

#

Functions

⇒ Takes input and give some output -

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

$$f(n) = \underline{2^{n+1}}$$

$f(x) = 2^{x+1}$

input
 $n=5$

output
 $2^{x+1} = 32$

parameters/arguments

```

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

return type

$$\text{sum}(2, 3) = 5$$

$$\text{sum}(1, 4) = 5$$

$$\text{sum}(9, 11) = 20$$

RAM

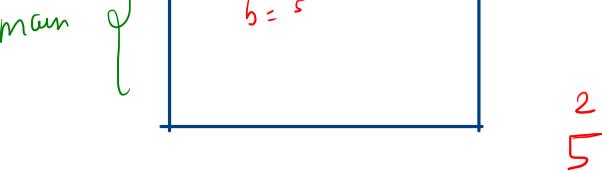
Stack

```
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);  
    }  
}
```

5

```
1- class Main {  
2-     public static int sum(int a, int b){  
3-         int sum = a + b;  
4-         return sum;  
5-     }  
6-  
7-     public static void fun(int a, int b){  
8-         int sum = a+b;  
9-         System.out.println(sum);  
10-  
11-         System.out.println("I am inside a void function");  
12-         System.out.println("I am inside a void function");  
13-     }  
14-  
15-     public static void main(String[] args) {  
16-         int a= 2;  
17-         int b= 3;  
18-         fun(a,b);  
19-     }  
20- }  
21 }
```

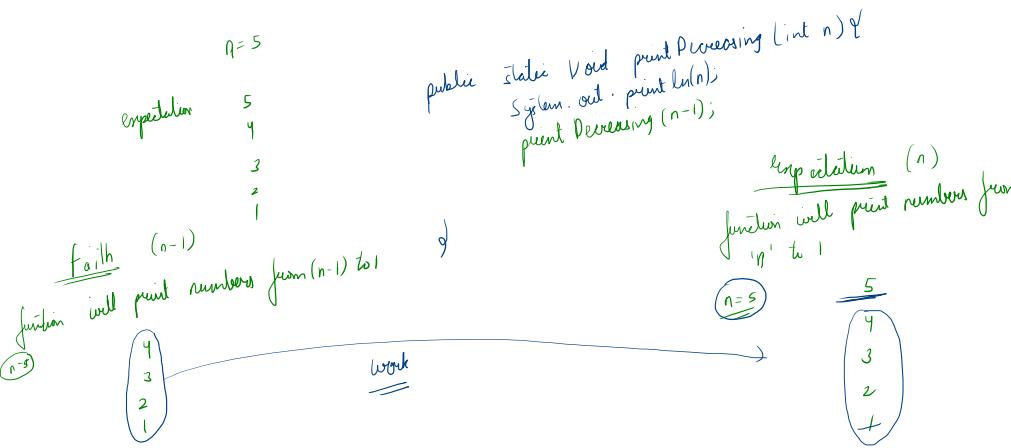
```
class Main {  
    public static void swap(int a, int b){  
        int temp = a;  
        a = b;  
        b = temp;  
    }  
    Run | Debug  
    public static void main(String[] args) {  
        int a= 2;  
        int b= 5;  
  
        swap(a, b);  
  
        System.out.println(a);  
 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



Recursion

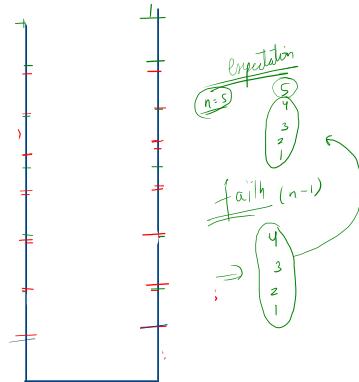
- How to solve any recursion problem
- 1) keep a faith that it works for a smaller problem (faith)
 - 2) solve for the smallest problem.
 - 3) solve for the actual problem (to reach your expectation)

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



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

stack overflow



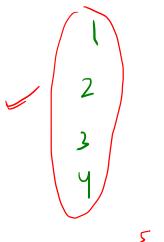
```
class Main {  
    public static void printDecreasing(int n){  
        if(n==0){  
            return;  
        }  
        System.out.println(n);  
        printDecreasing(n-1);  
    }  
}  
Run | Debug  
public static void main(String[] args) {  
    int n = 5; //  
    printDecreasing(n); 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

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 ' $n-1$ '



```
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

```
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.

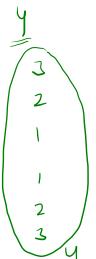
Faith $(n-1)$
It will print numbers
from $(n-1)$ to 1 and then 1 to $(n-1)$

$n=1$
3
2
1
1
2
3

$n=5$

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

}



Expectation

It will print numbers in decreasing
order then increasing order

$n=4$

4
3
2
1
1
2
3
4

```
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; 2  
    pdi(n);  
}
```

Ques find factorial of given number 'n'.

Faith $(n-1)$

It will return
factorial $(n-1)$

$1 \times 2 \times 3 \times 4$
smallAns

public static int fac(int n) {

int smallAns = fac(n-1); → faith
int ans = smallAns * n; → work
return ans;

}

work

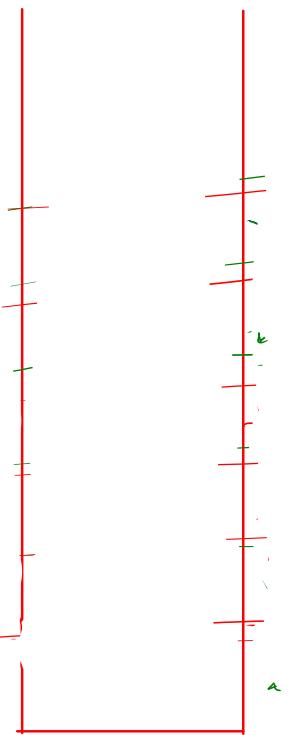
Expectation

It will return factorial
of given number

n=5

P

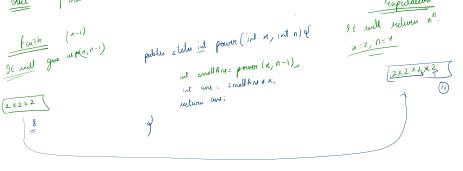
$1 \times 2 \times 3 \times 4 \times 5$



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



```
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 = 6;  
    int ans = power(x, n);  
    System.out.println(ans);  
}
```

faith ($n/2$)
it will calculate
 $x^{\frac{n}{2}}$ correctly
 $x=2, n=6$

actual
 $2 \times 2 \times 2$

public static int power(int x, int n){
it will return x^n .

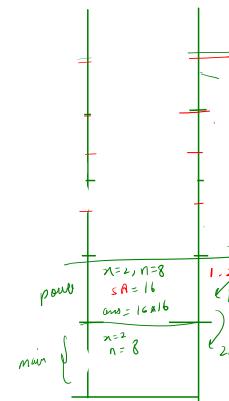
```
int smallAns = power(x, n/2);  
int ans = smallAns * smallAns;
```

$y = (n/2)^2 = 0$
 $y = ans + x$

return ans;

expectation
it will return x^n .
 $x=2, n=6$

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



```
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);  
}
```

$$\begin{aligned} 2^8 &\Rightarrow 2^2 \times 2^2 \\ &\Rightarrow 2^4 \times 2^4 \\ &\Rightarrow 2^8 \\ \\ 2^1 &\Rightarrow 2^{\frac{1}{2}} \times 2^{\frac{1}{2}} \\ &\Rightarrow 2^{\frac{3}{2}} \times 2^{\frac{1}{2}} \\ &\Rightarrow 2^6 \times 2^1 \end{aligned}$$

$$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \rightarrow \frac{n}{16} \dots \underset{\cancel{x}}{1}$$

$$\frac{n}{2^0} \rightarrow \frac{n}{2^1} \rightarrow \frac{n}{2^2} \rightarrow \frac{n}{2^3} \rightarrow \frac{n}{2^4} \rightarrow \frac{n}{2^5} \rightarrow \dots \underset{\cancel{x}}{1}$$

$$\Theta(\log n)$$

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

$$n = 2^k$$

$$2^k = n$$

$$\log_2 2^k = \log n$$

$$k \log_2 2 = \log n$$

$$k \log_2 2 = \log n$$

$$k \times 1 = \log n$$

$$k = \log n$$

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

$$\log_a^b \Rightarrow$$

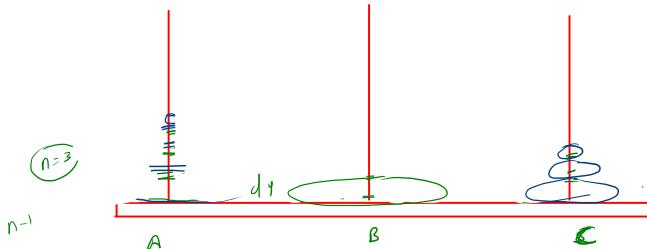
$$\Theta(\log n)$$

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 larger disc on top of a smaller disc



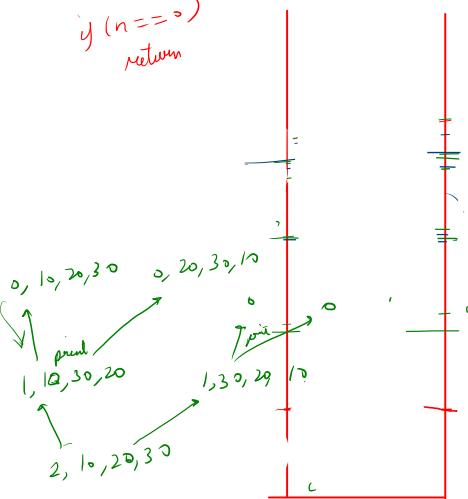
Path

move $(n-1)$ discs from A to C using B
→ move nth disc from A to B
→ move nth disc from A to B (move 3rd disc)
move $(n-1)$ discs from C to B (move 2nd disc)
move $(n-1)$ discs from C to B (move 2nd disc)
→ move $(n-1)$ discs from C to B (move 2nd disc)
→ move $(n-1)$ discs from C to B (move 2nd disc)

psv $\text{ToH}(n, \underline{\text{A}}, \underline{\text{B}}, \underline{\text{C}})$
→ $\text{ToH}(n-1, \underline{\text{A}}, \underline{\text{C}}, \underline{\text{B}}) \rightarrow$ Path
move nth disc from A to B
 $\text{ToH}(n-1, \underline{\text{C}}, \underline{\text{B}}, \underline{\text{A}})$

Expectation $\forall n$
function will move n discs
from 'A' to 'B' using 'C'
 $n=3$
with steps to move
3 discs from A to B

$y(n=3)$
return

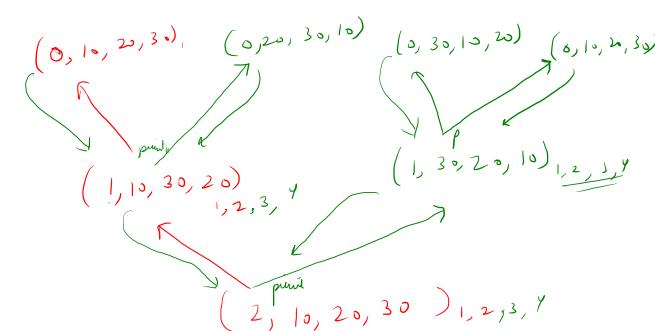


```
public static void towerOfHanoi(int n, int A, int B, int C){  
    if(n==0){  
        return;  
    }  
  
    towerOfHanoi(n-1, A, C, B);  
    // move nth disc from A to B  
    System.out.println("Moving ring " + n + " from " + A + " to " + B + " using " + C);  
    towerOfHanoi(n-1, C, B, A);  
}  
  
Run | Debug  
public static void main(String[] args) {  
    int n = 3;  
    towerOfHanoi(3, 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

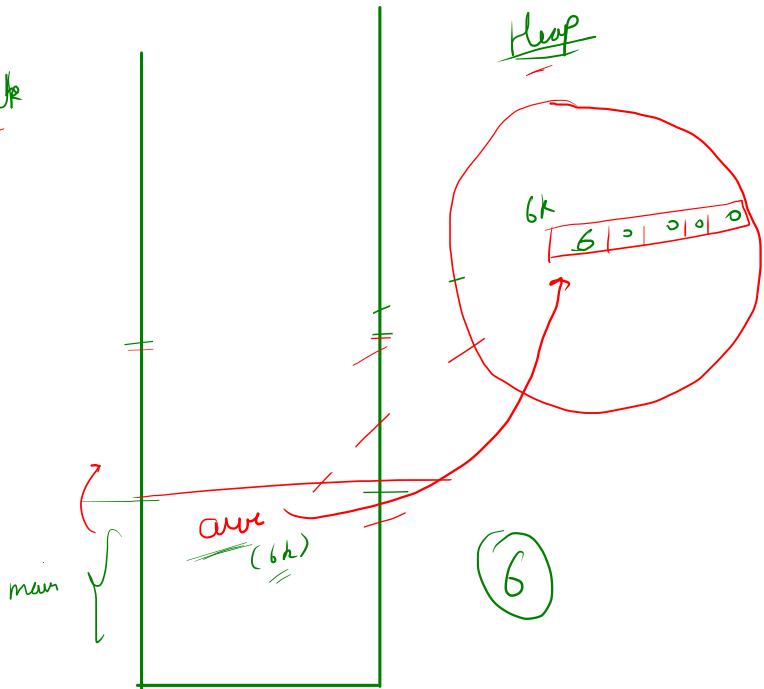


#

Recursion with Arrays

RAM

stack



(NEW)

```

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;

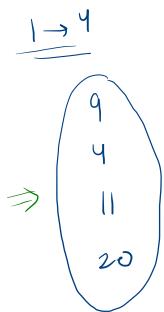
    int a = 5;
    increaseArray(arr);
    increaseInteger(a);

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

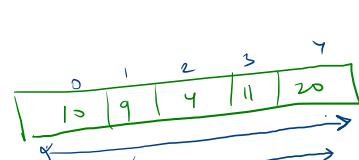
```

Ques Print array using recursion.

Faith (smaller area)
It will print array from
of $idx+1$ to end

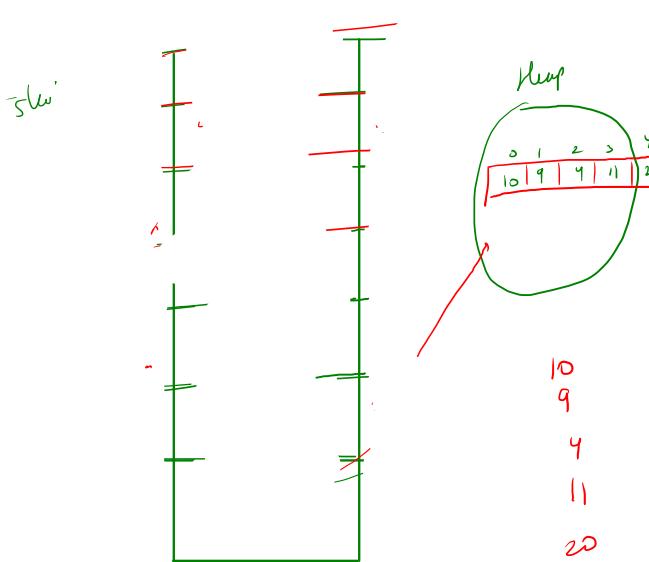
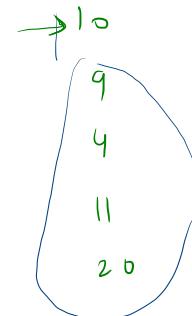


psu printArray(int[] arr, int idx){
 System.out.println(arr[idx]);
 printArray(arr, idx+1);
}



printArray(arr, 0)

expectation
function will print
array from 'idx' to end
(n-1)
from 0 → 4



```
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);  
}
```

Ours

Print array in reverse

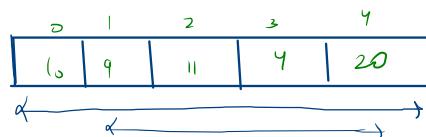
Fault

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

PSV
printReverse (int [] arr, int idx) {

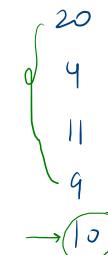
 printReverse (arr, idx+1);
 System.out.println (arr[idx]);

}



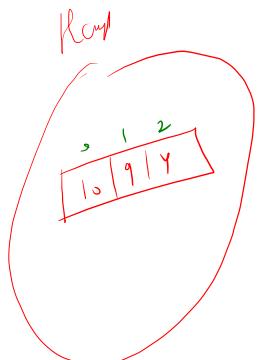
print reverse (arr, 0)

Expectation
It will print from 'idx'
to end in reverse
 $0 \rightarrow 4$ in reverse



if ($idx == arr.length$)
 return;

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



Man of { arr, 0

4
9
10