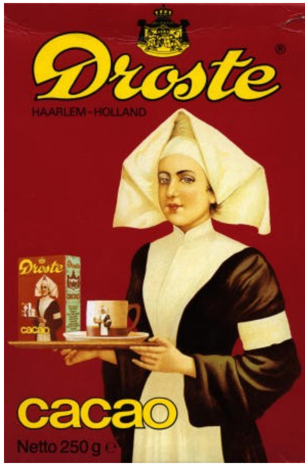
Merge Sort

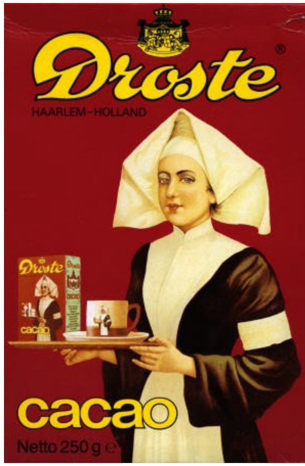
What is the output of this Python function ?




```
def printStars(n):
        if n == 1:
            print “*”,
        else:
            print “#”,
            printStars(n-1)

printStars(5)
```

What is the output of this Python function ?

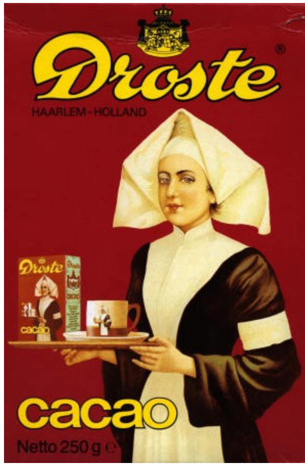



```
def printStars(n):
        if n == 1:
            print “*”,
        else:
            print “#”,
            printStars(n-1)

printStars(5)
```

Output: # # # # *

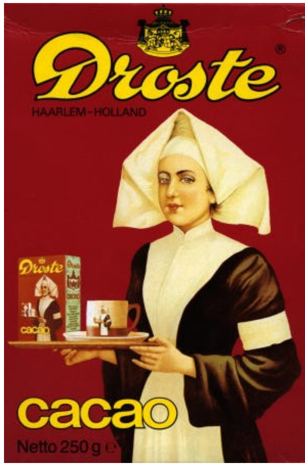
What does this method do?




```
def printStars(n):
        if n == 1:
            print “*",
        else:
            printStars(n-1)
            print “#”,

printStars(5)
```

Recursive Functions

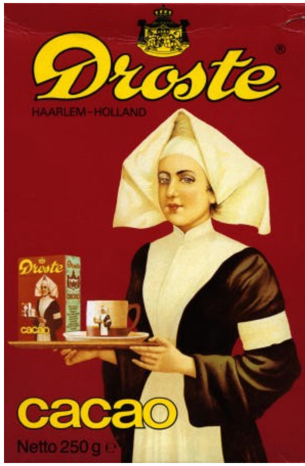



```
def printStars(n):
        if n == 1:
            print "*",
        else:
            printStars(n-1)
            print "#",

printStars(5)

Output: * # # # #
```

Recursive Functions




```
def printStars(n):
        if n == 1:
            print "*",
        else:
                printStars(n-1)
            print "#",

printStars(5)

Output: * # # # #
```

```
def printStars(n):
        if n == 1:
            print "*",
        else:
            print "#",
            printStars(n-1)

printStars(5)

Output: # # # # *
```

About 37,000,000 results (0.48 seconds)

Did you mean: ***recursion***

Dictionary

Search for a word

re·cur·sion

/rəˈkərZHən/

noun MATHEMATICS • LINGUISTICS

the repeated application of a recursive procedure or definition.

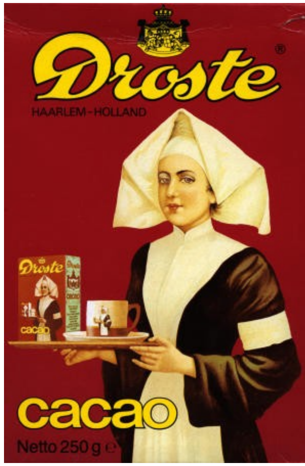
- a recursive definition.
  plural noun: **recursions**

Recursion

- **recursion**: The definition of an operation in terms of itself.
  - Solving a problem using recursion depends on solving smaller occurrences of the same problem.

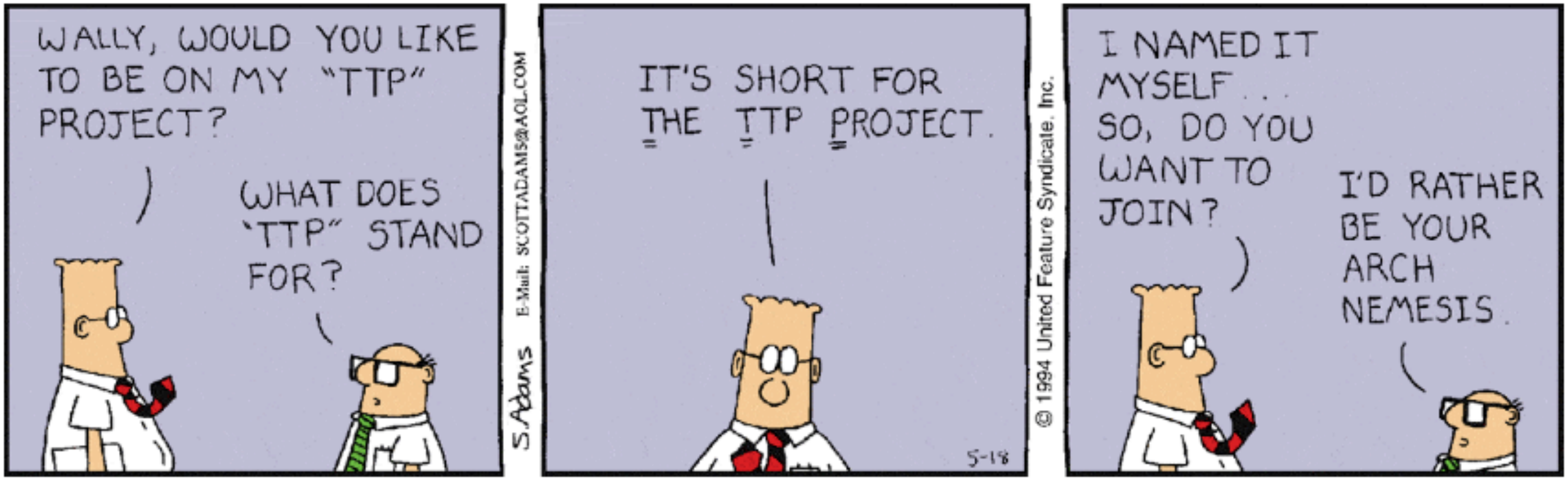
- **recursive programming**: Writing functions that call themselves
  - directly or indirectly
  - An equally powerful substitute for *iteration* (loops)
  - But sometimes much more suitable for the problem

Recursive Function




```
def printStars(n):
        if n == 1:
            print “*”,
        else:
            printStars(n-1)
            print “#”,
```

Recursive Acronyms



GNU — GNU's Not Unix
KDE — KDE Desktop Environment
PHP - PHP: Hypertext Preprocessor
PNG — PNG's Not GIF (officially "Portable Network Graphics")
PIP — PIP installs packages

http://search.dilbert.com/comic/Ttp

Beware of infinite repetition

Q: How did the programmer die in the shower?

Beware of infinite repetition

Q: How did the programmer die in the shower?

He read the shampoo bottle instructions: Lather. Rinse. Repeat.

Cases

- Every recursive algorithm has at least 2 cases:
  - **base case**: A simple instance that can be answered directly.
  - **recursive case**: A more complex instance of the problem that cannot be directly answered, but can instead be described in terms of smaller instances.
  - Can have more than one base or recursive case, but all have at least one of each.
  - A crucial part of recursive programming is identifying these cases.

Base and Recursive Cases: Example

```
def printStars(n):
        if n == 1:
            print "*",
        else:
            printStars(n-1)
            print “#",
```

Everything recursive can be done non-recursively

```
def printStars(n):
        for i in range(n):
            print "*",
```

Exercise

- Write a method `reverseLines` that accepts a file and prints to the lines of the file in reverse order.
  - Write the method recursively and without using loops.

  - Example input: Expected output:

```
this      no?
is        fun
fun       is
no?       this
```

  - What are the cases to consider?
    - How can we solve a small part of the problem at a time?
    - What is a file that is very easy to reverse?

Reversal pseudocode

- Reversing the lines of a file:
  - Read a line L from the file.
  - Print the rest of the lines in reverse order.
  - Print the line L.

- If only we had a way to reverse the rest of the lines of the file....

Reversal solution

```
def reverseLines():
    line = ifile.readline().strip("\n")
    if(line):
        reverseLines()
        print(line)


reverseLines()
```

- Where is the base case?

Tracing our algorithm

- **call stack**: The method invocations running at any one time.

```
def reverseLines():
    line = ifile.readline().strip("\n")
```

```
def reverseLines():
    line = ifile.readline().strip("\n")
```

```
def reverseLines():
    line = ifile.readline().strip("\n")
```

```
def reverseLines():
    line = ifile.readline().strip("\n")
```

```
def reverseLines():
    line = ifile.readline().strip("\n")
    if(line):    // false
        ...
```

output:

```
no?
fun
is
this
```

input file:

```
this
is
fun
no?
```

Divide and Conquer - Recursive Approach

```
def divideAndConquer(Instance Size):

    if (instance is trivial): // base case
            solve and return

    else: // recursive case

       part1 = divideAndConquer(first part of instance)
       part2 = divideAndConquer(second part of instance)

       combinedParts = combine part1 and part2

       return combinedParts
```

Binary search

```
def binarySearch(dictionary,  word):

    if (dictionary has one page): // base case
             scan the page for word

    else: // recursive case

       open the dictionary to a point near the middle
       determine which half of the dictionary contains word

       if (word is in first half of the dictionary):
          binarySearch(first half of dictionary, word)
       else:
          binarySearch(second half of dictionary, word)
```

Binary search

- Write a method `binarySearch` that accepts a sorted array of integers and a target integer and returns the index of an occurrence of that value in the array.
  - If the target value is not found, return -1

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

```
index  = binarySearch(data, 42)   // 10
index2 = binarySearch(data, 66)   // -1
```

Binary search

```
def binarySearch(self,sortedArr,num,startIndex,endIndex):
        if(startIndex > endIndex):
            return -1;
        else:
            mid = (startIndex + endIndex)/2
            if sortedArr[mid]== num:
                return mid
            elif sortedArr[mid] < num:
                return self.binarySearch(sortedArr,num,mid+1,endIndex)
            else:
                return self.binarySearch(sortedArr,num,startIndex,mid-1)

    def binSearch(self,sortedArr,num):
        return self.binarySearch(sortedArr,num,0,len(sortedArr)-1)
```

Divide and Conquer Algorithms (Example: Merge Sort)

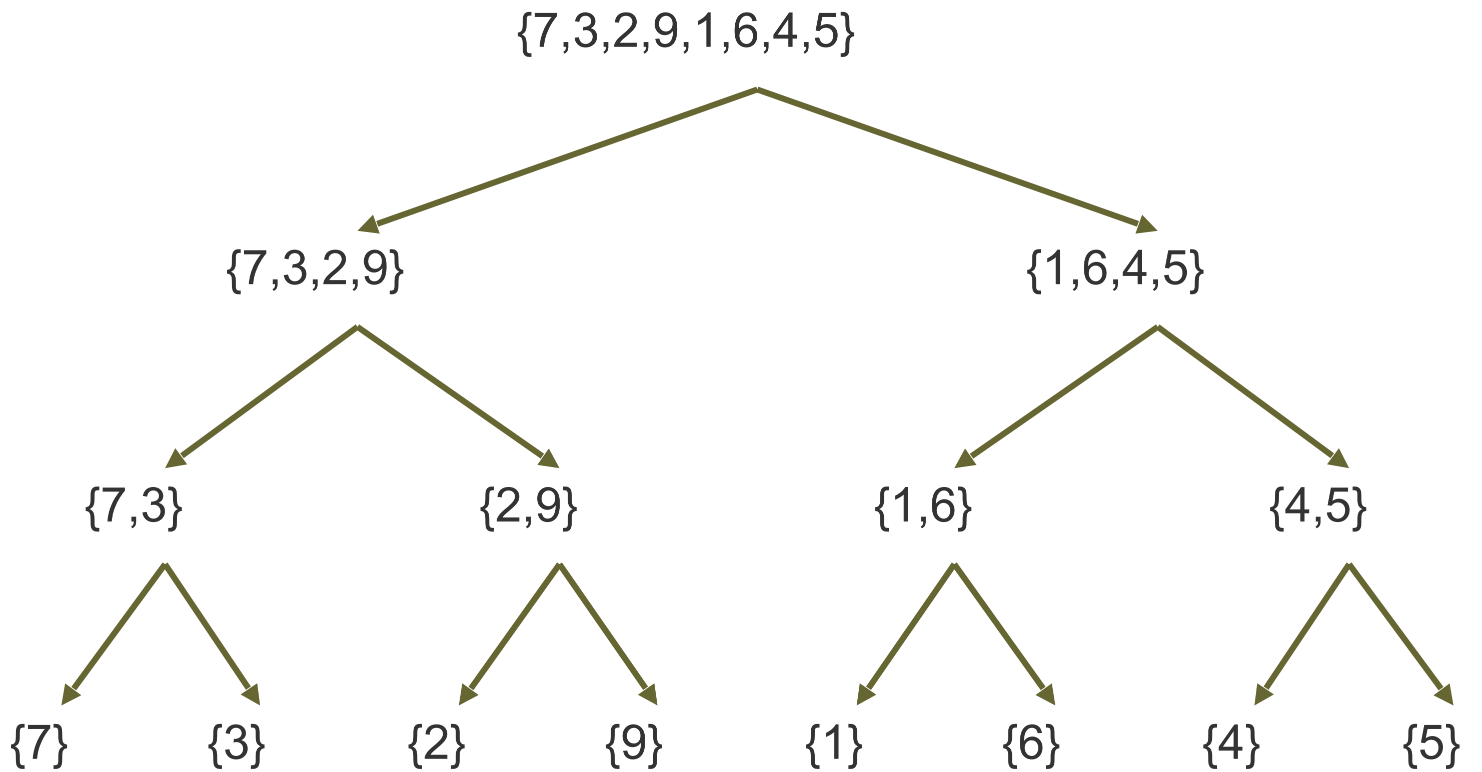
- Basic idea
  - Divide data into smaller subproblems
  - Conquer the subproblems recursively
  - Combine the solutions for the subproblems into a solution for the original problem

Mergesort

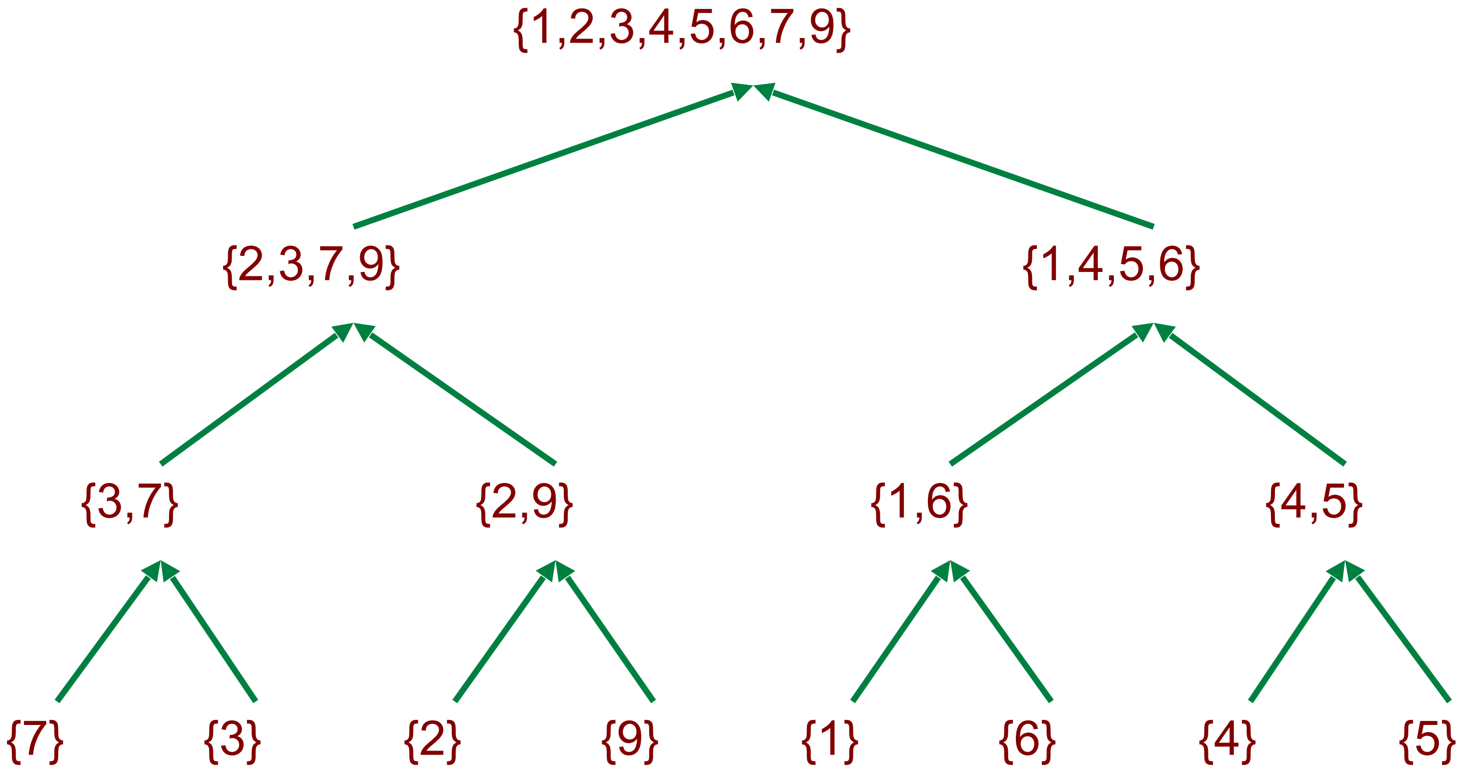
```
MergeSort(A):
     n <- length(A)
     if n <= 1:
       return A
     L <- mergeSort(A[1:n/2])
     R <- mergeSort(A[n/2 + 1 : n])
     return(merge(L,R))
```

```
merge(L,R):
     ll <- length(L)
     rl <- length(R)
     n <- ll + rl
     S <- Empty array of size n
     i <- 1
     j <- 1
     k <- 1
     while i <= ll and j <= rl:
         if L[i] < R[j]:
            S[k] <- L[i]
            i <- i + 1
         else:
            S[k] <- R[j]
            j <- j + 1
         k <- k + 1
     while i <= ll:
         S[k] <- L[i]
         i <- i + 1
         k <- k + 1
     while j <= rl:
         S[k] <- R[j]
         j <- j + 1
         k <- k + 1
     return(S)
```

Merge Sort - Divide



Merge Sort - Merge



Merge Sort - Divide