DDS: Algorithms and Data Structures

Tutorial 1

Welcome!

Instructor: Veerle Timmermans

- Postdoctoral researcher at RWTH Aachen
- Research: Algorithm Design and Algorithmic Game Theory
- eMail: veerle.tantimmermans@oms.rwth-aachen.de
- Office: B225, Kackertstrasse 7, 52072 Aachen.
- Office hours: directly after class

Today:

- Organizational Matters
- Sorting Algorithms and Pseudo-code

Organizational Matters

All course materials can be found on the Moodle

- Lecture Slides
- Tutorial Slides (including the homework)
- Optional: Python Code for algorithms we discussed.

Moodle:
`https://moodle.rwth-aachen.de/`

Organizational Matters

All course materials can be found on the Moodle

- Lecture Slides
- Tutorial Slides (including the homework)
- Optional: Python Code for algorithms we discussed.

Moodle:
`https://moodle.rwth-aachen.de/`

Online Python:
`https://repl.it/repls/ProudTechnologicalQueryoptimizer`

Organizational Matters

Goal of the exercises: To prepare you for:

- ... the exam on November 13
- ... job and job interviews involving algorithms
- ... problems that require you to design algorithms

Organizational Matters

Goal of the exercises: To prepare you for:

- ... the exam on November 13
- ... job and job interviews involving algorithms
- ... problems that require you to design algorithms

- Discuss homework exercises (except today)
- Summarize lecture and clarify new concepts.
- We solve some questions together
- You get more exercises to try at home

Homework is not obligated, but strongly advised.

Tentative Schedule:

1. Introduction to Algorithms and Pseudocode
2. Running Time Analysis
3. Divide and Conquer Algorithms
4. Interval Scheduling and Partitioning
5. Lists, Queues, Stacks and Heaps
6. Graphs and Shortest Paths
7. Search Trees and Hashing
8. Minimum Spanning Trees and Clustering

Lecture Summary and Learning Goals

Summary: We discussed two different sorting algorithms:

1. Insertion Sort
2. Merge Sort

Lecture Summary and Learning Goals

Summary: We discussed two different sorting algorithms:

1. Insertion Sort
2. Merge Sort

Learning Goals Tutorial:

Apply Sorting lists using Insertion Sort and Merge Sort

Skill Reading (and understanding) new pseudo-code.

Learn
- Two new sorting algorithms: Bubble Sort and Selection Sort
- Analyze and compare different sorting algorithms in terms of elemental operations (number of swaps and comparisons).

Lecture Summary and Learning Goals

Summary: We discussed two different sorting algorithms:

1. Insertion Sort
2. Merge Sort

Learning Goals Tutorial:

Apply Sorting lists using Insertion Sort and Merge Sort

Skill Reading (and understanding) new pseudo-code.

Learn
- Two new sorting algorithms: Bubble Sort and Selection Sort
- Analyze and compare different sorting algorithms in terms of elemental operations (number of swaps and comparisons).

Note: in this tutorial, we consider a list sorted if all numbers are in non-decreasing order.

Pseudo-code

Introduction to Pseudo-code

Problem with algorithm describtions:

- Normal text is not well structured, nor precise.
- Python/Java/C++/...-code contains unnecessary details, and is hard to understand.

Introduction to Pseudo-code

Problem with algorithm describtions:

- Normal text is not well structured, nor precise.
- Python/Java/C++/...-code contains unnecessary details, and is hard to understand.

Pseudo-code:

- Targetted at humans: easier to understand than code.
- Focus on algorithms: Precise enough to describe algorithms.
- Does not require the knowledge of a specific language.

Introduction to Pseudo-code

Problem with algorithm describtions:

- Normal text is not well structured, nor precise.
- Python/Java/C++/...-code contains unnecessary details, and is hard to understand.

Pseudo-code:

- Targetted at humans: easier to understand than code.
- Focus on algorithms: Precise enough to describe algorithms.
- Does not require the knowledge of a specific language.

Structure of Pseudo-code:

- Input: What is the algorithm given?
- Output: What does the algorithm compute?
- Sequence of computational steps: How is the output computed from the input?

High-Level Structure

Algorithm name (arguments):

Input: What the algorithm requires for computation
Output: What the algorithms compute

Sequence of computational steps

High-Level Structure

Algorithm name (arguments):

Input: What the algorithm requires for computation

Output: What the algorithms compute

Sequence of computational steps

Algorithm Maximum(a,b):

Input: Two numbers $a, b \in \mathbb{R}$

Output: The maximum of a and b

if $a > b$ **then**
    return a
else
    return b
end

Conditional Statements: if-else

Algorithm if-else construction:

if *(condition)* **then**
  | // Code here is only executed if condition is true
else
  | // Code here is only executed when condition is false
end

Comments:

- We use '//' to denote the start of a comment
- Comments provide explanation to the reader and are not part of the computational steps
- Pay special attention to the identation

Conditional Statements - While

Algorithm while-loop:

```
while (condition) do
    // If condition holds, do stuff
    // ⋮
    // at the end, jump back to the condition-check and repeat
end
// If condition does not hold, we jump here
```

Conditional Statements - While

Algorithm while-loop:

```
while (condition) do
    // If condition holds, do stuff
    // ⋮
    // at the end, jump back to the condition-check and repeat
end
// If condition does not hold, we jump here
```

While-loop:

- Repeats some commands, as long as the condition is fullfilled
- If condition is initially false, the commands inside the while-loop are never executed
- If the condition remains satisfied, the commands will be executed endlessly

For-loops

Algorithm For-loop:

for $i = 1$ *to* 10 **do**

| // do something for $i = 1, \ldots, 10$

end

For-loops

Algorithm For-loop:

for $i = \mathit{1\ to\ 10}$ **do**
    // do something for $i = 1, \ldots, 10$
end

- For-loop is a special case of a While-loop

For-loops

Algorithm For-loop:

for $i = 1$ *to* 10 **do**
  // do something for $i = 1, \ldots, 10$
end

- For-loop is a special case of a While-loop

Algorithm While-loop:

$i := 1$
while $i \leq 10$ **do**
  // do something for $i = 1, \ldots, 10$
  $//i := i + 1$
end

Assignments and Operators

Assignments

- We write $i := 1$ to assign the number 1 to variable i.
- We write $i := i + 1$ to increase the value of i by 1.

Assignments and Operators

Assignments

- We write $i := 1$ to assign the number 1 to variable i.
- We write $i := i + 1$ to increase the value of i by 1.

Arithmetic Operators

- $+$, $-$, $\cdot$, $/$, $!$, log, $\sqrt{}$, x^y, ...

Assignments and Operators

Assignments

- We write $i := 1$ to assign the number 1 to variable i.
- We write $i := i + 1$ to increase the value of i by 1.

Arithmetic Operators

- $+, -, \cdot, /, !, \log, \sqrt{}, x^y, \ldots$

Logical Operators

- AND (c1 && c2): both conditions need to hold
- OR (c1 || c2): at least one (or both) condition needs to hold
- NOT ($\neg$ c1): condition should be false

Example: Exercise

Algorithm DoSomething(a, b)**:**

Input: Two numbers $a, b \in \mathbb{N}$

Output:

```
i := 0
while a > 0 do
    i := i + b
    a := a − 1
end
return i
```

Q: Explain what the algorithm above does.

Example: Exercise

Algorithm DoSomething(a, b):

Input: Two numbers $a, b \in \mathbb{N}$

Output: $a \cdot b$

```
i := 0
while a > 0 do
    i := i + b
    a := a - 1
end
return i
```

Q: Explain what the algorithm above does.

Example: Exercise

Q: What does the Pseudocode below?

```
Algorithm DoSomething(n):
Input: A n ∈ ℕ
Output:
a := 1
for i := 1 to n do
|   a := a · i
end
return a
```

Example: Exercise

Q: What does the Pseudocode below?

Algorithm DoSomething(n):

Input: A $n \in \mathbb{N}$

Output: $n! = 1 \cdot 2 \cdot \ldots \cdot n$

```
a := 1
for i := 1 to n do
|   a := a · i
end
return a
```

Example: Exercise

Q: What does the Pseudocode below?

Algorithm DoSomething(n):

Input: A $n \in \mathbb{N}$

Output: $n! = 1 \cdot 2 \cdot \ldots \cdot n$

$a := 1$

for $i := 1$ *to* n **do**

  $a := a \cdot i$

end

return a

Python

```
def fak(n):
    a = 1
    for i in range(n):
        a *= i+1
    return a
```

Exercises

Apply Sorting lists using Insertion Sort and Merge Sort

Skill Reading (and understanding) new pseudo-code.

Learn
- Two new sorting algorithms: Bubble Sort and Selection Sort
- Analyze and compare different sorting algorithms in terms of elemental operations (number of swaps and comparisons).

Exercise 1: Apply

Apply Insertion Sort and Merge Sort to sort the following lists:

a) $[5, 2, 3, 4, 1]$

b) $[1, 4, 2, 5, 3]$

c) $[5, 4, 3, 2, 1]$

Exercise 2

The pseudocode below describes the sorting algorithm that is known as Bubble Sort. (Python Code can be found on the Moodle)

Algorithm Bubble Sort:

Input: A: list of n sortable items
Output: Ordered list A.
for $i = 1$ **to** $n - 1$ **do**
  for $j = 1$ **to** $n - i$ **do**
    if $A_{j-1} > A_j$ **then**
      Swap A_{j-1} and A_j
    end
  end
end
return A;

Exercise 2: Skill

a) Explain in words how Bubble Sort sorts an array.

b) Apply Bubble Sort on the following arrays.

$$[5, 2, 3, 4, 1]$$

$$[1, 4, 2, 5, 3]$$

Exercise 3: Learn

In this exercise, we compare the number of elemental operations that are needed to execute Insertion Sort and Bubble Sort.

a) How many comparisons and swaps does Insertion Sort need to sort a list with n values in the worst case? Give a list in which this worst case bound is obtained.

b) How many comparisons and swaps does Bubble Sort need to sort a list with n values in the worst case? Give a list in which this worst case bound is obtained.

Exercise 4: Learn

We analyze the algorithm Merge Sort.

a) How many comparisons are needed to merge two lists of length n in Merge Sort?

b) How many comparisons does Merge Sort need at most to sort a list of size 2^n?

Design a variant of Merge sort that divides the array in three almost equal parts instead of two, i.e., the arrays should be the same length if possible and differ in length by at most one otherwise.

c) Describe a new merge-operation that merges three arrays of length n into one of length $3n$ and that needs at most $6n - 3$ comparisons.

d) Think of a new merge-operation for question c) that, even in the worst case, needs strictly less than $6n - 3$ comparisons.

Running Time Analysis

Q: Why are we so interested in the number of swaps and comparisons needed?

Running Time Analysis

Q: Why are we so interested in the number of swaps and comparisons needed?

- They give us a good idea about the number of lines of code that need to be executed
- They give us a good idea on the amount with which the running time of the algorithms will increase in case the size of the input gets larger.
- Though all the algorithms we discussed give us the same results, they do not have the same running time!

Running Time Analysis

Q: Why are we so interested in the number of swaps and comparisons needed?

- They give us a good idea about the number of lines of code that need to be executed
- They give us a good idea on the amount with which the running time of the algorithms will increase in case the size of the input gets larger.
- Though all the algorithms we discussed give us the same results, they do not have the same running time!

Next lecture you will learn more on running time analysis

Running Time Analysis

Q: Why are we so interested in the number of swaps and comparisons needed?

- They give us a good idea about the number of lines of code that need to be executed
- They give us a good idea on the amount with which the running time of the algorithms will increase in case the size of the input gets larger.
- Though all the algorithms we discussed give us the same results, they do not have the same running time!

Next lecture you will learn more on running time analysis

A related interesting topic: space complexity!

Homework

Exercise 1: Apply

Apply Insertion Sort, Bubble Sort and Merge Sort to sort the following lists:

a) $[1, 5, 6, 2, 4, 3]$

b) $[6, 4, 1, 3, 2, 5]$

Exercise 2: Skill

Algorithm Selection Sort:

Input: A: list of n sortable items
Output: Ordered list A.

```
for j = 0 to n − 1 do
    i_min ← j
    for i = j + 1 to n − 1 do
        if A_i < A_{i_min} then
            i_min ← i
        end
    end
    if i_min ≠ j then
        Swap A_j and A_{i_min}
    end
end
return A
```

Exercise 2: Skill

The pseudocode on the previous slide describes the sorting algorithm named Selection Sort. (Python Code can be found on the Moodle)

a) Explain in words how Selection Sort sorts an array.

b) Apply Selection Sort on the following arrays.

$$[2, 6, 3, 4, 1, 5]$$

$$[1, 4, 2, 5, 6, 3]$$

Exercise 3: Learn

In this exercise, we compare the number of elemental operations that are needed to execute a sorting algorithm.

a) How many comparisons and swaps does Merge Sort need to sort a list with 2^n values in the best case? Give a list in which this best case bound is obtained.

b) How many comparisons and swaps does Selection Sort need to sort a list with n values in the best case? Give a list in which this best case bound is obtained.