



Design and analysis of algorithms

Edyta Szymańska

`edyta@cc.gatech.edu.pl`

Course overview

- ⑥ Efficient algorithms for fundamental computational problems.

Course overview

- ⑥ Efficient algorithms for fundamental computational problems.
- ⑥ NP-completeness.

Course overview

- ⑥ Efficient algorithms for fundamental computational problems.
- ⑥ NP-completeness.
- ⑥ Approximation algorithms.

Course overview

- ⑥ Efficient algorithms for fundamental computational problems.
- ⑥ NP-completeness.
- ⑥ Approximation algorithms.
- ⑥ Grading: Howeworks, Quiz(zes), Project (20%), Final exam

Course overview

- ⑥ Efficient algorithms for fundamental computational problems.
- ⑥ NP-completeness.
- ⑥ Approximation algorithms.
- ⑥ Grading: Howeworks, Quiz(zes), Project (20%), Final exam
- ⑥ Syllabus on Wednesday

On Algorithms

An Algorithm

On Algorithms

An **Algorithm** is a recipe or well-defined procedure for performing a calculation, or, in general, for transforming some input into a desired output.

On Algorithms

An **Algorithm** is a recipe or well-defined procedure for performing a calculation, or, in general, for transforming some input into a desired output.

Examples: algorithms for simple arithmetic operations

On Algorithms

An **Algorithm** is a recipe or well-defined procedure for performing a calculation, or, in general, for transforming some input into a desired output.

Examples: algorithms for simple arithmetic operations

algorithm comes from Al-Khwarizmi, a Persian mathematician who wrote a book on the decimal number system and algorithms for performing arithmetic (around 750 AD)

Example - multiplication algorithm

- ⑥ write the numbers side by side, say 75×29

Example - multiplication algorithm

- ⑥ write the numbers side by side, say 75×29
- ⑥ repeat

Example - multiplication algorithm

- ⑥ write the numbers side by side, say 75×29
- ⑥ repeat
 - △ divide the first number by 2 (no fractions)
 - △ multiply the second number by 2

Example - multiplication algorithm

- ⑥ write the numbers side by side, say 75×29
- ⑥ repeat
 - △ divide the first number by 2 (no fractions)
 - △ multiply the second number by 2
- ⑥ until the first number is 1

Example - multiplication algorithm

- ⑥ write the numbers side by side, say 75×29
- ⑥ repeat
 - △ divide the first number by 2 (no fractions)
 - △ multiply the second number by 2
- ⑥ until the first number is 1
- ⑥ cross out all rows in which the first entry is even

Example - multiplication algorithm

- ⑥ write the numbers side by side, say 75×29
- ⑥ repeat
 - △ divide the first number by 2 (no fractions)
 - △ multiply the second number by 2
- ⑥ until the first number is 1
- ⑥ cross out all rows in which the first entry is even
- ⑥ add all remaining entries of the second column

Example - multiplication algorithm

75	29
37	58
18	116
9	232
4	464
2	928
1	1856

2175

Example - multiplication algorithm

75	29
37	58
18	116
9	232
4	464
2	928
1	1856

2175

Analyzing the Algorithm

- ⑥ Does it halt ?

Analyzing the Algorithm

⑥ Does it halt ?

75 29

37 58

~~18 116~~

9 232

~~4 464~~

~~2 928~~

1 1856

2175

Analyzing the Algorithm

- ⑥ Does it halt ?
- ⑥ Is it correct ? (for every input it should halt with a correct output)

Analyzing the Algorithm

- Does it halt ?
- Is it correct ? (for every input it should halt with a correct output)

75	29	
37	58	
18	116	29
9	232	1001011
4	464	29
2	928	58
1	1856	232
<hr/>		1856
	2175	2175

Analyzing the Algorithm

- ⑥ Does it halt ?
- ⑥ Is it correct ? (for every input it should halt with a correct output)
- ⑥ How much time does it take ?

Analyzing the Algorithm

- ⑥ Does it halt ?
- ⑥ Is it correct ? (for every input it should halt with a correct output)
- ⑥ How much time does it take ?
- ⑥ How much memory does it use ?

Designing the Algorithm

- ⑥ Expressing algorithms
 - △ in English prose
 - △ **pseudo-code** (similar to Pascal, C, C++ and Java, see p.19-20 for conventions)

Designing the Algorithm

- ⑥ Expressing algorithms
 - △ in English prose
 - △ **pseudo-code** (similar to Pascal, C, C++ and Java, see p.19-20 for conventions)
- ⑥ How do we analyze an algorithm's running time ?
Let us consider an example first.

Computing the n th Fibonacci Number

The sequence F_0, F_1, F_2, \dots , where $F_0 = F_1 = 1$ and for all $n \geq 2$ $F_n = F_{n-1} + F_{n-2}$.

Computing the n th Fibonacci Number

The sequence F_0, F_1, F_2, \dots , where $F_0 = F_1 = 1$ and for all $n \geq 2$ $F_n = F_{n-1} + F_{n-2}$.
the numbers are: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

Computing the n th Fibonacci Number

The sequence F_0, F_1, F_2, \dots , where $F_0 = F_1 = 1$ and for all $n \geq 2$ $F_n = F_{n-1} + F_{n-2}$.

the numbers are: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

the Fibonacci numbers grow exponentially!

Computing the n th Fibonacci Number

The sequence F_0, F_1, F_2, \dots , where $F_0 = F_1 = 1$ and for all $n \geq 2$ $F_n = F_{n-1} + F_{n-2}$.

the numbers are: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

the Fibonacci numbers grow exponentially!

Suppose we want to compute the F_n for a given large integer n .

Computing the n th Fibonacci Number

The sequence F_0, F_1, F_2, \dots , where $F_0 = F_1 = 1$ and for all $n \geq 2$ $F_n = F_{n-1} + F_{n-2}$.

the numbers are: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

the Fibonacci numbers grow exponentially!

Suppose we want to compute the F_n for a given large integer n .

ALGORITHM 1:

```
function  $F(n : integer) : integer$ 
```

```
if  $n \leq 1$  then return 1
```

```
else return  $F(n - 1) + F(n - 2)$ 
```

Computing the n th Fibonacci Number

ALGORITHM 1:

```
function  $F(n : integer) : integer$   
if  $n \leq 1$  then return 1  
else return  $F(n - 1) + F(n - 2)$   
correct, always halts, but ...
```

Computing the n th Fibonacci Number

ALGORITHM 1:

function $F(n : \text{integer}) : \text{integer}$

if $n \leq 1$ then return 1

else return $F(n - 1) + F(n - 2)$

correct, always halts, but ...

Let $T(n)$ be the number of addition operations used by the algorithm on input n .

Computing the n th Fibonacci Number

ALGORITHM 1:

```
function  $F(n : integer) : integer$ 
```

```
if  $n \leq 1$  then return 1
```

```
else return  $F(n - 1) + F(n - 2)$ 
```

correct, always halts, but ...

Let $T(n)$ be the number of addition operations used by the algorithm on input n .

Express $T(n)$ in terms of smaller values of T . Thus,

$T(0) = 0$, $T(1) = 0$, for $n \geq 2$, we have

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

Computing the n th Fibonacci Number

ALGORITHM 1:

```
function  $F(n : integer) : integer$ 
```

```
if  $n \leq 1$  then return 1
```

```
else return  $F(n - 1) + F(n - 2)$ 
```

correct, always halts, but ...

Let $T(n)$ be the number of addition operations used by the algorithm on input n .

Express $T(n)$ in terms of smaller values of T . Thus,

$T(0) = 0$, $T(1) = 0$, for $n \geq 2$, we have

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

Solution: $T(n) \geq 2^{n/2}$ for $n \geq 4$!

Computing the n th Fibonacci Number - better ?

ALGORITHM 2:

```
function  $F(n : integer) : integer$   
array  $A[0..n]$  of integers, initially all 0  
 $A[0] := A[1] := 1$   
for  $i := 2$  to  $n$  do  
   $A[i] := A[i - 1] + A[i - 2]$   
return  $A[n]$ 
```

Computing the n th Fibonacci Number - better ?

ALGORITHM 2:

```
function  $F(n : integer) : integer$   
array  $A[0..n]$  of integers, initially all 0  
 $A[0] := A[1] := 1$   
for  $i := 2$  to  $n$  do  
   $A[i] := A[i - 1] + A[i - 2]$   
return  $A[n]$ 
```

Now $T(n) = n - 1$.

Computing the n th Fibonacci Number - better ?

We have reduced the exponential running time to a polynomial. Great !

Computing the n th Fibonacci Number - better ?

We have reduced the exponential running time to a polynomial. Great ! **BUT**... what was our model?
What constitutes an elementary step ?

Computing the n th Fibonacci Number - better ?

We have reduced the exponential running time to a polynomial. Great ! **BUT**... what was our model? What constitutes an elementary step ? We assumed that each arithmetic step takes unit time.

Computing the n th Fibonacci Number - better ?

We have reduced the exponential running time to a polynomial. Great ! **BUT**... what was our model? What constitutes an elementary step ? We assumed that each arithmetic step takes unit time. NOT TRUE if n is large

Computing the n th Fibonacci Number - better ?

We have reduced the exponential running time to a polynomial. Great ! **BUT**... what was our model? What constitutes an elementary step ? We assumed that each arithmetic step takes unit time. NOT TRUE if n is large Thus:

- ⑥ Algorithm 1: $O(nF_n)$
- ⑥ Algorithm 2: $O(n^2)$

Model of computation

- ⑥ **Turing machine** convenient in the Theory of Computation

Model of computation

- ⑥ **Turing machine** convenient in the Theory of Computation
- ⑥ **Random-access machine (RAM) model**

Model of computation

- ⑥ **Turing machine** convenient in the Theory of Computation
- ⑥ **Random-access machine (RAM) model**
 - △ sequential
 - △ arithmetic operations in unit time
 - △ data movement: load, store, copy in constant time
 - △ limited word size: integer n is represented by $c \log n$ bits for some $c \geq 1$

Next lecture

- ⑥ asymptotic notation

Next lecture

- ⑥ asymptotic notation
- ⑥ divide and conquer paradigm

Next lecture

- ⑥ asymptotic notation
- ⑥ divide and conquer paradigm
- ⑥ recurrences