CSCE 110: Programming I

David Kebo Hougninou

Algorithms and recursion
What is an algorithm?

An algorithm is a finite sequence of steps that solves a problem. It can be described in English or in pseudocode. Pseudocode is an intermediate language between English and the implementation of the steps in code.

- It is independent of the programming language
- It is more general than a specific programming language

Algorithms

What are we interested in?

The computational complexity of our algorithm:

- How much computing resources are needed to solve a problem?
- How long (time) and how much memory (space) does it take?
- We observe the behavior of algorithms as the input size grows
Some properties of algorithms

1. **Input**: An algorithm has input values from a specified set.

2. **Output**: An algorithm produces output values from a specified set. The output values are the *solution*.

3. **Correctness**: An algorithm should produce the *correct* output values for each set of input values.

4. **Finiteness**: An algorithm should produce the output after a *finite* number of steps for any input.

---

**Maximum-Finding Algorithm**

Find the maximum in the list:

4, 7, 3, 10, 9, 12, 6, 8, 32, 5, 2, 1
Maximum-Finding Algorithm

Find the maximum of list: $a_1, a_2, \ldots, a_n$

1: max $\leftarrow a_1$
2: for $i \leftarrow 2 \ldots n$ do
3: if max $<$ $a_i$ then
4: max $\leftarrow a_i$
5: end if
6: end for
7: return max

Activity

Does the maximum-finding algorithm have the desired properties of an algorithm?
The Searching Problem

Find 5 in the list:

4, 7, 3, 10, 9, 12, 6, 8, 32, 5, 2, 1

The Searching Problem

• **Input:** list of elements \( a_1, a_2, \ldots, a_n \) and a particular element \( x \)

• **Output:** return the index in the list where \( x \) appears; if \( x \) is not in the list then return -1

Assume that all the list elements are unique.
Linear Search Algorithm

Input: $a_1, a_2, \ldots, a_n$ and $x$

1:  $i \leftarrow 1$
2:  while $i \leq n$ and $x \neq a_i$ do
3:      $i \leftarrow i + 1$
4:  end while
5:  if $i \leq n$ then
6:      return $i$
7:  else
8:      return $-1$
9:  end if

Linear Search Algorithm

• Linear search can be slow: if $x$ is not in the list or is toward the end, we have to check all (or most) of the elements in the list.

• Informally, we can see that the running time is proportional to the number of elements in the list.
The Sorting Problem

- **Input:** list of elements $a_1, a_2, \ldots, a_n$ drawn from totally ordered set.
- **Output:** list of elements $b_1, b_2, \ldots, b_n$ that is a rearrangement of the input list such that $b_1 < b_2 < \ldots < b_n$.
- Assume all the list elements are unique.

Bubble Sort Algorithm

Bubble sort makes **multiple passes** through a list.

**Every pair** of elements that are out of order are **interchanged**.
**Bubble Sort**

**e.g.** Steps of bubble sort with: \{3, 2, 4, 1, 5\}

- At the first pass the largest element is put into the correct position
- At the end of the 2\textsuperscript{nd} pass, put the 2\textsuperscript{nd} largest element in the correct position
- In each subsequent pass, an additional element is put in the correct position

**Bubble Sort Algorithm**

Input: array A[1 ..n] of elements

1: for i:=1 to n-1
2: for j:=1 to n-i
3: if a\textsubscript{j} > a\textsubscript{j+1} then interchange a\textsubscript{j} and a\textsubscript{j+1}
4: end for
5: end for

\{a\textsubscript{1}, \ldots, a\textsubscript{n} is now in increasing order\}
Recursion

Recursively Defined Functions

A recursive function is a function that calls itself until a base condition is true, and the execution stops.

Each subsequent instance of the recursion gets closer to a stopping case.
Fibonacci numbers

The Fibonacci numbers are the numbers in the following integer sequence.
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Recursion example

**Basis step:** \( \text{fib}(0) = 0, \text{fib}(1) = 1 \)

**Inductive step:** \( \text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \) for all \( n \geq 2 \)

\[
\begin{align*}
\text{fib}(2) &= \text{fib}(1) + \text{fib}(0) = 1 + 0 = 1 \\
\text{fib}(3) &= \text{fib}(2) + \text{fib}(1) = 1 + 1 = 2 \\
\text{fib}(4) &= \text{fib}(3) + \text{fib}(2) = 2 + 1 = 3 \\
\text{fib}(5) &= \text{fib}(4) + \text{fib}(3) = 3 + 2 = 5 \\
&\quad \ldots
\end{align*}
\]
Recursive Algorithm: Fibonacci

1: function fib(n):
2:   if n = 0 then // stopping case
3:     return 0
4:   else if n = 1 then // stopping case
5:     return 1
6:   else
7:     return fib(n-1) + fib(n-2) // closer to stopping case
8:   end if
9: end function

Recursively Defined Functions

Example:
Give a recursive definition of the factorial function n!

Solution
f(1) = 1
f(n) = n × f(n-1)
Recursive Algorithm: $n!$

1: function factorial(n):
2:   if $n = 1$ then // stopping case
3:     return 1
4:   else
5:     return $n \times \text{factorial}(n-1)$ // closer to stopping case
6:   end if
7: end function

Recursively Defined Functions

Example:
Give a recursive definition of the function $a^n$

Solution
$a^1 = a$
$a^n = a \cdot a^{n-1}$
Recursive Algorithm: $a^n$

1: `function power(a, n):
2:     if n = 1 then                  // stopping case
3:         return a
4:     else
5:         return a * power(a, n-1)  // closer to stopping case
6:     end if
7:     end function

End