Exercises Week 8
Introduction to Algorithms

Hans Georg Schaathun

12th November 2015

Period 7–13 October 2015

Reading Stein *et al* cover this material in Chapter 4.

Programme This document details the programme for the week, including exercises and pointers to other material. It is available in two versions:

1. as a PDF document.
2. as a web site. This depends on MathML and may require firefox/iceweasel to display correctly.

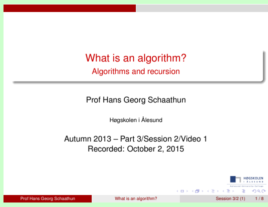
The web version includes inline video. The pdf version shows a still image from the video, providing a hyperlink directly to the video on the web site.

1 Wednesday 7 October 2015

Theory

A well-designed algorithm needs the following properties:

1. **Input** must be well-defined
2. **Output** must be well-defined
3. **Definiteness** i.e. the steps must be precisely defined.
4. **Correctness** i.e. the algorithm must produce the correct (intended) output
5. **Finiteness** the result should be reached within a finite number of steps
6. **Effectiveness** i.e. it must be possible to perform each step exactly and in a finite amount of time.
7. **Generality** i.e. the procedure must be applicable to any problem of the desired form (not just particular input values).



OGG MP4 Slides

Related reading: Rosen p. 193–196

Exercise 1.1 (Rosen p. 204, ex. 2) *Recall the characteristics Input, Output, Definiteness, Correctness, Finiteness, Effectiveness, and Generality, as defined in the video, or in Rosen's book p. 195.*

Consider the following algorithms, and determine for each one, which characteristics they posess and which they lack.

a)
```
procedure double(n : positive integer)
while n > 0
   n := 2n
```

b)
```
procedure divide(n : positive integer)
while n > 0
   m := 1/n
   n := n − 1
```

c)
```
procedure sum(n : positive integer)
sum := 0
while i < 10
   sum := sum + i
```

d)
```
procedure choose(a,b : positive integer)
x := either a or b
```

1. **procedure** `euclid`(a, b)
2. $r := a \mod b$
3. $a := b$
4. $b := r$
5. **If** $r = 0$, **then return** a
6. **else Goto** Line 2

Table 1: Euclid's Algorithm

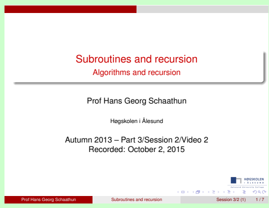
Exercise 1.2 *Review Euclid's (Table 1) algorithm and determine if it possesses each of the algorithmic properties above. Give reasons for your answers.*

THEORY

The key to problem solving is to split the problem into small pieces (subproblems) which are easier to solve.

We design algorithms to solve problems, and we design subroutines to solve subproblems.

Recursion is a very specific application of subroutines, where an algorithm uses itself as a subroutine. This is a critical concept in computer science.



OGG MP4 Slides

THEORY

The Tower of Hanoi is a classic puzzle, with a simple recursive algorithmic solution. You can try it out yourself in an interactive, online game at `http://haubergs.com/hanoi`.

Related reading: Stein p. 213+ or Rosen p. 264+

OGG

MP4

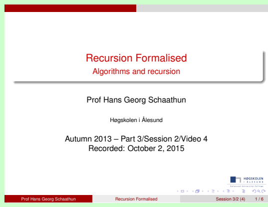
Slides

Exercise 1.3 *How many steps are required to move a Tower of Hanoi of ...*

1. *1 disk?*
2. *2 disks?*
3. *3 disks?*
4. *5 disks?*
5. *n disks? (this is difficult; we return to it later)*

THEORY

Having seen an example, we shall formalise the concept of recursion.



OGG

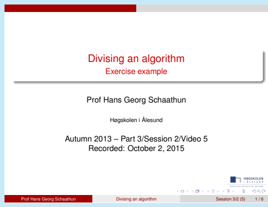
MP4

Slides

Related reading: Rosen p. 353–356

EXERCISE EXAMPLE

Problem 1.1 *Devise an algorithm which takes a sorted array as input, and outputs an array of all repeated elements in the input.*



OGG

MP4

Slides

Exercise 1.4 (Rosen p. 204, ex. 3) *Devise a recursive algorithm which finds the sum of all the integers in a list.*

Exercise 1.5 *Devise an algorithm which finds the common elements in two sorted arrays. The output should be an array.*

THEORY

```
procedure selectionsort( Array A of length n )
for i := 1 to n−1
    for j := i+1 to n
        if A[i] > A[j]
            swap A[i] with A[j]
```

Slides

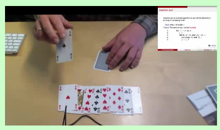
Related reading: Rosen p. 200

Exercise 1.6 *Consider the list* $[6, 2, 5, 4, 7, 1]$.

1. *Demonstrate how you sort the numbers step by step using selection sort.*
2. *Discuss: Is the algorithm recursive as you perform it?*

THEORY

```
procedure insertionsort( Array A_1, A_2, ..., A_n )
for i := 2 to n
    j := i
    while (j ≥ 2) and (A_j < A_{j−1})
        swap A_j and A_{j−1}
        j := j−1
```



Slides

Exercise 1.7 *(This exercise is similar to Exercise 1.6.) Consider the list* $[6, 2, 5, 4, 7, 1]$.

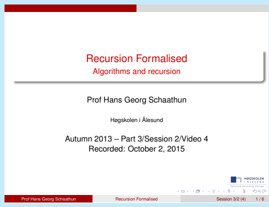
1. *Demonstrate how you sort the numbers step by step using insertion sort.*
2. *Discuss: Is the algorithm recursive as you perform it?*

2 Thursday 8 October 2015

2.1 More recursion

EXERCISE EXAMPLE

Exercise 2.1 *Rewrite the insertion sort algorithm in recursive form. (See the previous section for the iterative version.)*



OGG MP4 Slides

We observe that during the iteration $i = 2$ only A_i is moved (and only to the left). Hence elements to the left must already be sorted, and elements to the right are irrelevant. This is the core idea of a recursive approach.

We can break the algorithm into two steps. First we use $n-1$ iterations to sort $n-1$ elements. Then the nth iteration inserts the nth element into the subarray of $n-1$ elements, maintaining sort order. This is the basis for the following recursive algorithm.

1. **procedure** InsertionSort$(A_1, A_2, \ldots, A_n)$
2. if $n = 0$, then return []
3. else
4. InsertionSort$(A_1, A_2, \ldots, A_{n-1})$
5. insert$(A_n$ into $A_1, A_2, \ldots, A_{n-1})$

The insert algorithm is also defined recursively, as follows.

1. **procedure** insert$(e$ into $A_1, A_2, \ldots, A_n)$
2. if $n = 0$, then
3. $A_1 = e$
4. else if $e > A_n$, then
5. $A_{n+1} = e$
6. else
7. $A_{n+1} = A_n$
8. insert$(e$ into $A_1, A_2, \ldots, A_{n-1})$

Related reading: Rosen p. 358-359

Problem 2.1 *Consider the selection sort algorithm as described in the videos:*

1. *For* $i = 1, 2, \ldots, n-1$,
2. *for* $j = j+1, j+2, \ldots, n$,

3. *if $A_i > A_j$, then swap A_i with A_j*

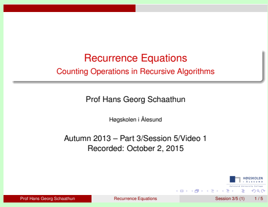
Rewrite the selection sort algorithm in recursive form.

Hint: Consider the array at the start of iteration i In the outer loop. Can you identify a subarray which has to be sorted?

2.2 Recurrences

THEORY

Problem 2.2 *How many moves (MoveDisc calls) are need to move an n-disk Tower of Hanoi?*



OGG

MP4

Slides

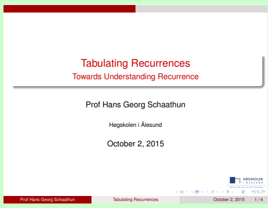
Related reading: Stein *et al.* p. 213-214 or Rosen p. 157+

1. procedure HanoiMove(n) from s to t using a
2. if $n = 1$, then MoveDisc from s to t
3. else
4. HanoiMove($n - 1$) from s to a using t
5. MoveDisc from s to t
6. HanoiMove($n - 1$) from a to t using s

Exercise 2.2 *Give recurrence equations to give the number of comparisons required to sort an n-element array using insertion sort.*

THEORY

A good start to understand a recurrence is to calculate a few values by hand and put them in a table.



OGG

MP4

Slides

Related reading: Stein *et al.* p. 213-214 or Rosen p. 157+

Exercise 2.3 *Tabulate the following recurrence for $n = 0, \ldots, 7$:*

$$T(n) = 0.5T(n-1) + 2, \tag{1}$$
$$T(0) = 0 \tag{2}$$

Can you spot a pattern? Try to guess a closed form expression.

1. **Algorithm** SquareNmultiply(x, e, n)
2. if $e = 1$, return x
3. $y :=$ SquareNmultiply$(x, \lfloor e/2 \rfloor, n)$
4. $y' := y^2 \mod n$
5. if $e \mod 2 = 1$,
6. $\qquad y'' := y' \cdot x \mod n$
7. else
8. $\qquad y'' := y'$
9. return y''

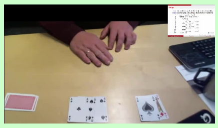
Table 2: Square-and-Multiply algorithm.

2.3 Split and Conquer

Exercise 2.4 *How many multiplications are needed to calculate* $a^x \mod n$ *in the worst case, using square-and-multiply? Give a recurrence equation in terms of* x*.*

THEORY

Merge sort sorts recursively, by splitting the array in two halves, sorting each half separately, and then merging the two halves together while preserving the sort order.



Slides

1. if $n == 1$, return A
2. $B =$ MergeSort$A_{1,2,\ldots,\lfloor n/2 \rfloor}$
3. $C =$ MergeSort$A_{\lfloor n/2 \rfloor+1, \lfloor n/2 \rfloor+2,\ldots,n}$
4. return Merge B, C

Related reading: Stein *et al.* p. 230 or Rosen p. 359–364

Exercise 2.5 *Give recurrence equations to give the number of comparisons required to sort an* n*-element array using merge sort.*

Exercise 2.6 *Consider a sorted array* A *as input. Devise an algorithm which finds a given element* k *in* A*.*

1. *Give an iterative formulation of your algorithm,*
2. *Give a recursive formulation of your algorithm,*

3. *How many comparisons does your algorithm require to find k? Would it be possible to make it faster?*

3 Compulsory Exercises (Tuesday 13 October 2015)

Exercise 3.1 *Consider the square-and-multiply algorithm as given in Table 2. What can you say about this algorithm with respect to Input, Output, Definiteness, Correctness, Finiteness, Effectiveness, and Generality.*

Exercise 3.2 *Consider the square-and-multiply algorithm which calculates $x^a \mod n$. Write pseudo code for an iterative version of this algorithm.*

Exercise 3.3 *Consider the following linear search algorithm:*

1. *procedure find(k,$A_1, A_2, \ldots, A_n$)*
2. *for $i := 1, 2, \ldots, n$*
3. *if $k = A_i$, return i*

It finds the index of an element k in an array $A_1, A_2, \ldots, A_n$. Rewrite the algorithm using recursion instead of a loop.

Exercise 3.4 *Give pseudo-code for a recursive version of Euclid's algorithm, and verify that the properties Input, Output, Definiteness, Correctness, Finiteness, Effectiveness, and Generality.*

Exercise 3.5 *How many comparisons are needed, in the worst case, to sort an array using*

1. *... insertion sort.*
2. *... selection sort.*
3. *... merge sort.*

Compare the three. Is any one faster than the others?

Exercise 3.6 *Tabulate the following recurrence for $n = 0, \ldots, 7$:*

$$T(n) = 2T(n-1) + 1, \tag{3}$$

$$T(0) = 0 \tag{4}$$

Can you spot a pattern? Try to guess a closed form expression.