Functions ⇒ Takes input and give some output.

$f(a, b) = a^2 + 2ab + b^2$

$f(x) = 2x + 1$

$f(n) = 2n + 1$

$x = 5$ → input

input = 5

output = 11

output $2 * 5 + 1 \Rightarrow 11$

```
public static int sum ( int a, int b) {
    int sum = a+b;
    return sum;
}

public static void fun ( int a) {

}
```

return type → int

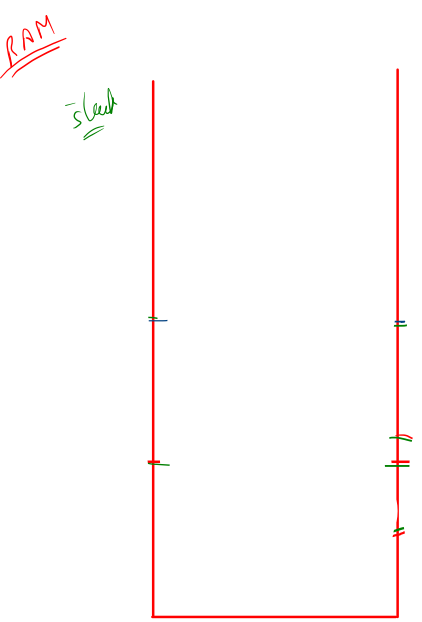
parameters / arguments

no return type → void

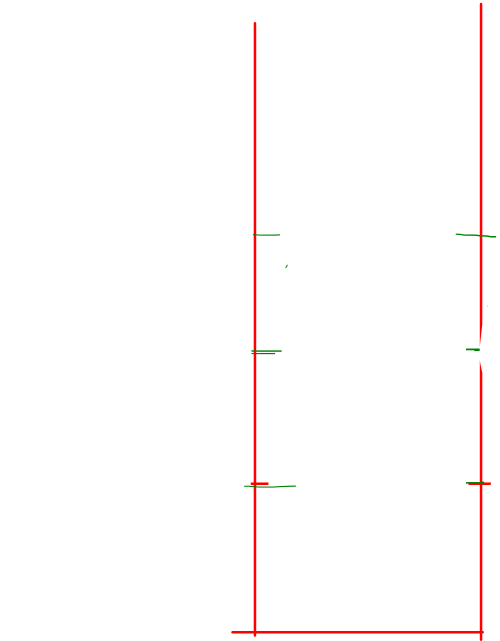
sum (2, 3) = 5

sum (1, 4) = 5

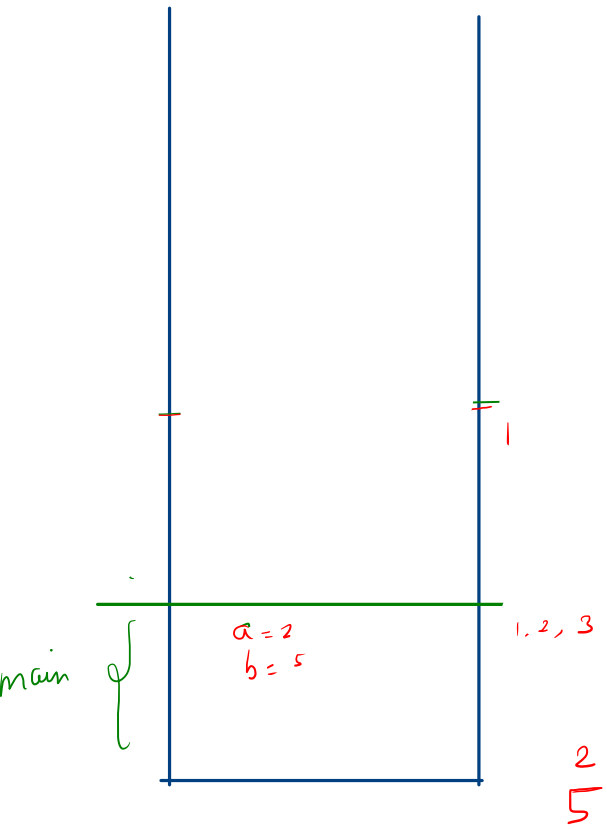
sum (9, 11) = 20



```java
class Main {
    public static int sum(int a, int b){
        int sum = a + b;
        return sum;
    }
    Run | Debug
    public static void main(String[] args) {
        int a= 2;
        int b= 3;

        int ans = sum(a,b);
        System.out.println(ans);
    }
}
```



```java
class Main {
    public static int sum(int a, int b){
        int sum = a + b;
        return sum;
    }

    public static void fun(int a, int b){
        int sum = a+b;
        System.out.println(sum);

        System.out.println("I am inside a void function");
        System.out.println("I am inside a void function");
    }

    public static void main(String[] args) {
        int a= 2;
        int b= 3;

        fun(a,b);
    }
}
```




```java
class Main {
    public static void swap(int a, int b){
        int temp = a;
        a = b;
        b = temp;
    }
    Run | Debug
    public static void main(String[] args) {
        int a= 2;
        int b= 5;

        swap(a, b);

        System.out.println(a);
        System.out.println(b);
    }
}
```

Recursion

⇒ How to solve any recursion problem

1) Keep a faith that it works for a smaller problem. (faith)
2) Solve for the smallest problem. {base case}
3) Solve for the actual problem (to reach your expectation)

Ques Print decreasing numbers from 'n' to 1 using recursion.

n = 5

expectation
5
4
3
2
1

```
public static void printDecreasing(int n){
    System.out.println(n);
    printDecreasing(n-1);
}
```

Faith (n-1)

function will print numbers from (n-1) to 1

(n=5)

4
3
2
1

work →

Expectation (n)

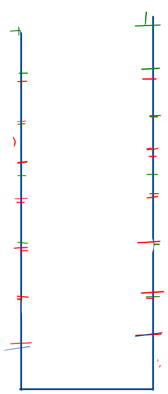
function will print numbers from 'n' to 1

(n=5)

5
4
3
2
1

you won't be able to learn recursion without dry-run

stack overflow



Expectation

(n=5)

5
4
3
2
1

faith (n-1)

4
3
2
1

```
class Main {
    public static void printDecreasing(int n){
      1 if(n==0){
            return;
        }
      2 System.out.println(n);
      3 printDecreasing(n-1);
    }
    Run | Debug
    public static void main(String[] args) {
        int n = 5; 3
        printDecreasing(n);
    }
}
```

3
2
1

Ques Print numbers in increasing order using recursion.

Faith (n-1)

It will work for (n-1)
print numbers from 1 to 'n-1'

1
2
3
4

5

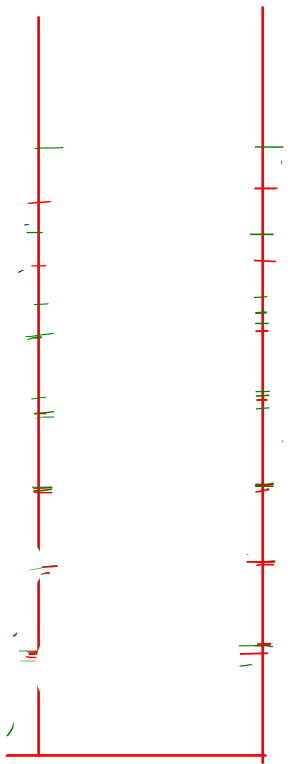
```
psv printIncreasing(int n){

    printIncreasing(n-1);
    System.out.println(n);

}
```

Expectation (n)

Function will print numbers from 1 to n

n=5

1
2
3
4
5



```java
public static void printIncreasing(int n){
    if(n==0){
        return;
    }
    printIncreasing(n-1);
    System.out.println(n);
}
Run | Debug
public static void main(String[] args) {
    int n = 3;
    printIncreasing(n);
}
```

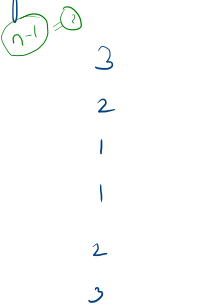
1
2
3

Ques Print numbers in decreasing order, then increasing using recursion.

n=5

Faith (n-1)

It will print numbers from (n-1) to 1 and then 1 to (n-1)



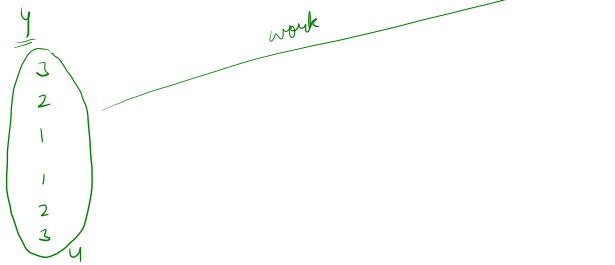

```
psv pdi(int n){
    System.out.println(n);
    pdi(n-1);
    System.out.println(n);
}
```

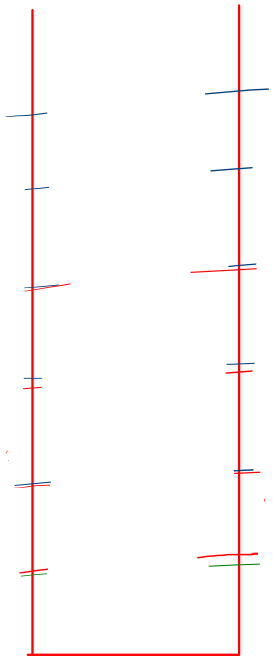
System.out.println(n); → fails

Expectation

It will print numbers in decreasing then increasing order

n=4





```java
public static void pdi(int n){
    if(n==0){
        return;
    }
    System.out.println(n);
    pdi(n-1);
    System.out.println(n);
}
Run | Debug
public static void main(String[] args) {
    int n = 4;
    pdi(n);
}
```

Ques Find factorial of given number 'n'.

Faith (n-1)

It will return factorial (n-1)

1 x 2 x 3 x 4

smallAns

```
public static int fac(int n){
    int smallAns = fac(n-1);
    int ans = smallAns * n;
    return ans;
}
```

faith

work

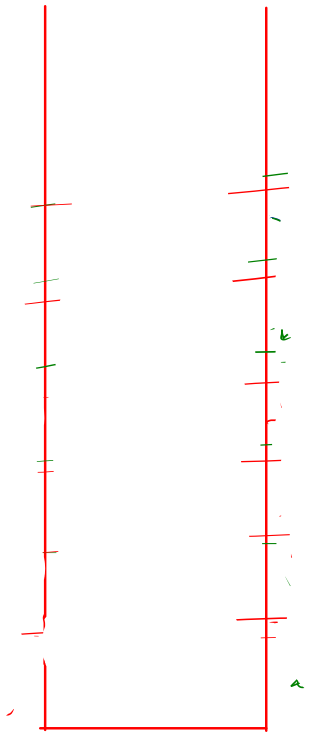
Expectation

It will return factorial of given number

n = 5

1 x 2 x 3 x 4 x 5

work



24

```
public static int factorial(int n){
    if(n==0){
        return 1;
    }
    int smallAns = factorial(n-1);
    int ans = smallAns * n;
    return ans;
}
Run | Debug
public static void main(String[] args) {
    int n = 4;
    int ans = factorial(n);
    System.out.println(ans);
}
```

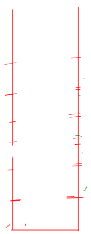
Ques: Find x^n using recursion

Faith: $(n-1)$ It will give x^{n-1}

public static int power(int x, int n){
int smallAns = power(x, n-1);
int ans = smallAns * x;
return ans;
}

Expectation: It will return x^n. $x=2, n=4$

2x2x2 → 2x2x2x2



```java
public static int power(int x, int n){
    if(n==0){
        return 1;
    }
    int smallAns = power(x, n-1);
    int ans = smallAns * x;
    return ans;
}
Run | Debug
public static void main(String[] args) {
    int x = 2;
    int n = 4; 3
    int ans = power(x,n);
    System.out.println(ans);
}
```

Faith $(n/2)$

It will calculate $x^{\frac{n}{2}}$ correctly

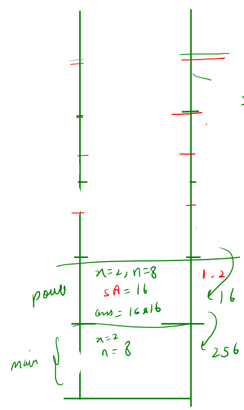
$x=2$ $n=6$

$2 \times 2 \times 2$ (smallAns)

public static int power(int x, int n){
int smallAns = power(x, $\frac{n}{2}$);
int ans = smallAns * smallAns;
if (n%2 != 0){
ans = ans * x;
}
return ans;
}

Expectation: It will return x^n. $x=2$, $n=6$

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



3

ans = 2*2 ⇒ 4

ans = 16 * 16 ⇒ 256

```java
public static int power2(int x, int n){
    if(n==0){
        return 1;
    }
    int smallAns = power2(x, n/2);
    int ans = smallAns * smallAns;

    if(n%2 != 0){
        ans = ans*x;
    }

    return ans;
}
Run | Debug
public static void main(String[] args) {
    int x = 2;
    int n = 6;
    int ans = power2(x,n);
    System.out.println(ans);
}
```

$O(\log n)$

$x^{\frac{n}{2}}$ x^n

$2^8 \Rightarrow 2^{\frac{8}{2}} * 2^{\frac{8}{2}}$
$\Rightarrow 2^4 * 2^4$
$\Rightarrow 2^8$

$2^7 \Rightarrow 2^{\frac{7}{2}} * 2^{\frac{7}{2}}$
$\Rightarrow 2^3 * 2^3$
$\Rightarrow 2^6 * 2$

$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \rightarrow \frac{n}{16}$ - - - - - - 1

$O(n)$ $O(n)$

$\frac{n}{2^0} \rightarrow \frac{n}{2^1} \rightarrow \frac{n}{2^2} \rightarrow \frac{n}{2^3} \rightarrow \frac{n}{2^4} \rightarrow \frac{n}{2^5} \rightarrow \frac{n}{2^6}$ - - - 1 (x steps)

$\frac{n}{2^x} = 1$

$n = 2^x$

$2^x = n$

$\log 2^x = \log n$

$x \log 2 = \log n$

$x \log_2 2 = \log_2 n$

$x \cdot 1 = \log_2 n$

$x = \log n$

$\log a^b = b \log a$

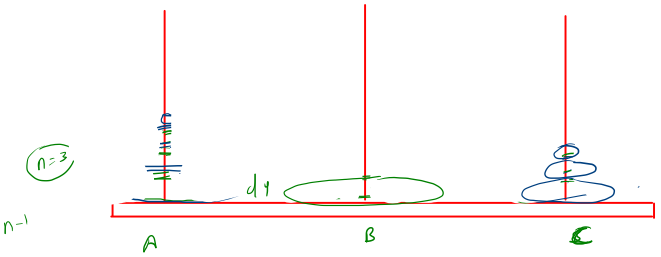
$\log_a a \Rightarrow 1$

$O(\log n)$

Ques Tower of Hanoi

shift n discs from tower A to tower B with the help of tower C

→ you can move one disc at a time
→ you can not put a longer disc on top of a smaller disc



Faith

move (n-1) disc from A → C using B (n-1 steps to move 2 disc from A to C)

→ move nth disc from A to B (move 3rd disc from A → B)

move (n-1) disc from C to B (move 2 disc from C → B)

psv TOH(n, A, B, C) ✓

→ TOH(n-1, A, C, B) → faith

move nth disc from A to B

→ TOH(n-1, C, B, A)

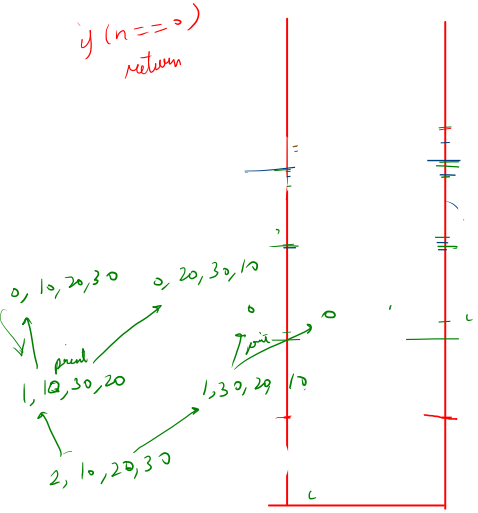
Expectation ✓ ns

function will move n discs from 'A' to 'B' using 'C'

n=3

write steps to move 3 disc from A to B

if (n==0) return

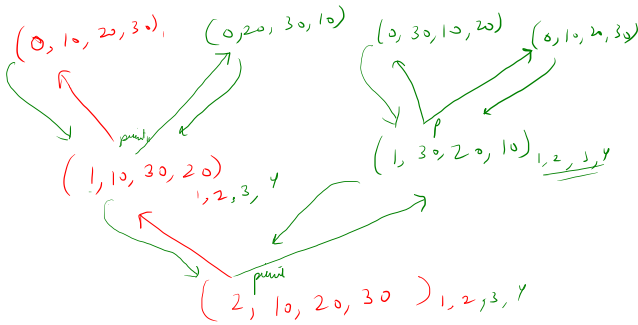



```java
public static void towerOfHanoi(int n, int A, int B, int C){
    if(n==0){
        return;
    }

    towerOfHanoi(n-1, A, C, B);
    // move nth disc from A to b
    System.out.println("Moving ring " + n + "from" + A + " to " + B + " using " + C);
    towerOfHanoi(n-1, C, B, A);
}

Run | Debug
public static void main(String[] args) {
    int n = 3;
    towerOfHanoi(n, A: 10, B: 20, C: 30);
}
```

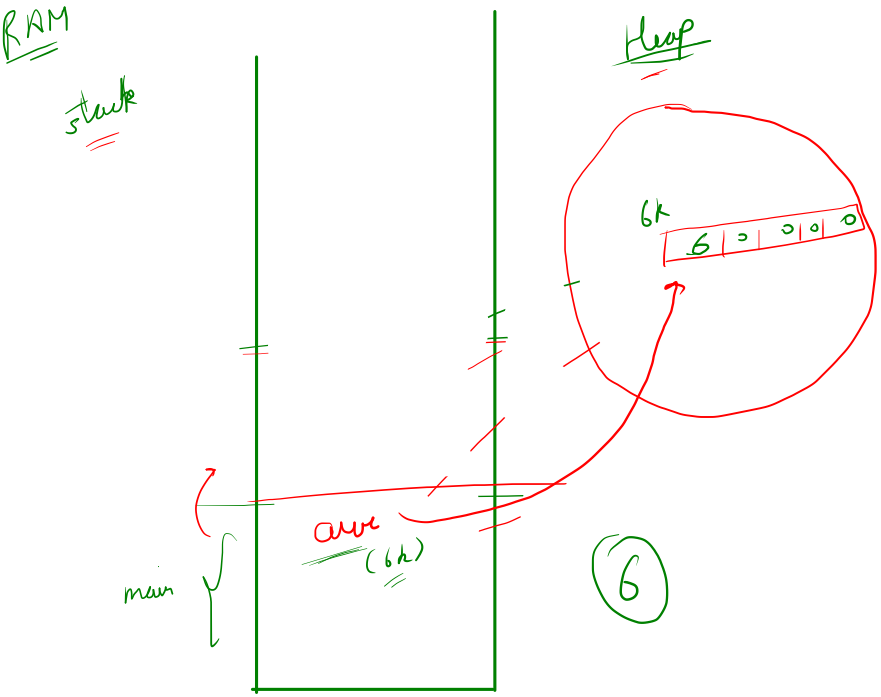
Moving ring 1 from 10 → 30 using 20
Moving ring 2 from 10 → 20
Moving ring 1 from 30 → 20

recursive tree diagram

→ color path



Recursion with Arrays




```java
public class Main {
    public static void increaseArray(int[] arr){
        arr[0]++;
    }

    public static void increaseInteger(int a){
        a++;
    }
    Run | Debug
    public static void main(String[] args) {
        int[] arr = new int[1];
        arr[0] = 5;

        int a = 5;
        increaseArray(arr);
        increaseInteger(a);

        System.out.println(arr[0]);
        System.out.println(a);
    }
}
```

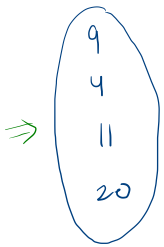
int [] arr = new int [5]

Ques Print array using recursion.

Faith (smaller area)

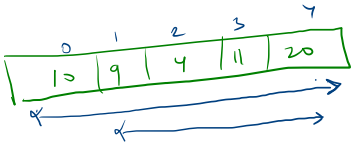
It will print array from idx+1 to end

1 → 4



psv

```
printArray(int[] arr, int idx){
    System.out.println(arr[idx]);
    printArray(arr, idx+1);
}
```

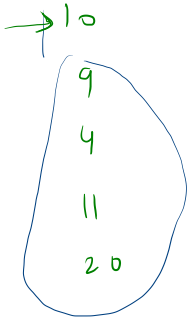


printArray(arr, 0)

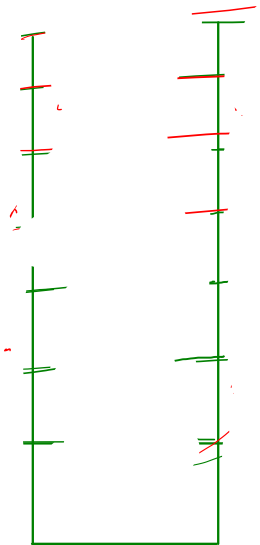
Expectation

function will print array from 'idx' to end (n-1)

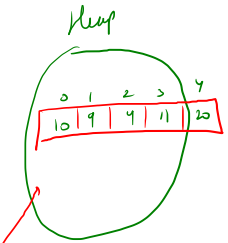
from 0 → 4



Flow



Stack



10
9
4
11
20

```java
public static void printArray(int[] arr, int idx){
    if(idx == arr.length){
        return;
    }

    System.out.println(arr[idx]);
    printArray(arr, idx+1);
}
Run | Debug
public static void main(String[] args) {
    int[] arr = {10,9,4,11,20};
    printArray(arr, 0);
}
```

Ques Print array in reverse

Faith

It will print from (idx+1) to end in reverse.

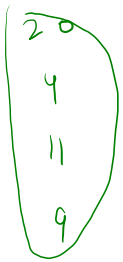
PSV

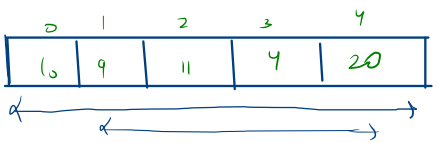
```
printReverse(int[] arr, int idx){
    printReverse(arr, idx+1);
    System.out.println(arr[idx]);
}
```

Expectation

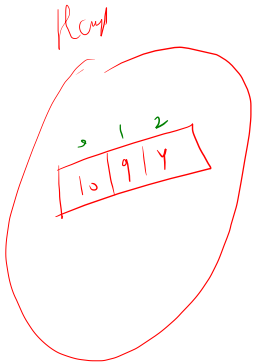
It will print from 'idx' to end in reverse

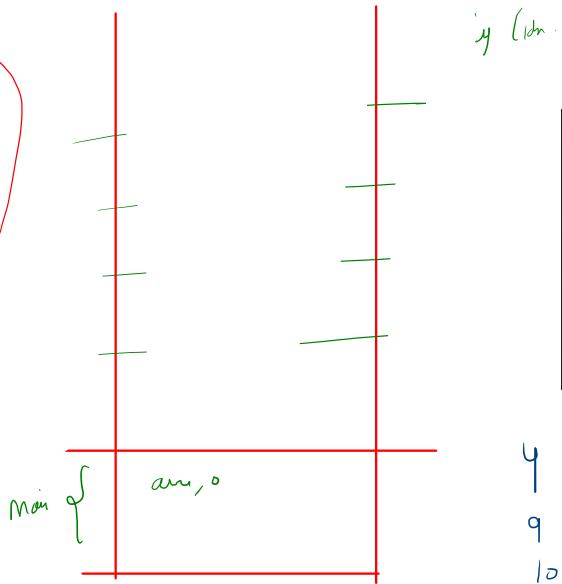
0 → 4 in reverse





printReverse(arr, 0)





if (idx == arr.length)
return;

```
public static void printReverse(int[] arr, int idx){
    printReverse(arr, idx+1);
    System.out.println(arr[idx]);
}
Run | Debug
public static void main(String[] args) {
    int[] arr = {10,9,4,20};
    printReverse(arr,idx: 0);
}
```

4
9
10

Ques Find max of array using recursion.

Faith
It will return max of a smaller array.

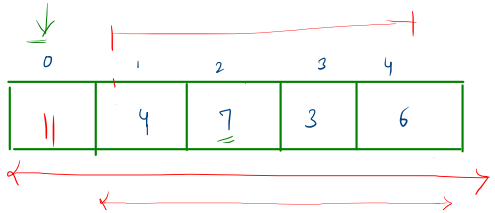
smallAns = 7

ps int maxofArray(int[] arr, int idx){

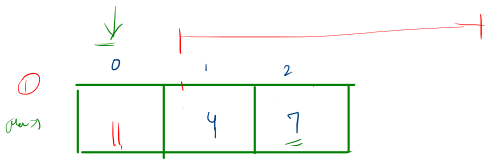
int smallAns = max(arr, idx+1) →
int ans = Math.max(arr[idx], smallAns);
return ans;

}

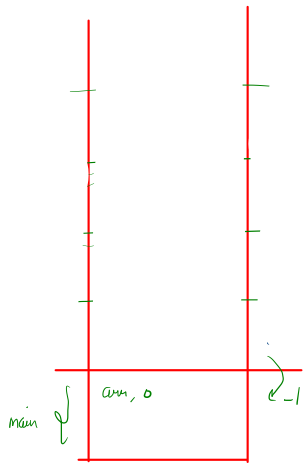
Expectation
It will return 'max' from 'idx' to end.
idx = 0 to n-1
ans = 11



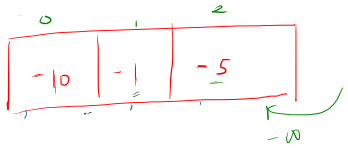
max(arr, 0)
0 → end
idx+1 to end




```
public static int maxofArray(int[] arr, int idx){
    if(idx == arr.length){
        return 0; Integer.MIN_VALUE
    }

    int smallAns = maxofArray(arr, idx+1);

    int ans = Math.max(arr[idx],smallAns);
    return ans;
}
Run | Debug
public static void main(String[] args) {
    int[] arr = {10,9,4,11,20};
    System.out.println(maxofArray(arr,idx: 0));
}
```

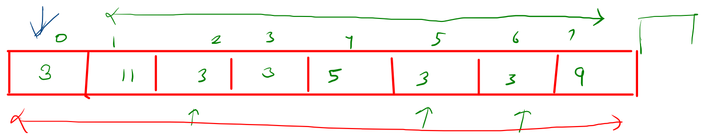


arr =>



Ques Find first index of given target element.

tar = 3



fun (arr, 0, 3) = 0

fun (arr, 8, 3)

Faith

It will return first index of tar from 'idx+1' to end

sA = 2

public static int firstIndex (int[] arr, int idx, int tar) {

int smallAns = fi(arr, idx+1, tar)

if (arr[idx] == target)
    ans = idx;
else
    ans = smallAns;

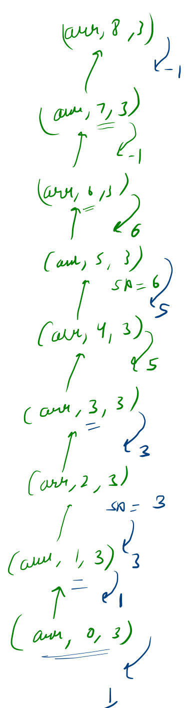
return ans;

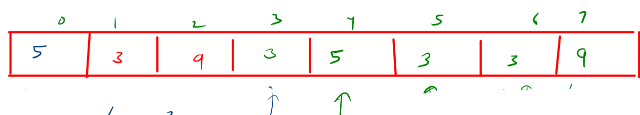
}

Expectation

It will return the first index of the given tar element.

ans = 0





tar = 3

```java
public static int firstIndex(int[] arr, int idx, int tar){
    int smallAns = firstIndex(arr, idx+1, tar);
    int ans = 0;

    if(arr[idx] == tar){
        ans = idx;
    } else {
        ans = smallAns;
    }

    return ans;
}
Run | Debug
public static void main(String[] args) {
    int[] arr = {10,9,4,11,20};
    System.out.println(maxofArray(arr,idx: 0));
}
```

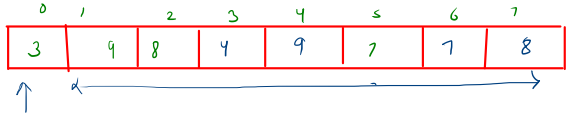
recursion on the way down

smallest problem

if (idx == arr.length)
    return -1

Ques Find last index of tar

tar = 3



Faith

from 'idx+1' to end it will find the last index of tar

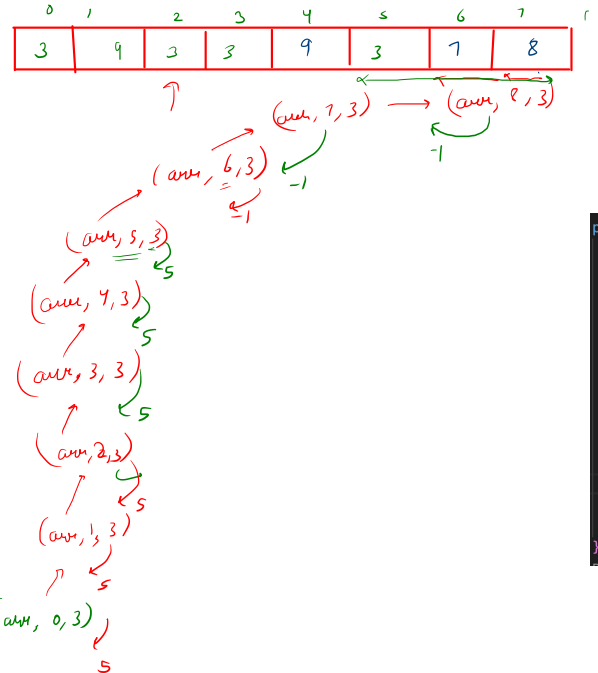
smallAns = -1

```
ps int lastIndex(int[] arr, int idx, int tar){
    int smallAns = lI(arr, idx+1, tar);

    int ans = 0;
    if(smallAns == -1){
        if(arr[idx] == tar)
            ans = idx;
    } else ans = smallAns;

    return ans;
}

lastIndex(arr, 0, tar);
```

Expectation

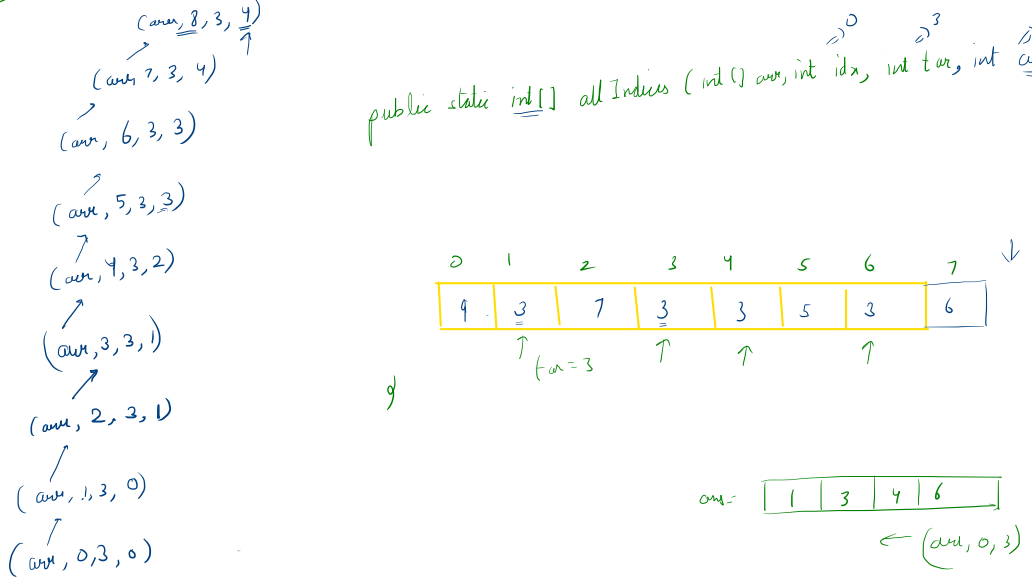
from '0' to end, it will return last index of target.

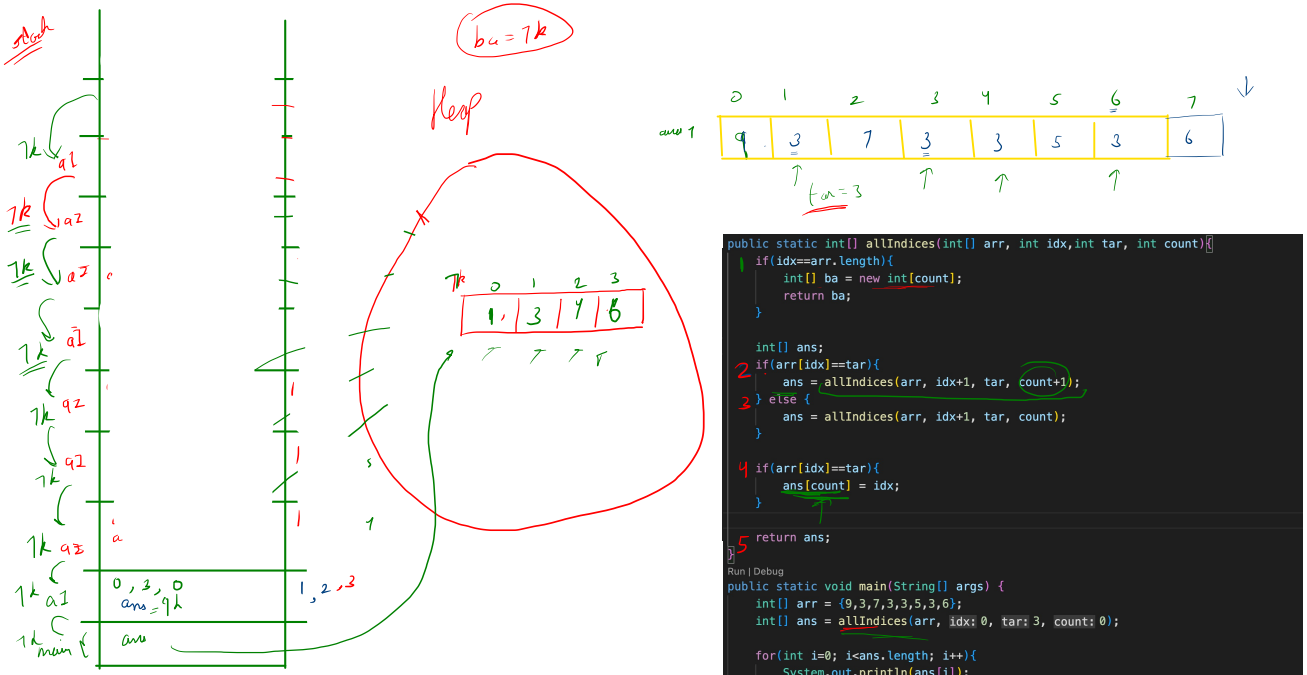
ans = 5




```java
public static int lastIndex(int[] arr, int idx, int tar){
    int smallAns = lastIndex(arr, idx+1, tar);

    int ans = 0;
    if(smallAns != -1){
        if(arr[idx]==tar){
            ans = idx;
        } else {
            ans = smallAns;
        }
    } else {
        ans = smallAns;
    }

    return ans;
}
```

Q) Find all indices of tar in array






```java
public static int[] allIndices(int[] arr, int idx,int tar, int count){
    if(idx==arr.length){
        int[] ba = new int[count];
        return ba;
    }

    int[] ans;
    if(arr[idx]==tar){
        ans = allIndices(arr, idx+1, tar, count+1);
    } else {
        ans = allIndices(arr, idx+1, tar, count);
    }

    if(arr[idx]==tar){
        ans[count] = idx;
    }

    return ans;
}
Run | Debug
public static void main(String[] args) {
    int[] arr = {9,3,7,3,3,5,3,6};
    int[] ans = allIndices(arr, idx: 0, tar: 3, count: 0);

    for(int i=0; i<ans.length; i++){
        System.out.println(ans[i]);
    }
}
```

Ques Find nth fibonacci number

0th	1	2	3	4	5	6	7	8	9	10	11	12
0	1	1	2	3	5	8	13	21	34	55	89	144

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

$f(6) = f(5) + f(4)$

(n-1)

Fourth

n=6

n-1 = 5

lt = 5

fib(n-1);

fib(n-2);

```
public static int fib ( int n) {
    int lt = fib(n-1);
    int lst = fib(n-2);
    int ans = lt + lst;
    return ans;
}
```

temp estation

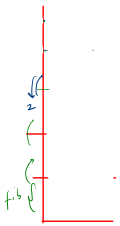
It will calculate nth fibonacci term.

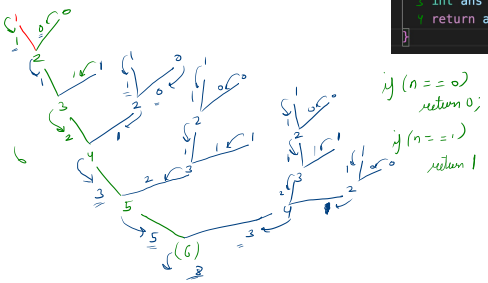
n=6

ans = 8

ans = 1+1 = 2

lt = 2



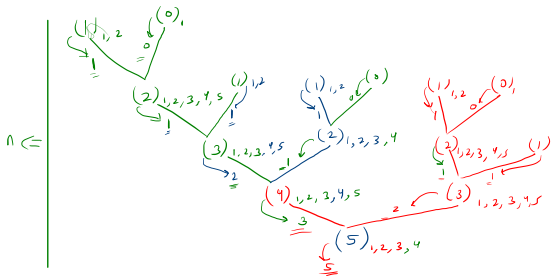


```
if (n == 0)
    return 0;
if (n == 1)
    return 1
```

```
// fibonacci number
public static int fib(int n){
    int lt = fib(n-1);
    int lst = fib(n-2);

    int ans = lt + lst;
    return ans;
}
```

TC => (number of calls) height of tree

2^n



```
// fibonacci number
public static int fib(int n){
    if(n==0){
        return 0;
    }
    if(n==1){
        return 1;
    }
    int lt = fib(n-1);
    int lst = fib(n-2);

    int ans = lt + lst;
    return ans;
}
```

Ques

nth even fibonacci number.

9-3 ⇒ n-3 → last even fib num
9-6 ⇒ n-6 → second last

0th	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	(2)	3	5	(8)	13	21	(34)	55	89	(144)	233	377	(610)

1st even fib number ⇒ 3rd fib number.
2nd even fib number ⇒ 6th fib number
3rd even fib number ⇒ 9th fib number.
4th even fib 〃 ⇒ 12 th fib

nth even fib number ⇒ 3 * n fib number

f(n-3) → last even fib num
f(n-6) → last 6 even fib ⇒

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

$f(n-2) + f(n-3) + f(n-3) + f(n-4)$

$f(n-3) + f(n-4) + f(n-3) + f(n-3) + f(n-4)$

$3*f(n-3) + f(n-4) + f(n-4)$

$3f(n-3) + f(n-5) + f(n-6) + f(n-4)$

$\Rightarrow 3f(n-3) + f(n-5) + f(n-4) + f(n-6)$

$\Rightarrow 3f(n-3) + f(n-3) + f(n-6)$

even

$f(n) \Rightarrow 4 f(n-3) + f(n-6)$ ⇒ nth even fib number ⇒ 4 * (last even fib number) + (second last even fib number)

$f_E(n) \Rightarrow 4 f_E(n-1) + f_E(n-2)$

$f_E(n-1)$ → last even fib number

$f_E(n-2)$ → Second last even fib number.

%

$10^9 + 7$ ↑ largest prime number (int)

x = 8 (on number line from 1 to 10)

ans % 10

ans = x + 4 ⇒ ans = 12

12 % 10 ⇒ 2

ans = n % 50

7

n % 2 < 0, 1

n % 9

18 % 9 = 0

9 % 9 ⇒ 0

27 % 9 ⇒ 0

3 % 9 ⇒ 3

last even ⇒ 10^9
last 〃 ⇒ 10^8

ans ⇒ $4*10^9 + 10^8$
⇒ $4*10^9$

last even ⇒ $7*10^9$
⇒ $3*10^9$

7×10^9

10^{10}

int range

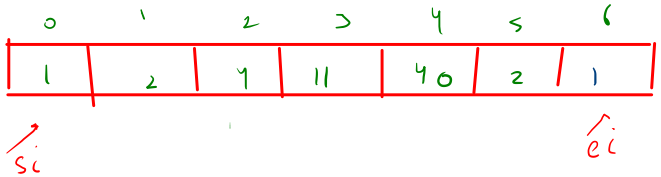
-2^{31}

$2^{31} - 1$ → 10^9

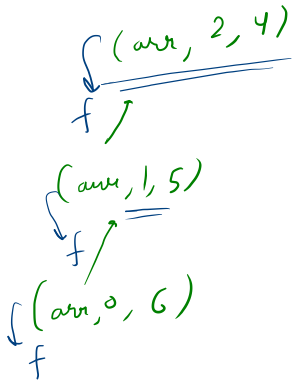
0 ———— $2^{31}-1$

10^{10} % (mod)

Ques Check whether the given array is palindrome or not.




```java
public static boolean isPalindromic(int[] arr, int si, int ei) {
    if(arr[si]!=arr[ei]){
        return false;
    }

    boolean ans = isPalindromic(arr, si + 1, ei - 1);
    return ans;
}
```



Dynamic array

data type

```
ArrayList<String> al = new ArrayList<>();
```

Dynamic array

size

```
al.size()
```

→ 3

add

```
al.add("abc")
al.add("def")
al.add("abcd")
```

get

```
al.get(idx);
```

	0	1	2
arr →	10	13	19

remove

```
al.remove(idx)
al.remove(1)
```

set

```
al.set(idx, value);
```

set 45 at 1st index

arr[1] = 45

Strings

str = "a b c d e"

substring => Continuous segment of string.

subsequence => discontinuous segment of string

"a b c d"

2^n (n = len of string)

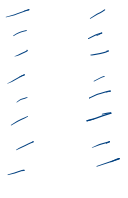
0 len: ""

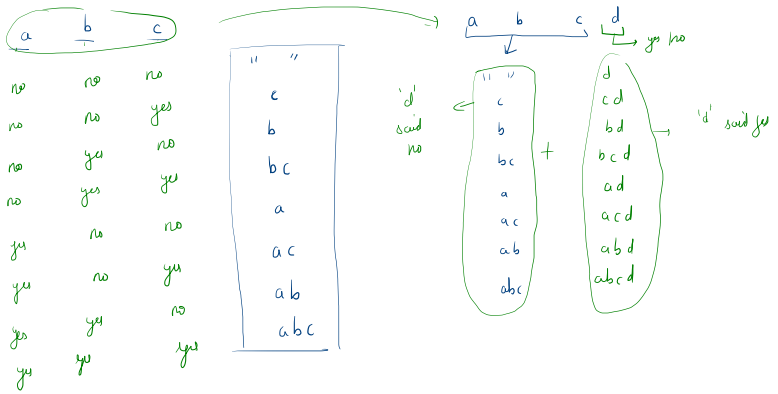
1 len: a, b, c, d

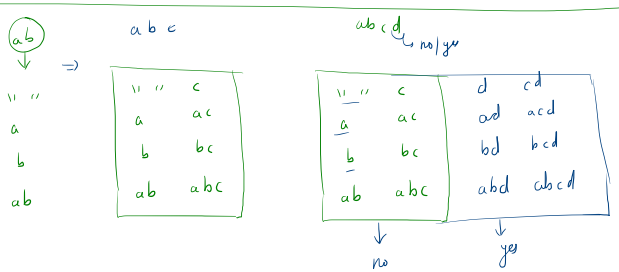
2 len: ab, ac, ad, bc, bd, cd

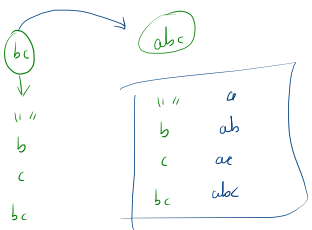
3 len: abc, abd, acd, bcd

4 len: abcd









Ques Get all the subsequences of a given string.

Faith (smaller value)

It will return all the subsequences for 'bc'

smallAns = {"", b, c, bc}

public static ArrayList<String> getSS(String str){

abc

Expectation

It will return all the subsequences of given string

str = "abc"

ans = {"", a, b, c, ab, bc, ac, abc}

work?

("")

{""}

(c)

{"", c}

(bc)

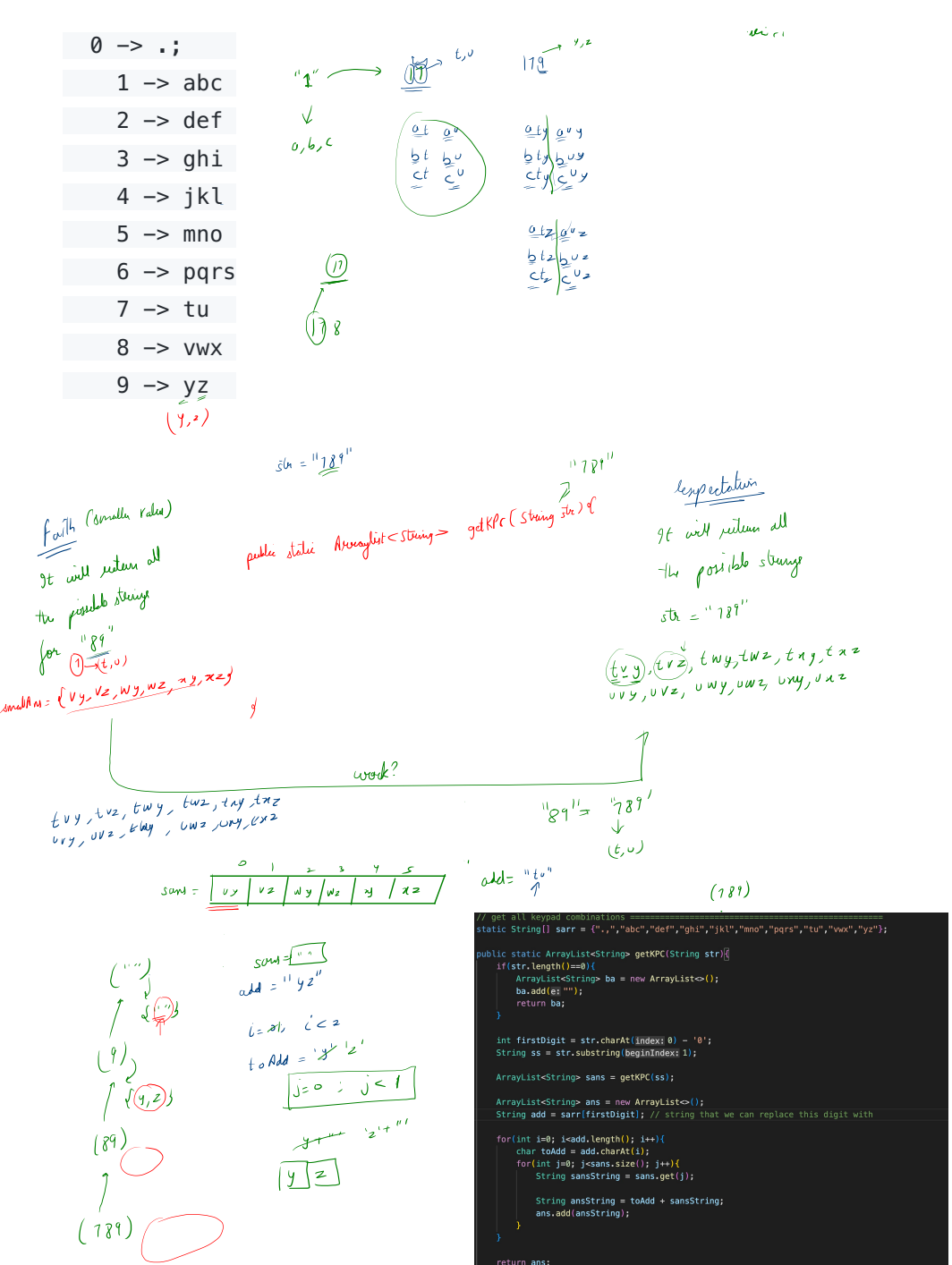
{"", c, b, bc}

(abc)

{"", c, b, bc, a, ac, ab, abc}

```java
public static ArrayList<String> getSS(String str){
    char fc = str.charAt(index: 0);
    String ss = str.substring(beginIndex: 1);

    ArrayList<String> sans = getSS(ss);

    ArrayList<String> ans = new ArrayList<>();
    // copy sans
    for(int i=0; i<sans.size(); i++){
        String ssub = sans.get(i);
        ans.add(ssub);
    }

    // add first char
    for(int i=0; i<sans.size(); i++){
        String ssub = sans.get(i);
        String sub = fc + ssub;
        ans.add(sub);
    }

    return ans;
}
```

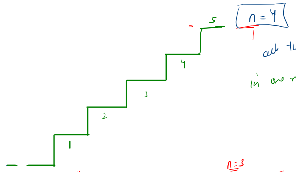
0 -> .;
1 -> abc
2 -> def
3 -> ghi
4 -> jkl
5 -> mno
6 -> pqrs
7 -> tu
8 -> vwx
9 -> yz



Ques Get all the stair paths.

path to climb

1 step ⇒ (n-1) [4 steps



all the path to get to nth step

in one move → 1 step
2 step
3 step

n=4
→ n=3 → 4
n=2 → 2
n=1 → 1

1 1 1 1 | 1 1 3
1 1 2
1 2 1
1 2 1 1
1 2 2
1 3 1

n = 4
1 1 1 1 | 1 3
1 1 2
1 2 1
2 1 1
2 2
3 1

n=3
1 1 1
1 2
2 1
3

n=2
1 1
2

n=1
1

n=5
→ (n-1)
(n-2)
(n-3)

2 step → paths to climb 3 steps.

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

3 step → paths to climb 2 steps

3 1 1
3 2

Faith
It will find all the possible path for (n-1) (n-2) (n-3)

public static ArrayList<String> getStairPath (int n) {

Expectation
It will return all the possible paths
n = 4

n=3
sans1 = { "111", "12", "21", "3" }

n-2 (2)
sans2 = { "11", "2" }

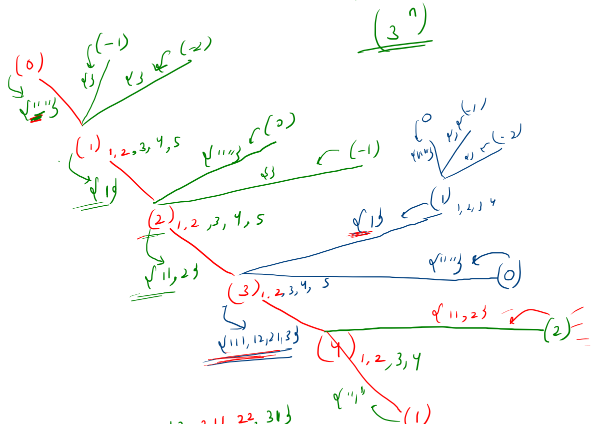
n-3 (1)
sans = { "1" }

number of path

(number of calls)

height of tree

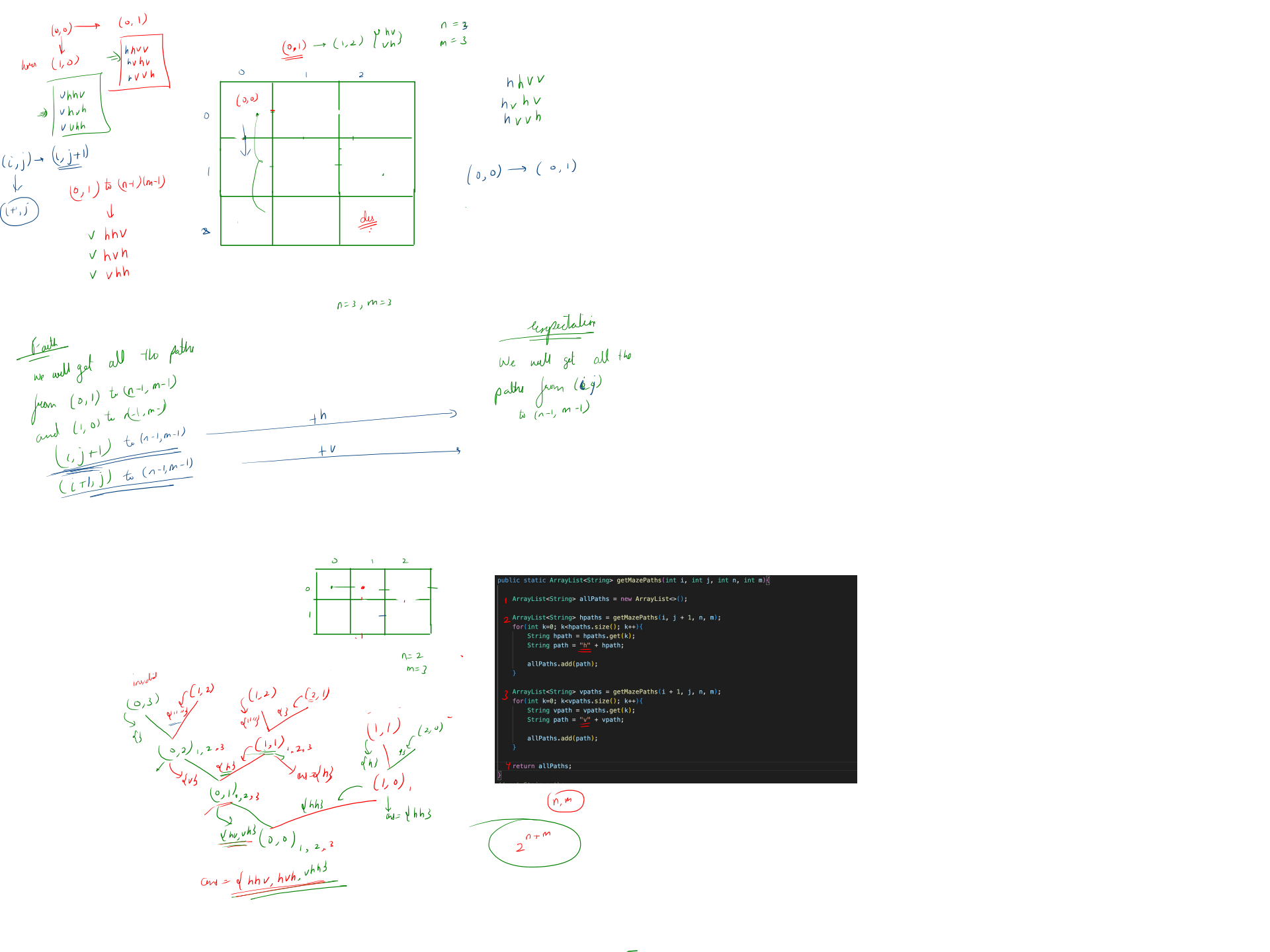
(3^n)



ans = {1111, 112, 121, 13, 211, 22, 31}

1
2

```java
public static ArrayList<String> getStairPaths(int n){
    if(n<0){ // no possible path
        ArrayList<String> ba = new ArrayList<>();
        return ba;
    }

    if(n==0){
        ArrayList<String> ba = new ArrayList<>();
        ba.add(e: ""); // not doing anything , just standing here (reached my destination)
        return ba;
    }
    ArrayList<String> ans = new ArrayList<>();
    ArrayList<String> sans = getStairPaths(n-1);

    for(int i=0; i<sans.size(); i++){ // take a jump of 1
        String spath = sans.get(i);
        String path = "1" + spath;
        ans.add(path);
    }

    sans = getStairPaths(n-2); // take a jump of 2
    for(int i=0; i<sans.size(); i++){
        String spath = sans.get(i);
        String path = "2" + spath;
        ans.add(path);
    }

    sans = getStairPaths(n-3); // take a jump of 3
    for(int i=0; i<sans.size(); i++){
        String spath = sans.get(i);
        String path = "3" + spath;
        ans.add(path);
    }

    return ans;
}
```



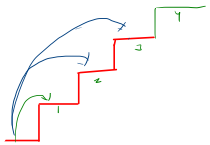

```java
public static ArrayList<String> getMazePaths(int i, int j, int n, int m){

1 ArrayList<String> allPaths = new ArrayList<>();

2 ArrayList<String> hpaths = getMazePaths(i, j + 1, n, m);
  for(int k=0; k<hpaths.size(); k++){
      String hpath = hpaths.get(k);
      String path = "h" + hpath;

      allPaths.add(path);
  }

3 ArrayList<String> vpaths = getMazePaths(i + 1, j, n, m);
  for(int k=0; k<vpaths.size(); k++){
      String vpath = vpaths.get(k);
      String path = "v" + vpath;

      allPaths.add(path);
  }

4 return allPaths;
}
```

(n, m)

2^{n+m}

Ques Get all stair paths.

⇒ Find all the number of paths

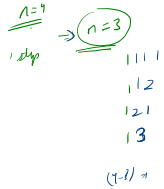


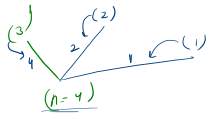
n → n-1 (added 1 in front of it)

↳ n-2 added 2 in front

↳ n-3 added 3 in front.

n=4 ⇒ 7





Ques Find the number of stair paths.

Faith

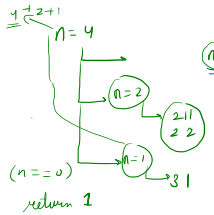
It will return number of paths for (n-1), (n-2) and for (n-3).

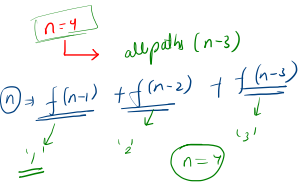
ps int noSp(int n){

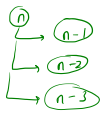
Arraylist<String>

Expectation

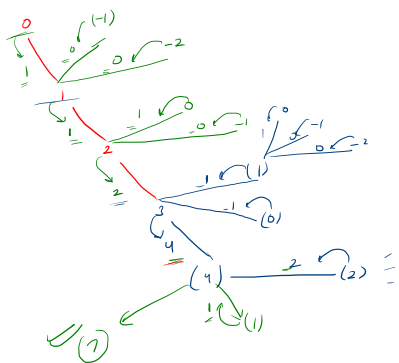
It will get all the number of paths.

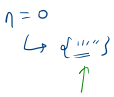






```
public static int numberOfPaths(int n){

    int numberOfPathsAfterOneStep = numberOfPaths(n-1);
    int numberOfPathsAfterTwoStep = numberOfPaths(n-2);
    int numberOfPathsAfterOneThreeStep = numberOfPaths(n-3);

    int numberOfPaths = numberOfPathsAfterOneStep + numberOfPathsAfterTwoStep + numberOfPathsAfterOneThreeStep;

    return numberOfPaths;
}
Run | Debug
public static void main(String[] args) {
    System.out.println(numberOfPaths(n: 4));
}
```



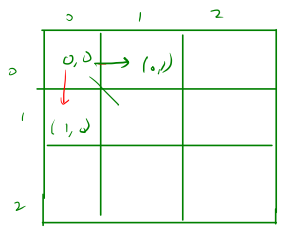


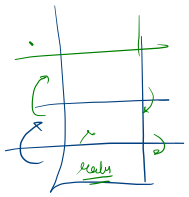
Ques Find all the different paths from (0,0) to (n-1, m-1);

(0,0) → (0,1)
↳ h h v v
h v h v
h v v h

(0,0) → (1,0)
↳ v h h v
v v h h
v h v h

number of paths





paths = hpaths + vpaths

ps int maxPaths (int i, int j, int n, int m) ✓

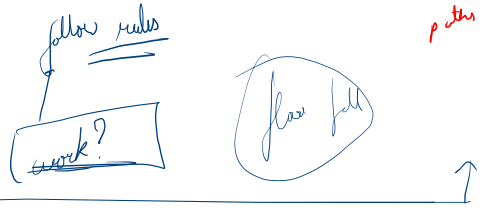
int hpaths = mp(i, j+1, n, m);
int vpaths = mp(i+1, j, n, m);

int ans = hp + vp;
return ans;

}

Faith

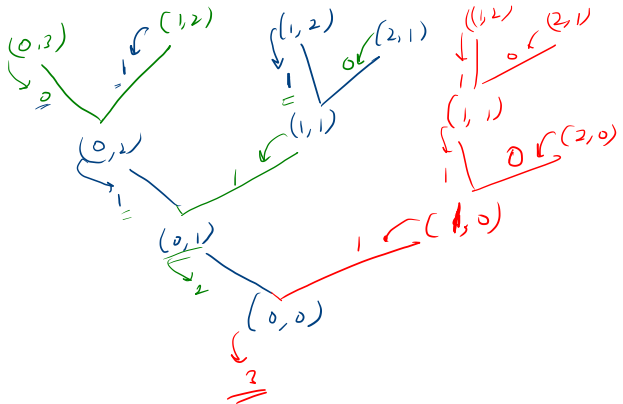
It will return
correct number of paths
from (i+1, j) to (n-1, m-1)
and (i, j+1) to (n-1, m-1);



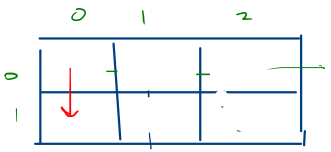
Expectation

It will return number of
paths from (i, j) to (n-1, m-1);

```
// number of maze Paths ========================================
public static int mazePaths(int i, int j, int n, int m){
    int hpaths = mazePaths(i, j+1, n, m);
    int vpaths = mazePaths(i+1, j, n, m);

    int allPaths = hpaths + vpaths;
    return allPaths;
}
```

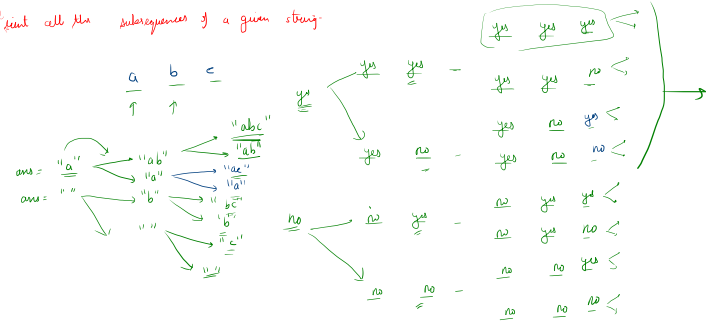


n = 2
m = 3



j == m

Q: Print all the subsequences of a given string

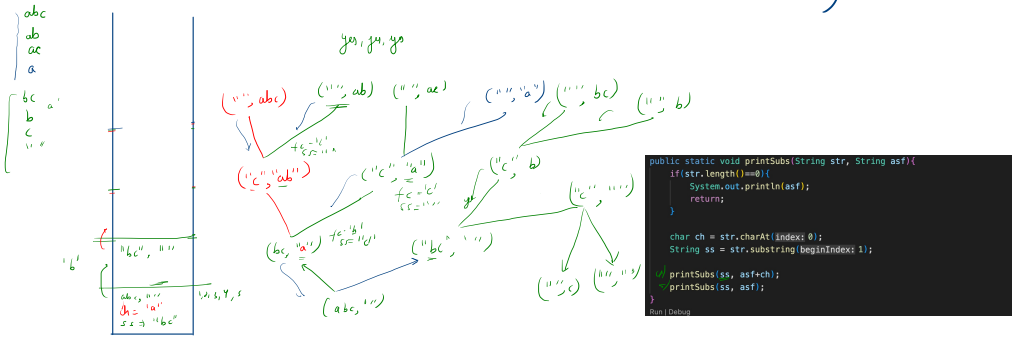


"print KPC"

psv printKPC(str, asf)

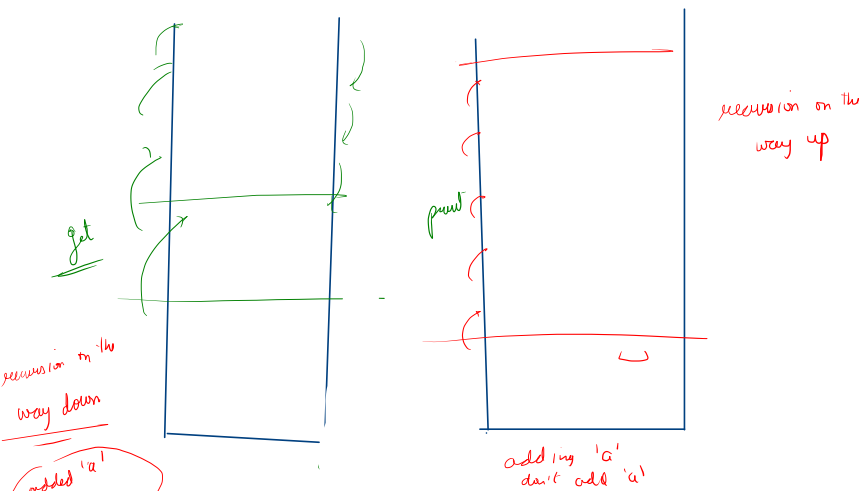
{

}

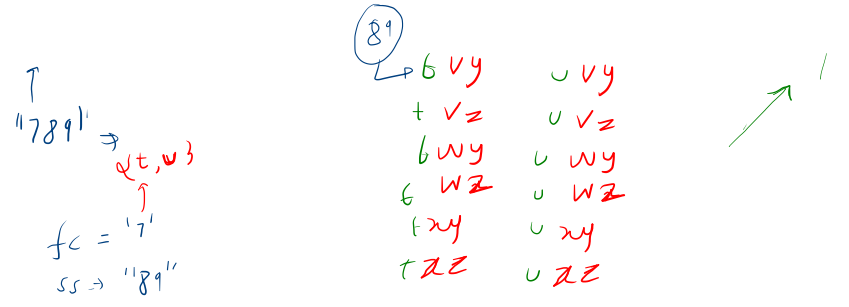



```java
public static void printSubs(String str, String asf){
    if(str.length()==0){
        System.out.println(asf);
        return;
    }

    char ch = str.charAt(index: 0);
    String ss = str.substring(beginIndex: 1);

    printSubs(ss, asf+ch);
    printSubs(ss, asf);
}
```

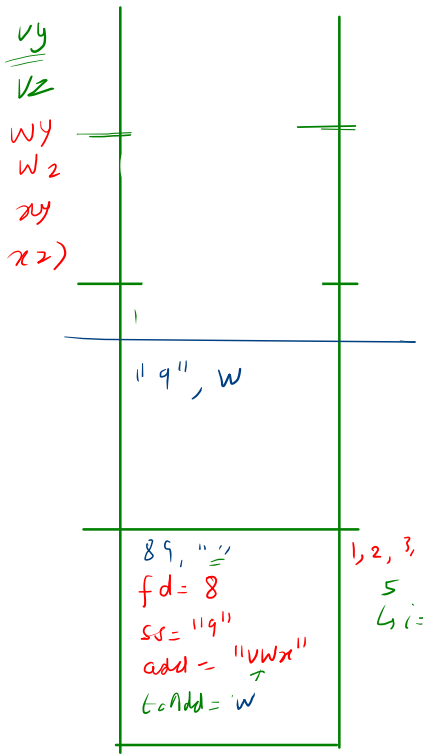
Run | Debug



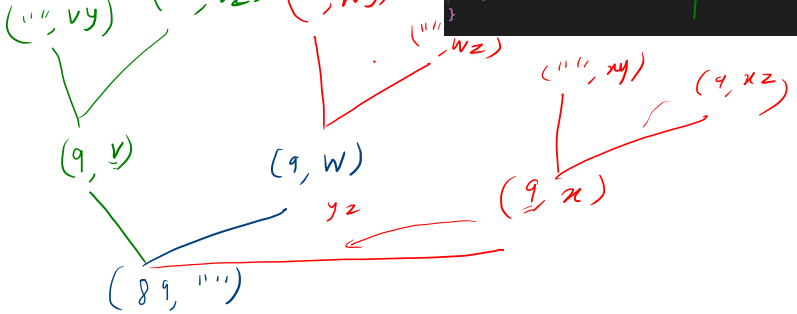
Ques Print all KPC

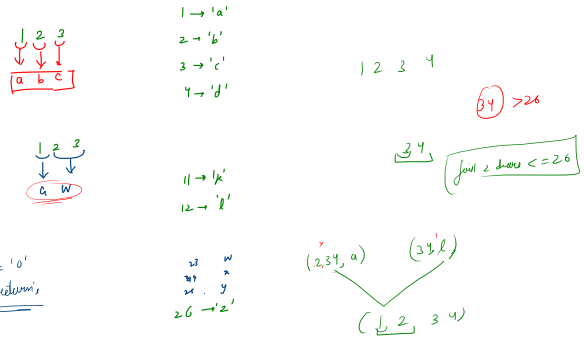


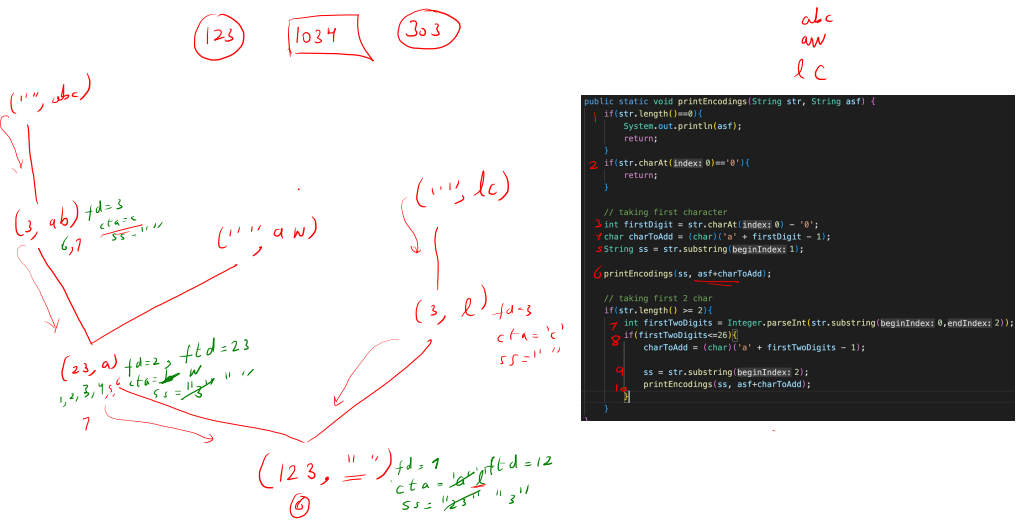

```
static String[] sarr = {".;","abc","def","ghi","jkl","mno","pqrs","tu","vwx","yz"};
```



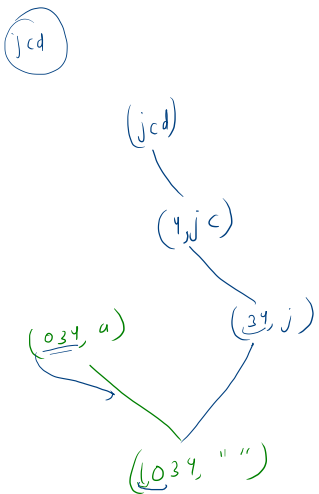

```
static String[] sarr = {".;","abc","def","ghi","jkl","mno","pqrs","tu","vwx","yz"};
public static void printKPC(String str, String asf){
    if(str.length()==0){
        System.out.println(asf);
        return;
    }
    int fd = str.charAt(index: 0) - '0';
    String ss = str.substring(beginIndex: 1);

    String add = sarr[fd];

    for(int i=0; i<add.length(); i++){
        char toAdd = add.charAt(i);
        printKPC(ss, asf + toAdd);
    }
}
```





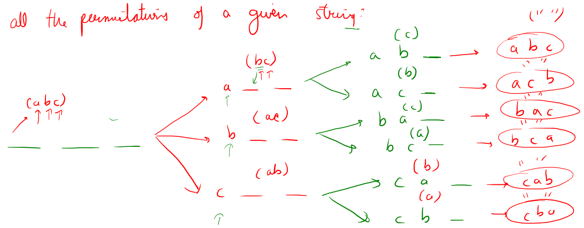



```java
public static void printEncodings(String str, String asf) {
    if(str.length()==0){
        System.out.println(asf);
        return;
    }
    if(str.charAt(index: 0)=='0'){
        return;
    }

    // taking first character
    int firstDigit = str.charAt(index: 0) - '0';
    char charToAdd = (char)('a' + firstDigit - 1);
    String ss = str.substring(beginIndex: 1);

    printEncodings(ss, asf+charToAdd);

    // taking first 2 char
    if(str.length() >= 2){
        int firstTwoDigits = Integer.parseInt(str.substring(beginIndex: 0,endIndex: 2));
        if(firstTwoDigits<=26){
            charToAdd = (char)('a' + firstTwoDigits - 1);

            ss = str.substring(beginIndex: 2);
            printEncodings(ss, asf+charToAdd);
        }
    }
}
```



Ques: Print all the permutations of a given string.

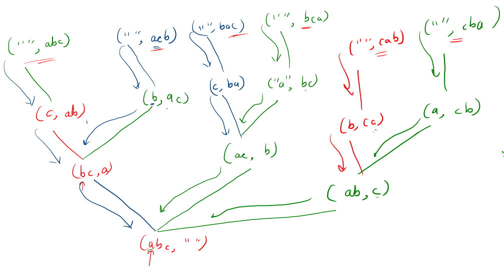
abc



n!

ros = bcd

abc
acb
bac
bca
cab
cba




```java
public static void permutationPrint(String ques, String asf) {
    if(ques.length()==0){
        System.out.println(asf);
        return;
    }

    for(int i=0; i<ques.length(); i++){
        char toAdd = ques.charAt(i);

        String leftPart = ques.substring(0, i);
        String rightPart = ques.substring(i+1, ques.length());

        String roq = leftPart + rightPart;
        permutationPrint(roq, asf+toAdd);
    }
}
```

pp (ques, asf)

expectation: fun will print all the permutations; asf will get all the permutations

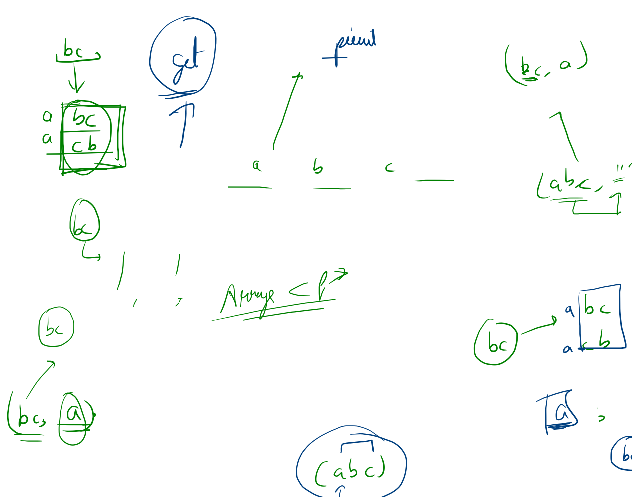
Faith: for a smaller ques, get all the arrangement in asf.

up

get ↓ — down, up

print, add

bc → a
"bc" → a
"ac" → b
"ab" → c



Always <P

print

(bc, a)

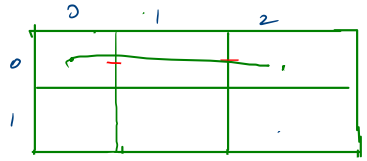
(abc, "")

bc
cb

a bc
a cb

expect "abc"

n jumps



h1h1v1 h1d1 v1h2
h1v1h1 h2v1 d1h1
v1h1h1

```java
public static void printMazePaths(int sr, int sc, int dr, int dc, String psf) {
    if(sr == dr && sc == dc){
        System.out.println(psf);
        return;
    }

    if(sr>dr || sc>dc){
        return;
    }

    // horizontal steps
    for(int jump = 1; jump<=dc; jump++){
        printMazePaths(sr, sc + jump, dr, dc, psf + "h"+ jump);
    }

    // vertical steps
    for(int jump = 1; jump<=dr; jump++){
        printMazePaths(sr+jump, sc, dr, dc, psf+"v"+jump);
    }

    // diagonal steps
    for(int jump=1; jump<=dc+dr; jump++){
        printMazePaths(sr + jump, sc + jump, dr, dc, psf+"d"+jump);
    }
}
```

different combinations

public static int ways (int n, int ld){

expect integers

combinations

expectation

It will return all combinations to make a sum of 7 using digits from (ld+1 to n)

1 → 1, 2, 4

}

Faith

for (int i = ld+1; i <= n; i++)

i = 1

ways (6, 1)

ways (5, 2) → (5)
(2, 5)

ways (4, 3)
(3, 4)

ways (3, 4)

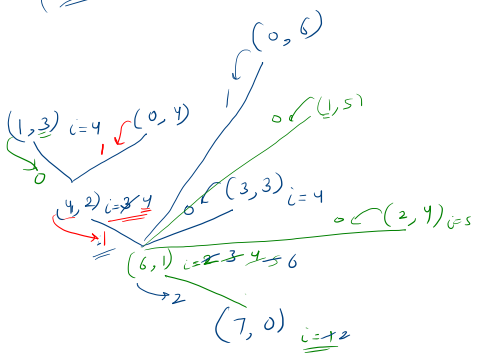
4 (n == 0)
return 0

```
public static long ways(int sum, int ld){
    long ans = 0;

    for(int i=ld+1; i<=sum; i++){
        long smallAns = ways(sum - i, i);
        ans = ans + smallAns;
    }

    return ans;
}
```

1, 2, 4

1, 6

⑦ last 3-4



and discipline

Consistency will take you places where motivation can't even imagine.