

# Recursive procedures

1. What is a recursive procedure?
2. Recursive versus iterative
3. An application: binary search

## 1. What is a recursive procedure?

A recursive procedure calls itself.

A recurrence function is defined using itself.

General form of simple recurrence function  $f()$ :

$$f(\text{argument}) = \begin{cases} \text{base case rule, where } f \text{ does not appear} & \text{when argument is 'small'} \\ \text{recurrence case rule, where } f \text{ appears} & \text{when argument is not 'small'} \end{cases}$$

**Example:** computing power of a number

Recurrence function for  $f(x, n) = x^n$ , when  $n$  is pos integer?

Basic idea:  $x^n = x(x^{n-1})$ .

$$f(x, n) = \begin{cases} 1 & \text{when } n = 0 \\ x \cdot f(x, n - 1) & \text{when } n > 0 \end{cases}$$

□

**Q:** When can we use a recursive procedure?

**A:** When the problem we want to solve can be expressed in terms of a smaller version of itself.

**Example:** computing power of a number (contd)

$$\begin{array}{c} x^n = x \cdot (x^{n-1}) \\ \uparrow \qquad \qquad \uparrow \\ \text{problem} = x \cdot (\text{smaller version of problem}) \end{array}$$



Features of recursive procedure:

- base case(s) or trivial case(s): procedure does not call itself
- recursion case(s): procedure calls itself
- recursion level
- each call has own parameters, own local variables
- call stack (what calls are still incomplete)

**Example:** computing power of a number (contd)

---

```
1 POWR(x, n)
2 input n is positive integer, x is some number output: xn
3 /* We compute xn recursively. */
4 if n == 0 then
5     return 1
6 else
7     z = POWR(x, n - 1)
8     return x · z
9 end
```

---

Compute POWR(2,3).

step	recursion level	code line(s)	computation	stack
1	1	7	$z = \text{POWR}(2, 2)$	$\text{POWR}(2,3)$
2	2	7	$z = \text{POWR}(2, 1)$	$\text{POWR}(2,2)$ $\text{POWR}(2,3)$
3	3	7	$z = \text{POWR}(2, 0)$	$\text{POWR}(2,1)$ $\text{POWR}(2,2)$ $\text{POWR}(2,3)$
4	4	5	return 1	$\text{POWR}(2,0)$ $\text{POWR}(2,1)$ $\text{POWR}(2,2)$ $\text{POWR}(2,3)$
5	3	7,8	$z = 1$ return 2	$\text{POWR}(2,1)$ $\text{POWR}(2,2)$ $\text{POWR}(2,3)$
6	2	7,8	$z = 2$ return 4	$\text{POWR}(2,2)$ $\text{POWR}(2,3)$
7	1	7,8	$z = 4$ return 8	$\text{POWR}(2,3)$

□

## 2. Recursive versus iterative

Usually a recursive procedure can be converted to an equivalent iterative procedure.

Alternative to a recursive procedure: *iterative (looping)* procedure: does not call itself

Correspondences:

recursive procedure	iterative procedure
recursion level	iteration
call stack	own stack (accumulator variables)
base case condition	iteration termination condition
recursion case(s)	in-loop computations

**Example:** computing power of a number (contd)

```
1 POWR(x,n)
2 input n is positive integer, x is some number output: x^n
3 /* We compute x^n recursively. */
4 if n==0 then
5     return 1
6 else
7     z = POWR(x,n-1)
8     return x*z
9 end
```

```
1 POWI(x,n)
2 input int n, some number x output: y
3 /* We compute x^n iteratively for positive integer n. */
4 y = 1
5 while n > 0
6     y = x*y, n = n-1
7 end
8 return y
```



Why prefer recursive procedure over iterative procedure?

- often recursive is more compact
- no need to handle stack
- often easier to write/understand

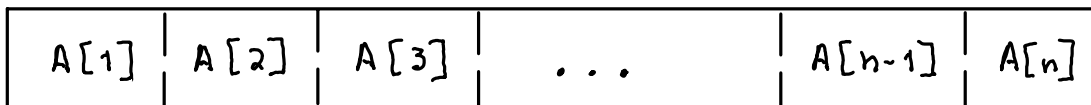
Why prefer iterative procedure over recursive procedure?

- often has better performance for some programming languages
- allows programmer better control over data structures

Ongoing debate: try searching 'why recursive is better than iterative'.

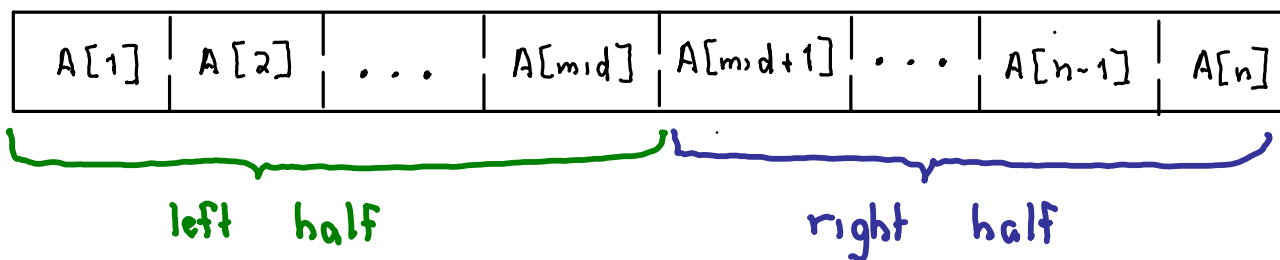
### 3. An application: binary search

Problem: find a value *key* in an sorted array  $A[1..n]$



Note :  $A[i] \leq A[i+1]$

Idea: split array in half



### Description

1. We start with array  $A[1..n]$  of numbers in sorted order from smallest to largest. We want to find location of  $key$  in  $A[1..n]$ , if it occurs.
2. The range of  $A$  we are looking in is  $A[L..R]$ . We compute an index roughly halfway between  $L$  and  $R$ : call it  $mid$ . If  $key \leq A[mid]$ , then we continue searching in  $A[L..mid]$ . If  $key > A[mid]$ , then we continue searching in  $A[(mid+1)..R]$ .
3. We keep halving the range we are searching in until either, we find the location of  $key$  or  $key$  does not occur in array.

---

```
1  BINSEARCH(A,L,R,key)
2  input A[1..n] is an array containing numbers; L and R are the
3  leftmost and rightmost indices that concern us; key is the value
4  we want to find
5  output: the index at which key occurs A or -1
6  /* The numbers in A must be in order from smallest to largest.
7  We search in array A[L..R] for key. If key is found, we return
8  its location, otherwise we return -1.*/
9  if L == R then
10     if A[L] == key then
11         return L
12     else
13         return -1
14     end
15 else
16     mid = [(L + R)/2]
17     if key ≤ A[mid] then
18         return BINSEARCH(A,L,mid,key)
19     else
20         return BINSEARCH(A,mid + 1,R,key)
21     end
22 end
```

---

### Example

Compute BINSEARCH( A,1, 7, 12) where

A =

-21	-20	-18	-6	5	12	13
-----	-----	-----	----	---	----	----

step	recursion level	code line(s)	computation	array A	$\uparrow = \text{mid}$	$\uparrow = L$	$\uparrow = R$							
—	—	—	$L = 1$ $R = 7$	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">-21</td> <td style="padding: 2px 10px;">-20</td> <td style="padding: 2px 10px;">-18</td> <td style="padding: 2px 10px;">-6</td> <td style="padding: 2px 10px;">5</td> <td style="padding: 2px 10px;">12</td> <td style="padding: 2px 10px;">13</td> </tr> </table>	-21	-20	-18	-6	5	12	13			
-21	-20	-18	-6	5	12	13								
1	1	16 20	$\text{mid} = \lfloor 8/2 \rfloor = 4$ return $\text{BINSEARCH}(A, 5, 7, 12)$	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">-21</td> <td style="padding: 2px 10px;">-20</td> <td style="padding: 2px 10px;">-18</td> <td style="padding: 2px 10px;">-6</td> <td style="padding: 2px 10px;">5</td> <td style="padding: 2px 10px;">12</td> <td style="padding: 2px 10px;">13</td> </tr> </table>	-21	-20	-18	-6	5	12	13			
-21	-20	-18	-6	5	12	13								
2	2	16 18	$\text{mid} = \lfloor 12/2 \rfloor = 6$ return $\text{BINSEARCH}(A, 5, 6, 12)$	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">-21</td> <td style="padding: 2px 10px;">-20</td> <td style="padding: 2px 10px;">-18</td> <td style="padding: 2px 10px;">-6</td> <td style="padding: 2px 10px;">5</td> <td style="padding: 2px 10px;">12</td> <td style="padding: 2px 10px;">13</td> </tr> </table>	-21	-20	-18	-6	5	12	13			
-21	-20	-18	-6	5	12	13								
3	3	16 20	$\text{mid} = \lfloor 11/2 \rfloor = 5$ return $\text{BINSEARCH}(A, 6, 6, 12)$	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">-21</td> <td style="padding: 2px 10px;">-20</td> <td style="padding: 2px 10px;">-18</td> <td style="padding: 2px 10px;">-6</td> <td style="padding: 2px 10px;">5</td> <td style="padding: 2px 10px;">12</td> <td style="padding: 2px 10px;">13</td> </tr> </table>	-21	-20	-18	-6	5	12	13			
-21	-20	-18	-6	5	12	13								
4	4	11	return 6	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">-21</td> <td style="padding: 2px 10px;">-20</td> <td style="padding: 2px 10px;">-18</td> <td style="padding: 2px 10px;">-6</td> <td style="padding: 2px 10px;">5</td> <td style="padding: 2px 10px;">12</td> <td style="padding: 2px 10px;">13</td> </tr> </table>	-21	-20	-18	-6	5	12	13			
-21	-20	-18	-6	5	12	13								

Tämä teos on lisensoitu Creative Commons Nimeä-EiKaupallinen-EiMuutoksia 4.0 Kansainvälinen -lisenssillä. Tarkastele lisenssiä osoitteessa <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

tekijä: Frank Cameron

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

made by Frank Cameron

