WS 2020/21

Efficient Algorithms and Data Structures

Debarghya Ghoshdastidar

Fakultät für Informatik
TU München

Winter Term 2020/21

Part I

Organizational Matters

Part I

Organizational Matters

- Modul: IN2003
  - Name: "Efficient Algorithms and Data Structures"
    "Effiziente Algorithmen und Datenstrukturen"
  - ECTS: 8 Credit points (4+2 SWS)
- Lectures:
  - Recording on BigBlueButton
    Link: `https://bbb.in.tum.de/deb-av3-3u3`
    New videos on Monday and Friday morning
- All information on Moodle for lecture

- Required knowledge:
  - IN0001, IN0003
    “Introduction to Informatics 1/2”
    “Einführung in die Informatik 1/2”
  - IN0007
    “Fundamentals of Algorithms and Data Structures”
    “Grundlagen: Algorithmen und Datenstrukturen” (GAD)
  - IN0011
    “Basic Theoretic Informatics”
    “Einführung in die Theoretische Informatik” (THEO)
  - IN0015
    “Discrete Structures”
    “Diskrete Strukturen” (DS)
  - IN0018
    “Discrete Probability Theory”
    “Diskrete Wahrscheinlichkeitstheorie” (DWT)

The Lecturer

- Debarghya Ghoshdastidar
- Email: ghoshdas@in.tum.de
- Room: 03.11.043
- Q&A session: Mo 11:15–11:45, Fr 11:15–11:45

  Q&A open for everyone (no appointment needed)

Assistants / Tutors

- PhD Assistants:
  - Balasubramanian A. R.
  - bala.ayikudi@tum.de
  - Office hours (via BBB): Mo 12:15–13:45, Fr 12:15–13:45

  - Pascal Mattia Esser
  - esser@in.tum.de
  - Office hours (via BBB): Mo 12:15–13:45, Fr 12:15–13:45
- Appointment via Moodle
- Contact via email only for administrative problems

Assistants / Tutors

- Student tutors:
  - Mitja Daniel Krebs
  - Office hours (via BBB): Mo 14:15–15:45, Tu 10:15–11:45
  - Mahalakshmi Sabanayagam
  - Office hours (via BBB): Tu 14:15–15:45, We 10:15–11:45
- Appointment via Moodle

Assessment

- In order to pass the module you need to pass a written exam
  - Remote pen-and-paper exam with mandatory video supervision
  - OR On-site pen-and-paper exam
- Repeat exam (before next term begins)
  - Only for absentees or failures in end term exam
- Oral exam
  - No oral substitute for written exam
  - May be conducted to validate grade if we suspect dishonesty in written exam

Mode of interaction

- Lecture videos available on BBB:

  BBB Link: `https://bbb.in.tum.de/deb-av3-3u3`
- Moodle:
  - Important announcements
  - Q&A forum: Ask questions, help others
  - Weekly assignment submission
  - Feedback: Send (anonymous) feedback about course
- Q&A session with lecturer on BBB
- Tutorial: Solution video/slides uploaded on Moodle
- Q&A for assignment grading: Via BBB (use appointment)
- Contact us if you cannot access Moodle

Weekly assessment

Submit weekly assignment to improve your grade by 0,3 or 0,4

- Written exam 1,7 + bonus = Final grade 1,3
- Written exam 3,3 + bonus = Final grade 3,0

Requirements for Bonus

- Written exam grade between 1,3 to 4,0
- 50% of the points are achieved on submissions 2–8
- 50% of the points are achieved on submissions 9–14
- Each team member has written at least 4 solutions
- OR met requirements for bonus in WS-19/20

Weekly assessment

Assignment Sheets:

- Assignment available on Monday on Moodle / course website
  - Assignment 1 – Due on 9.11; Not graded
  - Assignment 2 – Graded; Available on 9.11; Due on 16.11
- Solutions to be submitted via Moodle

  Deadline: 08:00 on following Monday
- You must submit solutions in teams of **2** people

  Solutions have to be given in English

  Can be hand written (scanned)

Weekly assessment (Group submission)

Important only if you submit assignments

Registration (groups on Moodle):

- Find team partner in first week (use Moodle forum)
- Register in one of 8 groups on Moodle (Fr 6.11 – Fr 13.11)
  - Based on which time you can attend Q&A with tutors
  - Both members must register in same group
- Cannot change group / partner during semester

Weekly submission:

- Mention name, student id number for every team member in submission
- Mention who wrote the submission

1 Contents

- Foundations
  - Machine models
  - Efficiency measures
  - Asymptotic notation
  - Recursion
- Higher Data Structures
  - Search trees
  - Hashing
  - Priority queues
  - Union/Find data structures
- Cuts/Flows
- Matchings

2 Literatur

- Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman:
  The design and analysis of computer algorithms,
  Addison-Wesley Publishing Company: Reading (MA), 1974
- Thomas H. Cormen, Charles E. Leiserson, Ron L. Rivest, Clifford Stein:
  Introduction to algorithms,
  McGraw-Hill, 1990
- Michael T. Goodrich, Roberto Tamassia:
  Algorithm design: Foundations, analysis, and internet examples,
  John Wiley & Sons, 2002

2 Literatur

- Ronald L. Graham, Donald E. Knuth, Oren Patashnik:
  Concrete Mathematics,
  2. Auflage, Addison-Wesley, 1994
- Volker Heun:
  Grundlegende Algorithmen: Einführung in den Entwurf und die Analyse effizienter Algorithmen,
  2. Auflage, Vieweg, 2003
- Jon Kleinberg, Eva Tardos:
  Algorithm Design,
  Addison-Wesley, 2005
- Donald E. Knuth:
  The art of computer programming. Vol. 1: Fundamental Algorithms,
  3. Auflage, Addison-Wesley, 1997

2 Literatur

Donald E. Knuth:
The art of computer programming. Vol. 3: Sorting and Searching,
3. Auflage, Addison-Wesley, 1997

Christos H. Papadimitriou, Kenneth Steiglitz:
Combinatorial Optimization: Algorithms and Complexity,
Prentice Hall, 1982

Uwe Schöning:
Algorithmik,
Spektrum Akademischer Verlag, 2001

Steven S. Skiena:
The Algorithm Design Manual,
Springer, 1998

Part II

Foundations

3 Goals

- Gain knowledge about efficient algorithms for important problems, i.e., learn how to solve certain types of problems efficiently.
- Learn how to analyze and judge the efficiency of algorithms.
- Learn how to design efficient algorithms.

4 Modelling Issues

What do you measure?

- Memory requirement
- Running time
- Number of comparisons
- Number of multiplications
- Number of hard-disc accesses
- Program size
- Power consumption
- . . .

4 Modelling Issues

How do you measure?

- Implementing and testing on representative inputs
  - How do you choose your inputs?
  - May be very time-consuming.
  - Very reliable results if done correctly.
  - Results only hold for a specific machine and for a specific set of inputs.

- Theoretical analysis in a specific model of computation.
  - Gives asymptotic bounds like “this algorithm always runs in time $\mathcal{O}(n^2)$”.
  - Typically focuses on the worst case.
  - Can give lower bounds like “any comparison-based sorting algorithm needs at least $\Omega(n \log n)$ comparisons in the worst case”.

4 Modelling Issues

Input length

The theoretical bounds are usually given by a function $f : \mathbb{N} \to \mathbb{N}$ that maps the input length to the running time (or storage space, comparisons, multiplications, program size etc.).

The input length may e.g. be

- the size of the input (number of bits)
- the number of arguments

Example 1

Suppose n numbers from the interval $\{1, \ldots, N\}$ have to be sorted. In this case we usually say that the input length is n instead of e.g. $n \log N$, which would be the number of bits required to encode the input.

Model of Computation

How to measure performance

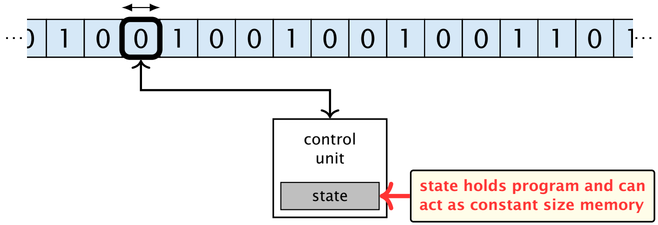
1. Calculate running time and storage space etc. on a simplified, idealized model of computation, e.g. Random Access Machine (RAM), Turing Machine (TM), . . .
2. Calculate number of certain basic operations: comparisons, multiplications, harddisc accesses, . . .

Version 2. is often easier, but focusing on one type of operation makes it more difficult to obtain meaningful results.

Turing Machine

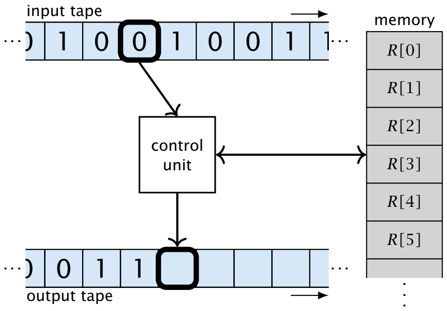
- Very simple model of computation.
- Only the “current” memory location can be altered.
- Very good model for discussing computabiliy, or polynomial vs. exponential time.
- Some simple problems like recognizing whether input is of the form xx, where x is a string, have quadratic lower bound.

$\Rightarrow$ **Not a good model for developing efficient algorithms.**



Random Access Machine (RAM)

- Input tape and output tape (sequences of zeros and ones; unbounded length).
- Memory unit: infinite but countable number of registers $R[0], R[1], R[2], \ldots$.
- Registers hold integers.
- Indirect addressing.



Note that in the picture on the right the tapes are one-directional, and that a READ- or WRITE-operation always advances its tape.

Random Access Machine (RAM)

Operations

- input operations (input tape $\rightarrow R[i]$)
  - `READ` i
- output operations ($R[i] \rightarrow$ output tape)
  - `WRITE` i
- register-register transfers
  - $R[j]$ `:=` $R[i]$
  - $R[j]$ `:=` 4
- **indirect** addressing
  - $R[j]$ `:=` $R[R[i]]$
    loads the content of the $R[i]$-th register into the j-th register
  - $R[R[i]] := R[j]$
    loads the content of the j-th into the $R[i]$-th register

Random Access Machine (RAM)

Operations

- branching (including loops) based on comparisons
  - `jump` x
    jumps to position x in the program;
    sets instruction counter to x;
    reads the next operation to perform from register $R[x]$
  - `jumpz` x $R[i]$
    jump to x if $R[i] = 0$
    if not the instruction counter is increased by 1;
  - `jumpi` i
    jump to $R[i]$ (indirect jump);
- arithmetic instructions: $+$, $-$, $\times$, $/$
  - $R[i]$:= $R[j]$ + $R[k]$;
    $R[i]$:= -$R[k]$;

The jump-directives are very close to the jump-instructions contained in the assembler language of real machines.

Model of Computation

- uniform cost model
  Every operation takes time 1.
- logarithmic cost model
  The cost depends on the content of memory cells:
  - The time for a step is equal to the largest operand involved;
  - The storage space of a register is equal to the length (in bits) of the largest value ever stored in it.

Bounded word RAM model: cost is uniform but the largest value stored in a register may not exceed 2^w, where usually $w = \log_2 n$.

The latter model is quite realistic as the word-size of a standard computer that handles a problem of size n must be at least $\log_2 n$ as otherwise the computer could either not store the problem instance or not address all its memory.

4 Modelling Issues

Example 2

Algorithm 1 RepeatedSquaring(n)

```
1: $r \leftarrow 2$;
2: for $i = 1 \rightarrow n$ do
3:     $r \leftarrow r^2$
4: return $r$
```

- running time (for Line 3):
  - uniform model: n steps
  - logarithmic model:
    $2 + 3 + 5 + \cdots + (1 + 2^n) = 2^{n+1} - 1 + n = \Theta(2^n)$
- space requirement:
  - uniform model: $\mathcal{O}(1)$
  - logarithmic model: $\mathcal{O}(2^n)$

There are different types of complexity bounds:

- best-case complexity:

$$C_{\mathrm{bc}}(n) := \min\{C(x) \mid |x| = n\}$$

  Usually easy to analyze, but not very meaningful.

- worst-case complexity:

$$C_{\mathrm{wc}}(n) := \max\{C(x) \mid |x| = n\}$$

  Usually moderately easy to analyze; sometimes too pessimistic.

- average case complexity:

$$C_{\mathrm{avg}}(n) := \frac{1}{|I_n|} \sum_{|x|=n} C(x)$$

  more general: probability measure μ

$$C_{\mathrm{avg}}(n) := \sum_{x \in I_n} \mu(x) \cdot C(x)$$

μ is a probability distribution over inputs of length n.

$C(x)$	cost of instance x
$\lvert x\rvert$	input length of instance x
I_n	set of instances of length n

There are different types of complexity bounds:

- amortized complexity:
  The average cost of data structure operations over a worst case sequence of operations.
- randomized complexity:
  The algorithm may use random bits. Expected running time (over all possible choices of random bits) for a fixed input x. Then take the worst-case over all x with $|x| = n$.

$C(x)$	cost of instance x
$\lvert x\rvert$	input length of instance x
I_n	set of instances of length n

μ is a probability distribution over inputs of length n.

4 Modelling Issues

Bibliography

[MS08] Kurt Mehlhorn, Peter Sanders:
Algorithms and Data Structures — The Basic Toolbox,
Springer, 2008

[CLRS90] Thomas H. Cormen, Charles E. Leiserson, Ron L. Rivest, Clifford Stein:
Introduction to algorithms (3rd ed.),
McGraw-Hill, 2009

Chapter 2.1 and 2.2 of [MS08] and Chapter 2 of [CLRS90] are relevant for this section.

5 Asymptotic Notation

We are usually not interested in exact running times, but only in an asymptotic classification of the running time, that ignores constant factors and constant additive offsets.

- We are usually interested in the running times for large values of n. Then constant additive terms do not play an important role.
- An exact analysis (e.g. *exactly* counting the number of operations in a RAM) may be hard, but wouldn't lead to more precise results as the computational model is already quite a distance from reality.
- A linear speed-up (i.e., by a constant factor) is always possible by e.g. implementing the algorithm on a faster machine.
- Running time should be expressed by simple functions.

Asymptotic Notation

Formal Definition

Let f, g denote functions from $\mathbb{N}$ to $\mathbb{R}^+$.

- $\mathcal{O}(f) = \{g \mid \exists c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geq n_0 :\ [g(n) \leq c \cdot f(n)]\}$
  (set of functions that asymptotically grow not faster than f)
- $\Omega(f) = \{g \mid \exists c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geq n_0 :\ [g(n) \geq c \cdot f(n)]\}$
  (set of functions that asymptotically grow not slower than f)
- $\Theta(f) = \Omega(f) \cap \mathcal{O}(f)$
  (functions that asymptotically have the same growth as f)
- $o(f) = \{g \mid \forall c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geq n_0 :\ [g(n) \leq c \cdot f(n)]\}$
  (set of functions that asymptotically grow slower than f)
- $\omega(f) = \{g \mid \forall c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geq n_0 :\ [g(n) \geq c \cdot f(n)]\}$
  (set of functions that asymptotically grow faster than f)

Asymptotic Notation

There is an equivalent definition using limes notation (assuming that the respective limes exists). f and g are functions from $\mathbb{N}_0$ to $\mathbb{R}_0^+$.

- $g \in \mathcal{O}(f)$: $\quad 0 \leq \lim_{n\to\infty} \frac{g(n)}{f(n)} < \infty$
- $g \in \Omega(f)$: $\quad 0 < \lim_{n\to\infty} \frac{g(n)}{f(n)} \leq \infty$
- $g \in \Theta(f)$: $\quad 0 < \lim_{n\to\infty} \frac{g(n)}{f(n)} < \infty$
- $g \in o(f)$: $\quad \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$
- $g \in \omega(f)$: $\quad \lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty$

- Note that for the version of the Landau notation defined here, we assume that f and g are positive functions.
- There also exist versions for arbitrary functions, and for the case that the limes is not infinity.

Asymptotic Notation

Abuse of notation

1. People write $f = \mathcal{O}(g)$, when they mean $f \in \mathcal{O}(g)$. This is **not** an equality (how could a function be equal to a set of functions).
2. People write $f(n) = \mathcal{O}(g(n))$, when they mean $f \in \mathcal{O}(g)$, with $f : \mathbb{N} \to \mathbb{R}^+, n \mapsto f(n)$, and $g : \mathbb{N} \to \mathbb{R}^+, n \mapsto g(n)$.
3. People write e.g. $h(n) = f(n) + o(g(n))$ when they mean that there exists a function $z : \mathbb{N} \to \mathbb{R}^+, n \mapsto z(n), z \in o(g)$ such that $h(n) = f(n) + z(n)$.

2. In this context $f(n)$ does **not** mean the function f evaluated at n, but instead it is a shorthand for the function itself (leaving out domain and codomain and only giving the rule of correspondence of the function).

3. This is particularly useful if you do not want to ignore constant factors. For example the median of n elements can be determined using $\frac{3}{2}n + o(n)$ comparisons.

Asymptotic Notation

Abuse of notation

4. People write $\mathcal{O}(f(n)) = \mathcal{O}(g(n))$, when they mean $\mathcal{O}(f(n)) \subseteq \mathcal{O}(g(n))$. Again this is not an equality.

2. In this context $f(n)$ does **not** mean the function f evaluated at n, but instead it is a shorthand for the function itself (leaving out domain and codomain and only giving the rule of correspondence of the function).

3. This is particularly useful if you do not want to ignore constant factors. For example the median of n elements can be determined using $\frac{3}{2}n + o(n)$ comparisons.

Asymptotic Notation in Equations

How do we interpret an expression like:

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$$

Here, $\Theta(n)$ stands for an anonymous function in the set $\Theta(n)$ that makes the expression true.

Note that $\Theta(n)$ is on the right hand side, otw. this interpretation is wrong.

Asymptotic Notation in Equations

How do we interpret an expression like:

$$2n^2 + \mathcal{O}(n) = \Theta(n^2)$$

Regardless of how we choose the anonymous function $f(n) \in \mathcal{O}(n)$ there is an anonymous function $g(n) \in \Theta(n^2)$ that makes the expression true.

Asymptotic Notation in Equations

The $\Theta(i)$-symbol on the left represents one anonymous function $f : \mathbb{N} \to \mathbb{R}^+$, and then $\sum_i f(i)$ is computed.

How do we interpret an expression like:

$$\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$$

Careful!

"It is understood" that every occurence of an $\mathcal{O}$-symbol (or $\Theta, \Omega, o, \omega$) on the left represents one anonymous function.

Hence, the left side is **not** equal to

$$\Theta(1) + \Theta(2) + \cdots + \Theta(n-1) + \Theta(n)$$

$\Theta(1)+\Theta(2)+\cdots+\Theta(n-1)+\Theta(n)$ does not really have a reasonable interpretation.

Asymptotic Notation in Equations

We can view an expression containing asymptotic notation as generating a set:

$$n^2 \cdot \mathcal{O}(n) + \mathcal{O}(\log n)$$

represents

$$\Big\{ f : \mathbb{N} \to \mathbb{R}^+ \mid f(n) = n^2 \cdot g(n) + h(n) \text{ with } g(n) \in \mathcal{O}(n) \text{ and } h(n) \in \mathcal{O}(\log n) \Big\}$$

Recall that according to the previous slide e.g. the expressions $\sum_{i=1}^{n} \mathcal{O}(i)$ and $\sum_{i=1}^{n/2} \mathcal{O}(i) + \sum_{i=n/2+1}^{n} \mathcal{O}(i)$ generate different sets.

Asymptotic Notation in Equations

Then an asymptotic equation can be interpreted as containement btw. two sets:

$$n^2 \cdot \mathcal{O}(n) + \mathcal{O}(\log n) = \Theta(n^2)$$

represents

$$n^2 \cdot \mathcal{O}(n) + \mathcal{O}(\log n) \subseteq \Theta(n^2)$$

Note that the equation does not hold.

Asymptotic Notation

Lemma 3

Let f, g be functions with the property $\exists n_0 > 0\ \forall n \geq n_0 : f(n) > 0$ (the same for g). Then

- $c \cdot f(n) \in \Theta(f(n))$ *for any constant* c
- $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$
- $\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n))$
- $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(\max\{f(n), g(n)\})$

The expressions also hold for Ω. Note that this means that $f(n) + g(n) \in \Theta(\max\{f(n), g(n)\})$.

Asymptotic Notation

Comments

- Do not use asymptotic notation within induction proofs.
- For any constants a, b we have $\log_a n = \Theta(\log_b n)$. Therefore, we will usually ignore the base of a logarithm within asymptotic notation.
- In general $\log n = \log_2 n$, i.e., we use 2 as the default base for the logarithm.

Asymptotic Notation

In general asymptotic classification of running times is a good measure for comparing algorithms:

- If the running time analysis is tight and actually occurs in practise (i.e., the asymptotic bound is not a purely theoretical worst-case bound), then the algorithm that has better asymptotic running time will always outperform a weaker algorithm for large enough values of n.
- However, suppose that I have two algorithms:
  - Algorithm A. Running time $f(n) = 1000 \log n = \mathcal{O}(\log n)$.
  - Algorithm B. Running time $g(n) = \log^2 n$.

  Clearly $f = o(g)$. However, as long as $\log n \le 1000$ Algorithm B will be more efficient.

Multiple Variables in Asymptotic Notation

Sometimes the input for an algorithm consists of several parameters (e.g., nodes and edges of a graph (n and m)).

If we want to make asympotic statements for $n \to \infty$ and $m \to \infty$ we have to extend the definition to multiple variables.

Formal Definition

Let f, g denote functions from $\mathbb{N}^d$ to $\mathbb{R}_0^+$.

- $\mathcal{O}(f) = \{g \mid \exists c > 0\ \exists N \in \mathbb{N}_0\ \forall \vec{n}$ with $n_i \geq N$ for some i : $[g(\vec{n}) \leq c \cdot f(\vec{n})]\}$

  (set of functions that asymptotically grow not faster than f)

Multiple Variables in Asymptotic Notation

Example 4

- $f : \mathbb{N} \to \mathbb{R}_0^+$, $f(n, m) = 1$ und $g : \mathbb{N} \to \mathbb{R}_0^+$, $g(n, m) = n - 1$
  then $f = \mathcal{O}(g)$ does not hold
- $f : \mathbb{N} \to \mathbb{R}_0^+$, $f(n, m) = 1$ und $g : \mathbb{N} \to \mathbb{R}_0^+$, $g(n, m) = n$
  then: $f = \mathcal{O}(g)$
- $f : \mathbb{N}_0 \to \mathbb{R}_0^+$, $f(n, m) = 1$ und $g : \mathbb{N}_0 \to \mathbb{R}_0^+$, $g(n, m) = n$
  then $f = \mathcal{O}(g)$ does not hold

5 Asymptotic Notation

Bibliography

[MS08] Kurt Mehlhorn, Peter Sanders:
Algorithms and Data Structures — The Basic Toolbox,
Springer, 2008

[CLRS90] Thomas H. Cormen, Charles E. Leiserson, Ron L. Rivest, Clifford Stein:
Introduction to algorithms (3rd ed.),
McGraw-Hill, 2009

Mainly Chapter 3 of [CLRS90]. [MS08] covers this topic in chapter 2.1 but not very detailed.