

# Unit 4 **Recursion**

Data Structures and Algorithms



## Agenda

- What is recursion?
- Some examples of recursion.
  - Factorial function
  - Multiply 2 numbers using addition
  - Calculating the sum of an array of integers
  - Power
  - Binary Search

### What is recursion?

- Recursion is a stratigy for solving problems.
- Recursion is an alternative to iteration. Any recursive function can be solved using iteration.
- A recursive method requires fewer lines than using a loop.
- Loops are more efficient (recursion consumes more memory).
- Some problems are very difficult to solve by iteration.

```
for (i=0; i < cond; ++i){
Statements

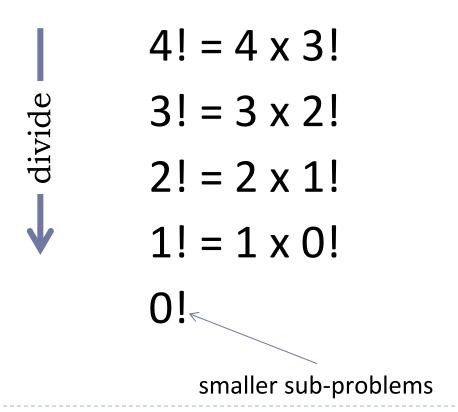
if (cond)

fewer
else:
myFunction()

fewer
lines
```

## What is recursion?

 Recursion divides a complex problem into smaller sub-problems that can be solved directly.



#### **Base Case vs Recursive Case**

- The Case base is a smaller sub-problems that can be solved directly.
- The recursive case is where the function calls itself again and again until it reaches the base case.



Find the factorial of 4!?

```
4! = 4 \times 3!
    3! = 3 \times 2!
divide
   2! = 2 \times 1!
    1! = 1 \times 0!
    ()!
```

Find the factorial of 4!?



Find the factorial of 4!?

$$n! = \begin{cases} 1 & \text{Recursive definition} \\ n \cdot (n-1)! & \text{if } n = 0 \\ if n \ge 1. \end{cases}$$

```
def factorial(n):
    if n==0:#base case
      return 1
    else: #recursive case
      return n*factorial(n-1)
```

Big-O for factorial function?

O(n): There is n+1 recursive calls (each one counts as 1 operation).

$$5 \times 3 = 5 + 5 + 5 = 15$$

Recursive case x+(x, n-1)

#### **Recursive definition**

$$x * n = \begin{cases} x, & if n = 1 \\ x + (x, n - 1) & if n \ge 1 \end{cases}$$

def multiplyRec(x, n):

return x

O(?)

else:

return x + multiplyRec(x, n-1)

# Example: Calculating the sum of an array of integers.

Find sum of [1,3,5,7,9]?

```
sum [1,3,5,7,9] = 1 + sum [3,5,7,9]

sum [3,5,7,9] = 3 + sum [5,7,9]

sum [5,7,9] = 5 + sum [7,9]

sum [7,9] = 7 + sum [9]

sum [9] = 9
```

# Example: Calculating the sum of an array of integers.

Find sum of [1,3,5,7,9]?

sum 
$$[1,3,5,7,9] = 1 + sum [3,5,7,9] = 25$$
  
sum  $[3,5,7,9] = 3 + sum [5,7,9] = 24$   
sum  $[5,7,9] = 5 + sum [7,9] = 21$   
sum  $[7,9] = 7 + sum [9] = 16$   
sum  $[9] = 9$ 

# Example: Calculating the sum of an array of integers.

#### Recursive definition

$$\sum_{i=0}^{n} a_i = \begin{cases} a_0, & if \ n = 1 \\ a_0 + (a_{1,n-1}) & if \ n > 1 \end{cases}$$

```
def sumArray (a):
  if len(a)==1:
    return a[0]
  else:
    return a[0] + sumArray(a[1:])
```

Time complexity: O(n)

#### Example: power (base, exponente)

 $\geq$  Power function: = 3<sup>4</sup>, power(x,n)=x<sup>n</sup>

```
power (3, 4) = 3 * power (3, 3)
```

power 
$$(3, 3) = 3 * power (3, 2)$$

power 
$$(3, 2) = 3 * power (3, 1)$$

power 
$$(3, 1) = 3 * power (3, 0)$$

$$power (3, 0) = 1$$

### Example: power (base, exponente)

 $\geq$  Power function: = 3<sup>4</sup>, power(x,n)=x<sup>n</sup>

#### Example: power (base, exponente)

#### Recursive definition

$$x^n = \begin{cases} 1, & if \ n = 0 \\ x * (x, n - 1) & if \ n > 1 \end{cases}$$

```
def power(x, n):
  if n==0:
    return 1
  else:
    return x*power(x,n-1)
```

Time complexity: O(n)

Input: a <u>sorted</u> array of integers and a number

$$x = 23$$

$$start = 0$$

$$mid = (0 + 8) / 2 = 4$$

end = 8

#### 1<sup>st</sup> Iteration

- 1) If x == A[4], Found!!!
- 2) If x < A[4], search from start to mid-1
- 3) If x > A[4], search from mid+1 to end

Input: a <u>sorted</u> array of integers and a number

$$x = 23$$

2<sup>st</sup> Iteration

$$mid = (5 + 8) / 2 = 6$$

- 1) If x == A[6], Found!!!
- 2) If x < A[6], search from start to mid-1
- 3) If x > A[6], search from mid+1 to end

Input: a <u>sorted</u> array of integers and a number

x = 7 (which does not exist in the list)

start=0

$$mid = (0 + 8) / 2 = 4$$

end=8

1<sup>st</sup> Iteration

- 1) If x == A[4], Found!!!
- 2) If x < A[4], search from start to mid-1
- 3) If x > A[4], search from mid+1 to end

Input: a <u>sorted</u> array of integers and a number

x = 7 (which does not exist in the list)

start=0 end=3 
$$mid=(0+3)/2=1$$

#### 2<sup>nd</sup> Iteration

- 1) If x == A[1], Found!!!
- 2) If x < A[1], search from start to mid-1
- 3) If x > A[1], search from mid+1 to end

Input: a <u>sorted</u> array of integers and a number

x = 7 (which does not exist in the list)

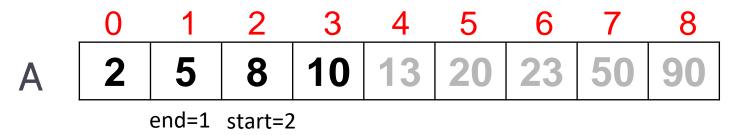
start=2 end=3 mid=(2+3)/2=2

#### 3<sup>rd</sup> Iteration

- 1) If x == A[2], Found!!!
- 2) If x < A[2], search from start to mid-1
- 3) If x > A[2], search from mid+1 to end

Input: a sorted array of integers and a number

x = 7 (which does not exist in the list)



#### 4<sup>th</sup> Iteration

start > end!!! : the array does not contain it!!!

### Example: implementation of binary search

- 1) If x == A[mid], Found!!! Base case!!!
- 2) If x < A[mid], search from start to mid-1
- 3) If x > A[mid], search from mid+1 to end

Recursive cases!!!

```
def binary search(data,x):
  if len(data)==0:
    return False
 #integer division
 mid=len(data)//2
  if x==data[mid]: #base case
    return True #found!!!
  elif x<data[mid]: #recursive case,
    #search at the first half of the array
    return binary search(data[0:mid],x)
  else: #x>data[mid], recursive case
    #search at the second half of the array
    return binary search(data[mid+1:],x)
```

# Example: Analysis of binary search function

> Big-Oh for binary search function?

At each iteration, the array is divided by half.

- At Iteration 1, Length of array = n/2<sup>1</sup>
- At Iteration 2, Length of array = n/2<sup>2</sup>
- At Iteration 3, Length of array = n/2<sup>3</sup>

----

After k divisions, the length of array becomes 1
 Length of array = n/2<sup>k</sup> = 1
 n = 2<sup>k</sup>
 k = log<sub>2</sub> (n)

## Types of recursion

- Linear Recursion: a recursive call could produce at most one new recursive call. Example: Factorial, Power, Binary Search, etc.
- Binary recursion: a recursive call can generate two new recursive calls. Example, Fibonacci Numbers.
- Multiple Recursion: a recursive call can generate three or more recursive calls. Example: exploring a file system.

## **Binary Recursion:**

Fibonacci Numbers

### Binary Recursion: Fibonacci Numbers

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
 Fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ Fib(n-1) + Fib(n-2) & \text{if } n>1 \end{cases}
def fib(n):
                                  Is it an efficient way to compute fib(50)?
     if n<=1:
          return n;
     else:
          return fib(n-1) + fib(n-2)
```

### Binary Recursion: Fibonacci Numbers

No, no, no! This code is inefficient.

 $O(2^{n})$ 

