Algorithms and Datastructures Runtime analysis Minsort / Heapsort, Induction

THE STATE OF THE S

Albert-Ludwigs-Universität Freiburg

Prof. Dr. Rolf Backofen

Bioinformatics Group / Department of Computer Science Algorithms and Datastructures, October 2018

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 computional problems
- **Datastructure**

 Representation of data on computer

Example 1: Sorting



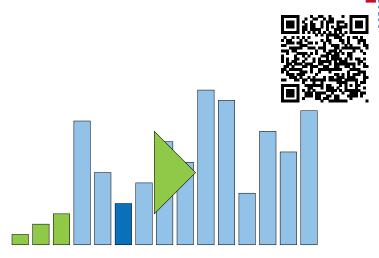


Figure: Sorting with *Minsort*

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



Figure: Navigationplan © OpenStreetMap

Content of the Lecture 1 / 2



General:

- Most of you had a lecture on basic progamming ... performance was not an issue
- Here it is going to be:
 - 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

- Priority Queue
- Linked Lists
- Pathfinding / Dijkstra Algorithm
- Search Trees

Mathematics:

- Runtime analysis
- Ø-Notation

Proof of correctness

After the lecture ...



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

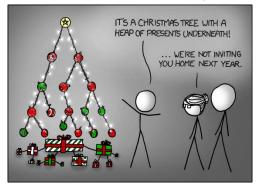


Figure: Comic @ xkcd/835

■ Hopefully your parents will still invite you



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

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 \Rightarrow 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



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
 - 2 Build system
 - 3 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
- I, Claudis Korzen or one of the tutors will reply as fast as possible
- Link: Forum

Checkstyle / Linting (flake8):

■ Installation: python3 -m pip install flake8

■ Check file: python3 -m flake8 path/to/files/*.py

■ Link: flake8

Why unit tests?

- A non-trivial method without an unit test is probably wrong
- Simplifies debugging
- We and you can automatic 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

doctest

Testing (doctest):

```
def subOne(n):
    """Subtracts 1 from n
    >>> subOne(5)
    >>> subOne(3)
    . . .
    return n-2
if __name__ == "__main ":
    print("2 minus 1: %d" % subOne(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 uploded code runs
 - Especially whether all unit test pass
 - And if checkstyle (flake8) is statisfied
- Will be shown in the first exercise
- Link: Jenkins

- Input: n elements $x_1, ..., x_n$
- 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, \dots, x_n sorted with operator

Example

Input: 14, 4, 32, 19, 8, 44, 65

Output:

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

Informal description:

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

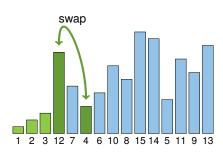


Figure: Minsort

Minsort in Python:

```
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]:</pre>
                 minimum = i
        if minimum != i:
             [st[i], [st[minimum] = \]
                 Ist[minimum], Ist[i]
    return 1st
```



How long does our program run?

- We test it for different input sizes
- Observation: It is going to be "disproportional" slower the more numbers are being sorted

Table: Runtime for *Minsort*

n	Runtime / ms			
2×10^3	5.24			
4×10^3	16.92			
6×10^3	39.11			
8×10^3	67.80			
10×10^3	105.50			
12×10^3	150.38			
14×10^3	204.00			
16×10^3	265.98			
18×10^3	334.94			



How long does our program run?

- We test it for different input sizes
- Observation: It is going to be "disproportional" slower the more numbers are being sorted

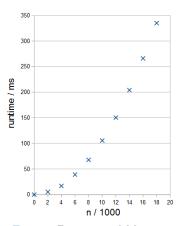


Figure: Runtime of *Minsort*

Runtime analysis:

- As a first example serves this diagram for *Minsort*
 - Thats 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

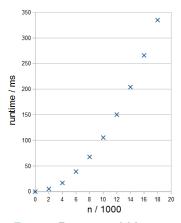


Figure: Runtime of *Minsort*

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

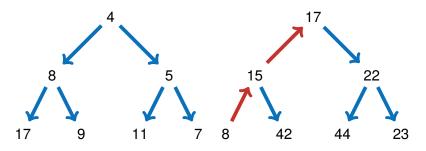
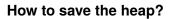


Figure: Valid min heap

Figure: Invalid min heap



- 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 floor $\left(\frac{i-1}{2}\right)$

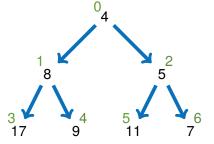


Table: Elements can be stored in array

0	1	2	3	4	5	6
4	8	5	17	9	11	7

Repairing after taking the smallest element: heap.pop()

- Remove the smallest element (root node)
- Replace the root with the last node
- Sift the new root node down until the heap property is statisfied

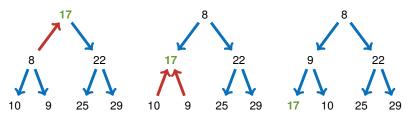


Figure: Repair of a min heap



- 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 like previously described
- Output: 4, 5, ...

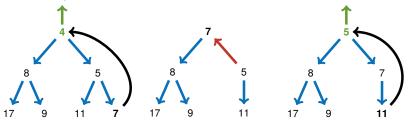


Figure: One iteration of Heapsort

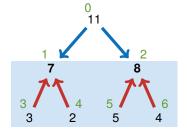
Creation of a heap:

- This operation is called heapify
- \blacksquare The *n* elements are already in the containing array
- Interpret this field als binary heap where the heap property is not yet statisfied
- We repair the heap from bottom up (in layers) with sift



Table: Input in array

0	1	2	3	4	5	6
11	7	8	3	2	5	4



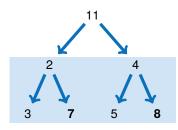
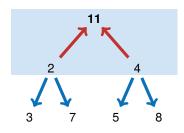


Figure: Heapify lower layer



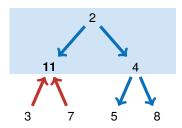


Figure: Heapify upper layer

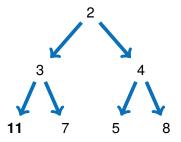


Figure: Resulting heap

Finding the minimum is intuitive:

- Minsort: Iterate through all non-sorted elements
- **Heapsort:** Finding the minimum is trivial (concept)

 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

■ General for this Lecture

[CRL01] Thomas H. Cormen, Ronald L. Rivest, and Charles E. Leiserson. Introduction to Algorithms. MIT Press, Cambridge, Mass, 2001.

[MS08] Kurt Mehlhorn and Peter Sanders. **Algorithms and Data Structures.**Springer, Berlin, 2008.

https://people.mpi-inf.mpg.de/~mehlhorn/ftp/Mehlhorn-Sanders-Toolbox.pdf.

Sorting

[Wika] Wikipedia - Heapsort

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/