WS 2019/20

Efficient Algorithms and Data Structures

Harald Räcke

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2019WS/ea/

Winter Term 2019/20

Part I

Organizational Matters

Part I

Organizational Matters

- Modul: IN2003
- Name: "Efficient Algorithms and Data Structures"
  "Effiziente Algorithmen und Datenstrukturen"
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 10:00–12:00 (Room Interim2)
    Fri 10:00–12:00 (Room Interim2)
- Webpage: `http://www14.in.tum.de/lehre/2019WS/ea/`

Part I

Organizational Matters

- Modul: IN2003
- Name: “Efficient Algorithms and Data Structures”
  “Effiziente Algorithmen und Datenstrukturen”
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 10:00–12:00 (Room Interim2)
    Fri 10:00–12:00 (Room Interim2)
- Webpage: `http://www14.in.tum.de/lehre/2019WS/ea/`

Part I

Organizational Matters

- Modul: IN2003
- Name: "Efficient Algorithms and Data Structures"
  "Effiziente Algorithmen und Datenstrukturen"
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 10:00–12:00 (Room Interim2)
    Fri 10:00–12:00 (Room Interim2)
- Webpage: `http://www14.in.tum.de/lehre/2019WS/ea/`

Part I

Organizational Matters

- Modul: IN2003
- Name: "Efficient Algorithms and Data Structures"
  "Effiziente Algorithmen und Datenstrukturen"
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 10:00–12:00 (Room Interim2)
    Fri 10:00–12:00 (Room Interim2)
- Webpage: http://www14.in.tum.de/lehre/2019WS/ea/

Part I

Organizational Matters

- Modul: IN2003
- Name: "Efficient Algorithms and Data Structures"
  "Effiziente Algorithmen und Datenstrukturen"
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 10:00–12:00 (Room Interim2)
    Fri 10:00–12:00 (Room Interim2)
- Webpage: `http://www14.in.tum.de/lehre/2019WS/ea/`

- 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)

- 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)

- 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)

- 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)

- 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)

- 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

- Harald Räcke
- Email: raecke@in.tum.de
- Room: 03.09.044
- Office hours: (by appointment)

Tutorials

A01 Monday, 12:00–14:00, 00.08.038 (Stotz)

A02 Monday, 12:00–14:00, 00.09.038 (Guan)

A03 Monday, 14:00–16:00, 02.09.023 (Stotz)

B04 Tuesday, 10:00–12:00, 00.08.053 (Czerner)

B05 Tuesday, 14:00–16:00, 00.08.038 (Czerner)

C06 Wednesday, 10:00–12:00, 03.11.018 (Guan)

E07 Friday, 12:00–14:00, 00.13.009 (Stotz)

Assignment sheets

In order to pass the module you need to pass an exam.

Assessment

Assignment Sheets:

- An assignment sheet is usually made available on Monday on the module webpage.
- Solutions have to be handed in in the following week before the lecture on Monday.
- You can hand in your solutions by putting them in the mailbox "Efficient Algorithms" on the basement floor in the MI-building.
- Solutions have to be given in English.
- Solutions will be discussed in the tutorial of the week when the sheet has been handed in, i.e, sheet may not be corrected by this time.
- You should submit solutions in groups of up to 2 people.

Assessment

Assignment Sheets:

- An assignment sheet is usually made available on Monday on the module webpage.
- Solutions have to be handed in in the following week before the lecture on Monday.
- You can hand in your solutions by putting them in the mailbox "Efficient Algorithms" on the basement floor in the MI-building.
- Solutions have to be given in English.
- Solutions will be discussed in the tutorial of the week when the sheet has been handed in, i.e, sheet may not be corrected by this time.
- You should submit solutions in groups of up to 2 people.

Assessment

Assignment Sheets:

- An assignment sheet is usually made available on Monday on the module webpage.
- Solutions have to be handed in in the following week before the lecture on Monday.
- You can hand in your solutions by putting them in the mailbox "Efficient Algorithms" on the basement floor in the MI-building.
- Solutions have to be given in English.
- Solutions will be discussed in the tutorial of the week when the sheet has been handed in, i.e, sheet may not be corrected by this time.
- You should submit solutions in groups of up to 2 people.

Assessment

Assignment Sheets:

- An assignment sheet is usually made available on Monday on the module webpage.
- Solutions have to be handed in in the following week before the lecture on Monday.
- You can hand in your solutions by putting them in the mailbox "Efficient Algorithms" on the basement floor in the MI-building.
- Solutions have to be given in English.
- Solutions will be discussed in the tutorial of the week when the sheet has been handed in, i.e, sheet may not be corrected by this time.
- You should submit solutions in groups of up to 2 people.

Assessment

Assignment Sheets:

- An assignment sheet is usually made available on Monday on the module webpage.
- Solutions have to be handed in in the following week before the lecture on Monday.
- You can hand in your solutions by putting them in the mailbox "Efficient Algorithms" on the basement floor in the MI-building.
- Solutions have to be given in English.
- Solutions will be discussed in the tutorial of the week when the sheet has been handed in, i.e, sheet may not be corrected by this time.
- You should submit solutions in groups of up to 2 people.

Assessment

Assignment Sheets:

- An assignment sheet is usually made available on Monday on the module webpage.
- Solutions have to be handed in in the following week before the lecture on Monday.
- You can hand in your solutions by putting them in the mailbox "Efficient Algorithms" on the basement floor in the MI-building.
- Solutions have to be given in English.
- Solutions will be discussed in the tutorial of the week when the sheet has been handed in, i.e, sheet may not be corrected by this time.
- You should submit solutions in groups of up to **2** people.

Assessment

Assignment Sheets:

- Submissions must be handwritten by a member of the group. Please indicate who wrote the submission.
- Don't forget name and student id number for each group member.

Assessment

Assignment Sheets:

- Submissions must be handwritten by a member of the group. Please indicate who wrote the submission.
- Don't forget name and student id number for each group member.

Assessment

Assignment can be used to improve you grade

Requirements for Bonus

- 50% of the points are achieved on submissions 2–8,
- 50% of the points are achieved on submissions 9–14,
- each group member has written at least 4 solutions.

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

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

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

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.

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.

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.

(2)

Input $\longrightarrow$ Output

I $\qquad$ $\mathcal{O}$

$f_A : I \to \mathcal{O}$

- A holds on every $x \in I$
- $f_A = f_P$

Problem : $f_P : I \to \mathcal{O}$

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

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

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

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

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

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

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

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

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

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

Suppose n numbers from the interval $\{1, \dots, N\}$ 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.

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

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.

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 $\{1, \dots, N\}$ 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.

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.

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

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

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

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.

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.

Model of Computation

How to measure performance

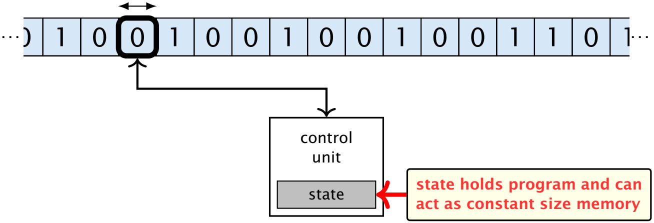
1. Calculate running time and storage space etc. on a simplified, idealized model of computation, e.g. Random Access Machine (RAM), Turing Machine (TM), ...
2. Calculate number of certain basic operations: comparisons, multiplications, harddisc accesses, ...

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.

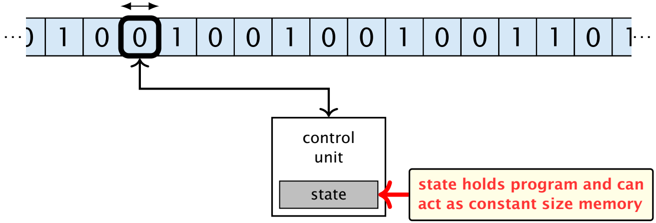
$\Rightarrow$ **Not a good model for developing efficient algorithms.**



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.

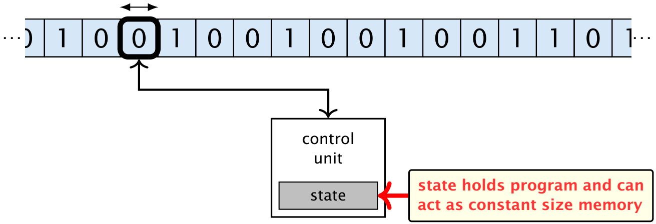
$\Rightarrow$ **Not a good model for developing efficient algorithms.**



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.

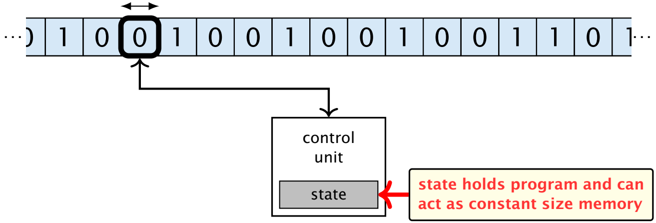
$\Rightarrow$ **Not a good model for developing efficient algorithms.**



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.

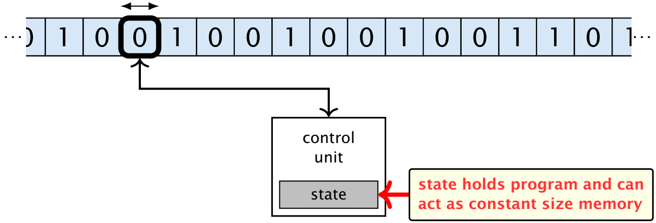
$\Rightarrow$ Not a good model for developing efficient algorithms.



Turing Machine

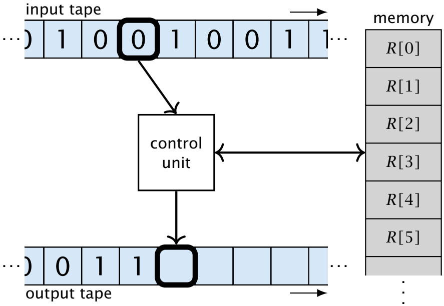
- Very simple model of computation.
- Only the "current" memory location can be altered.
- Very good model for discussing computabiliy, or polynomial vs. exponential time.
- Some simple problems like recognizing whether input is of the form xx, where x is a string, have quadratic lower bound.

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



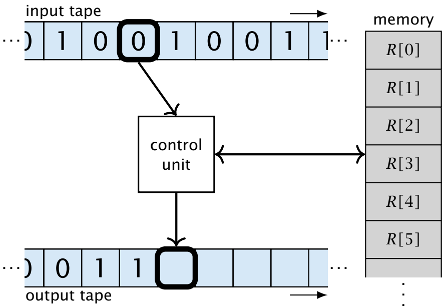
Random Access Machine (RAM)

- 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.



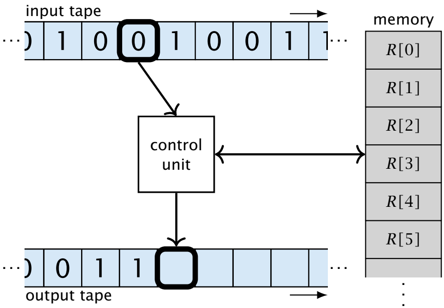
Random Access Machine (RAM)

- 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.



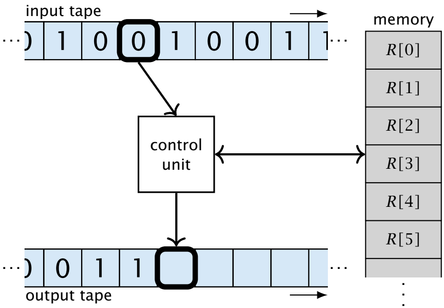
Random Access Machine (RAM)

- 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.



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.



Random Access Machine (RAM)

Operations

- input operations (input tape → $R[i]$)
  - READ i
- output operations ($R[i]$ → 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

- input operations (input tape → $R[i]$)
  - READ i

Random Access Machine (RAM)

Operations

- input operations (input tape → $R[i]$)
  - READ i
- output operations ($R[i]$ → 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

- input operations (input tape $\to R[i]$)
  - READ i
- output operations ($R[i] \to$ 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

- 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

- 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

- input operations (input tape $\to R[i]$)
  - READ i
- output operations ($R[i] \to$ output tape)
  - WRITE i
- register-register transfers
  - $R[j]$:= $R[i]$
  - $R[j]$:= 4
- indirect addressing

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

- 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

- input operations (input tape → $R[i]$)
  - `READ` i
- output operations ($R[i]$ → 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: $+$, $-$, $\times$, $/$
  - $R[i] := R[j] + R[k]$;
    $R[i] := -R[k]$;

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: $+$, $-$, $\times$, $/$
  - $R[i] := R[j] + R[k]$;
    $R[i] := -R[k]$;

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: $+$, $-$, $\times$, $/$
  - $R[i] := R[j] + R[k];$
    $R[i] := -R[k];$

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: $+, -, \times, /$
  - $R[i] := R[j] + R[k];$
    $R[i] := -R[k];$

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: $+$, $-$, $\times$, $/$
  - $R[i]$:= $R[j]$ + $R[k]$;
    $R[i]$:= -$R[k]$;

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: $+$, $-$, $\times$, $/$
  - $R[i]$ `:=` $R[j]$ `+` $R[k]$`;`
    $R[i]$ `:=` `-`$R[k]$`;`

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 $w = \log_2 n$.

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 $w = \log_2 n$.

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 $w = \log_2 n$.

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 $w = \log_2 n$.

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 $w = \log_2 n$.

4 Modelling Issues

Example 2

Algorithm 1 RepeatedSquaring(n)

```
r ← 2;
for i = 1 → n do
    r ← r^2
return r
```

4 Modelling Issues

Example 2

Algorithm 1 RepeatedSquaring(n)

$r \leftarrow 2$;
for $i = 1 \rightarrow n$ **do**
    $r \leftarrow r^2$
return r

- running time:
  - uniform model: n steps
  - logarithmic model: $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1 = \Theta(2^n)$
- space requirement:
  - uniform model: $\mathcal{O}(1)$
  - logarithmic model: $\mathcal{O}(2^n)$

Example 2

Algorithm 1 RepeatedSquaring(n)

$r \leftarrow 2$;
for $i = 1 \rightarrow n$ **do**
    $r \leftarrow r^2$
return r

- running time:
  - uniform model: n steps
  - logarithmic model: $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1 = \Theta(2^n)$
- space requirement:
  - uniform model: $\mathcal{O}(1)$
  - logarithmic model: $\mathcal{O}(2^n)$

4 Modelling Issues

Example 2

Algorithm 1 RepeatedSquaring(n)

```
1: r ← 2;
2: for i = 1 → n do
3:     r ← r^2
4: return r
```

- running time:
  - uniform model: n steps
  - logarithmic model: $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1 = \Theta(2^n)$
- space requirement:
  - uniform model: $\mathcal{O}(1)$
  - logarithmic model: $\mathcal{O}(2^n)$

4 Modelling Issues

Example 2

Algorithm 1 RepeatedSquaring(n)

```
r ← 2;
for i = 1 → n do
    r ← r^2
return r
```

- running time:
  - uniform model: n steps
  - logarithmic model: $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1 = \Theta(2^n)$
- space requirement:
  - uniform model: $\mathcal{O}(1)$
  - logarithmic model: $\mathcal{O}(2^n)$

4 Modelling Issues

Example 2

Algorithm 1 RepeatedSquaring(n)

```
r ← 2;
for i = 1 → n do
    r ← r^2
return r
```

- running time:
  - uniform model: n steps
  - logarithmic model: $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1 = \Theta(2^n)$
- space requirement:
  - uniform model: $\mathcal{O}(1)$
  - logarithmic model: $\mathcal{O}(2^n)$

4 Modelling Issues

Example 2

Algorithm 1 RepeatedSquaring(n)

```
1: r ← 2;
2: for i = 1 → n do
3:     r ← r^2
4: return r
```

- running time:
  - uniform model: n steps
  - logarithmic model: $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1 = \Theta(2^n)$
- space requirement:
  - uniform model: $\mathcal{O}(1)$
  - logarithmic model: $\mathcal{O}(2^n)$

There are different types of complexity bounds:

- best-case complexity:

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

  Usually easy to analyze, but not very meaningful.

- worst-case complexity:

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

  Usually moderately easy to analyze; sometimes too pessimistic.

- average case complexity:

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

  more general: probability measure μ

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

There are different types of complexity bounds:

- best-case complexity:

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

  Usually easy to analyze, but not very meaningful.

- worst-case complexity:

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

  Usually moderately easy to analyze; sometimes too pessimistic.

- average case complexity:

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

  more general: probability measure μ

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

$f : \mathbb{N} \to \mathbb{N}$

↑ input length

$f(n)$

There are different types of complexity bounds:

- best-case complexity:

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

  Usually easy to analyze, but not very meaningful.

- worst-case complexity:

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

  Usually moderately easy to analyze; sometimes too pessimistic.

- average case complexity:

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

  more general: probability measure μ

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

There are different types of complexity bounds:

- best-case complexity:

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

  Usually easy to analyze, but not very meaningful.

- worst-case complexity:

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

  Usually moderately easy to analyze; sometimes too pessimistic.

- average case complexity:

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

  more general: probability measure μ

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