Classifying algorithms

Yuhua Li
School of Computer Science & Informatics

(Content adapted from slides by Dr Louise Knight)

Different types of algorithms

There are many different ways to classify algorithms:

- Control structures used in the program – iterative, recursive, etc.
- Type of problem they solve – e.g. sorting, searching, packing, scheduling
- Approach to solving a problem – greedy, divide-and-conquer, backtracking, branch-and-bound, dynamic programming, etc.
- Whether they solve a problem exactly or approximately
- How fast/slow they run

Different control structures

- Selection sort and insertion sort are examples of iterative algorithms
- Merge sort is an example of a recursive algorithm

Approach to solving a problem

- Greedy
- Divide-and-conquer
- Backtracking
- Branch-and-bound
- Dynamic programming

Does the algorithm solve the problem exactly?

- Exact methods produce the best solution to a problem, where a solution exists
- Approximate methods concentrate on getting a reasonable solution in a reasonable amount of time when exact methods are not feasible. Approximate methods come in two classes:
  - Heuristic methods – simple rules of thumb are used
  - Metaheuristic methods – includes simulated annealing, tabu search and genetic algorithms

Control structures: recursive algorithms

- Recursive algorithms involve functions/procedures/methods that call themselves
- E.g. sum of squares

$f(n) = 1^2 + 2^2 + 3^2 + \ldots + n^2$

- Another way of writing this is as $\sum_{i=1}^{n} i^2$
- Iterative algorithm or recursive?

$f(n) = n^2 + f(n - 1)$

Sum of squares pseudocode

SumSquares(n)

Input: An integer, n.

Output: The sum of squares $1..n$.

if $n = 1$ **then return** (1)

else return $(n * n + \text{SumSquares}(n - 1))$

Factorial function

- Factorial n, or $n!$

$f(n) = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

Algorithm Factorial(n)
    Input: An integer, n.
    Output: $n!$.
    if $n = 1$ **then return** (1)
        else return ($n*$ Factorial($n-1$))

Fibonacci numbers

$f(n) = f(n-1) + f(n-2)$

for $n \geq 2$

$f(0) = 0$

$f(1) = 1$

Write down the first 12 in the sequence.

Recursive Fibonacci

Algorithm Fibonacci(n)

Input: An integer, n.

Output: n^{th} fibonacci number.

if $n = 0$ **then return** (0)

if $n = 1$ **then return** (1)

else return (Fibonacci($n - 1$)

$+$ Fibonacci($n - 2$))

Recursive vs iterative

- Use of recursion results in a very compact algorithm
- A recursive algorithm can also be written in an iterative form
- The iterative version is easier to analyse

Iterative Fibonacci

Algorithm Fibonacci(n)

Input: An integer, n.

Output: n^{th} fibonacci number.

if $n = 0$ **then return** (0)

if $n = 1$ **then return** (1)

$f0 \leftarrow 0$; $f1 \leftarrow 1$

for $i \leftarrow 2$ **to** n **do**

$f \leftarrow f1 + f0$

$f0 \leftarrow f1$

$f1 \leftarrow f$

return (f)

Analysis of iterative Fibonacci

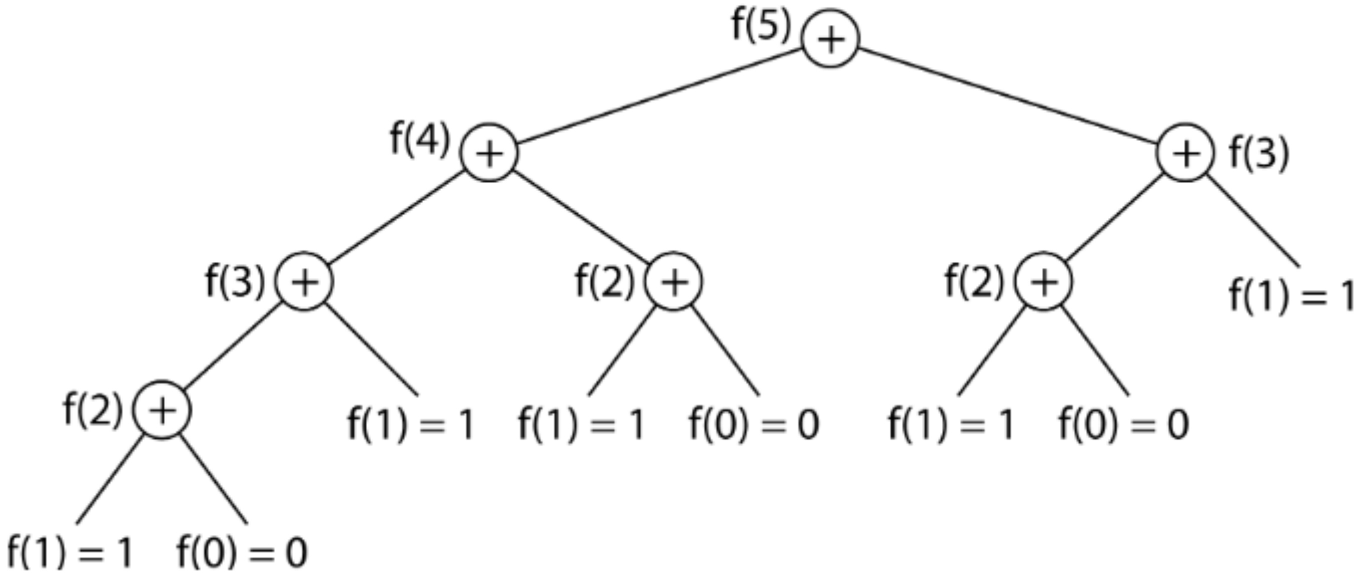
- 2 conditional statements
- 2 assignment statements
- 1 addition per iteration
- 1 branch per iteration
- 2 assignment statements per iteration
- Loop counter – increment, test
- Enter/exit function

Algorithm Fibonacci(n)
  Input: An integer, n.
  Output: n^{th} fibonacci number.
  if $n = 0$ **then return** (0)
  if $n = 1$ **then return** (1)
  $f0 \leftarrow 0$; $f1 \leftarrow 1$
  for $i \leftarrow 2$ **to** n **do**
    $f \leftarrow f1 + f0$
    $f0 \leftarrow f1$
    $f1 \leftarrow f$
  return (f)

$6 + 4(n - 1) + n = 5n + 2 = O(n)$

Analysis of recursive Fibonacci

- $n = 5$
- View the algorithm bottom-up using the recursion tree



Analysis of recursive Fibonacci

- If b_n is number of additions in recursion tree for finding $f(n)$, then how might we represent the number of additions as a formula?

Implementation tradeoff

- For divide-and-conquer
- Recursion is elegant solution – small code
- Not efficient for repeated computations, excessive use of memory
- Importance of “sorting in place”