Introduction and Getting Started

Introduction

- Algorithm is a tool for solving a well-specified computational problem
- Algorithms as a technology
- Basic questions about an algorithm
  1. Does it halt?
  2. Is it correct?
  3. Is it fast?
  4. How much memory does it use?
  5. How does data communicate?

Getting started: case study 1

- Problem: computing the nth Fibonacci number F_n
- Definition:

$$
\begin{aligned}
&F_0 = 0, \\
&F_1 = 1, \\
&F_n = F_{n-1} + F_{n-2} \quad \text{for} \quad n \geq 2
\end{aligned}
$$

- Algorithms:
  1. Recursive ("top-down")
  2. Memorize the recursive – iterative ("bottom-up")
  3. Divide-and-conquer
  4. Approximate

Getting started: case study 2

- Problem: sorting
- Definition:

  Input: a sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
  Output: a permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the a-sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

- Algorithms:
  1. Insert sort
  2. Merge sort

Getting started: case study 2

Insert sort algorithm

- Idea: incremental approach
- Pseudocode (*expressing algorithm*)

```
InsertionSort(A)
n = length(A)
for j = 2 to n
   key = A[j]
   // insert ``key'' into sorted array A[1...j-1]
   i = j-1
   while i > 0 and A[i] > key do
      A[i+1] = A[i]
      i = i-1
   end while
   A[i+1] = key
end for
return A
```

Getting started: case study 2

Remarks:

- Correctness: argued by “loop-invariant” (a kind of induction)
- Complexity analysis:
  best-case, worst-case, average-case
- Insert sort is a “sort-in-place”, no extra memory necessary
- Importance of writing a good pseudocode;
  “*expressing algorithm to human*”
- There is a recurisve version of insert sort, see Homework 1.

Getting started: case study 2

Merge sort algorithm

- Idea: divide-and-conquer approach
- Pseudocode

```
MergeSort(A,p,r)               // Merge-sort of array A[p..r]
if p < r then                  // check for base case
   q = flooring( (p+r)/2 )     // divide
   MergeSort(A,p,q)            // conquer
   MergeSort(A,q+1,r)          // conquer
   Merge(A,p,q,r)              // combine
end if
```

Getting started: case study 2

- Pseudocode, cont'd

```
Merge(A,p,q,r)
n1 = q-p+1;  n2 = r-q
for i = 1 to n1         // create arrays L[1...n1+1] and R[1...n2+1]
    L[i] = A[p+i-1]
end for
for j = 1 to n2
    R[j] = A[q+j]
end for
L[n1+1] = infty; R[n2+1] = infty // mark the end of arrays L and R
i = 1; j = 1
for k = p to r          // Merge arrays L and R to A
    if L[i] <= R[j] then
       A[k] = L[i]
       i = i+1
    else
       A[k] = R[j]
       j = j+1
    end if
end for
```

Getting started: case study 2

- Merge sort is a divide-and-conquer algorithm consisting of three steps: divide, conquer and combine
- To sort the entire sequence A[1...n], we make the initial call

  MergeSort(A,1,n)

  where n = length(A).
- Complexity analysis:

$$T(n) = 2T\left(\frac{n}{2}\right) + n - 1 = O(n \lg(n))$$