★★★★★

Recursion

- V.V.I ✓ while the func. is not finished executing it will remain in stack.

✓ when a func. finishes executing it is removed from stack and the flow of program is restored to where the fn. was called.

Stack

print5(5)
print4(4)
print3(3)
print2(2)
print1(1)
main

O/P
1
2
3
4
5

Jaise sab ek-ek karke store ~~et~~ hua hai
i.e. main → print1(1) → ....

✓ usi hisab se execute hone ~~ke~~ baad nikalta jayega

→ Ulta. i.e. print5(5) → print4(4) → .... → main().

Concept of recursion in stack → main ~~ne~~ print1(1) ~~ko~~ call kiya ~~&~~ print 1 ~~ne~~ ander ek aur call ~~kr~~ diya jab tak wo execute na ho tab tak print1(1) nhi niklega aur uske baad main niklega (last) main.

✓ (main is the last fn. i.e. removed from stack).

✓ ✓ recursion → when a function call itself.

Stack memory in case of recursion →

```
print(n+1);
```

print(6)
print(5)
print(4)
print(3)
print(2)
print(1)
main()

O/P
1
2
3
4
5
6

∴ we need a base condition

✓ base condition in recursion : condition where our recursion will stop making new calls. (simple if condition).

```
if (n==5){
    cout (5);
    return; → Matlb dont call now
}
```

Now will empty as :-
print(5) → print(4) → direct sab before main
↓
main().

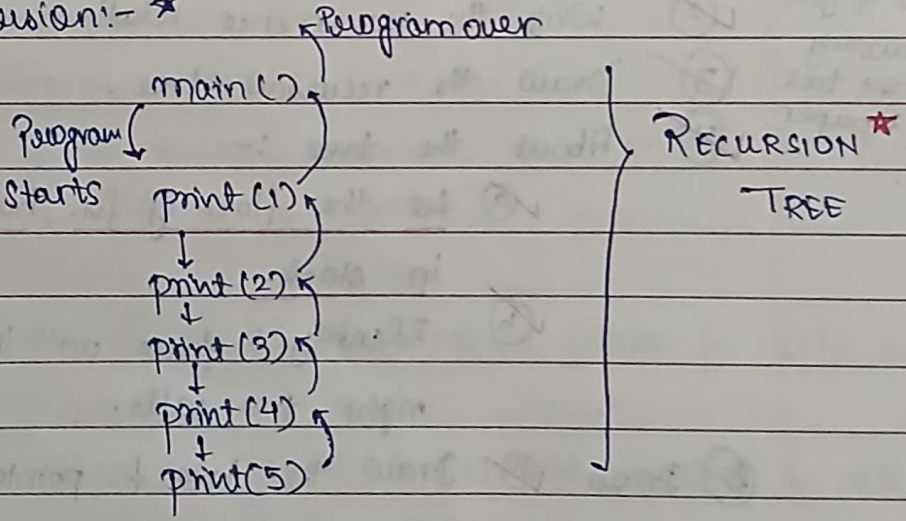
✓ if you are calling again & again, you can treat it as a separate call in the stack.

✓ No base cond^{n} → func. call will be keep happening.
→ stack will be filled again and again.
(every call of func. takes memory)
→ after some time, memory of computer will exceed to certain limit.
i.e. Stack overflow error.
when there is no base condition

Q. Why recursion?

- ✓ It helps us in solving bigger/complex problems in a simple way.
- ✓ You can covert any recursion sol^{n} into ≠ iteration & vice-versa.
- ✓ Space complexity is not constant because of recursive calls.
- ✓ It helps us in breaking down bigger problems into smaller problems.

✓ Visualising Recursion:- ★



- General concept for Fibonacci no.

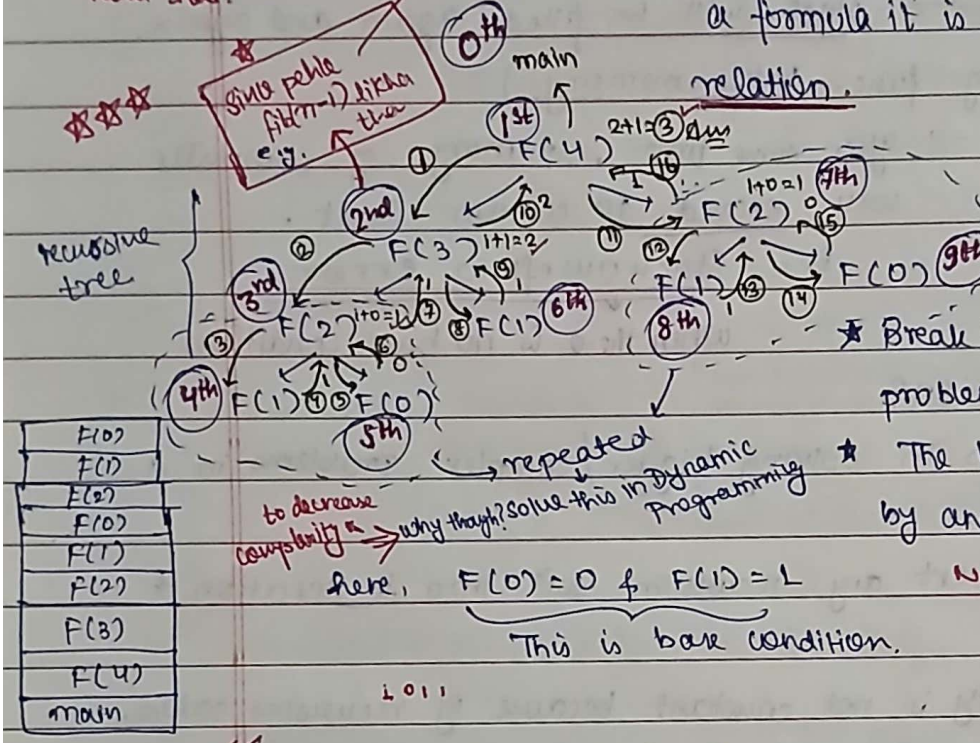
is basic maths

$$Fibo(N) = Fibo(N-1) + Fibo(N-2)$$

kai sochne ki jaroret nhi hai → seedha dikh raha hai.

when you write a recursion in a formula it is called recurrence relation.

Sirf pehle fib(n-1) likha tha e.g.



repeated → to decrease complexity → why though? solve this in Dynamic Programming

here, $F(0) = 0$ & $F(1) = 1$ — This is base condition.

★ Break it down into smaller problems.

★ The base cond^{n} is represented by answers we already have.

Notice- black arrows for filling in stack.

VVVI. How to understand and approach a problem?

No one can explain recursion to you if you're not drawing recursion tree in pen & paper.

1. Identify if you can break down problem into smaller problems
2. Write the recurrence relation if needed.
3. Draw the recursive tree.
4. About the tree :-
   a. See the flow of fn., how they are getting in stack.
   b. Identify & focus on left tree calls & right tree calls.
5. ~~Dread~~ Draw the tree & pointer again & again using pen & paper.
   a. use a debugger to see the flow.
5. See how values are returned at each step & what type of values

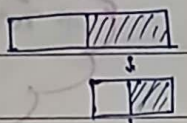
See where the fn. call will come out of.
In the end ~~you~~ it will come out of the main fn.

Tip! Do not over think.

VVVVVI Variables: (1) Arguments
(2) Return types
(3) Body of func.

Understand the code

Q. Binary search with recursion.



1. (1) Comparison in constant time i.e. O(1) ← search in the array of N/2.
(2) Dividing into 2 half

2. recurrence rel^n → $F(N) = O(1) + F(N/2)$
comparison

i.e. to find binary search in array of size 'n' → ~~take~~ do a step that takes constant amount of time + search in 1/2 of the array.

- Types of recurrence relation:-
  - Linear recurrence relation → e.g. Fibo
  - ~~Fibo~~ Divide & conquer recurrence rel^n → eg. Binary search (reduced by a factor)

⋮

⋆⋆ Whatever variables you need to pass in future fn. calls put inside the argument.
And variables that are considered valuable in that fn. call only that you don't need to pass into future recursion calls put inside body of function.
Here middle 'm' is in body & start 's' & end 'e' in argument.
↳ for future rec. calls

return 0; ⇒ signifies success or that the operation completed successfully without errors.

e.g. filed closed successfully

Note: this & return -1; will return 0 & -1 only respectively but it indicates their meaning as value 0 & -1 ⇒ convention in C/C++/Java.

✓ Make sure to return the result of a fn. call of the return type.

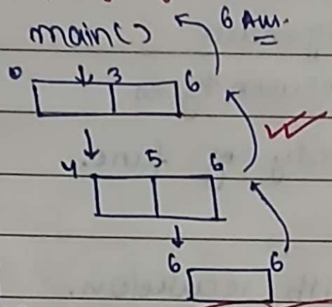
arr: [1, 2, 3, 4, 55, 66, 78]   target: 78   mid. m: 3

(indices: 0 1 2 3 4 5 6)

Left takes (start → middle)
Right takes (m+1 → end)

↓ thought only ↓ but check code!!

✓ code ko debug karke samjho aise samaj nhi aayega.



// return 6 → way of showing error in form of integer.

* return -1; ⇒ indication that a certain condition has not been met;

e.g. error opening file   e.g. invalid input data

here → e.g. element not found

$\frac{s+e}{2} = s + \frac{(e-s)}{2}$

Time and Space Complexity

✓ What is time complexity?

reference ex. →

Old Computer	M1 macbook
✓ data: 1,000,000 elements in array	"
✓ Algorithm, linear search for target that doesn't exist in array	"
✓ Time taken → 10 sec	Time: 1 sec.

Both have same Time Complexity i.e. O(n)

because [Time Complexity ≠ Time Taken]

It's basically gt a function that gives us the relationship about how the time will grow as input grows

O(N^3) → upper bound that means complexity will never exceed N^3. e.g. N^2, N^2 log N x

- Worst case complexity : $O(N^2)$
- Best " " : $O(N)$
- use case : Adaptive i.e. steps get reduced if array is sorted. no. of swaps reduce as comparison to Bubble sort
- Stable Algo
- Used for smaller values of N => works good when array is partially sorted + it takes part in hybrid sorting algorithms (e.g. quick sort, merge sort, heapsort then insertion)

* Recursion LvL-1 Ques.

Sum Digit → e.g. n = 1342 → 1+3+4+2 = 10

2 + f(134) N/10

N%10 4 + f(13)

3 + f(1)

1 + f(0)

stack pushing

Ans. 2+8 = 10, 4+4 = 8, 3+1 = 4

popping stack

f(1) = 1 ⇐ [f(0) = 0]

Base cond^n

Similarly Prod Digit. n = 1342 → 1×3×4×2 = 24

2 × f(134)

4 × f(13)

3 × f(1)

1 × f(0)

Base condition [f(0) = 1]

2342

2 × f(234)

4 × f(23)

3 × f(2)

2 × f(0)

alternate base condition →

```
n%10 == n
return n;
```

e.g. n = 505

5 × f(50)

0 × f(5)

Concept :-

If in recursion call `n--;`

then means pass n then sub. 1

∴ it will call n again & again & again ....

∴ use `(--n)`.

- reverse a no. :-

eg. n = 1824 ⇒ 4281

4 + f(182)

2 + f(18)

8 + f(1)

1 + f(0)

Method :1 →

eg. 123

(123) false	false(12)	false(1)	true
			return
rem = 3	rem = 2	rem = 1	sum
sum = 0+3 = 3	sum = 30+2 = 32	sum = 320+1 = 321	(321)

```
sum = 0
fun(n){
if (n==0) {
    return;
}
rem = n%10;
sum = sum*10 + rem
fun(n/10);
}
```

Method :2 → Sometimes you might need some additional variables in argument in that case make another function. (helper function) → eg. swap, etc.

n = 1234

4 + 123

4×1000 + 123

[4×10^3 + f(123)]

→ [3×10^2 + f(12)]

→ [2×10 + f(1)]

[f(n, arg) = rem*10^(arg-1) + f(n/10, arg-1)]

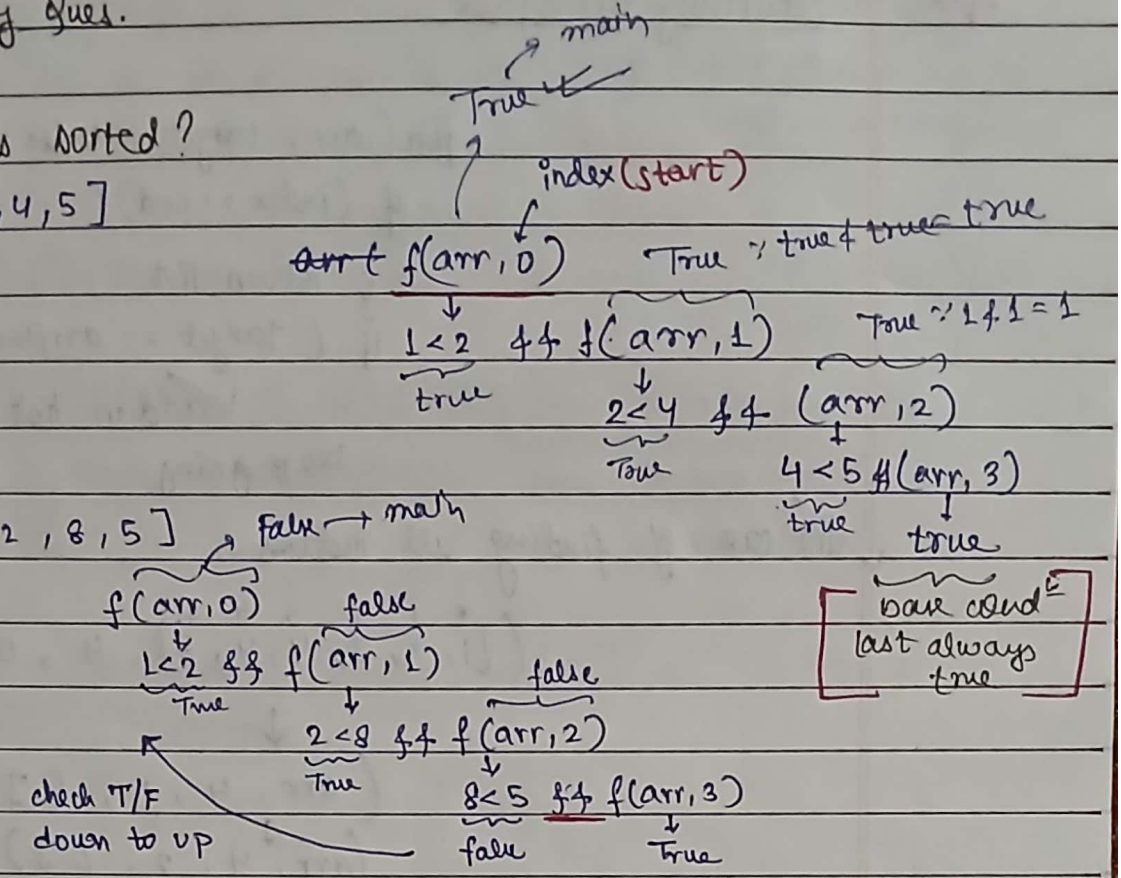
base condition → `if (n%10==n) return n;`

Date 17/07/24

* Recursion - Array Ques.

Q. Check array is sorted?

→ arr = [1,2,4,5]



Linear search with recursion

arr = [3, 2, 1, 18, 9] , target = 18

f(arr, target, 0) ← index

arr[0] == target || (arr, target, 1) ← aur sirf agar ek bhi jaga target mil gya to True!

arr[1] == target || (arr, target, 2)

⋮

false ← till index becomes out of bound

returning Arraylist

```
fun( arr, target, index, list)
    if (index == end){
        return list;
    }
    if ( target == arr[index]){
        add in list
    keep going
```

in case of finding all indexes.

([1, 2, 3, 4, 4, 8], 4, 0, [ ]) → main

↓

(arr, 4, 1, [ ])

↓

(arr, 4, 2, [ ])

↓

∴ list.add Yes! ← (arr, 4, 3, [ ])

⋮ return

(arr, 4, 6, [3, 4])

adding on the same object just via diff. reference variable.

e.g. ⇒

↓

1. PASSING IT IN ARGUMENT

```
static ArrayList<Integer> findAllIndex(int[] arr, int target,
                                int index, ArrayList<Integer> list){
    if (index == arr.length){
        return list;
    }
    if (arr[index] == target){
        list.add(index);
    }
    return findAllIndex(arr, target, index +1, list);
}
```

[return type]

∵ last tak check hoga agar nhi mila to usko aage wale ↓ return pahuchkar

list → returned

type ArrayList

Check B.S. Many Q. videos

Date 14/04/24
Page

Rotated Binary Search

watch B.S. video if you don't know this is rotated array.

	0	1	2	3	4	5	6	7
arr =	5	6	7	8	9	1	2	3

① if arr[s] ≤ arr[mid]

if key ≥ arr[s] & ≤ arr[mid]

end = mid - 1

else

s = m + 1

e.g. 7

mid = 7/2 = 3.5

↓

floorvalue = 3 i.e. '8'

② if key ≥ arr[mid] & ≤ arr[e]

s = m + 1

else

e = m - 1 .

e.g. 2

Pattern + Bubble + Selection

Q.

	0	1	2	3
4	☆	☆	☆	☆
3	☆	☆	☆	
2	☆	☆		
1	☆			

row, column

f(r, c)

f(4,0)

↓

proceed if 0 < 4

↓

f(4,1) 1<4

↓

f(4,2)

↓

f(4,3)

↓

new line, again start with 0.

row-1 ← (3,0)

south)

.....

illy for correct triangle → check code.

Q. What Arrays.copyOfRange() does?

→ Sol: 1 ctrl + click to check

Sol 2 Copies the specified range into a new array.

→ check code!. + 4 mostly check for merge in place code!

★ 2. (Continuing return ArrayList) Passing it in body (not Argument

↳ very Imp.

Goal: return the list but don't take in argument.

∴ Problem: Every call will have new arrayList.

e.g. arr[1,2,3,4,4,8], target = 4, index = 0

main → [3,4]

① (arr, target, 0)

[4,3] [list = []] → body of function

② (arr, t, 1)

[4,3] [list = []] → empty since, pehla 4 → 3rd index pe aayega.

③ (arr, t, 2)

[4,3] [list = []] → new lists are created at every function call.

④ (arr, t, 3)

[list = [3]] → 3 added in the list will be there for only this function; no dependency with past & future.

∵ here it is different obj for different function call

take mine [4] → using add all func.

⑤ (arr, t, 4)

[list = [4]]

(arr, t, 5) → so the answer is present in individual function call: Problem

[list = []]

return type list ∴ []

How to connect these ans? see red pen answers.

base condition if (index == arr.length){ return list; }

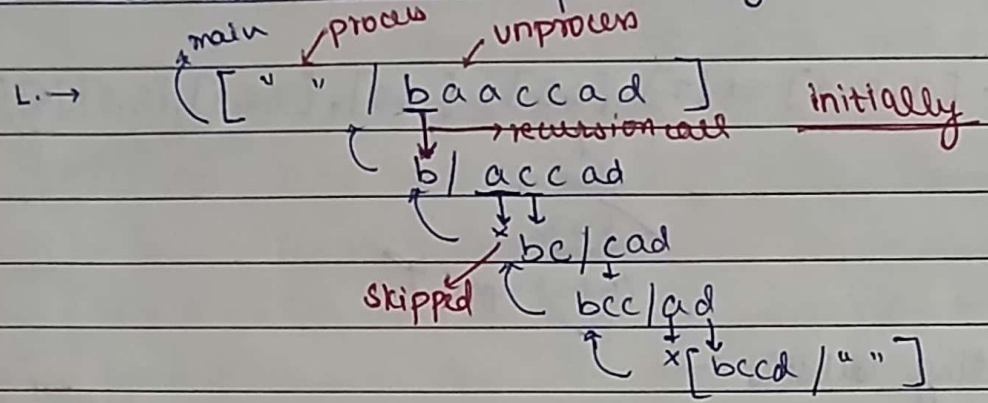
CHECK CODE!

Q. Remove 'a' from a string.

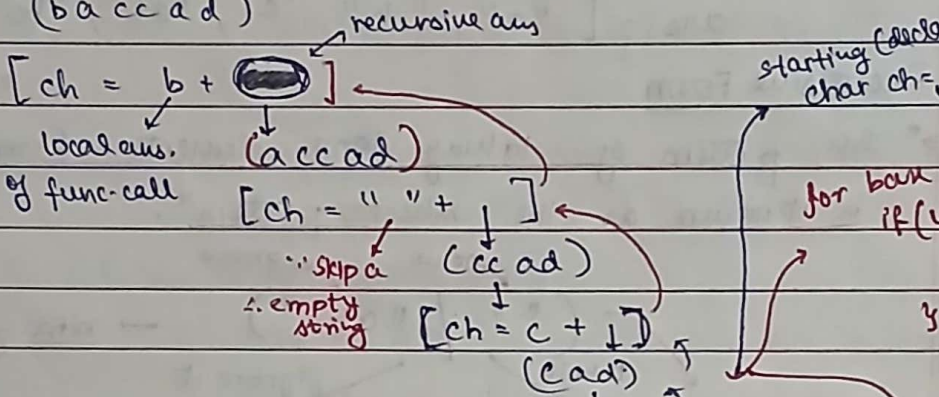
eg. str = baccad ans = bccd.

→ ① Pass the ans. string in argument → passed for future calls.

② Create ans. variable in func. body → new



2.→ (baccad)



→ substring: creates a new object & removes first character.

↳ same as slice method in J.S.

eg. String s = "0 1 2 3 4 5 6 7 8 9 10 Hello World";

sout(s.substring(6)); // World

sout(s.substring(0,5)); // Hello

[inclusive] exclusive

for 1 element always inclusive for it to end.

→ for skip :-

if(ch == 'a'){

skip(p, up.substring(1));

will remain as it is

now consider next index i.e. 1 to check (skips on index where a is)

Method: 2 using return type :- (take only up as arg)

base condition → ~~if(ch == 'a')~~ if(up.isEmpty){

return "";

if a → if(ch == 'a'){

return skipRet(up.substring(1));

}else{

return ch + skipRet(up.substring(1));

for not 'a' :-

else{

skip(ptch, up.substring(1))

}

added in process

NOTE :- To skip a word directly → e.g. Apple

↳ if(up.startsWith("apple")){

return skipApple(up.substring(5));

else{

return up.charAt(0) + skipApple(up.substring(1));

19/07/24

for Arrays ↗ for strings

* Subsets | Subsequence

✓ deals with permutations & Combinations

✓ Subsets → Non-adjacent collection.

e.g. [3,5,9] → [3], [3,5], [3,9], [3,5,9], [5,9], [5], [9]

[5,3] same as [3,5] (so dont repeat)

✓ Creating subsets :-

str = "abc"

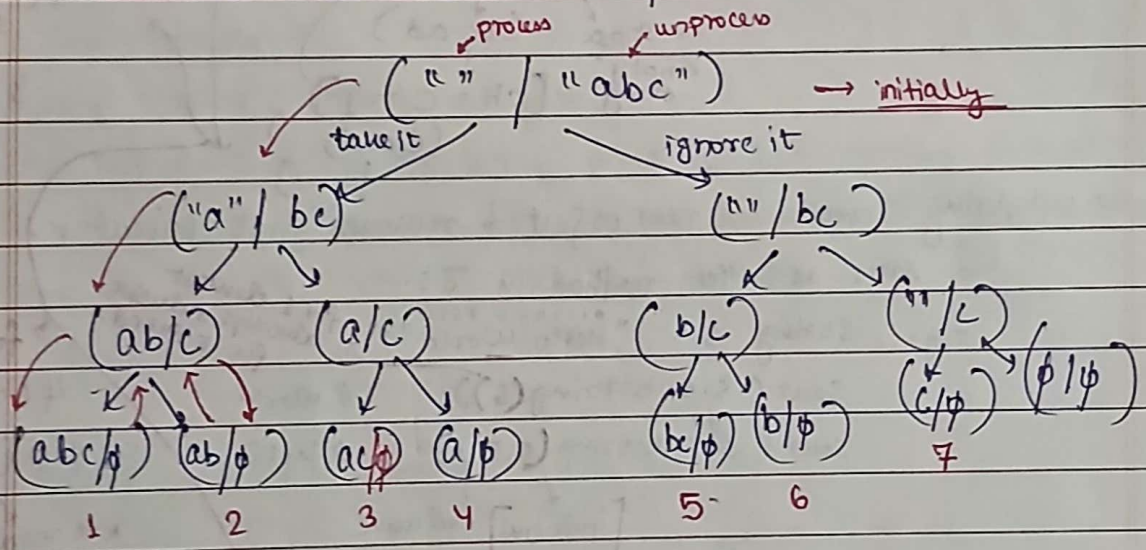
ans = ["a", "b", "c", "ab", "bc", "ca", "abc"]

ca: i.e either ac or ca not both

order can't be i.e. ca

I. Recursive Form

* ⇒ "This pattern of taking some elements & removing some is known as the subset pattern".



→ (no return type) base condition : if (up.isEmpty()){

dout(p);

return;

}

char ch = up.charAt(0);

take it :→ subseq(p+ch, up.substring(1));

ignore it :→ " (p, " ");

to print Ascii value → " (p+(ch+0), up.substring(1));

along with

→ for return type: ~~Arraylist of String~~ → 1. Pass it in argument
→ 2. Pass it in body

```
static ArrayList<String> subseqRet(String p, String up){
base cond^n →   if(up.isEmpty()){
                    ArrayList<String> list = new ArrayList<>();
                    list.add(p);
                    return list;
                }
                char ch = charAt(0);
```

check ArrayList return in body note

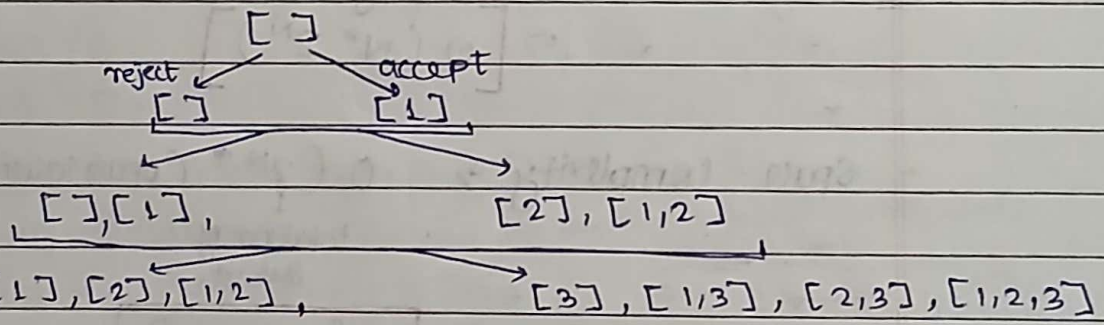
aus. from below calls { take it → `ArrayList<String> left = subseqRet(ch+p, up.substring(1));`
ignore it → `" " right = " (p, " );`

select both + add → `left.addAll(right);` / or `right.addAll(left);`
`return left;` / `return right;`
}

for ascii → `ArrayList<String> ascii = subseqRet(p+(ch+0), up.substring(1));`
`left.addAll(right);`
`left.addAll(ascii);`
`return left;`

II. Iterative Form

arr = [1, 2, 3]



Ques.

From above observation:- second half is added in the array.

list of list

⇒ static List<List<Integer>> subset(int[] arr){

bahar wala list
List<List<Integer>> outer = new ArrayList<>();

initially → [ ] ← outer.add(new ArrayList<>());

just an empty list

for(int num : arr){ → for every no. in my array

Jitna outer(bahar) list hai utna internal(right) list banao.

int n = outer.size();

for(int i=0; i<n; i++){

List<Integer> internal = new ArrayList<>();

COPY outer.get(i)

add the current no. you are at. ← internal.add(num);

for that make a copy of outerlist in internal list & add no. in it. (outer.get(i))

outer.add(internal); → now combine

}

}

return outer;

⇒ How to call?

int[] arr = {1,2,3};

List<List<Integer>> ans = subset(arr);

for every lists of list in answer → print that list

for(List<Integer> list : ans){

sout(list);

}

- Time complexity → O(N * (total subsets))

copy n elements

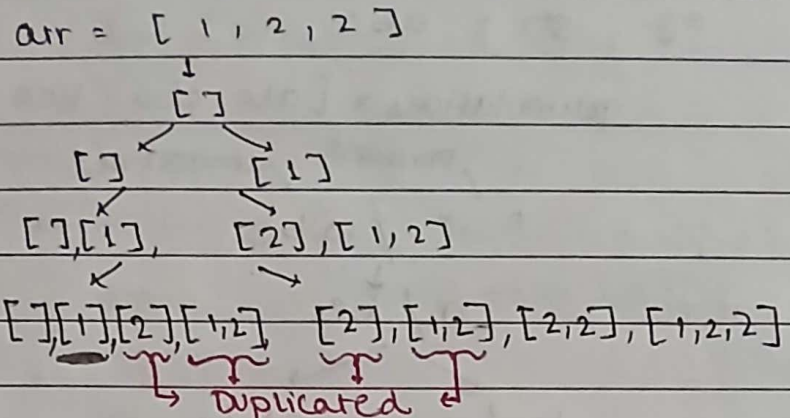
⇒ [O(N * 2^N)]

Space complexity → O(2^N * (space taken by every subsets))

total no. of subsets

⇒ O(2^N * N)

- Subsequence of strings with duplicate elements



How to solve this?

Soln → only add 2 in internal list (right one).

Hence, → when you find a duplicate element, only add it in the newly created subset (internal) of previous step.

But, condition → Duplicates have to be together.

↓

How to ~~solve~~ solve. this ~~not~~ problem now?

↳ Soln → SORT the array.

Chalk code!

Note:- Direct sorting of array in Java → Arrays.sort(arr);

```
int start = 0;
int end = 0;
for (int i = 0; i < arr.length ; i++){
    start = 0;
if current & previous element → if (i>0 && arr[i] == arr[i-1]){
same
                                    start = end + 1 ;
                                }
                                end = outer.size() - 1;
int n = outer.size() ;
for (int j=0 ; j<n ; j++){
List<Integer> internal = new ArrayList<>(outer.get(j));
internal.add(arr[i]);
return outer; } } outer.add(internal);
```

base condition (otherwise out of bound)

(end pehle outer loop ka last tha so ab start internal loop me start hoga rather outer loop ka start se)

* Permutations

e.g. str = abc

permutations = [abc, cba, bca, acb, ...]

processed | unprocessed

" " / abc → initially

a / bc

ba / c ab / c

cba/ϕ (1) bca/ϕ (2) bac/ϕ (3) cab/ϕ (4) acb/ϕ (5) abc/ϕ (6)

3! = 6 ✓

Different from rest others → Here the no. of function call is dependent on size of process.

variable no. of recursive calls.

And no. of func./recursive call is size of process + 1.

e.g. using for loop in _ba_

for(i=0; i<=2; i++){

[0,0) + ch + [0,2] , ba + ch + (0,1] , ba + ch + (0,2] + ch + (2,2]

(1+2]

⇒ ch+ba

Hence, first = p.substring(0,i)

second = p.substring(i,p.length())

p = first + ch + second.

Check

eg. abc → p = "" , up = "abc"

(1) up="abc" char ch = up.charAt(0); → ch='a'

f="" , i=0

s="" ""

p = ""+'a'+"" = 'a'

(2) up="bc" ch='b'

f=""

s="a"

P = ch+s = ba

(3) up="c" ch='c'

f="" (0,0)

s="ba"

P=cba

up="" → empty so(up) → return

up="c", p="ba" (earlier)

(4) ch='c'

Now for i=1.

```
static void permutations(String p, String up){
    if(up.isEmpty()){
        Sout(p);
    } return;
    Char ch = up.charAt(0); ✓
variable ← for(int i=0; i<=p.length(); i++){
rec. calls      String f = p.substring(0,i);
                String s = "" (i, p.length());
                permutations(f+ch+s, up.substring(1));
    }
```

⇒ do yourself for returning in case of ArrayList (by own)
↳ check screenshot if gave up.

→ check rec. level ⇒ Ques of counting zeros (with helper func.).

→ find counts ⇒

```
static int permutationsCount(String p, String up){
    if(up.isEmpty()){
        return 1;
    }
    int count = 0;
    char ch = up.charAt(0);
    for(int i=0; i<=p.length(); i++){
    String f = ✓""
      "" s = ✓""
    count = count + permutationsCount(f+ch+s, up.substring(1));
    }
    return count;
}
```

* Numpad Ques. GOOGLE Q.

Leetcode : Letter combination of a phone no.

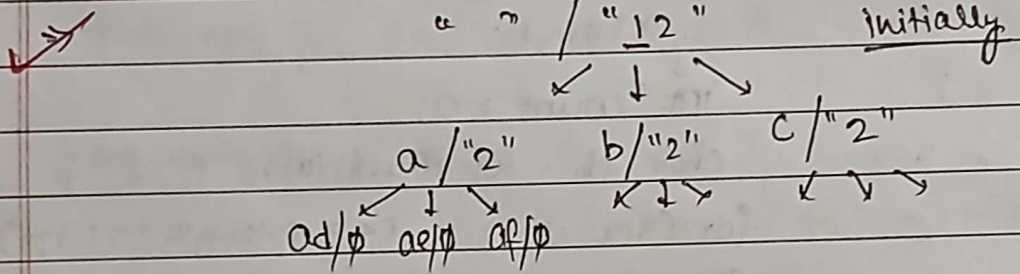
for simplicity, → (solve actual by your own)

1	2	3
abc 0 1 2	def 3 4 5	ghi 6 7 8
4 jkl	5 mno	6 pqr
7 stu	8 vwx	9 yz

e.g. "12"

ad, ae, af, bd, be, bf, cd, ce, cf

⇒ same as earlier concept of taking d & removing bc ← (process/unprocess).



⇒ range of each digit → [(digit-1)*3, digit*3) (index wise)

e.g. 5 → [12, 15) → 12, 13, 14. → m n o

e.g. 1 → [0, 3) → 0, 1, 2 → a b c

base cond$ → when index = 26 stop.

eg. here i will run from [0, 3) → 0, 1, 2 → a b c

to get character from no.
just do = 'a' + index

debug & check which is doing what

```
static void pad (String p, String up){
    if (up.isEmpty()){
        sout (p)
        return;
    }
    int digit = up.charAt(0) - '0';   → convert '2' into 2.
                                      convert up(String) to no. & store in digit.
    for(int i=(digit - 1)*3 ; i< digit*3 ; i++){
        char ch = (char)('a' + i);
        pad (p+ch, up.substring(1));
                                      ∴ up = "12" (String)
    }
}
```

converting first placed string into no.

Ofly do for list in arguments (easy)

In this ques. we've to return in list. Already covered just copy paste).

```
static ArrayList<String> padRet ("", "" ){
    if (up.isEmpty()){
        ArrayList<String> list = new ArrayList<>();
        list.add(p);
        return list;
    }
    int digit = up.charAt(0) + '0';
    ArrayList<String> list = new ArrayList<>();
    for (int i = ..... "" "" ){
        char ch = "" "";
        list.addAll (padRet (p+ch, up.substring(1));
    }
    return list;
```

Can also perform for count of pad.

Q. now 1 → no letter, abc from 2, 7 has pqrs & 9 has wxyz.
(22 23 24 25)

Start from 2. char ch = (char)('a'+i-3)

a=0, b=1, c=2

∴ (digit-1)*3 = 18 ∴ i=3

Char → 'a' + i-3

till 8 →

∴ digit*3 = 21

↳ to get 18 → i=3.

∴ we read p@18 ∴ [p=15] ∴ i=3 for 15 ∴ -3

pqrs

↳ Ofly for 9 → [24, 27] ∴ (-2)

∴ 8 → tuv (19 20 21)

↳ [21, 24]

∴ -2 4 i < digit*3 (not equal)

25/07/24

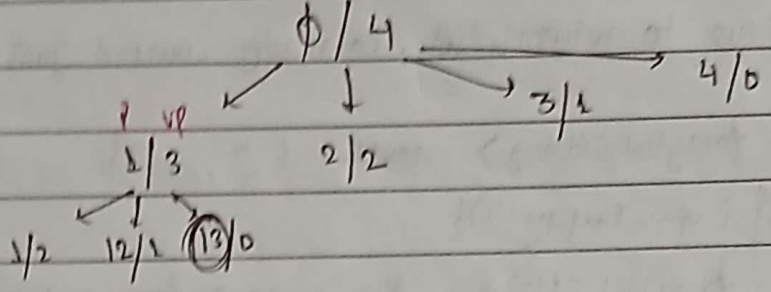
* Number on a Dice — AMAZON Q.

Die has → [1, 2, 3, 4, 5, 6] ⟶ some initial data

eg: 4 on a die possible combinations:-

[4, 13, 22, 31..] → taking something ignoring already used!

aim should be 4.



condition : i<= target

```
static void dice (String p, int target){
    if (target == 0){
        sout(p);
    } return;
    for (int i = 1 ; i<=6 && i<= target ; i++){
        dice( p+i , target-i );
    }
}
```

debug & check what's happening

target = 4, i = 3

" + 4 → 4+3=7 → 11

4 3 2 1 ∴

1 3 6 10

CHECK

⇒ Ily, do for returning via list. (yourself)

⇒ If die face is more than 6.

↳ pass another argument of int face.

↳ for(int i=1 ; i<= face && i<= target ; i++){
    diceFace (p+i , target - i , face);
}

Date 25/07/24
Page

* Backtracking

- Maze Problem

	0	1	2
0	S		
1			
2			D

can move only Right & down.

Ans. ⇒ RRDD
DDRR
RDDR
⋮

Q. How many ways of going from (0,0) to (2,2)?

Breaking down into ~~diff~~ smaller approaches :-

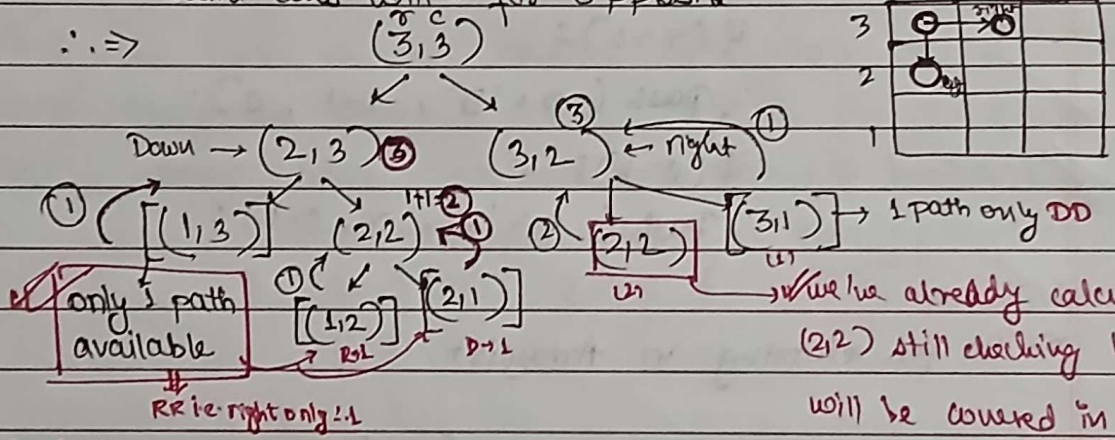
- if you are ~~this~~ at last column → only possible way is to Down
- " " " " " row → " " " " " Right.

⇒ if RD + ( )
DR + ( ) → Path till that point.

For no. of ways :-

⇒ Note :- same ans. will be for opposite.

∴ ⇒

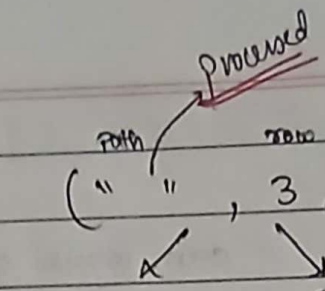


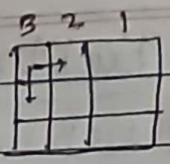

```
static int count (int r, int c){
    if (r == 1 || c == 1){
        return 1;
    }
    int left = count(r-1, c);   DOWN
    int right = " (r, c-1);     Right
    return left + right;
}
```

How (memoization + tabulation)

Gives no. of ways.

For Path :→



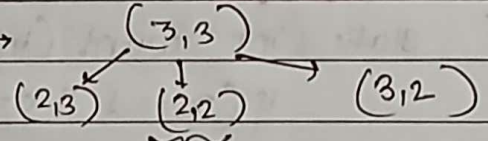


Ans. [when both row & col = 1] ← base cond^{n}.

```
static void path (String p, int r, int c){
    if (r==1 && c==1){
        sout(p); // earlier for count it was "ans"
        return;
    }
    if (r>1){
        path(p+'D', r-1, c);
    }
    if(c>1){
        path(p+'R', r, c-1);
    }
```

⇒ Do for returning in ArrayList.

⇒ For going diagonally →

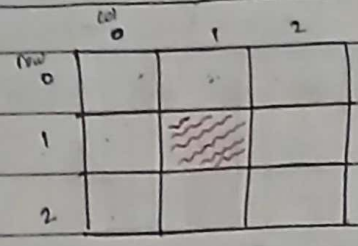


i.e. you can't go diagonal ← condition: (r>1 && c>1)
when you are at last col
and row.

↳ just add one more condition → `if(r>1 && c>1){ path(p+'X', r-1, c-1); }`
★ ★

Date 25/04/24

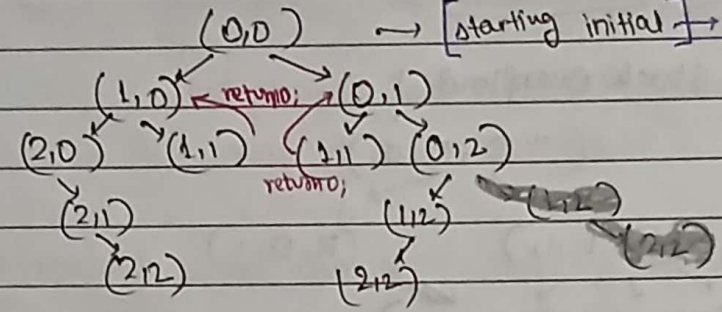
- Maze with obstacles



(false → river) → Boolean

Note:- when you land on a new cell, check whether that is river or not.

If you land on river, stop recursion for that call. (return from river)



Now this time we are starting from (0,0) to (2,2) earlier we were doing the opp.

⇒

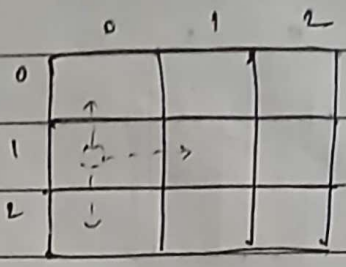
```
static void pathRestrictions(String p, boolean[][] maze, int r, int c){
    if (r == maze.length-1 && c == maze[0].length-1){
        Sout(p);
    }  → reached at the end of maze.
    return;

    if (maze[r][c] == false){
    }
    return;

    if (r < maze.length-1){
        pathRestriction(p+'D', maze, r+1, c);
    if (c < maze[0].length-1){
        "path" (p+'R', maze, r, c+1);
    }}
```

Create board in main :→ boolean[][] board = { {true, true, true},
{ ", false, "},
{ ", true, "}
};

pathRestrictions("", board, 0, 0);

15/09/24

Maze for all Directions



Left, right, up, down.

If you are going:-

up ⇒ r-1, c (condition: row > 0)

left ⇒ r, c-1 (" : col > 0)

right & down → we know.

result: Stack Overflow !

reason:→ ("", 0, 0)

(D, 1, 0) (R, 0, 1)

(DD, 2, 0), (DR, 1, 1), (DU, 0, 0)

Problem ⟶ again reached (0,0) Hence, never ending!

∴ Sol:⟶ Do not move back the same path.

All cells that are visited, mark those as false, so that it does not go there.

↳ same concept as we did in maze with obstacle.

∴ put all those (row, col) as false which are already visited

Another Problem → To humlog (DD, 2, 0) ki chale the false kar-kar ke wo to false reh gya, So when we traverse from (DR, 1, 1) ∴ uske baad down aane pr false marked hai but ye upar kabhi gaye hi nhi still!

* ∴ So when previous path is over & now you're in another rec^n. call, then these cells should not be false.

Solution:-

⇒ While you're moving back, you restore the maze as it was.

↓

So, when do we go back? → when fn. is returned

when you come out of recursive func. → you're now in above recursion call. Hence, while going back( ) → remark the cells as true!

THIS IS KNOWN AS BACKTRACKING

↓

(what if I had not taken this path what would my Array look like).

• You're making some changes while going from below recursive calls, so when you go outside that recursion calls, changes that were made via that recursion calls should also not be available. This is known as Backtracking :→ Thought process: make a change, reverse the change when that work is done.

↳ check all paths code!!

Print all paths with numbers

e.g.

1		
2	5	6
3	4	7

- Take a step variable.
- Update the path array.
- print it in base cond^k.

⇒ Backtracking (as array, integer) DDRURD

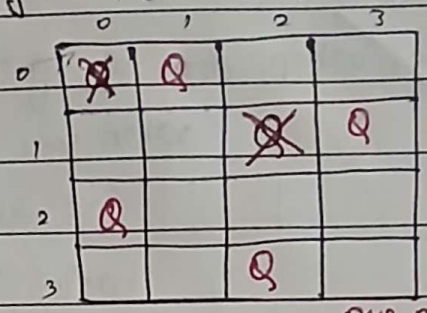
- pass path & step in argument.
- with 'every' rec^k. call step+1.
- initially path[r][c] = step, after(for) backtrack → path[r][c] = 0;
- for base cond^k → for every path in arr → print (last step) that array.

Note: in main: declare path as array of size board! → check code.

* N-Queen Problem

"Place 2 Queens on board so that no 2 Queens with their path eliminate each other."

eg. N=4 (place 4 Queens)



↳ one of the ~~two~~ answer.

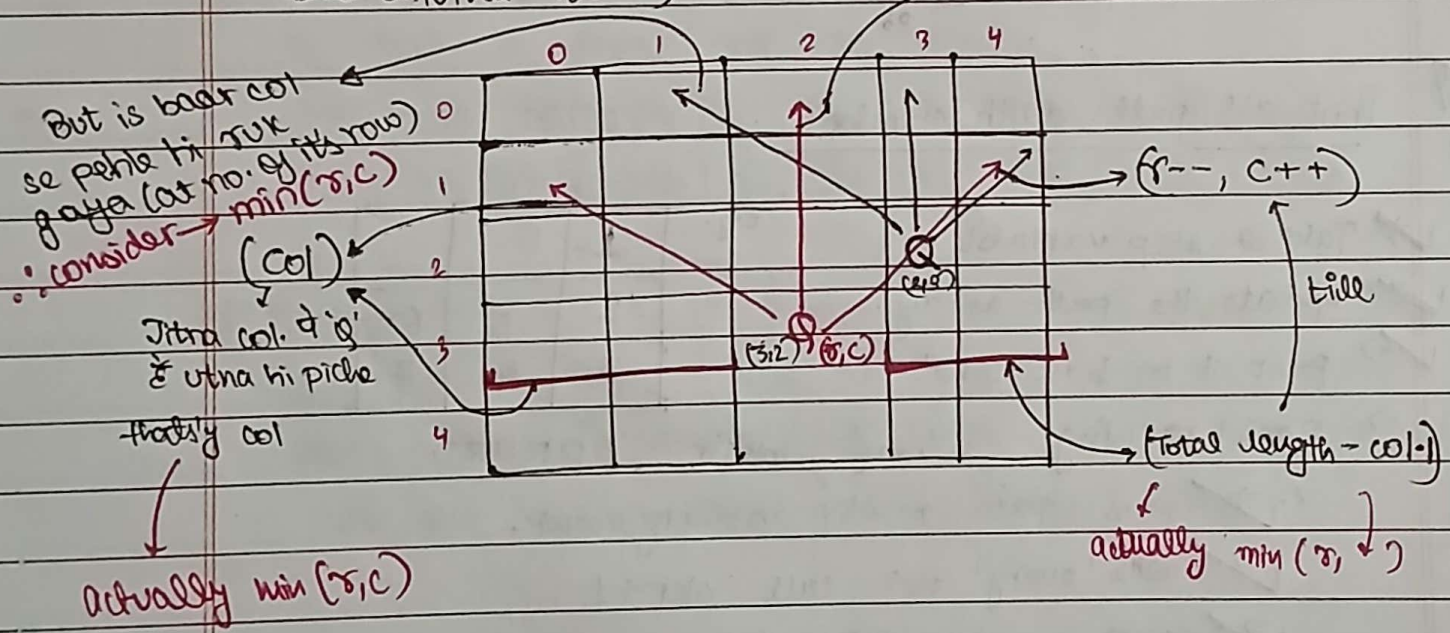
How, recursion? → checking again & adding/ignoring.

How, Backtracking? → (what changes you're making in that ~~rec~~ rec^{n}. call when that ~~the~~ call is over those changes should also be gone).

here, (1,2) was not a feasible path so Q placed there should remove.

∴ putting there we can't put 4 Queens

⇒ some imp. checks e.g. if Q is in center of 3×3 matrix then no other Queen will be placed i.e. (no need to check ~~#~~ another rows)



⇒ static int queens (boolean[][] board, int row){

all 'Q' placed ← if (row == board.length){

display (board);
sout(); → add a new line after displaying board for another
return 1;
}

int count = 0; → local variable

placing the queen & checking for each row & col. → for (int col=0; col< board.length; col++){

if (isSafe(board, row, col)){ → place 'Q' if safe

board [row][col] = true; → placed, now so below ones can be placed or not

count = count + queens (board, row+1);

coming out of func. calls 'Backtrack' → board[row][col] = false;
}
}
return count;
}

static boolean isSafe (boolean[][] board, int row, int col){

Check vertical row→[0,r] → for (int i =0; i< row; i++){

if (board[i][col]) → mtlb if this is true = board[i][col] == true

return false;

Not safe to place it ∵ its a 'Q' over there

}
}

for max. time you can go diagonal left min (r,c) → int maxLeft = Math.min (row, col);

for (int i = 1; i<= maxLeft; i++){

if (board[row-i][col-i]{

return false

}}

Lastly, diagonal right (length - col - 1) with (r--, c++) → int maxRight = Math.min (row, board.length-col-1);

for (int i=1; i<= maxRight; i++){

if (board[row-1][col-1]{

return false;

}}

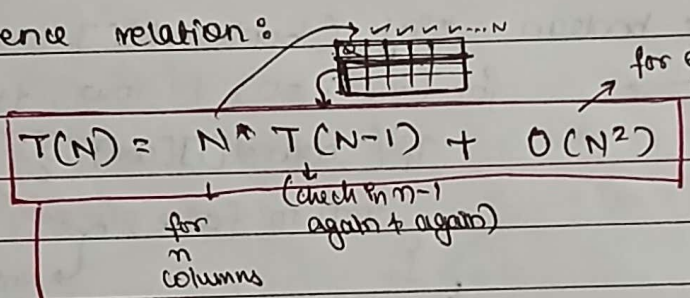
If none of the check returns false ∴ ans. is true → return true;
}

display ⇒ static void display (boolean[][] board){
for every row on board ⇒ for (boolean[] row : board){
for (boolean element : row){
if (element){
sout ("Q");
}else{
sout ("x");
}
}
sout(); → next line
}

Main ⇒ int n = 4; → no. of Q's. → on 8×8 or 4×4 board
boolean[][] board = new boolean[n][n];
sout (queens (board, 0));
}
↳ row initially.

→ Space complexity O(n×n).

→ recurrence relation:



Solve Time Complexity using Akra-Bazzi formula.

↓

$O(N^3 + N!) = \boxed{O(N!)}$ Ans

→ Note:- For these problems: you can eliminate for loops with conditions, but then you need another variable in argument.

e.g. (board, row, col, target)

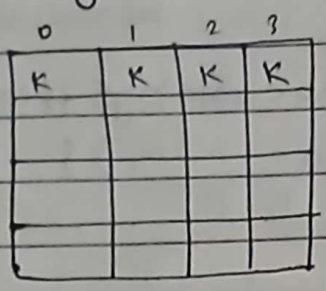
eg. Ans. found when target = 0.

(0,0,4)

(1,0,3)

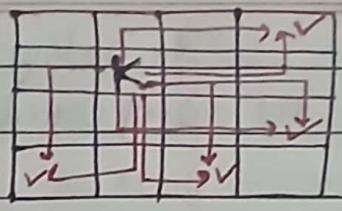
Date 26/07/24
Page

* N-Knights Problem



rows cols knights to be placed

(0,0,4)
↓
(0,1,3)
↓
(0,2,2)
↓
(0,3,1)
↓
(0,4,0)
↳ ∴ one of the ans. is found



Now, whenever you reached end → try for a new line.

∴ (0,0,4) → (0,1,3), (0,2,2), (1,1,1) ←

safe ⇒ [r-2, c-1 ; r-2, c+1 ; r-1, c+2 ; r-1, c-2]

⇒
```
static void knight(boolean[][] board, int row, int col, int knights){
    if(knights == 0){
        display(board); ⟶ make display same as of queen
        sout()
        return;
    }
    if(col == board.length){ → if reached end of col, increase row by 1 & check for another.
        knight(board, row+1, 0, knights);
        return;
    }
    if(isSafe(board, row, col)){
        board[row][col] = true;
        knight(board, row, col+1, knights-1);
        board[row][col] = false;
    }
    knight(board, row, col+1, knights);
}
```

make if reach end of row & col just return

place it when safe & col++

Backtracking →

If not safe ∴ make → ahead & do not reduce knight ∵ you didn't placed it

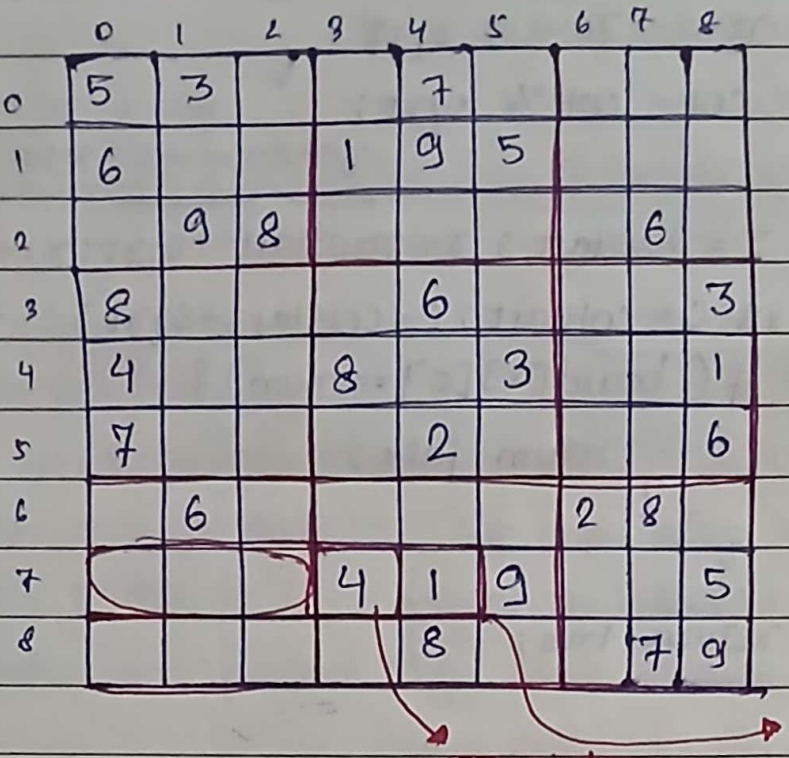
for checking places are → static boolean isValid (boolean[][] board, int row, int col){
not out of bounds in Matrix

```
if (row>=0 && row < board.length && col>=0
                && col < " " ) {

} return true;
return false;
}
```

for checking safe in → static boolean isSafe (" , " , "){
valid places only.

```
if (isValid(board, row-2, col-1)){
    if(board[row-2][col-1] ){
        return false;
    }
}
if (isvalid( " , row-1 , col-2){
    " " [row-1][col-2]) {
        " "
    }}
" " " " (" , row-2 , col+1){
    " " (row-2][col+1]) {
    " "
" " " " (" , row-1 , col+2){
    " " [row-1][col+2]) {
    " "
}
return true; → if all not safe valid.
}
```

* Sudoku Solver



Rules: You know ✓

We know how to check rows & col but for each block.

eg. I'm at (7,4) we've to find starting of the box where '1' lies.

⇓

$x - x \% 3$

$y - y \% 3$

$\Rightarrow 7 - 7 \% 3 = 6$

$4 - 4 \% 3 = 3$

$\Rightarrow (6,3)$ ✓

eg. we've at (6,8)

Starting $\Rightarrow$ $6 - 6 \% 3 = 6$

$8 - 8 \% 3 = 6$

$\Rightarrow (6,6)$ ✓

remainder thats'y this formula.

and 3 bcoz $\sqrt{9}$

to check if same no. already here

```
static boolean isSafe(int[][] board, int row, int col, int num){
```

1. Check the row → `for(int i=0; i<board.length; i++){`
   Check if the no. is in the row → `if(board[row][col] == num){`
   `return false;`
   `}}`

2. Check the col → `for(int[] nums: board){`
   Check if the no. is in the col (by checking diff. row of same col) → `if(nums[col] == num){`
   `return false;`
   `}}`

3.
```
int sqrt = (int)(Math.sqrt(board.length));
int rowStart = row - row % sqrt;
int colStart = col - col % sqrt;
```
e.g. 9 → √9 = 3 ek block में 3×3 का ही matrix ban payega

```
    for (int r = rowStart; r < rowStart + sqrt; r++){
        for (int c = colStart; c < colStart + sqrt; c++){
            if (board[r][c] == num) {
                return false;
            }
        }
    }
    return true;
}
```

Now,
```
static boolean solve(int[][] board){
    int n = board.length;
    int row = -1;
    int col = -1;
    boolean emptyLeft = true;
```

This is how we're replacing r,c from arguments.

```
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            if (board[i][j] == 0){
                row = i;
                col = j;
```

r value stored there ←
```
                emptyLeft = false;
                break;
            }
        }
```

If some empty element is found in row ~~as already present~~ →
```
        if (emptyLeft == false){
            break;
        }
```

if all filled (sudoku solved) →
```
    if (emptyLeft == true){
        return true;
    }
```

backtrack → for (int number = 1; number <= 9; number++){

if(isSafe(board, row, col, number)){

put no. if safe board[row][col] = number;

print board if answer ← if (solve(board) == true) or (solve(board)){

"found answer" ← display(board);

return true;

}else{

backtrack (pichle-jaise ← board[row][col] = 0;

ke bata te jao) jab tak }

sahi wala no. sahi }

place pe na ho) ← return false;

sudoku can't be solved ← }

display → static void display (int[][] board){

for every row in board → for (int[] row : board){

" " no. in row → for (int num : row){

sout(num + " ");

}

sout(); → new line (arranged)

}}

main → int[][] board = new int[][]{

{3, 0, 6, . . .

⋮

};

~~if(solve(board))~~ sout(solve(board));

~~sout(solve~~

if(solve(board)){

display(board);

}else{

sout("cannot solve");

}}

- Complexity: Total 9 nos.
  for every no. → n^2 possibility.

Time : $O(9^{n^2})$, Space : $O(N^2)$