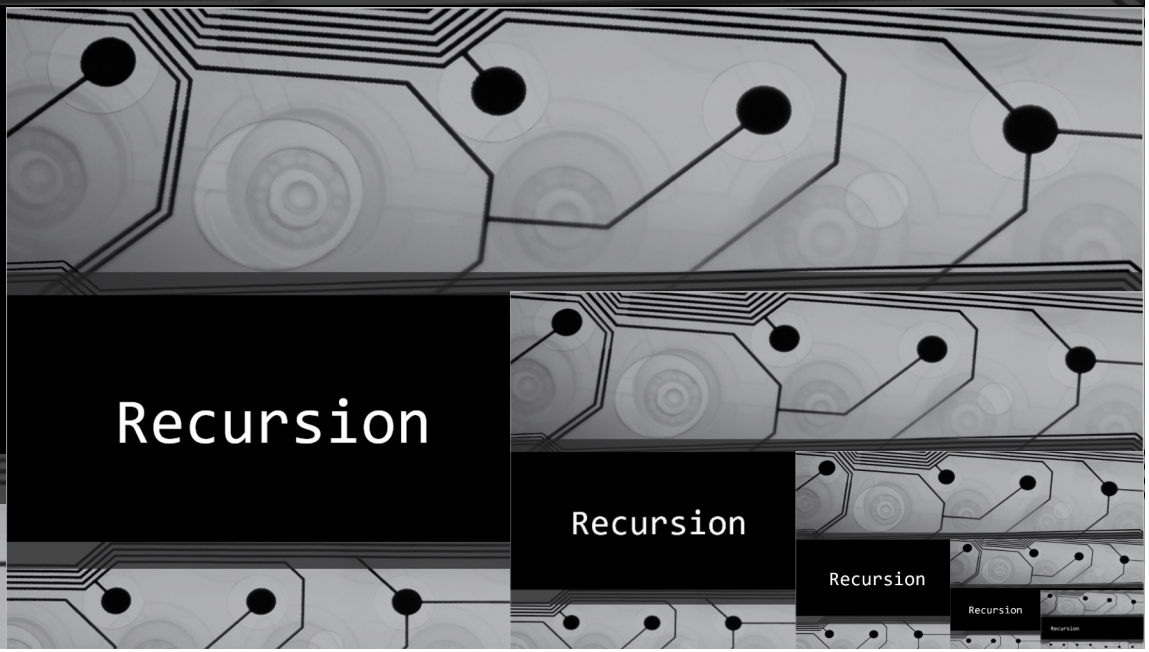
# Recursion

3/1/24



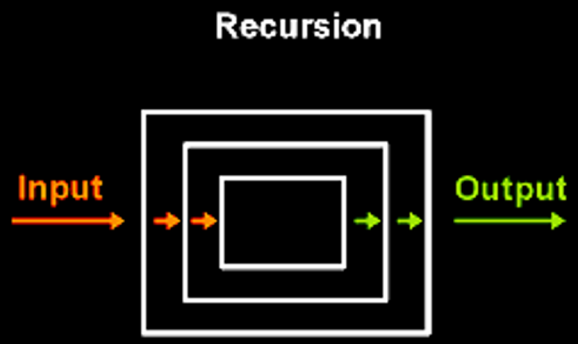
# Administrivia

- Lab due Tuesday
- No new lab Tuesday (we'll be there to answer questions) and course review in class on Wednesday
- Final exam: Monday(3/11) afternoon (1:30-4:30pm)
  - Meet in the downstairs labs (5th period in Cat lab, 6th period in Stellyes)
  - Open note, closed other stuff
  - Comprehensive, but more heavily weighting stuff from since the 2nd exam
  - Sample exam out today: 9 questions (50% more than midterm)
    - Expect similar, but not necessarily identical kinds of questions

# Recursion

**recursion** occurs when a thing is defined in terms of itself

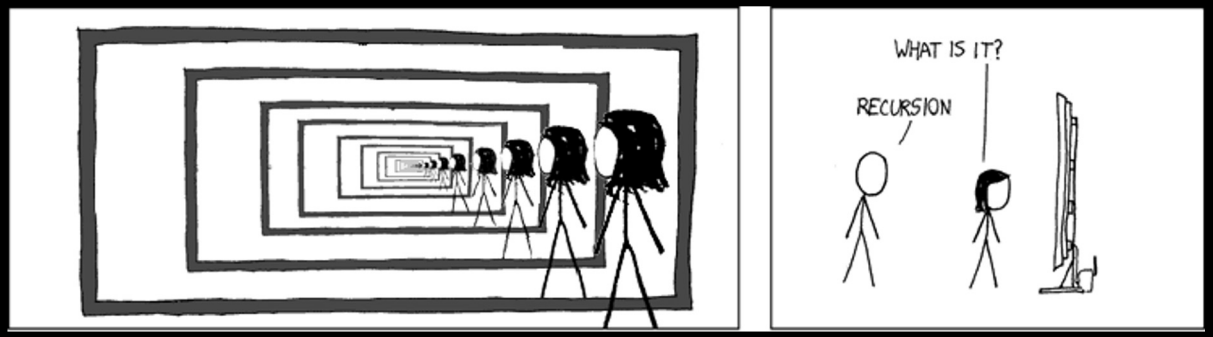
**recursive function:** is a function that calls itself with a smaller input than what it had originally received



# Recursion is another way to break big problems into small problems

If a problem is small, solve it directly. (we call that a "**base case**", something that you immediately know the answer to)

Otherwise, break off a piece of the problem that you can figure out, and then try solving the smaller remaining portion (by breaking off another piece and so on, over and over… that is our "**recursive step**").

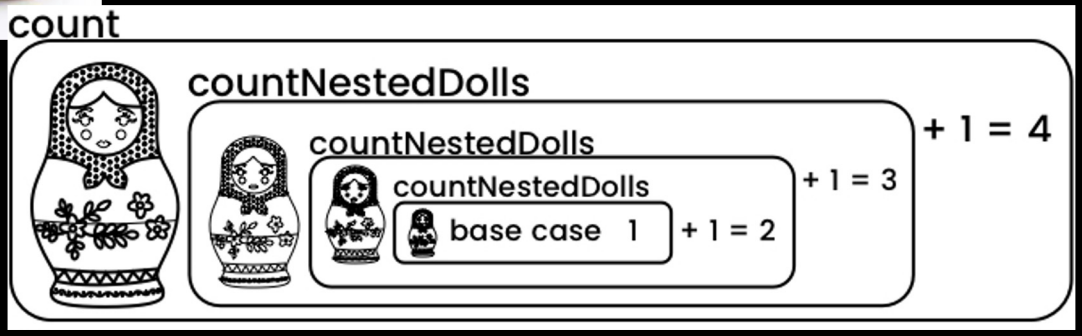


# Recursion intuition: take one step toward the solution and ask your clone to handle the rest

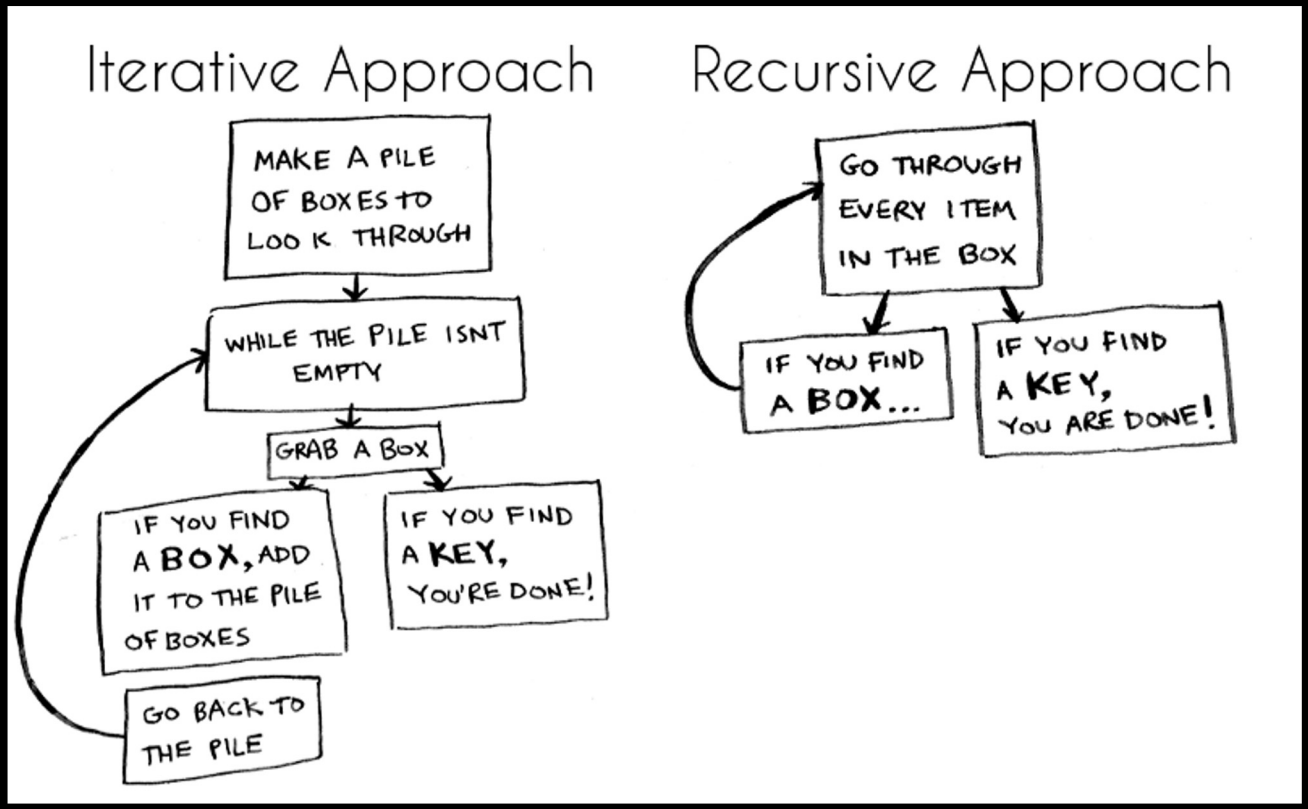
How many Russian nesting dolls do we have inside some large nesting doll?

No clue but 1 for this one + all the ones inside it!

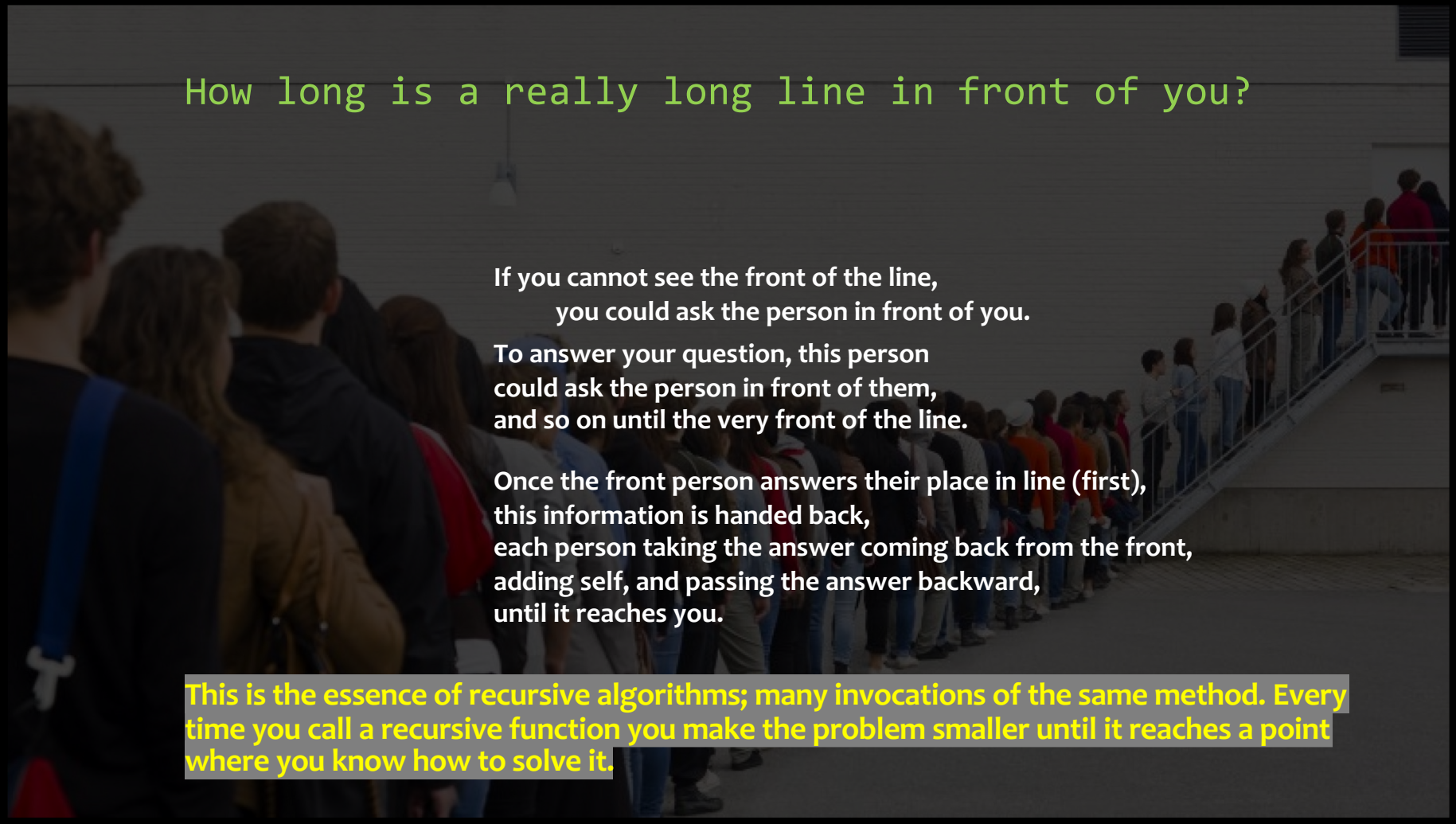
Keep opening dolls and adding one until we reach the tiny one that doesn't open ← Base Case of 1 doll.

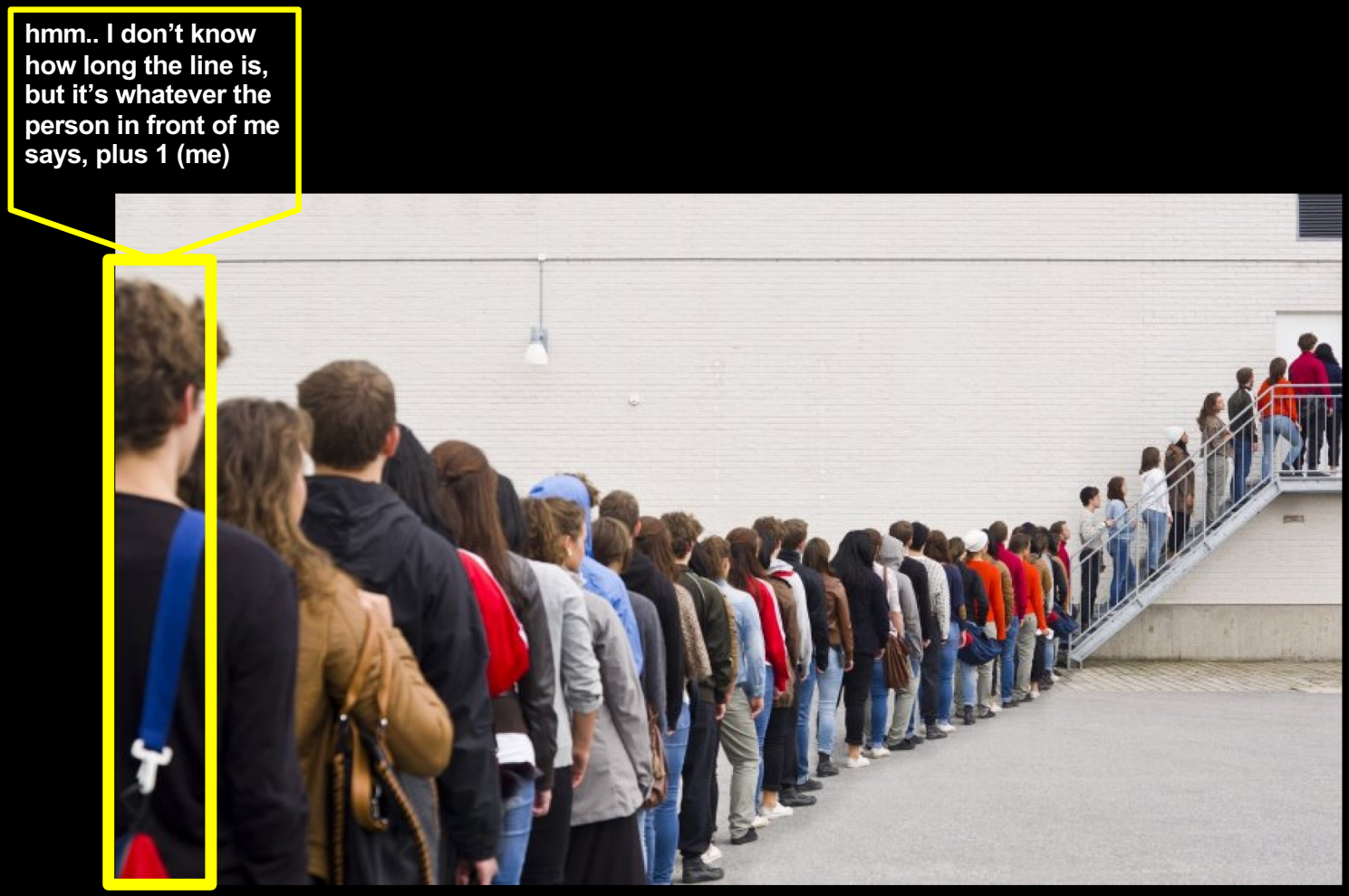


# Iteration vs Recursion:
# two ways to "loop" through your code




https://www.freecodecamp.org/news/how-recursion-works-explained-with-flowcharts-and-a-video-de61f40cb7f9/

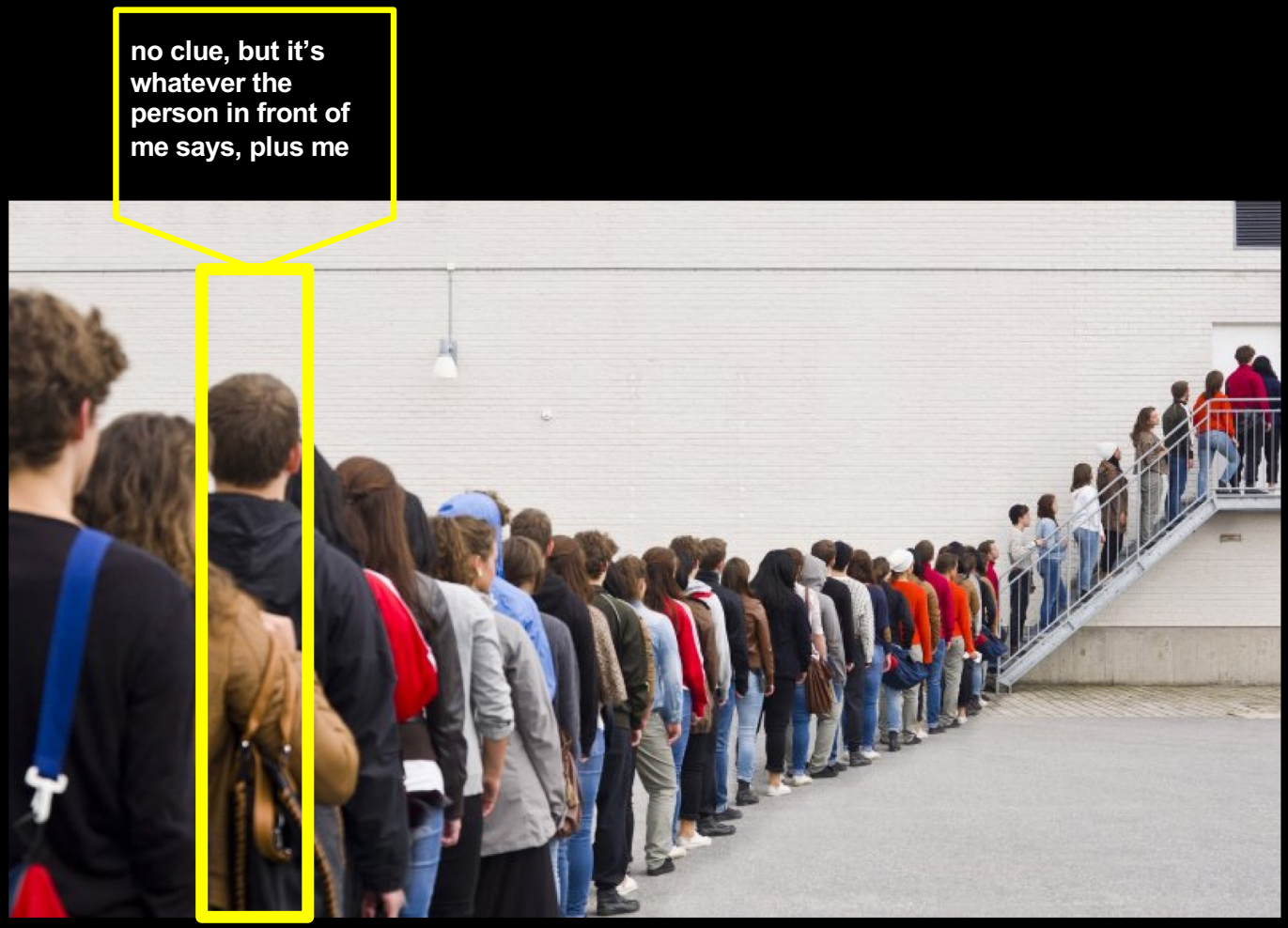

How long is a really long line in front of you?
If you cannot see the front of the line,
you could ask the person in front of you.
To answer your question, this person
could ask the person in front of them,
and so on until the very front of the line.
Once the front person answers their place in line (first),
this information is handed back,
each person taking the answer coming back from the front,
adding self, and passing the answer backward,
until it reaches you.
This is the essence of recursive algorithms; many invocations of the same method. Every time you call a recursive function you make the problem smaller until it reaches a point where you know how to solve it.




https://www.inc.com/jill-krasny/why-waiting-in-lines-makes-things-more-appealing.html

I don’t know either, but it’s whatever the person in front of me says, plus me

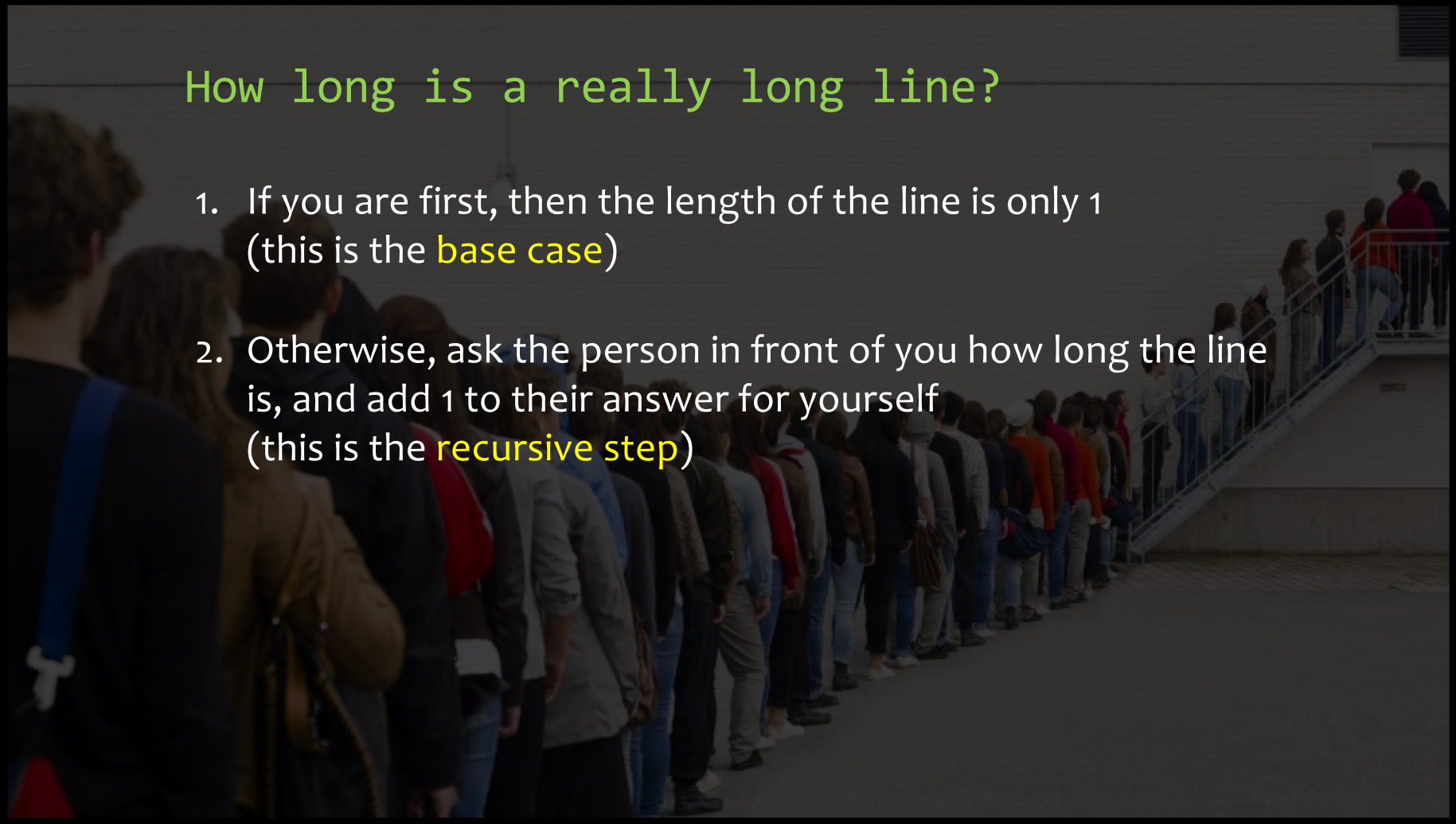
https://www.inc.com/jill-krasny/why-waiting-in-lines-makes-things-more-appealing.html




https://www.inc.com/jill-krasny/why-waiting-in-lines-makes-things-more-appealing.html


https://www.inc.com/jill-krasny/why-waiting-in-lines-makes-things-more-appealing.html


https://www.inc.com/jill-krasny/why-waiting-in-lines-makes-things-more-appealing.html



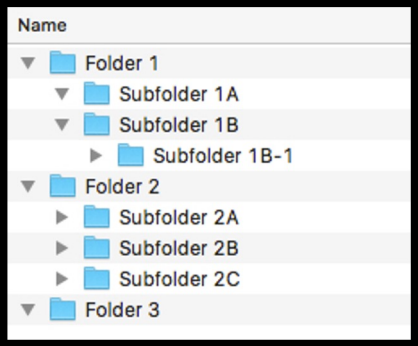
## How long is a really long line?

1. If you are first, then the length of the line is only 1
   (this is the base case)

2. Otherwise, ask the person in front of you how long the line is, and add 1 to their answer for yourself
   (this is the recursive step)

# How to show everything in a folder and all of its subfolders

1. Show everything in folder
2. Use #1 on each subfolder




The file hierarchy is recursive!

https://zapier.com/blog/organize-files-folders/

## Pseudocode to list all files in a folder

```
listAllFiles(folderOrFile):
      if folderOrFile is file:
            print folderOrFile
      else: //now we know it's a folder instead
            for each item in folderOrFile:
                  listAllFiles(item)
```

## Pseudocode to list all files in a folder

```
listAllFiles(folderOrFile):
      if folderOrFile is file:
            print folderOrFile
      else: //now we know it's a folder instead
            for each item in folderOrFile:
                  listAllFiles(item)
```

Base case:
When we see a file, just print it

# Pseudocode to list all files in a folder

```
listAllFiles(folderOrFile):
      if folderOrFile is file:
            print folderOrFile
      else: //now we know it's a folder instead
            for each item in folderOrFile:
                  listAllFiles(item)
```

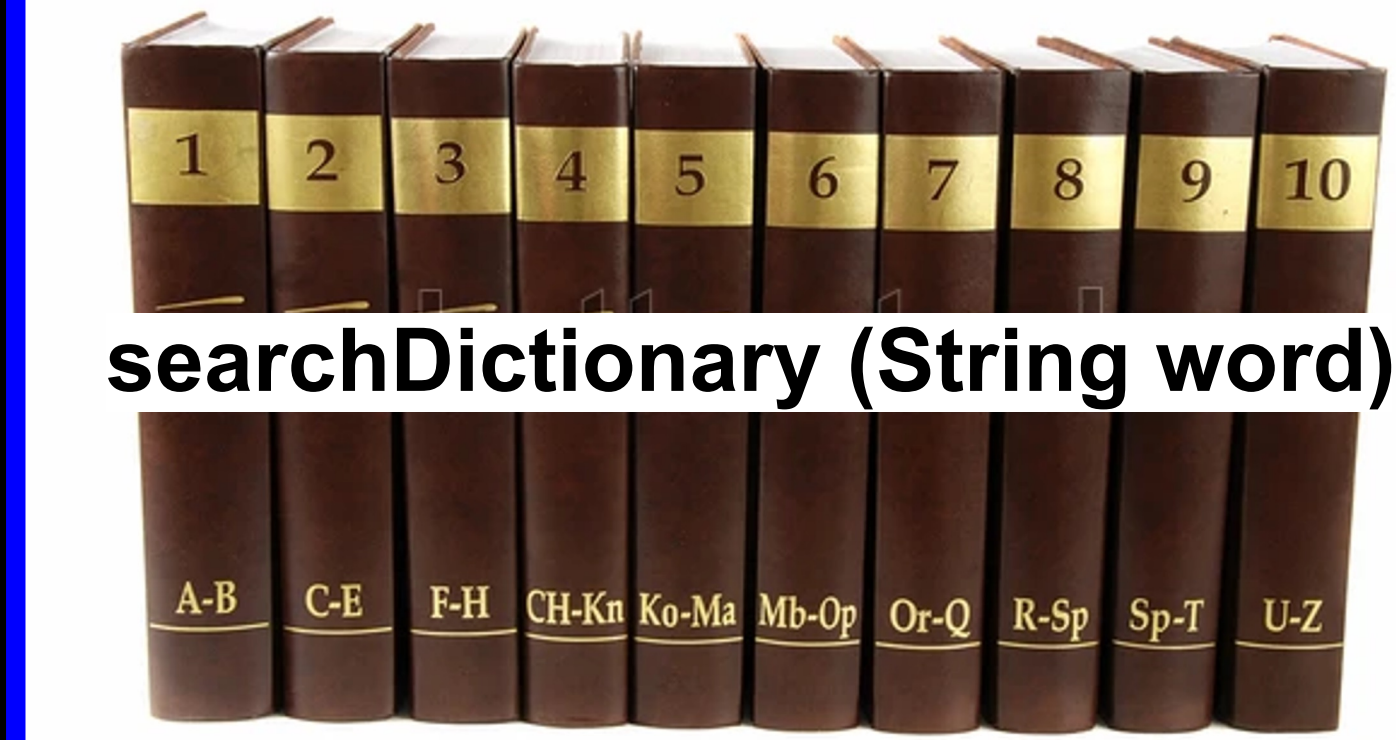
Recursive step:
When we see a folder,
call the listAllFiles function on it
to list the files

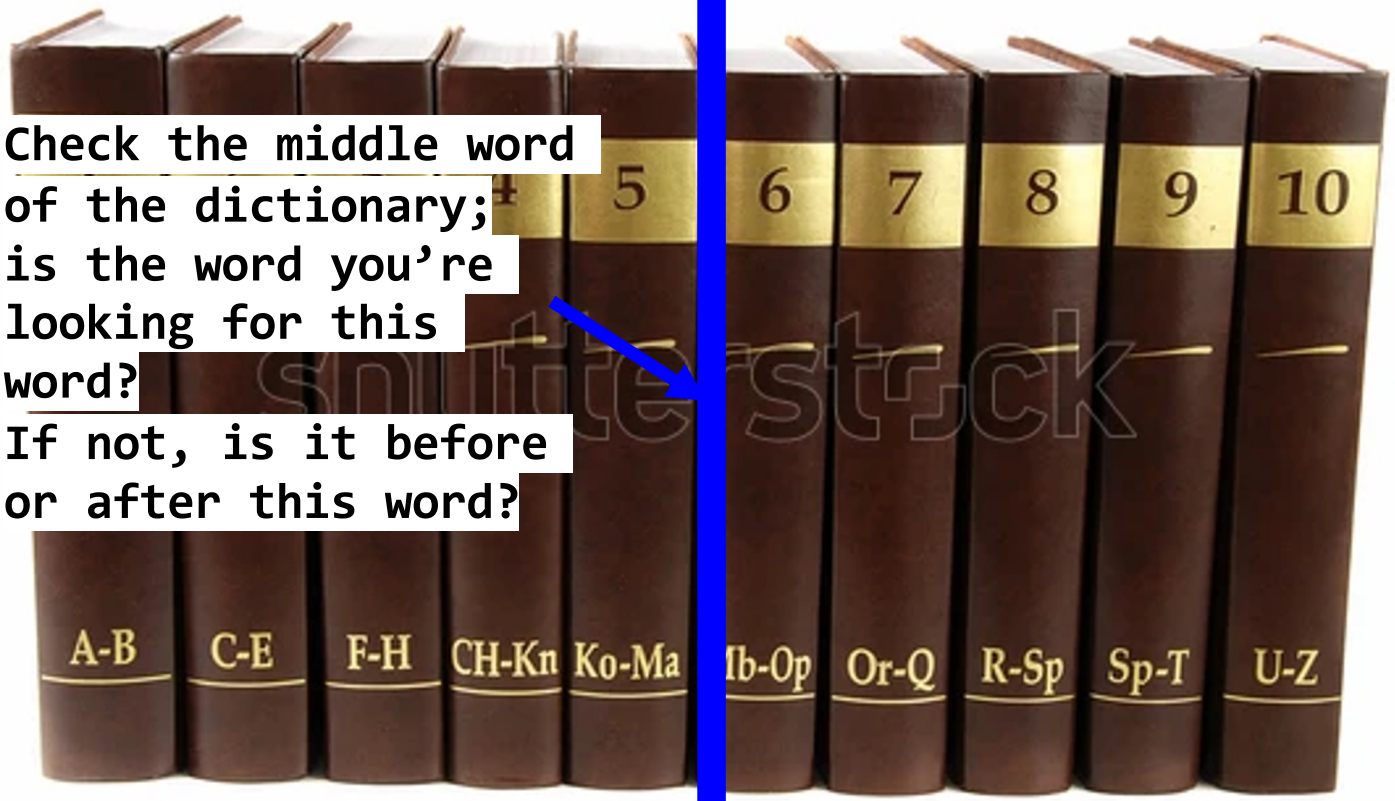
## search for a word in a dictionary

1. Base Case:

   If there's no words in the dictionary then your word is not in it

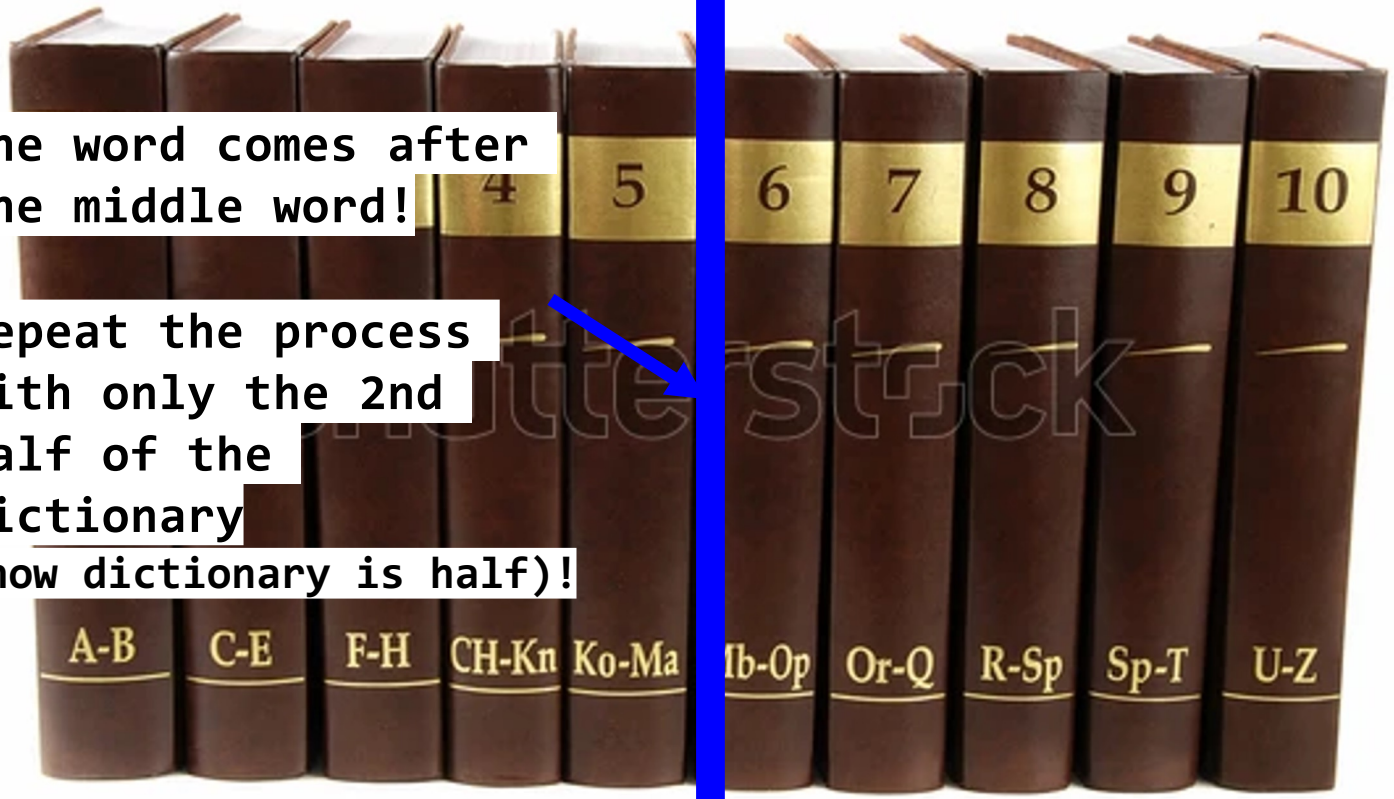
1. Check the word in the middle of dictionary:
   if it's your word then you're done!
   else
      if your word is alphabetically earlier, search in left half
      else search in right half

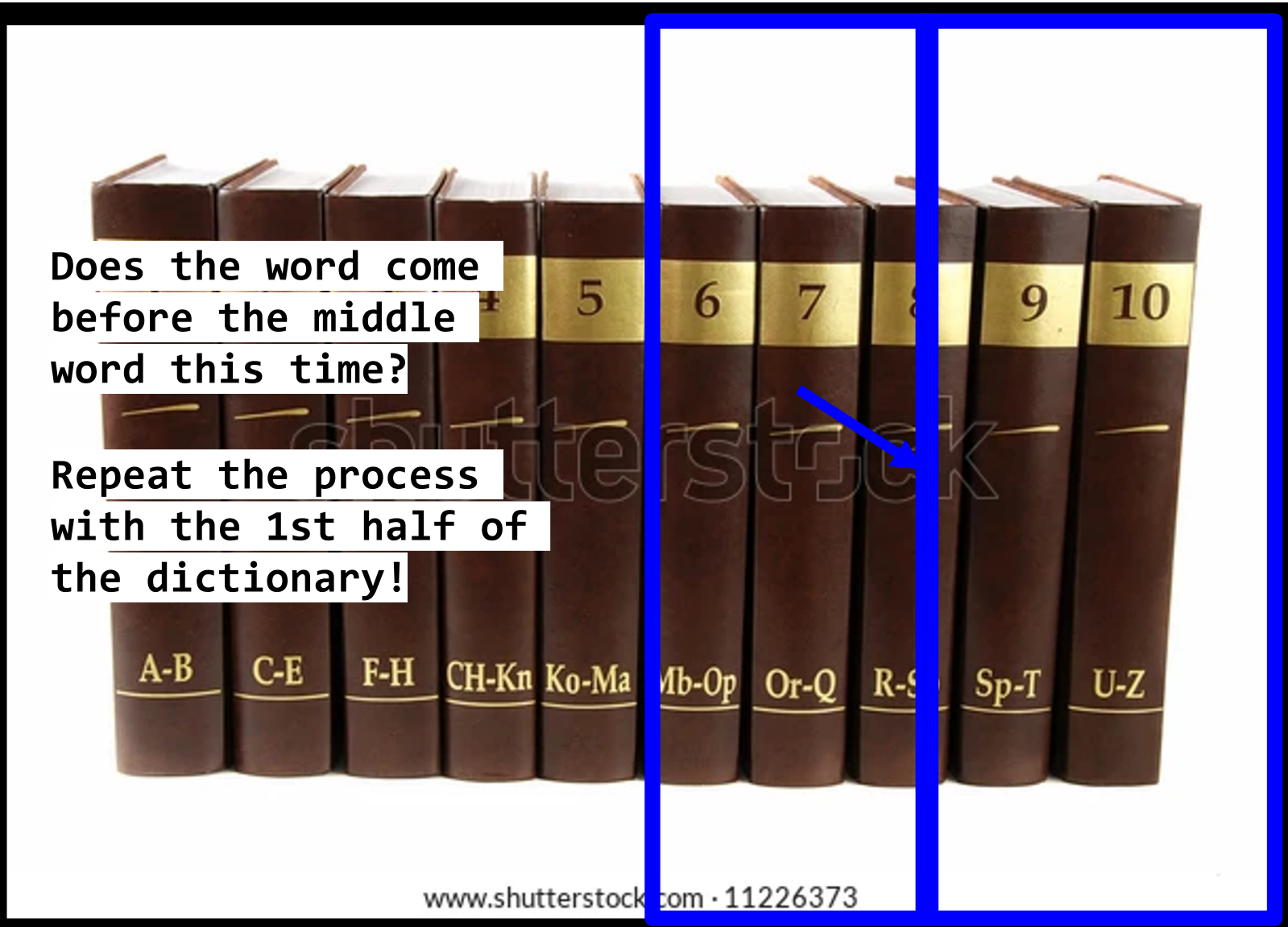

1
2
3
4
5
6
7
8
9
10
searchDictionary (String word)
A-B
C-E
F-H
CH-Kn
Ko-Ma
Mb-Op
Or-Q
R-Sp
Sp-T
U-Z
www.shutterstock.com · 11226373

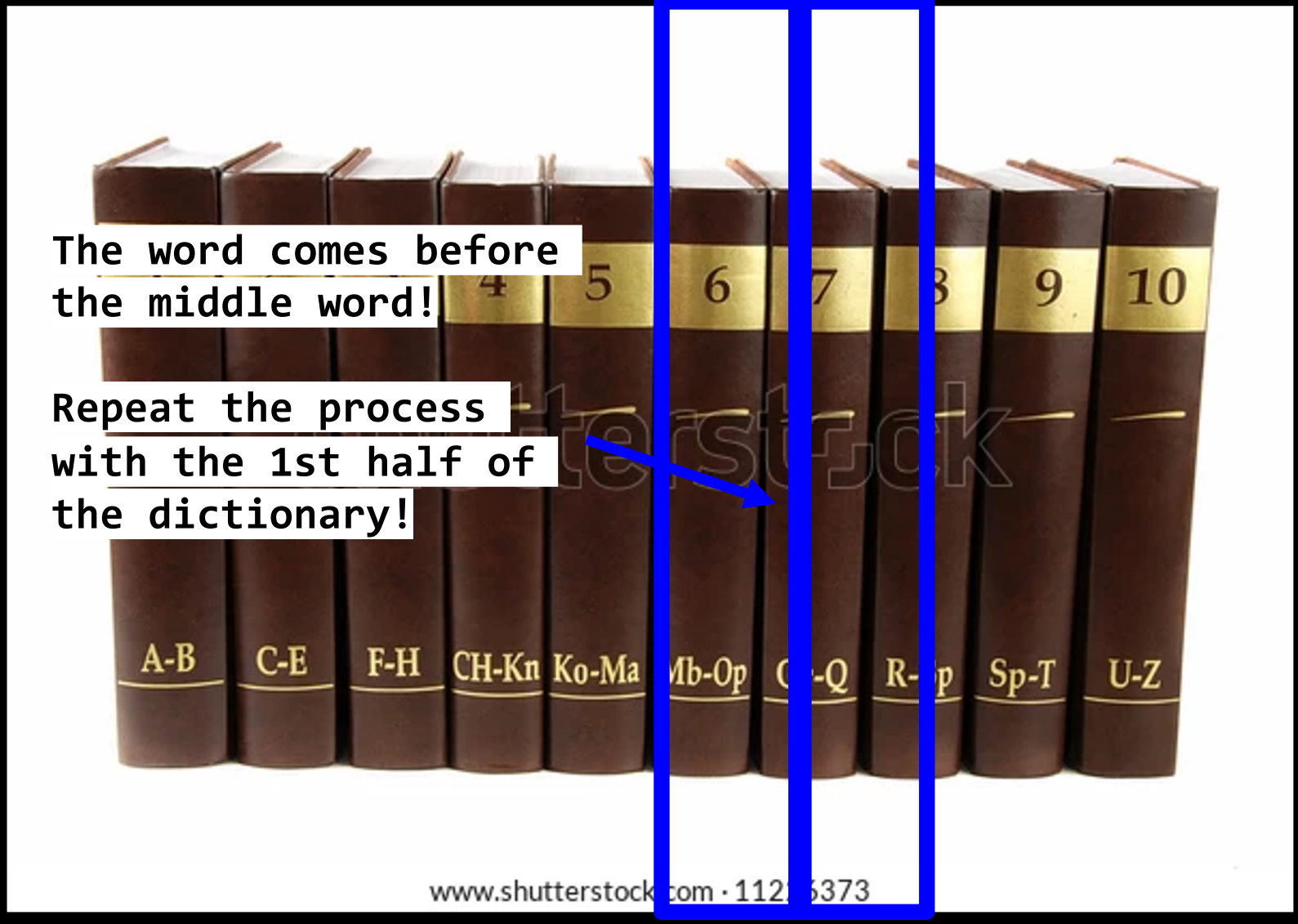

Check the middle word of the dictionary; is the word you're looking for this word?
If not, is it before or after this word?
5
6
7
8
9
10
A-B
C-E
F-H
CH-Kn
Ko-Ma
lb-Op
Or-Q
R-Sp
Sp-T
U-Z
www.shutterstock.com · 11226373

The word comes after the middle word!

Repeat the process with only the 2nd half of the dictionary (now dictionary is half)!




Does the word come before the middle word this time?
Repeat the process with the 1st half of the dictionary!
5
6
7
9
10
A-B
C-E
F-H
CH-Kn
Ko-Ma
Mb-Op
Or-Q
Sp-T
U-Z
www.shutterstock.com · 11226373


The word comes before the middle word!
Repeat the process with the 1st half of the dictionary!
A-B
C-E
F-H
CH-Kn
Ko-Ma
Sp-T
U-Z

# Other possible uses of recursion:

## Mathematical Summations

Sum all values from n to 1:

sumAll(n) =
n + sumAll(n-1) =
n + ((n-1) + sumAll(n-2)) =
…
n + (n-1) + (n-2) + … + 2 + 1

## How do we implement recursion in Java?

A. nested for loops
B. a for loop in a while loop
C. a function with a loop
D. a function that calls itself
E. lambda-R (the recursion operator)

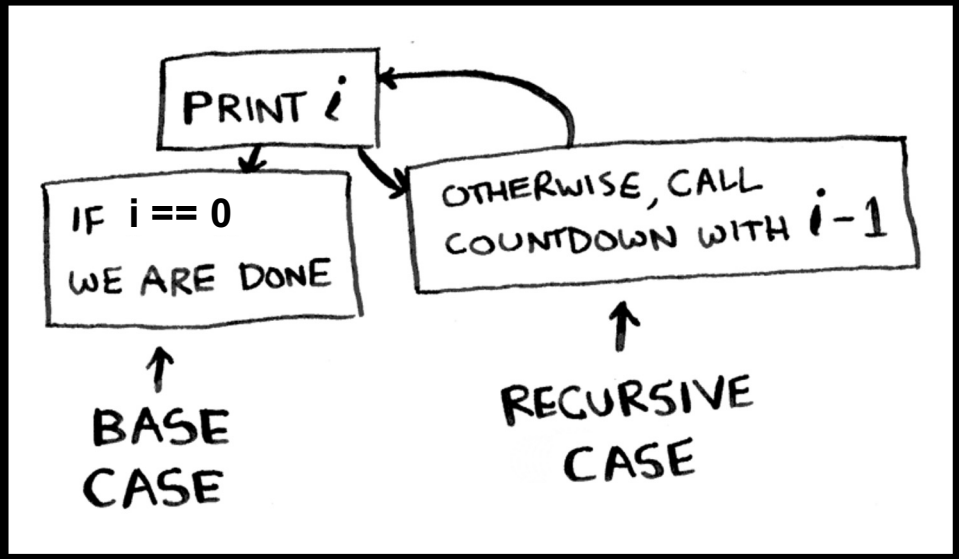
## How do we implement recursion in Java?

A. nested for loops
B. a for loop in a while loop
C. a function with a loop
**D. a function that calls itself**
E. lambda-R (the recursion operator)

There is no lambda-R, the recursion operator. I made that up...

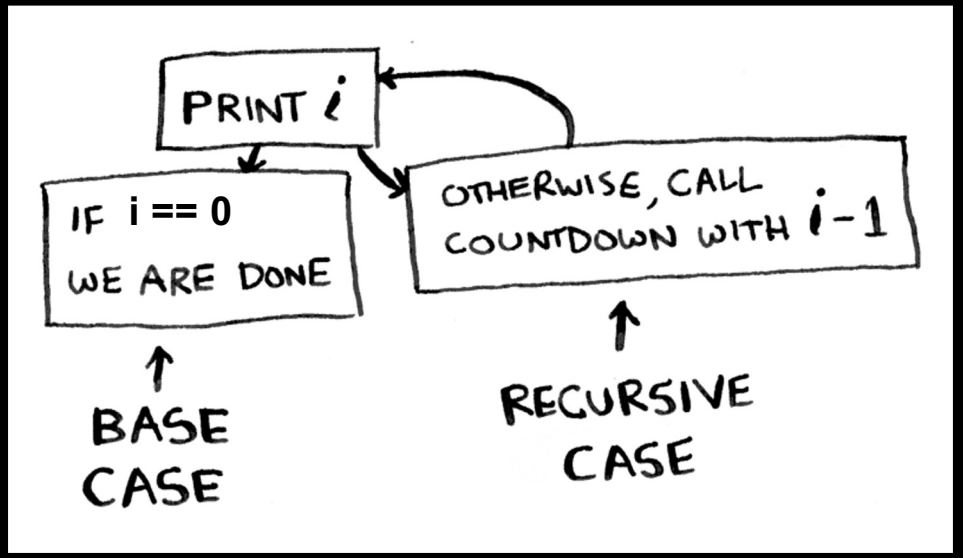
## `blastoff(int countdown)`

1. If the countdown is zero, yell BLASTOFF!
2. Otherwise, yell the countdown, decrease the countdown by 1, and redo #1



# blastoff(int countdown)

```
public static void blastoff(int countdown) {
    if (countdown == 0) //base case
        System.out.println("Blastoff!");
    else {
        System.out.println(countdown+"!");
        countdown(n-1); //recursive step
    }
}
```




https://repl.it/@jspacco/Blastoff-Recursively

# Other possible uses of recursion:
## Factorials

Factorial of 5 =

factorial(5) =

5 * factorial(4) =

5 * 4 * factorial(3) =

5 * 4 * 3 * factorial(2) =

5 * 4 * 3 * 2 * factorial(1) =

5 * 4 * 3 * 2 * 1

## Other possible uses of recursion

Factorial in math

5! = 5 * 4 * 3 * 2 * 1

A. 5! = 5 * 4!
B. 4! = 4 * 3!
C. 3! = 3 * 2!
D. 2! = 2 * 1!
E. 1! = 1

Which is the base case?

# Other possible uses of recursion

Factorial in math

5! = 5 * 4 * 3 * 2 * 1

A. 5! = 5 * 4!
B. 4! = 4 * 3!
C. 3! = 3 * 2!
D. 2! = 2 * 1!
**E. 1! = 1**

Which is the base case?

factorial(1)=1

## Other possible uses of recursion

Factorial in math

5! = 5 * 4 * 3 * 2 * 1

A. 5! = 5 * 4!
B. 4! = 4 * 3!
C. 3! = 3 * 2!
D. 2! = 2 * 1!
E. 1! = 1

Which is the recursive step?

## Other possible uses of recursion

Factorial in math

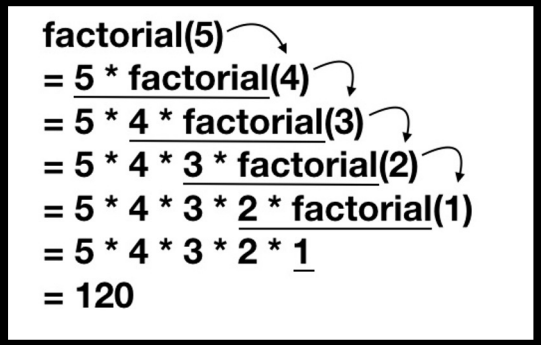
5! = 5 * 4 * 3 * 2 * 1

**A.** 5! = 5 * 4!
**B.** 4! = 4 * 3!
**C.** 3! = 3 * 2!
**D.** 2! = 2 * 1!
E. 1! = 1

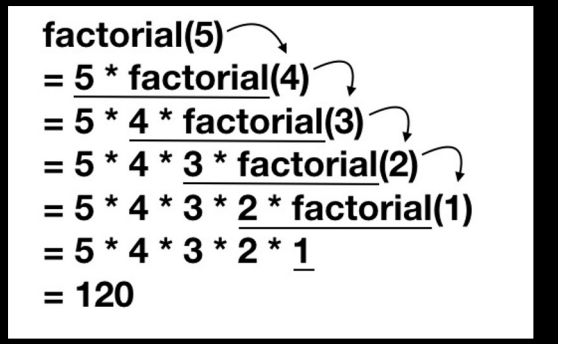
Which is the recursive step?

## factorial in Java

```
static int fact(int n) {
            // Base case
            // Recursive step
}
```



```
static int fact(int n) {
            // Base case
            // Recursive step
}
```




Which is the best base case for calculating factorial?

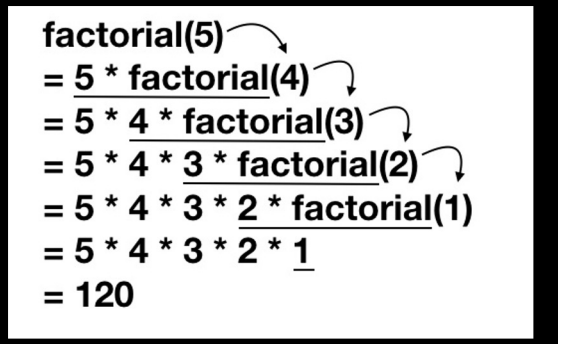
A
```
if (n<=1) {
      return 1;
}
```

C
```
return fact(n-1);
```

B
```
return n-1;
```

D
```
return n * fact(n-1);
```

```
static int fact(int n) {
            // Base case
            // Recursive step
}
```




Which is the best base case for calculating factorial?

A
```
if (n<=1) {
      return 1;
}
```

C
```
return fact(n-1);
```

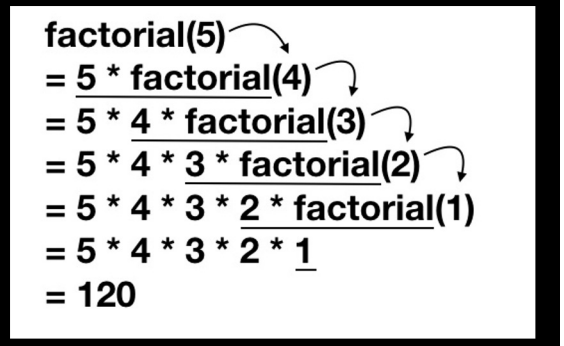
B
```
return n-1;
```

D
```
return n * fact(n-1);
```

```
static int fact(int n) {
      if (n<=1) { //base case
            return 1;
      }
      // Recursive step
}
```

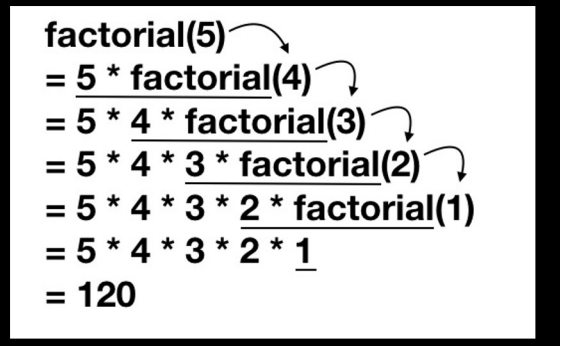
Which should be the recursive step?

```
A if (n >= 1) {
      return n;
  }
B return n * n-1;
C return n * fact(n-1);
D return fact(n) * fact(n-1);
E none of the above
```



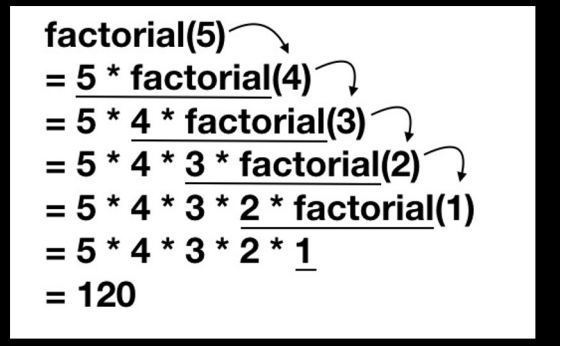
```
static int fact(int n) {
      if (n<=1) { //base case
            return 1;
      }
      // Recursive step
}
```

Which should be the recursive step?



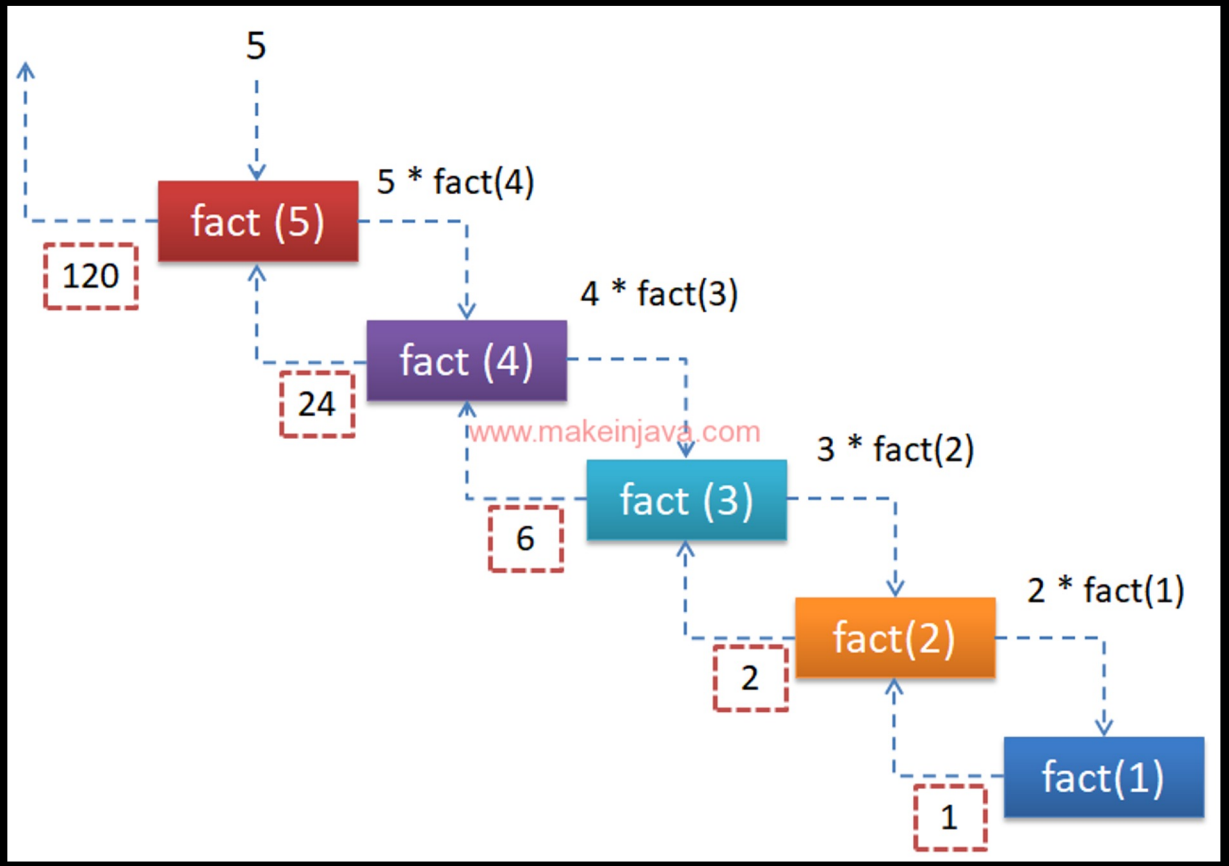

A
```
if (n >= 1) {
      return n;
  }
```
B
```
return n * n-1;
```
**C**
```
return n * fact(n-1);
```
D
```
return fact(n) * fact(n-1);
```
E none of the above

Recursion can’t go on forever.
Eventually we get a call
where n<=1 ← BASE CASE




```
static int fact(int n) {
      if (n<=1) { //base case
            return 1;
      }
      return n * fact(n-1); //recursive step
}
```

https://repl.it/@jspacco/141-factorial


5
fact (5)
5 * fact(4)
120
fact (4)
4 * fact(3)
24
www.makeinjava.com
fact (3)
3 * fact(2)
6
fact(2)
2 * fact(1)
2
fact(1)
1