Recursive function is one that calls itself.

```
public static int f(int n) {
    int tmp = f(n)
}
```

↓ Problem - if a recursive function ALWAYS makes a call to itself, none of the calls would ever finish!

Have to be cases where the function doesn't call itself!

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

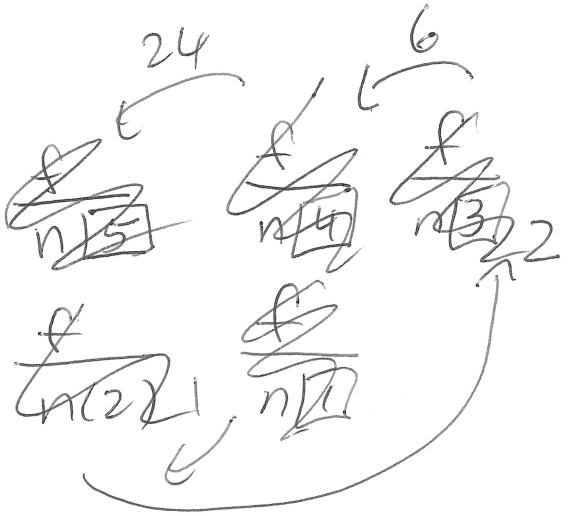
// Base case(s)

$$n! = \begin{cases} 1, & n=1 \\ n*(n-1)!, & n>1 \end{cases}$$

main

```
int ans = f(5);
```

ans 120



f(1)
f(2)
f(3)
f(4)
f(5)
main()

// Calculate $base^{exp}$. exp ≥ 0. int

$3^5 = 3 \times 3^4$

1. Break problem down into pieces such that at least 1 piece is a problem of the exact same nature.

2. Identify some cases where the problem is so easy, it's best just to solve it directly.

```
public static double mypow(double base, int exp) {
    if (exp == 0) return 1;
    return mypow(base, exp-1) * base;
}
```

cost	tip
$10	$1.50
$11	$1.65
$12	$1.80
⋮	
$50	$7.50

tipchart(int start, int end, double rate)

Base case: start > end

towers(int start, int end, int n)

start tower, end tower, # of disks

$T(1) = 1$

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

$T(2) = 2 \cdot 1 + 1 = 3$

$T(3) = 2 \cdot 3 + 1 = 7$

$T(4) = 2 \cdot 7 + 1 = 15$

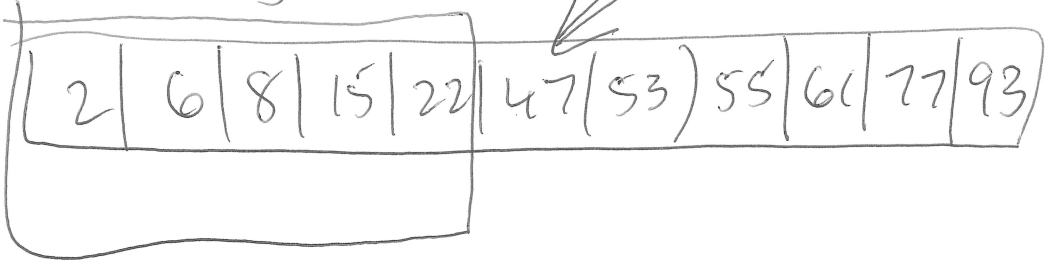
$T(n) = 2^n - 1$

towers(start, other, n-1);

⇒ move disk n from start to end.

towers(other, end, n-1);

Binary Search




```
public static boolean search(int[] arr,
                             int lowIndex,
                             int highIndex,
                             int value) {

    if (lowIndex > highIndex)
        return false;
    int mid = (lowIndex + highIndex)/2;
    if (value > arr[mid])
        return search(arr, mid+1, high, value);
    else if (value < arr[mid])
        return search(arr, low, mid-1, value);
    else
        return true;
}
```