Algorithms and Datastructures
Runtime analysis Minsort / Heapsort, Induction
Structure

Algorithms and Datastructures
  Structure
  Links
  Organisation
    Daphne
    Forum
    Checkstyle
    Unit Tests
    Version management
    Jenkins

Sorting
  Minsort
  Heapsort
Topics of the Lecture:

■ Algorithms and Data Structures
  Efficient data handling and processing
  … for problems that occur in practical any larger program / project

■ Algorithm ≈ Solving of complex computational problems

■ Datastructure ≈ Representation of data on computer
Example 1: Sorting

Figure: Sorting with Minsort
Example 2: Navigation

- **Datastructures:** How to represent the map as data?
- **Algorithms:** How to find the shortest / fastest way?

**Figure:** Navigation plan © OpenStreetMap
General:

- Most of you had a lecture on basic progamming ... performance was not an issue

- Here it is going to be:
  1. How fast is our program?
  2. How can we make it faster?
  3. How can we proof that it will always be that fast?

- **Important** issues:
  - Most of the time: application runtime
  - Sometimes also: resource / space consumption
Content of the Lecture 2 / 2

Algorithms:
- Sorting
- Dynamic Arrays
- Associative Arrays
- Hashing
- Edit distance
- Priority Queue
- Linked Lists
- Pathfinding / Dijkstra Algorithm
- Search Trees

Mathematics:
- Runtime analysis
- $\mathcal{O}$-Notation
- Proof of correctness
After the lecture . . .

- . . . you should be able to understand the joke

*Figure: Comic © xkcd/835*

- Hopefully your parents will still invite you
Links

Homepage:
- Exercise sheets
- Lectures
- Materials

Link to Homepage
Lecture:
- Tuesday, 12:00 - 14:00, HS 00 006, Build. 082
- Recordings of the lecture will be uploaded to the webpage

Exercises:
- One exercise sheet per week
- Submission / Correction / Assistance online
- Tutorial: (if needed)
  Wednesday, 13:00-14:00 - HS 00 006, Build. 082

Exam:
- Planned: Sa. 23th March 2019, 10:00-12:00, Build. 101, Lec. theater 026 & 036
Exercises:
- 80% practical, 20% theoretical
- We expect everyone to solve every exercise sheet

Exam:
- 50% of all points from the exercise sheets are needed
- Content of exam: whole lecture and all exercises
Exercises:

- Tutors: Tim Maffenbeier, Till Steinmann, Tobias Faller
- Coordinators: Michael Uhl, Florian Eggenhofer and Björn Grüning
- Deadline: ESE: 1 week, IEMS: none
Organisation - Exercises 3 / 5

Exercises:

- Post questions into the forum (link later)
- Submission via “commit” through svn and Daphne
- Feedback one week after deadline through “update” (svn)
- Unit test / checkstyle via Jenkins
Exercises - Points:

- Practical:
  - 60% functionality
  - 20% tests
  - 20% documentation, Checkstyle, etc.
  - Program is not running ⇒ 0 points

- Theoretical (mathematical proof):
  - 40% general idea / approach
  - 60% clean / complete
Effort:

- 4 ECTS (ESE), 6 ECTS (IEMS)
- 120 / 180 working hours per semester
- 14 Lectures each 6h / 8h + exam
- 4h / 6h per exercise sheet (one per week)
Daphne:

- Provides the following information:
  - Name / contact information of your tutor
  - Download of / info needed for exercise sheets
  - Collected points of all exercise sheets
  - Links to:
    - Coding standards
    - Build system
    - The other systems

- Link: Daphne
Forum:

- Please don’t hesitate to ask if something is unclear
- Ask in the forum and not separate. Others might also be interested in the answer
- The tutors or the coordinators will reply as soon as possible
- Link: Forum
Checkstyle / Linting (flake8):

- Installation: `python3 -m pip install flake8`
- Check file: `python3 -m flake8 path/to/files/*.py`
- Link: [flake8](https://flake8.pycqa.dev/)
Unit Tests

Why unit tests?

1. A non-trivial method without a unit test is probably wrong
2. Simplifies debugging
3. We and you can automatically check correctness of code

What is a good unit test?

- Unit test checks desired output for a given input
- At least one typical input
- At least one critical case
  
  E.g. double occurrence of a value in sorting
Testing (doctest):

def subtract_one(n):
    """Subtracts 1 from n"
    >>> subtract_one(5)
    4
    >>> subtract_one(3)
    2
    """
    return n-1

if __name__ == "__main__":
    print("2 - 1 = %d" % subtract_one(2))

- Tests are contained in docstrings
- Module doctest runs them
- Run check with:
  python3 -m doctest
  path/to/files/*.py -v
Version management (subversion):

- Keeps a history of code changes
- Initialize / update directory: `svn checkout <URL>`
- Add files / folders: `svn add <file> --all`
- Create snapshot: `svn commit -m "<Your Message>"`
  Data is uploaded to Jenkins automatically
- Link: Subversion
Jenkins:

- Provides our build system
- You can check if your uploaded code runs
  - Especially whether all *unit test* pass
  - And if *checkstyle* (*flake8*) is satisfied
- Will be shown in the first exercise
- Link: Jenkins
Problem:
- Input: $n$ elements $x_1, \ldots, x_n$
- Transitive operator "<" which returns true if the left value is smaller than the right one
  - Transitivity: $x < y, y < z \rightarrow x < z$
- Output: $x_1, \ldots, x_n$ sorted with operator

Example

<table>
<thead>
<tr>
<th>Input:</th>
<th>14, 4, 32, 19, 8, 44, 65</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td></td>
</tr>
</tbody>
</table>
Why do we need sorting?

- Nearly every program needs a sorting algorithm
- **Examples:**
  - Index of a search engine
  - Listing filesystem in explorer / finder
  - (Music) library
  - Highscore list
Minsort - Algorithm

Informal description:

- Find the minimum and switch the value with the first position
- Find the minimum and switch the value with the second position
- ...

Figure: Minsort
Minsort in Python:

```python
def minsort(lst):
    for i in range(0, len(lst) - 1):
        minimum = i

        for j in range(i + 1, len(lst)):
            if lst[j] < lst[minimum]:
                minimum = j

        if minimum != i:
            lst[i], lst[minimum] = lst[minimum], lst[i]

    return lst
```
How long does our program run?

- We test it for different input sizes
- **Observation:**
  - It is going to be “disproportionately” slower the more numbers are being sorted

**Table: Runtime for Minsort**

<table>
<thead>
<tr>
<th>$n$</th>
<th>Runtime / ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2 \times 10^3$</td>
<td>5.24</td>
</tr>
<tr>
<td>$4 \times 10^3$</td>
<td>16.92</td>
</tr>
<tr>
<td>$6 \times 10^3$</td>
<td>39.11</td>
</tr>
<tr>
<td>$8 \times 10^3$</td>
<td>67.80</td>
</tr>
<tr>
<td>$10 \times 10^3$</td>
<td>105.50</td>
</tr>
<tr>
<td>$12 \times 10^3$</td>
<td>150.38</td>
</tr>
<tr>
<td>$14 \times 10^3$</td>
<td>204.00</td>
</tr>
<tr>
<td>$16 \times 10^3$</td>
<td>265.98</td>
</tr>
<tr>
<td>$18 \times 10^3$</td>
<td>334.94</td>
</tr>
</tbody>
</table>
MinSort - Runtime

How long does our program run?

- We test it for different input sizes

- **Observation:**
  It is going to be “disproportionately” slower the more numbers are being sorted

**Figure:** Runtime of *MinSort*
MinSort - Runtime

Runtime analysis:

- *Minsort* runtime depicted in a diagram
  - That is what you should do in the first exercise sheet

- **We observe:**
  - The runtime grows faster than linear
  - With double the input size we need four times the time

*Figure: Runtime of *Minsort*
Heapsort:

- The principle stays the same
- Better structure for finding the smallest element quicker

Binary heap:

- Preferably a complete binary tree
- **Heap property:** Each child is smaller (larger) than the parent element
Min heap:

- **Heap property**: Each child is smaller (larger) than the parent element
- A valid heap fulfills the property at each node

**Figure: Valid min heap**

**Figure: Invalid min heap**
How to save the heap?

- We number all nodes from top to bottom and left to right starting at 0
- The children of node $i$ are $2i + 1$ and $2i + 2$
- The parent node of node $i$ is $\left\lfloor \frac{i-1}{2} \right\rfloor$

Table: Elements can be stored in array

```
0 1 2 3 4 5 6
4 8 5 17 9 11 7
```
Repairing after taking the smallest element: \texttt{heap.pop()}

- Remove the smallest element (root node)
- Replace the root with the last node
- \textbf{Sift} the new root node down until the heap property is satisfied

\begin{figure}
\centering
\begin{tikzpicture}
  \node (root) {17} child {node {8} child {node {10} child {node {10}} child {node {9}}} child {node {25}}} child {node {22} child {node {29}}};
  \node (root2) at (2,0) {8} child {node {17} child {node {10} child {node {10}}} child {node {9}}} child {node {22} child {node {25}} child {node {29}}};
  \node (root3) at (4,0) {8} child {node {9} child {node {10}}} child {node {22} child {node {25}} child {node {29}}};
\end{tikzpicture}
\caption{Repairing a min heap}
\end{figure}
Heapsort:

- Organize the $n$ elements as heap
- While the heap still contains elements
  - Take the smallest element
  - Move the last node to the root
  - Repair the heap as described
- Output: 4, 5, …

Figure: One iteration of Heapsort
Creating a heap:

- This operation is called heapify
- The $n$ elements are already stored in an array
- Interpret the array as binary heap where the heap property is not yet satisfied
- We repair the heap from bottom up (in layers) with sift
Table: Input in array

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure: Heapify lower layer
Heapsort - Algorithm 8 / 10

Figure: Heapify upper layer
Heapsort - Algorithm 9 / 10

Figure: Resulting heap
Finding the minimum is intuitive:
- **Minsort**: Iterate through all non-sorted elements
- **Heapsort**: Finding the minimum is trivial (concept)
  \[
  \text{Just take the root of the heap}
  \]

Removing the minimum in Heapsort:
- Repair the heap and restore the **heap property**
  - We don’t have to repair the whole heap
- More of this in the next lecture
Further Literature

- **Course literature**
  
  
  
Further Literature

Sorting

https://en.wikipedia.org/wiki/Heapsort

[Wikb] Wikipedia - Selectionsort
https://de.wikipedia.org/wiki/Selectionsort
Further Literature

- **Subversion**

  [Apa] Apache Subversion
  https://subversion.apache.org/