

$1! = 1$

$\{1, 2, 4\}$, $K = 4/2 = 2$

$3! = 6$

$4 \% 2 = 0$

$\{1, 2\}$ $\quad K = 0/1 = 0$

$K = 0 \% 0 = 0$

$2! = 2$

$\{2\}$ $\quad k = 0$

$(3\ 4\ 1\ 2)$

] 6

] 6

] 4 ← $16 \% 6 = 4$

] 6

] 2 $\quad$ 0-1

] 2 $\quad$ 2-3

] 2 $\quad$ ✓ (4-5)

16th

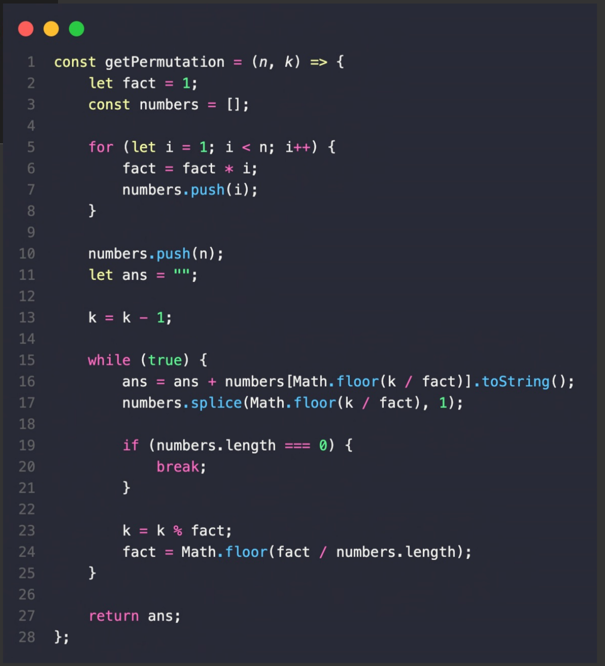
] 1

]

TC ⇒ $O(N) \times O(N)$

⇒ $O(N^2)$

SC → $O(N)$



```javascript
const getPermutation = (n, k) => {
    let fact = 1;
    const numbers = [];

    for (let i = 1; i < n; i++) {
        fact = fact * i;
        numbers.push(i);
    }

    numbers.push(n);
    let ans = "";

    k = k - 1;

    while (true) {
        ans = ans + numbers[Math.floor(k / fact)].toString();
        numbers.splice(Math.floor(k / fact), 1);

        if (numbers.length === 0) {
            break;
        }

        k = k % fact;
        fact = Math.floor(fact / numbers.length);
    }

    return ans;
};
```