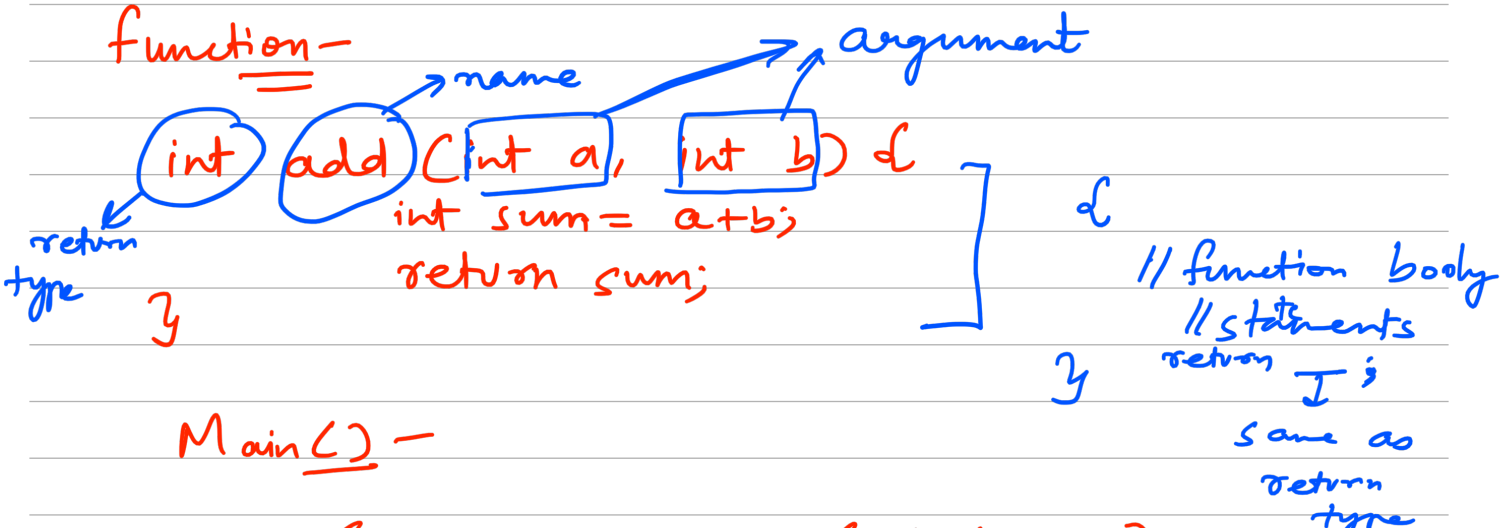
Recursion-

A process in which a method calls itself continuously.

Java,
A method that calls itself

function-



Main() -

```
System.out.println(add(5,10));

int add5And10 = add(5,10);
system.out.println(add5And10);
```

```
void printSum(int a, int b) {
    int sum = a+b;
    System.out.println(sum);
}
```

Write a function to print "WORK OUT <DAY NO>" for 60 days and then print "MARRIAGE".

```
Main() {
    for(int day = 1; day <= 60; day += 1) {
```

1 — initialisation <one time>
2 — condition
4 — after the code

8 5
9 6

s1 executed > s3

system.out.println("WORKOUT " + day);

3
7
10

}

Once that condition is no longer satisfied, we come here and print "MARRIAGE"

system.out.println("MARRIAGE");

}

workout(int day) {

S2 Terminating Condition { if (day == 61) {
print("MARRIAGE");
return;
}

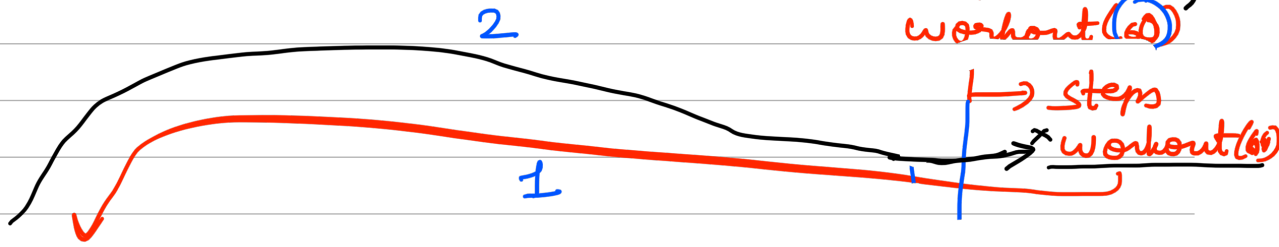
Body { System.out.println("WORKOUT " + day);

workout(day+1); s3

}

workout(1); s1

workout(1)
- steps
- workout(2)
  - steps
  - workout(3)
    - steps
    - workout(4) ← ...
      - ... steps
      - workout(59) ← workout(60)
        - steps
        - workout(61)




```
workout(61)
    ↳ if (day == 61) {
          print("MARRIAGE");
          return;
      }
```

- we designed an algorithm to solve a problem by solving a smaller instance of the same problem, unless the problem is so small that we can just solve it directly.

Earlier, we started workout at day 1

what it felt like is that we are at day 59

Starting workout at day 50 and will do another 10 days of workout.

At day 61,
The problem got so small that we can solve it directly.

This technique is called recursion.

Template for a recursion -

```
fn () {
    // base case
    // fn(); //recursion
}
```

Fibonacci -

$F_6 = F_5 + F_4$

$F_5 = F_4 + F_3$

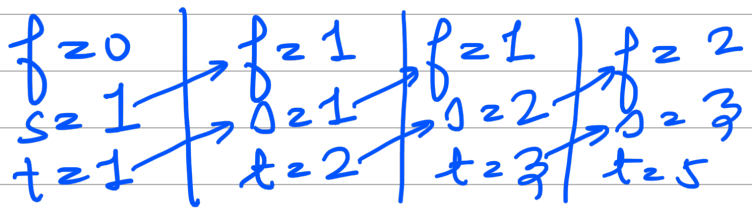
⋮

$F_n = F_{n-1} + F_{n-2}$

Iterative approach-

```
int getNFib(int n) {
    int first = 0;
    int second = 1;
    int third = 1
    for(int num = 3; num <= n;
                    num += 1) {
        third = first + second;
        first = second;
        second = third;
    }
    return third;
}
```

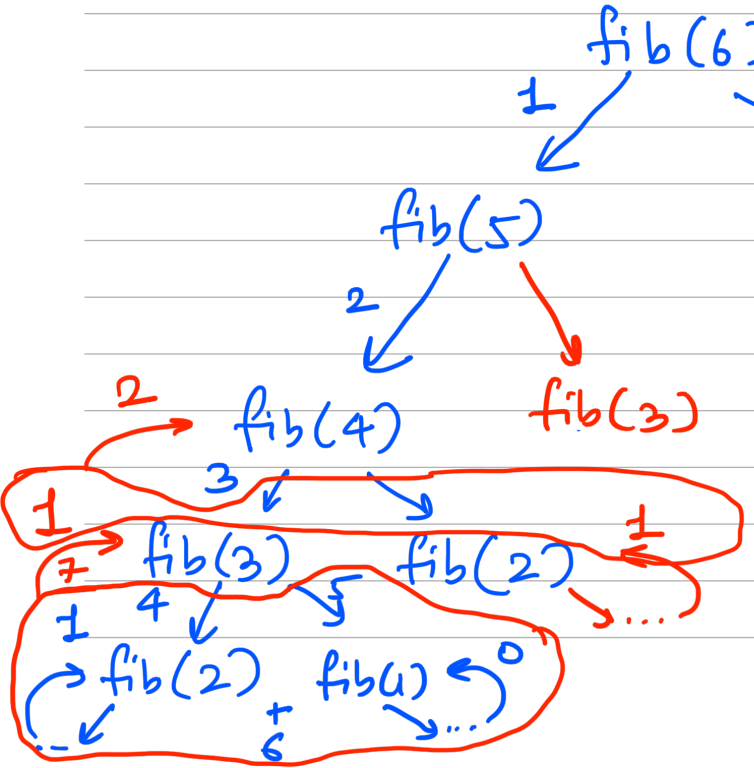
0 1 1 2 3 5



```
int fib(n) {
    if (n==1) {
        return 0;
    }
    if (n==2) {
        return 1;
    }
    return fib(n-1) + fib(n-2);
}
```

$$fib(n) = \begin{cases} 0, & n=1 \\ 1, & n=2 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

<< BAD IMPLEMENTATION OF FIBONACCI NUMBERS >>



redundant work.. As fib(3) was already calculated before...??

find sum of n numbers using recursion

$$Sum(n) = \begin{cases} 1 & \text{if } n \geq 1 \\ n + sum(n-1) & \text{otherwise} \end{cases}$$

sum(5)
→ 5 + sum(4)
→ 4 + sum(3)
→ 3 + sum(2)
→ 2 + sum(1)
1
↘...

```
int sum(int n) {
    if (n>=1) {
        return 1;
    }
    return n + sum(n-1);
}
```

In-class assignments-

① → Multiply M and N, only using addition operation and recursion.

```
int multiply (int M, int N) {
    if(M==1) {
        return N;
    }
    return N + multiply(M-1, N);
}
```

$$Mul(M,N) = \begin{cases} N, & \text{if } M==1 \\ N + mul(M-1,N) \end{cases}$$

8 6

8 × 6 = 48

M N

6 + (42)

6 + (7 × 6)
↳ 6 + (6 × 6)
↳ 6 + (5 × 6)
↳ 6 + (4 × 6)

6 + (7 × 6)
↳ 6 + (6 × 6)
↳ 6 + (5 × 6)
↳ 6 + (4 × 6)

6 + (3 × 6)
↳ 6 + (2 × 6)
↳ 6 + (1 × 6)

6+6+6+6+6+6+6+6

(2) → Palindrome recursive –

"torch" ×

"appa" ✓

"tape" ×

"madam" ✓

```
isPalindrome(s) {
    int left = 0,
        right = s.length()-1;
    while (left <= right) {
        if (s.charAt(left) != s.charAt
                (right)) {
            return false;
        }
        left += 1;
        right -= 1;
    }
    return true;
}
```

isPalindrome(s, l, r)

⇓

true : l >= r

false : s.at(l) != s.at(r)

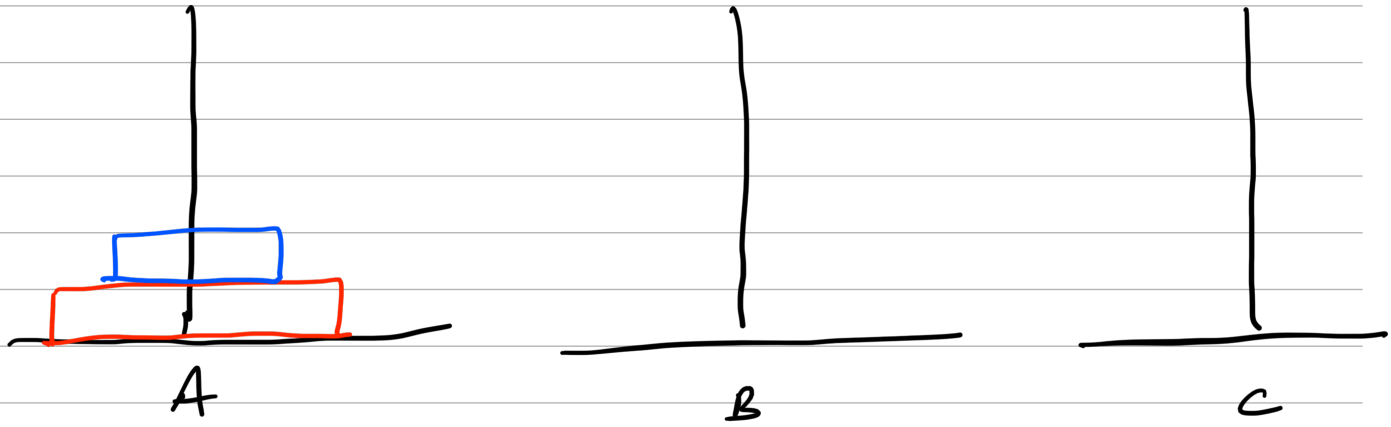
isPalindrome(s, l+1, r-1)

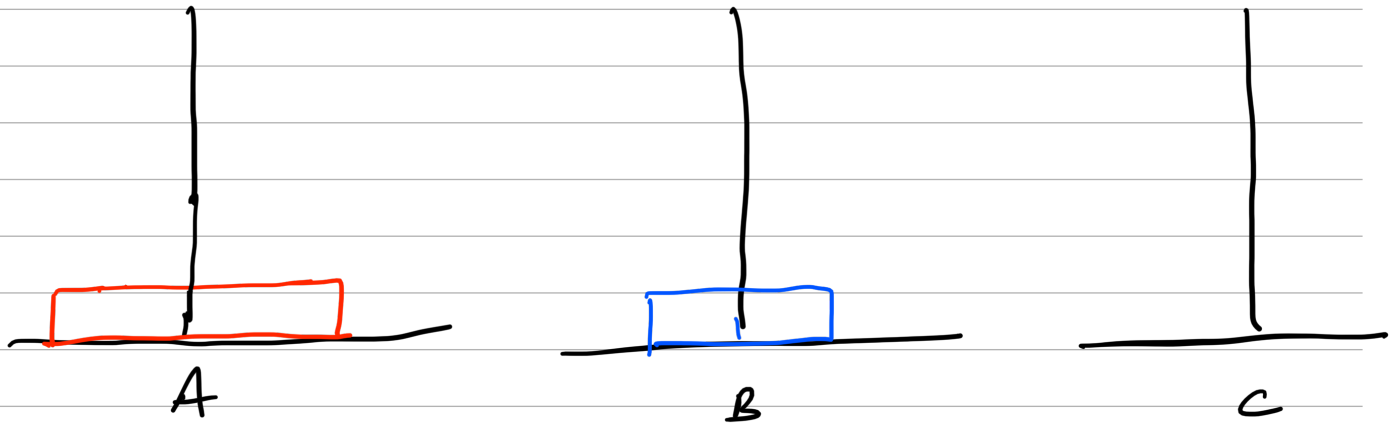
```
boolean isPalindrome(String s, int l, int r) {
        if (l >= r) {
            return true;
        }
        if (s.charAt(l) != s.charAt(r)) {
            return false;
        }
        return isPalindrome(s, l+1,
                            r-1);
    }
```

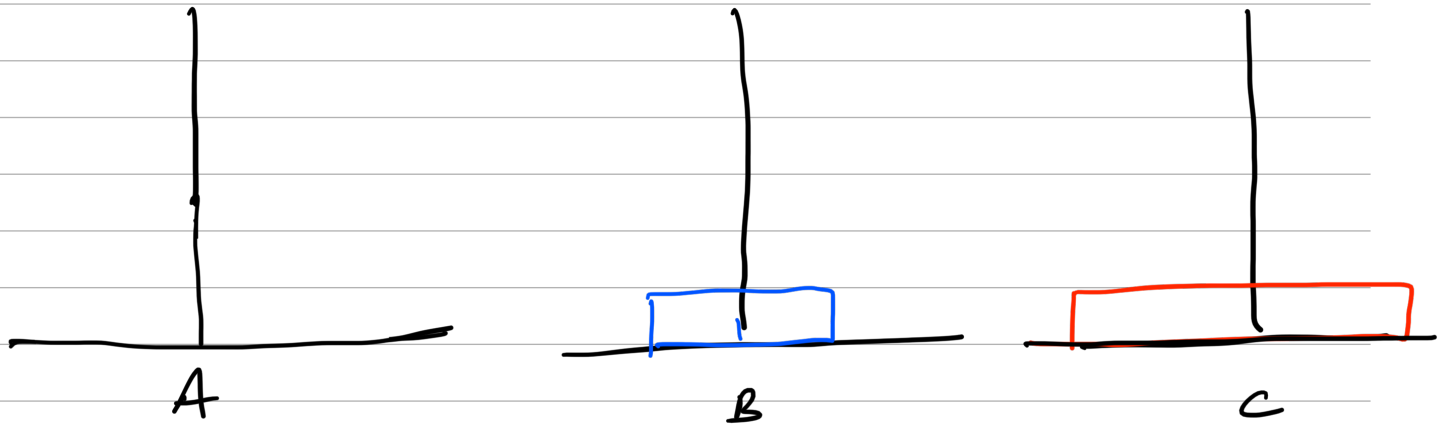
- A to B

(3) → Tower of Hanoi –

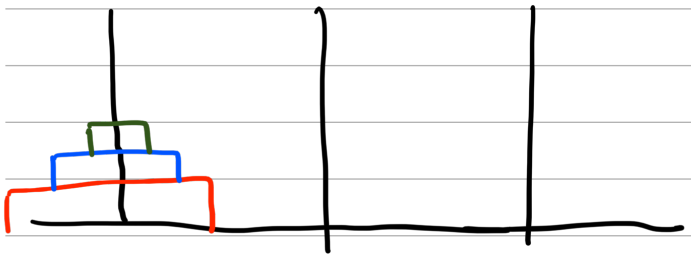
- can't place big disk on top of small disk



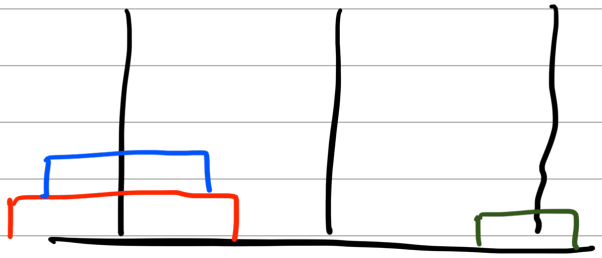




3 Disks -



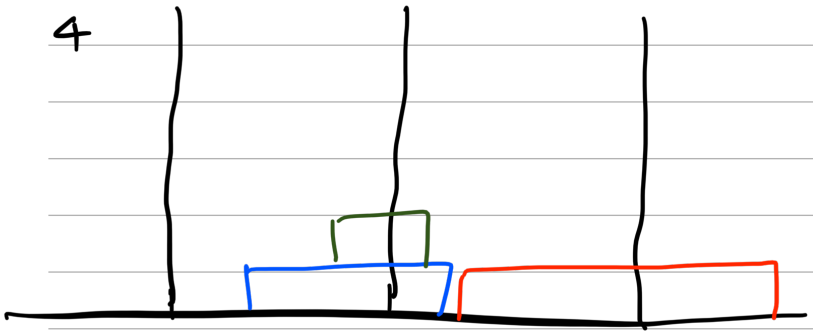
1



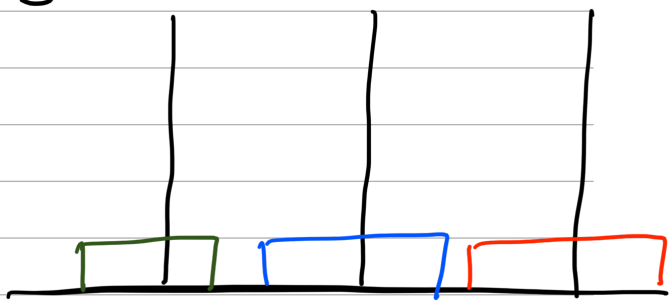
2

3

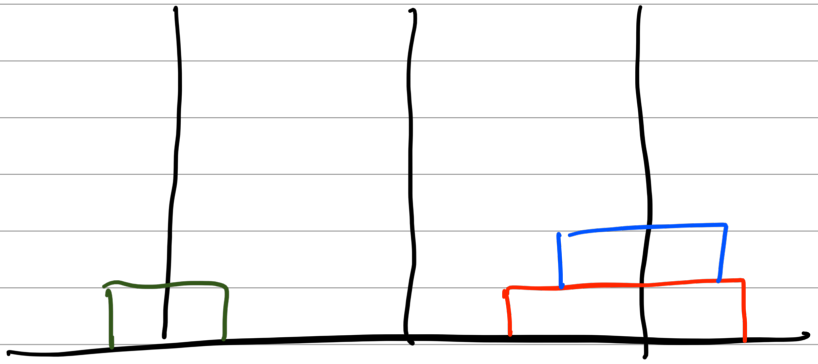
4



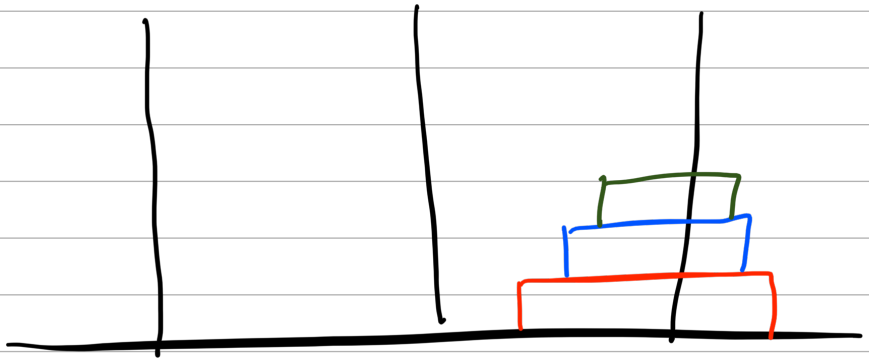
5



6



7



```
void towerOfHanoi(int n, int fromTower,
        int toTower, int extraTower) {
    if(n==0) {
        return;
    }
    towerOfHanoi(n-1, fromTower,
                 extraTower, toTower);
    System.out.println(n + ": " +
            fromTower + "->" +
            toTower);
    towerOfHanoi(n-1, extraTower,
                 toTower, fromTower);
}
```