Exercises Week 9
Mathematical Induction

Hans Georg Schaathun

12th November 2015

Period 14–20 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 Mathematical Induction (Wednesday 14 October 2015)

THEORY

The natural numbers have a peculiar property. Any set of natural numbers, whether finite or infinite, has one element which is smaller than any of the others.

Related reading: Stein *et al.* p. 191-195 or Rosen p. 307+

The smallest
Mathematical induction

Prof Hans Georg Schaathun

Høgskolen i Ålesund

October 9, 2015

Prof Hans Georg Schaathun | The smallest | October 9, 2015 1 / 7

OGG

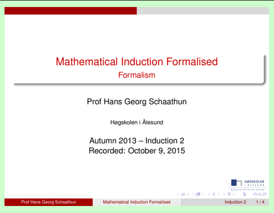
MP4

Slides

THEORY

The fact that any set of natural numbers has a smallest element makes it possible to use *mathematical induction* to prove predicates $p(x)$ for all $x \in \mathbb{N}$.

Related reading: Stein *et al.* p. 195-198 or Rosen p. 307+



Mathematical Induction Formalised
Formalism

Prof Hans Georg Schaathun

Høgskolen i Ålesund

Autumn 2013 – Induction 2
Recorded: October 9, 2015

Prof Hans Georg Schaathun | Mathematical Induction Formalised | Induction 2 1 / 4

OGG

MP4

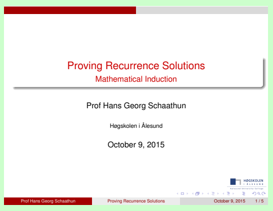
Slides

THEORY

Problem 1.1 *Use mathematical induction to prove that the following recurrence:*

$$T(n) = \begin{cases} 1, & \textit{when } n = 1, \\ 2T(n-1)+1, & \textit{otherwise}, \end{cases}$$

is equivalent to $T(n) = 2^n - 1$.


Proving Recurrence Solutions

Mathematical Induction

Prof Hans Georg Schaathun

Høgskolen i Ålesund

October 9, 2015

Prof Hans Georg Schaathun | Proving Recurrence Solutions | October 9, 2015 1 / 5

OGG MP4 Slides

Related reading: Stein *et al.* p. 191-195 or Rosen p. 307+

Exercise 1.1 *Consider the following recurrence.*

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

Prove that $T(n) = 4 - 2^{2-n}$

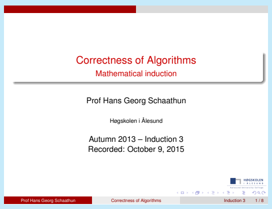
Exercise 1.2 *Describe the main steps of a proof by induction.*

Exercise 1.3 *Consider a ladder. If you stand on the ground, it is possible to step onto the first rung. If you stand at the* n*th rung, it is possible to climb onto the* $(n+1)$*st rung.*

Prove that you can climb to the top of the ladder, starting on the ground, using a proof by contradiction.

Exercise Example

Problem 1.2 *Prove that the output array of insertion sort is sorted in increasing order.*


Correctness of Algorithms
Mathematical induction

Prof Hans Georg Schaathun

Høgskolen i Ålesund

Autumn 2013 – Induction 3
Recorded: October 9, 2015

Prof Hans Georg Schaathun Correctness of Algorithms Induction 3 1 / 8

OGG MP4 Slides

```
procedure insertionsort( Array $A_1, A_2, \ldots, A_n$ )
for i := 2 to n
    j := i
    while $(j \geq 2)$ and $(A_j < A_{j-1})$
        swap $A_j$ and $A_{j-1}$
        j := j-1
```

Solution: To conduct a proof by induction, we need some predicate describing partial success of the algorith, Loop indices provide a reasonable link. 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. Thus we can propose the following invariant $P(i)$:

$$P(n) := \text{the subarray } A_1, A_2, \ldots, A_n \text{ is sorted at the start of iteration } n+1$$

If $P(n)$ is true, the entire array is sorted after the last iteration, and the algorithm is correct. Therefore we prove $P(n)$ by induction.

Firstly, we note that for $i = 2$, there is a single element to the left, which makes a trivially sorted subarray. Hence $P(1)$ is true.

To prove $P(n) \Rightarrow P(n+1)$, i.e. that the subarray remains sorted after the iteration, we need to consider the old subarray $A_1, \ldots, A_n$, and the new one, $A'_1, \ldots, A'_{n+1}$. It is sufficient to prove that $A'_{j-1} \leq A'_j$ for any j.

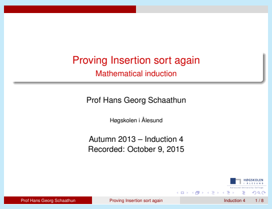
Considering an arbitrary j, there are three possibilities:

1. A'_{j-1} and A'_j correspond to adjacent elements in the original array. Then $A'_{j-1} \leq A'_j$ as required, because the original array was sorted.
2. $A'_j = A_{n+1}$ is the new element. In this case $A'_j \geq A'_{j-1}$, lest the while-condition be true, and the two elements swapped.
3. $A'_{j-1} = A_{n+1}$ is the new element. In this case $A'_j < A'_{j-1}$, since the two elements have been swapped by the while loop.

In all three cases, the sort order is satisfied, and we conclude that the loop maintains sort order. InsertionSort is correct by mathematical induction.

EXERCISE EXAMPLE

Problem 1.3 *Prove that the output array of insertion sort is sorted in increasing order. Base your proof on the recursive version below.*


Proving Insertion sort again
Mathematical induction

Prof Hans Georg Schaathun

Høgskolen i Ålesund

Autumn 2013 – Induction 4
Recorded: October 9, 2015

Prof Hans Georg Schaathun | Proving Insertion sort again | Induction 4 1 / 8

OGG MP4 Slides

Related reading: Stein *et al.* p. 203-206 or Rosen p. 357-358

Solution: We base the proof on the recursive formulation of the algorithm.

To conduct a proof by induction, we need some predicate describing partial success of the algorith, The a variable should be in the set of natural numbers. In the case of recursion, we can typically link the predicate to the size of the input, as follows:

$$P(n) := \text{Insertion sort can correctly sort } n \text{ numbers}$$

If we can prove $P(n)$ for all n, then the algorithm is correct for any (valid) input.

We note that for $n = 0$ we have an empty array, which is trivially sorted. Hence $P(0)$ is trivially true. It remains to prove that $P(n-1) \Rightarrow P(n)$.

In the recursive case, the algorithm performs two operations. First, an InsertionSort on $n-1$ elements, which is correct by the inductive hypothesis $P(n-1)$; and secondly an insert into a sorted array of $n-1$ elements. If we can prove that the insert algorithm is correct, then correctness of InsertionSort will follow.

The proof of the insert algorithm is simular

$$Q(n) := \text{The insert algorithm correctly inserts an element into an array of } n \text{ numbers}$$

Again, we note that $Q(0)$ is true, because $n = 0$ gives a singleton array which is trivially sorted. To prove $Q(n-1) \Rightarrow Q(n)$, we note that there are two cases. If e is larger than A_n, it is larger than every element in A because it is sorted. Thus e is correctly inserted as a last element. If e is not larger than A_n, A_n has to be the last element, at position $n+1$, and e is inserted into the subarray of $n-1$ elements. By the inductive hypothesis, $P(n-1)$, e is correctly inserted, and $P(n)$ follows for any non-negative n by mathematical induction.

Since insert is correct, InsertionSort is correct, as argued above.

Exercise 1.4 *Prove that the Tower of Hanoi algorithm is correct.*

Exercise 1.5 *Prove that the Square-and-Multiply algorithm is correct as given in Table 1.*

1. *Identify a base case, and argue that the algorithm returns the correct answer in the*

```
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.         $y'' := y' \cdot x \mod n$
7. else
8.         $y'' := y'$
9. return $y''$
```

Table 1: Square-and-Multiply algorithm.

base case.

2. *Consider Line 3 Write y as an expression in terms of x.*
3. *Consider Lines 4–9. Write y'' as a mathematical expression in terms of y.*
4. *Using the answers from the two previous questions, write y'' as an expression in terms of x.*
5. *Argue that y'' as given in your previous answer is equal to $x^e \mod n$. Which arithmetic rules are you using?*
6. *Use your previous answers to argue that the inductive case is correct.*
7. *Give the structure of an inductive proof and finalise the argument that the algorithm in the table be correct.*

2 Thursday

THEORY

One way to solve a recurrence is to iterate it.

Related reading: Stein *et al.* p. 217-218 or Rosen p. 162-167

Iterating a recurrences
Recurrence

Prof Hans Georg Schaathun

Høgskolen i Ålesund

Autumn 2013 – Part 3/Session 5/Video 3
Recorded: October 9, 2015

Prof Hans Georg Schaathun | Iterating a recurrences | Session 3/5 (3) 1 / 5

OGG

MP4

Slides

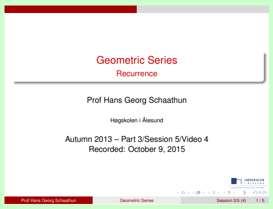
THEORY

Theorem 1 *If a recurrence $T(n)$ is given as,*

$$T(n) = \begin{cases} b, & \text{when } n = 0, \\ r \cdot T(n-1) + a, & \text{otherwise.} \end{cases}$$

then we can write

$$T(n) = r^n \cdot b + a\frac{1-r^n}{1-r}$$



Geometric Series
Recurrence

Prof Hans Georg Schaathun

Høgskolen i Ålesund

Autumn 2013 – Part 3/Session 5/Video 4
Recorded: October 9, 2015

Prof Hans Georg Schaathun | Geometric Series | Session 3/5 (4) | 1 / 5

OGG

MP4

Slides

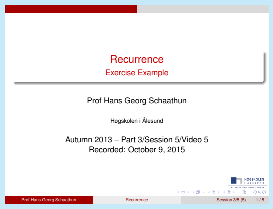
Related reading: Stein *et al.* p. 217-218 or Rosen p. 162-167

EXERCISE EXAMPLE

Problem 2.1 *Iterate the recurrence to solve the following equation*

$$T(n) = 1.5T(n-1) + 1 \quad T(0) = 0.$$

Use mathematical induction to prove that the solution is correct.



Recurrence
Exercise Example

Prof Hans Georg Schaathun

Høgskolen i Ålesund

Autumn 2013 – Part 3/Session 5/Video 5
Recorded: October 9, 2015

Prof Hans Georg Schaathun | Recurrence | Session 3/5 (5) | 1 / 5

OGG

MP4

Slides

Recall the recurrence for the Tower of Hanoi:

$$M(n) = 2M(n-1) + 1 \quad M(1) = 1$$

Exercise 2.1 *Solve the following recurrences, and explain how they differ from the recurrence of the Tower of Hanoi (above):*

$$M(n) = 2M(n-1) + 1 \quad M(0) = 0, \tag{3}$$
$$M(n) = 3M(n-1) + 1 \quad M(1) = 1, \tag{4}$$
$$M(n) = M(n-1) + 2 \quad M(1) = 1, \tag{5}$$
$$T(n) = 0.5T(n-1) + 2, \quad T(0) = 0. \tag{6}$$

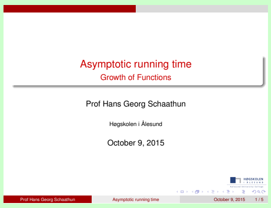
Exercise 2.2 *Prove the following formula for geometric series using mathematical induction:*

$$\sum_{i=0}^{n-1} r^i = \frac{1-r^n}{1-r}$$

Theory

When we evaluate algorithms, we are not so interested in the precise number of operations, but more in the behaviour of this number when the problem size grows.

Related reading: Rosen p. p. 206-210


Asymptotic running time
Growth of Functions

Prof Hans Georg Schaathun

Høgskolen i Ålesund

October 9, 2015

OGG

MP4

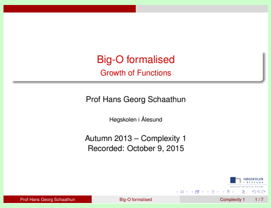
Slides

Theory

Definition 1 *Let $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{N} \to \mathbb{R}$. We say that $f(n)$ is $O(g(n))$ if there are constants c and k such that*

$$\forall n \geq k, |f(n)| \leq c \cdot |g(n)|$$

Related reading: Rosen p. p. 206-210


Big-O formalised
Growth of Functions

Prof Hans Georg Schaathun

Høgskolen i Ålesund

Autumn 2013 – Complexity 1
Recorded: October 9, 2015

OGG

MP4

Slides

Exercise 2.3 *Consider the function $f(n) = n^2 + 2n + 14$. Show that $f(n) = O(n^2)$. You have to review the definition of Big-O and find suitable constants c and k.*

Exercise 2.4 *Review the number of swaps needed for insertion sort in the worst case. What is the Big-O expression of this number? This is known as the (asymptotic) complexity of the algorithm.*

Exercise 2.5 *Consider each of the recurrences in Exercise 2.1 and give Big-O expressions for each of them.*

3 Compulsory Exercises (Tuesday 20 October 2015)

Exercise 3.1 *Write down Euclid's algorithm in recursive form and use mathematical induction to prove that it correctly returns the highest common factor.*

Exercise 3.2 *Use mathematical induction to prove that*

$$T(n) = 2T(n-1) + 1 \quad T(0) = 0$$

is equivalent to

$$T(n) = 2^n - 1.$$

Exercise 3.3 *Use mathematical induction to prove that*

$$T(n) = T(n-1) + n \quad T(0) = 0$$

is equivalent to

$$T(n) = \frac{(n+1)n}{2}.$$

Exercise 3.4 *Solve the following recurrences:*

$$T(n) = 4 \cdot T(n-1) + 1, \quad T(0) = 0 \tag{7}$$
$$T(n) = 2 \cdot T(n-1) + n, \quad T(0) = 0. \tag{8}$$

Exercise 3.5 *Solve the following recurrences:*

$$T(n) = r \cdot T(n-1) + 1, \quad T(0) = 0 \tag{9}$$
$$T(n) = r \cdot T(n-1) + s^n, \quad T(0) = 0 \tag{10}$$

Exercise 3.6 *Review the number of swaps needed for selection sort in the worst case. What is the Big-O expression of this number? This is known as the (asymptotic) complexity of the algorithm.*