Recursion

↳ when a function calls itself

1) Steps

⇒ Solve it for the smallest problem.

⇒ Assume that it will work properly for smaller problem (faith)

⇒ Solve it for a smaller solution (your work)

Ques Take an input x, print numbers from x to 1.

⇒

```
↳ public static void fun (int x) {
    if (x == 1) {                      → base case
        System.out.println(x);
        return;
    }
    Sysout(x);                         step ③
    fun(x-1);                          step ②
}                                      step ①
```

3
2
1

---

x-1

Expectation ↓ it will print numbers from x to 1 properly

work

Faith ↓ it will print numbers from (x-1) to 1

x=3

5
4
3
2
1

faith (x-1)

4
3
2
1

Ques Print numbers from 1 to x

work

x=5

Faith

I to will print numbers from 1 to x-1 properly.

Expectation (x)

It will print numbers from 1 to x properly

1
2
3
4
5

psvm print (int x) {

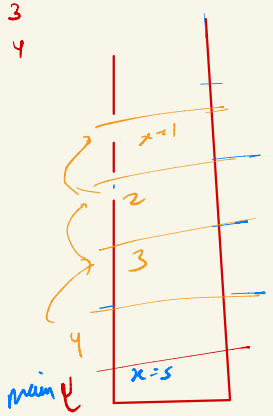
1) print(x-1);
2) System.out.println(x);

}

if (x==1) {
System.out(x);
return;
}

1
2
3
4

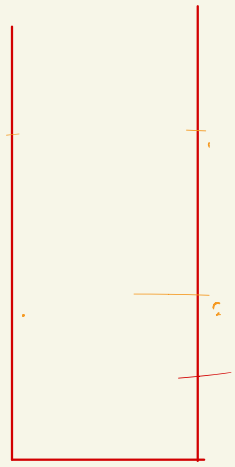
1
2
3
4
5



psv print (int x) {

1) if (x==1) {
System.out(x);
return;
}
2) print(x-1);
3) System.out(x);

}



1
2
3
4

Ques Print numbers from x to 1 using recursion.

print decreasing

Expectation: n=5

It will print numbers from n to 1

5
4
3
2
1

psv pd(int n){

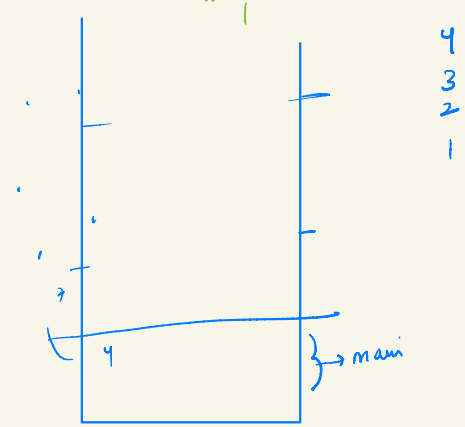
1) Sysout(n);
2) pd(n-1);
3 }

if(n==1){
Sysout(n);
return;
}

Fault

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

4
3
2
1



Dry n=5

first print decreasing, then increasing

5
4
3
2
1
2
3
4
5

n=4

4
3
2
1
2
3
4

Expectation (x=5)

It will print decreasing from x→1, then 1→x

Faith (x=4)

It will print (x-1) to 1 and then 1 to (x-1)

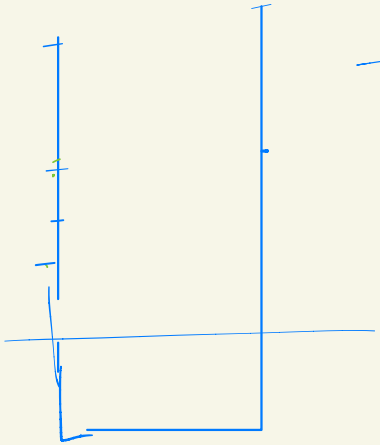
```
public static void printDecreasingIncreasing(int x){
    if(x==1){
        System.out.println(x);
        return;
    }
    System.out.println(x);
    printDecreasingIncreasing(x-1);
    System.out.println(x);
}
```

5 4 3 2 1 2 3 4 5

4 3 2 1 2 3 4

if (x==1) sysout(x) return

3 2 1 2 3



Ques Write a function to find factorial n!

expectation (n=5)

It will return (120) n!

5 x 4 x 3 x 2 x 1

```
public static int fac(int n){
    if(n==0) return 1;
    int f = fac(n-1);
    int ans = n*f
    return ans;
}
```

Faith

It will return (n-1)!

4 → 4x3x2x1 → 24

Ques Write function to calculate a raise to power b

Expectation

a=2
b=4

It will calculate and return a^b

2^4
↓
2×2×2×2

Faith

It will calculate a^{b-1}

```
ps int power(int a, int b){
1) if(b==0) return 1;
2) int f = power(a, b-1);
3) int ans = a*f
   return ans;
}
```

O(b)

Expectation a=2, b=4

It will calculate a^b.

2×2×2×2

ans = f*f

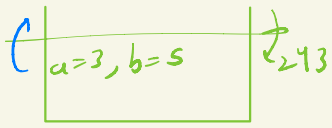
Faith

It will calculate $a^{\frac{b}{2}}$

2×2

```
psi power_better(a,b){
   if(b==0) return 1
1) int f = pb(a, b/2);
2) int ans = f*f;
3) if(b%2==1){
      ans = a*ans;
   }
   return ans
}
```

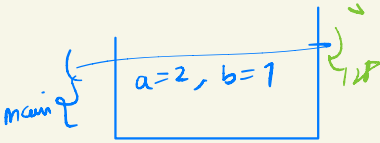
O(log b)

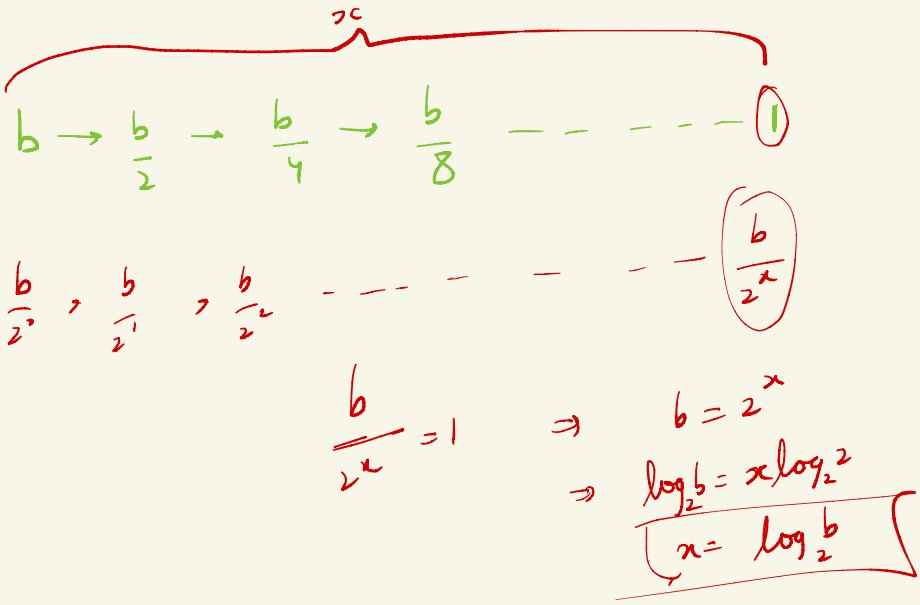


f=9

ans = 9×9 → 81

ans = 3×81 → 243





Ques Print array using recursion

arr = [10, 29, 11, 5, 43]
0 1 2 3 4

Expectation

Print elements from index 0 to (n-1);

10
29
11
5
43

Faith

Print elements from index 1 to n-1

```
if (idx == arr.length) return;
psv print_array (int[] arr, int idx)
    0
    1) System.out.println (arr[idx]);
    2) print_array (arr, idx+1);
```

Recursion on the way up

10
29
11
5
43

0 → n-1 reverse

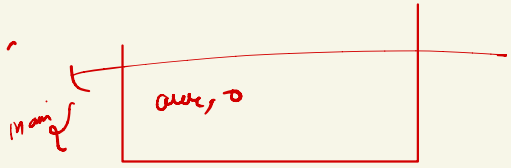
10, 29, 11, 5, 43

1 → n-1 reverse

43
5
11
29
10

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

43
5
11
29
10



Qus Find max of an array using recursion.

arr => { 10, 9, 4, 7, 6 }
0 1 2 3 4

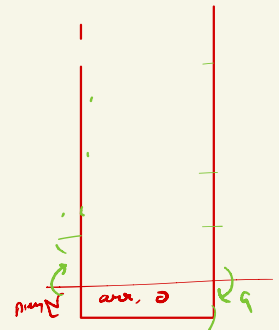
→ [Integer.MIN_Value]

Expectation

It will find the maximum element b/w (0 - 4) index.

Faith

It will find the maximum element b/w (1-4) index.




```
int max(int[] arr, int idx) {
    1) int max_ocage = max(arr, idx+1)
    2) int ans = Math.max(arr[idx], max_ocage);
    return ans
}
```

Ques Find at which index, tar exists -

arr = {1, 2, 11, 9, 45}
0 1 2 3 4

tar = 9, ans = 3

tar = 12, ans = -1

Expectation

It will find idx of tar b/w 0 and 4

Faith

It will find idx of tar b/w 1 and 4

```java
// tar=7 0 1 2 3 4 5 6
//  arr={2,3,5,7,5,7,24};
public static int find_rec(int[] arr, int idx, int tar){
    1) if(idx==arr.length){
        return -1;
    }

    2) int f=find_rec(arr, idx+1,tar);

    3) if(f==-1){
        if(arr[idx]==tar){
            return idx;
        }
    }

    4) return f;
}
```

=> last occurrence

=> first occurrence

recursion on the way up

recursion on the way down

0 1 2 3 4 5 6

2, 3, 5, 7, 5, 7, 24

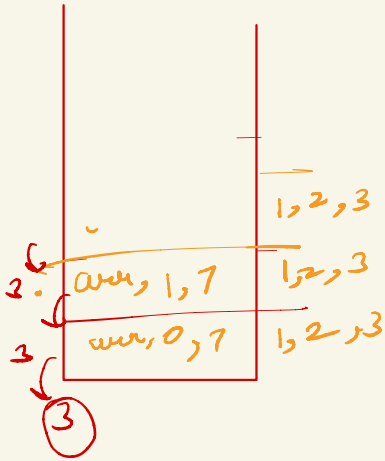
```java
public static int find_fo(int[] arr, int idx, int tar){
    if(idx==arr.length){
        return -1;
    }

    int f=find_fo(arr, idx+1,tar);

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

    return f;
}
```

f = 3

⇒ first occurrence using recursion on the way down.

```java
// first occurrence using recursion on the way up
public static int find_fo_better(int[] arr, int idx, int tar){
    if(idx==arr.length){
        return -1;
    }

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

    int ans=find_fo_better(arr, idx+1, tar);
    return ans;
}
```

ans = 3



Q) Find sum of array using recursion

Expectation

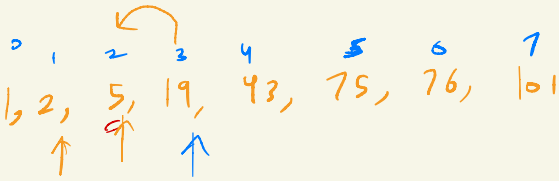
Find sum of elements from (0 - n)

Faith

Find sum of elements from 1 - n

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

    int sum_aage=sum(arr,idx+1);

    int ans=sum_aage+arr[idx];

    return ans;
}
```

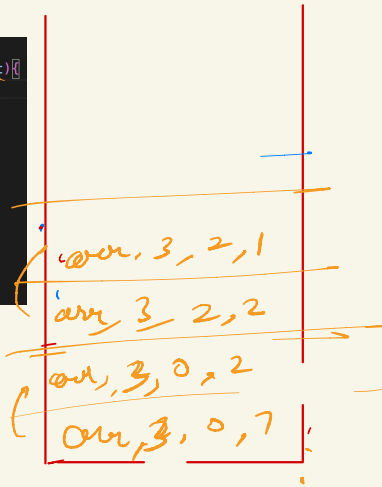
Ques Implement binary search using recursion.



left = 0
right = n-1 = 7

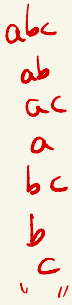
```java
// binary search using recursion
public static int bs_rec(int[] arr, int tar, int left, int right){

    int mid=(left+right)/2;

    if(arr[mid]==tar){
        return mid;
    } else if(arr[mid]>tar){
        return bs_rec(arr, tar, left, mid-1);
    } else {
        return bs_rec(arr, tar, mid+1, right);
    }
}
```

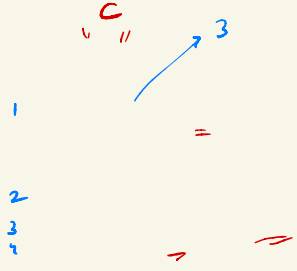
mid = 2

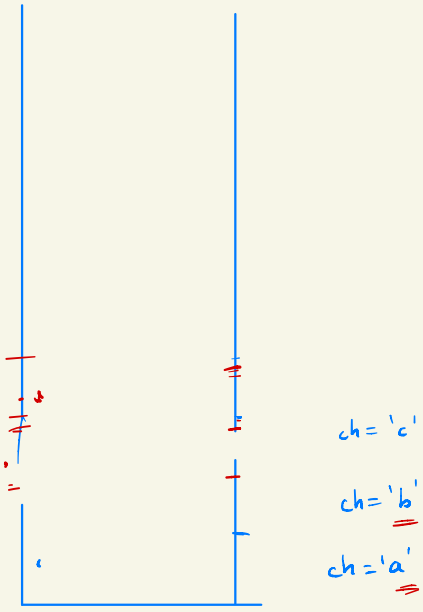


Ques Print all the subsequence of a given string.

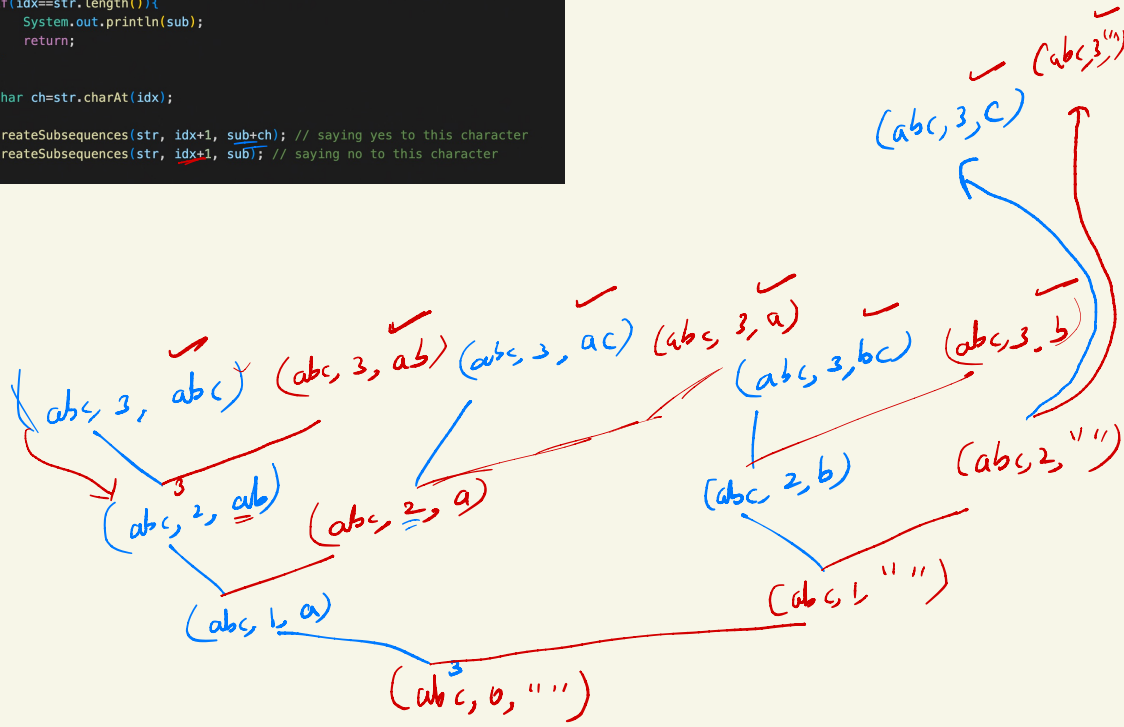
str = "a b c"
0 1 2





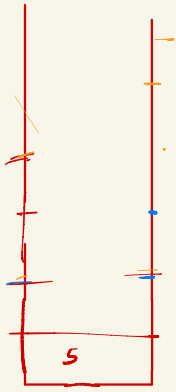


```java
// Subsequence
public static void createSubsequences(String str, int idx, String sub){
    if(idx==str.length()){
        System.out.println(sub);
        return;
    }

    char ch=str.charAt(idx);

    createSubsequences(str, idx+1, sub+ch); // saying yes to this character
    createSubsequences(str, idx+1, sub); // saying no to this character
}
```



min = ~~5~~ ~~3~~ ~~2~~ 5

```java
public void push(int x) {
    if(st.size()==0){
        min=x;
        st.push(x);
        return;
    }

    if(x<min){
        int new_min=x;
        int to_push=(x-min)+x;

        st.push(to_push);
        min=new_min;
    } else {
        st.push(x);
    }
}

public void pop() {
    if(st.peek()<min){ // we are popping encrypted value
        int tp=st.pop();

        int x=min;
        int old_min=2*x-tp;

        min=old_min;
    } else {
        st.pop();
    }
}
```



$tp = (x - min) + x$

$tp < x$

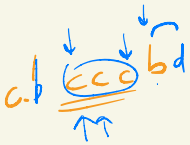
$tp = x - min + x$

$tp = 2x - min$

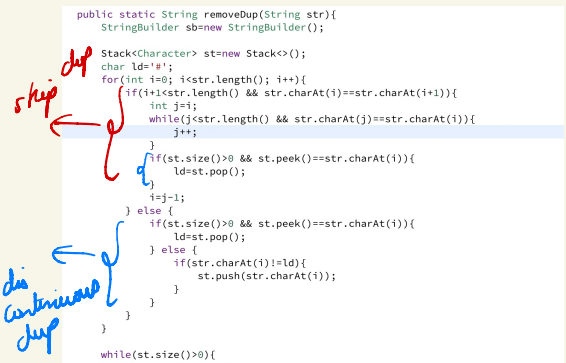
$min = 2x - tp$

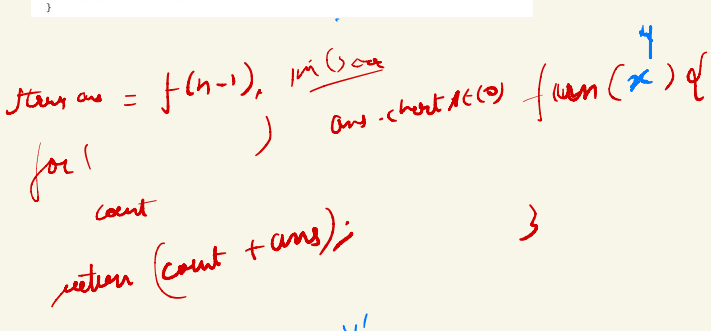
if (x < min) {
to_push = (x - min) + x;
}

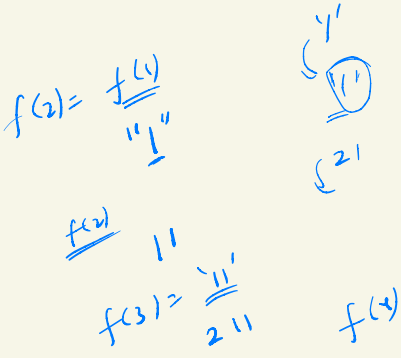
1) we can retrieve old_min
2) $tp < x$

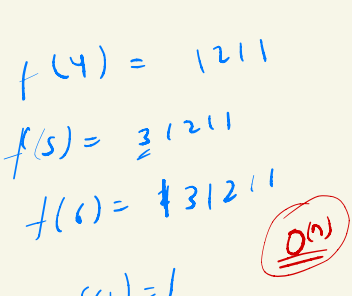


```java
public static String removeDup(String str){
    StringBuilder sb=new StringBuilder();

    Stack<Character> st=new Stack<>();
    char ld='#';
    for(int i=0; i<str.length(); i++){
        if(i+1<str.length() && str.charAt(i)==str.charAt(i+1)){
            int j=i;
            while(j<str.length() && str.charAt(j)==str.charAt(i)){
                j++;
            }
            if(st.size()>0 && st.peek()==str.charAt(i)){
                ld=st.pop();
            }
            i=j-1;
        } else {
            if(st.size()>0 && st.peek()==str.charAt(i)){
                ld=st.pop();
            } else {
                if(str.charAt(i)!=ld){
                    st.push(str.charAt(i));
                }
            }
        }
    }

    while(st.size()>0){
        sb.append(st.pop());
    }

    sb.reverse();

    return sb.toString();
}
```









s = 1

11

21

1211

3121

13112111

41311211