CS 281
Prof. Alex Pothen
Webbook:
www.cs.odu.edu/~webbook
www.cs.odu.edu/~pothen/Courses/CS281
CLASS NOW IN CONSTANT 120
Wed Oct 28, 1998
Lecture 17
Sec 3.3 Recursive Definitions
Sec 3.4 Recursive Algorithms
Next Lecture: 3.3 Recursive Definitions
Recursive definitions

• A recursive definition involves
  (1) one or more basis rules, which define some simple objects, and
  (2) one or more inductive rules, whereby larger objects are defined in terms of smaller objects in the collection.

• Fibonacci numbers

  Basis: \( f_0 = 0, f_1 = 1, \)
  Induction: \( f_i = f_{i-1} + f_{i-2}, \) for \( i \geq 2. \)

Hence \( f_2 = 1, f_3 = 2, f_4 = 3, f_5 = 5, \)
\( f_6 = 8, f_7 = 13, f_8 = 21, f_9 = 34, \)
\( f_{10} = 55, \ldots. \)

\[ f_n > \left( \frac{1 + \sqrt{5}}{2} \right)^n \approx 1.6^n. \]
Fibonacci numbers
Recursive Algorithm

procedure \( \text{fibonacci}(n) \)
  if \( n == 0 \) then
    \( \text{fib} := 0; \)
  else if \( n == 1 \) then
    \( \text{fib} := 1; \)
  else
    \( \text{fib} := \text{fibonacci}(n - 1) + \text{fibonacci}(n - 2); \)
  end if
return \( \text{fib}; \)
Fibonacci numbers
Complexity: Recursive Algorithm

• Let $a_n$ denote the number of additions to compute $f_n$.

• We prove by induction that $a_n = f_{n+1} - 1$, for $n \geq 2$.

• Base case: for $n = 2$, $f_2 := f_1 + f_0$. Hence $a_2 = 1$.

• Ind. hyp: Assume $a_k = f_{k+1} - 1$, for $k \geq 2$. 
Complexity: Recursive Algorithm II

• Ind. step: Prove that $a_{k+1} = f_{k+2} - 1$.

  $f_{k+1} := f_k + f_{k-1}$. By the ind. hyp., $a_k = f_{k+1} - 1$, and $a_{k-1} = f_k - 1$. Hence

  \[
  a_{k+1} = (f_{k+1} - 1) + (f_k - 1) + 1 = f_{k+1} + f_k - 1 = f_{k+2} - 1.
  \]
Binary Search

• Given a sorted list of $n$ integers $A(1 : n)$, and an integer $item$, find if $item$ is in the list.
  Return the index of (one occurrence of) $item$ if it is in the list; otherwise return ‘not found’.

• Assume that the list is in increasing order:
  $$a_1 \leq a_2 \leq \ldots \leq a_n.$$
**Binary Search**

**Iterative Algorithm**

BinSearch := proc(A::list(integer),
item::integer)
left := 1; right := n; found := false;
{search A[left : right] in each iteration}
while (left ≤ right and not found) do
    mid := floor((left + right)/2);
    if (A[mid] = item) then {success!}
        found := true;
    else if (A[mid] < item) then
        {search right sublist}
        left := mid + 1;
    else {A[mid] > item, search left sublist}
        right := mid - 1; fi ;
    od ;
Return ([found, mid]);
Binary Search
Recursive Algorithm

procedure BinSearch(A, item, left, right)
    mid := [(left + right)]/2;
    if A[mid] == item then
        loc := mid;
    else if item < A[mid] and left < mid then
        {search left subarray}
        loc := BinSearch(A, item, left, (mid - 1));
    else if item > A[mid] and right > mid then
        {search right subarray}
        loc := BinSearch(A, item, (mid + 1), right);
    else
        loc := 0;
    end if
return loc;
Complexity of Binary Search
Iterative Algorithm

- Let $k$ be the smallest value such that $n \leq 2^k$.

- At the end of each step, the number of elements in the subarray to be searched is reduced by a factor of 2.

- At the end of the first step, we need search $n/2 \leq 2^k/2 = 2^{k-1}$ elements. At the end of the second step, only $n/4 \leq 2^{k-2}$ elements remain. At the end of the $k$th step, only $n/2^k \leq 2^k/2^k = 1$ element remains.

- Hence the algorithm takes $k = \lceil \log_2 n \rceil$ steps.

- Each step takes only a (small) constant number of operations. Hence the algorithm has time complexity $O(\log_2 n)$. 