4 Modelling Issues

What do you measure?

- Memory requirement

4 Modelling Issues

What do you measure?

- Memory requirement
- Running time

4 Modelling Issues

What do you measure?

- Memory requirement
- Running time
- Number of comparisons

4 Modelling Issues

What do you measure?

- Memory requirement
- Running time
- Number of comparisons
- Number of multiplications

4 Modelling Issues

What do you measure?

- Memory requirement
- Running time
- Number of comparisons
- Number of multiplications
- Number of hard-disc accesses

4 Modelling Issues

What do you measure?

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

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?

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.

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.).

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

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)

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

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

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), ...

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, ...

Model of Computation

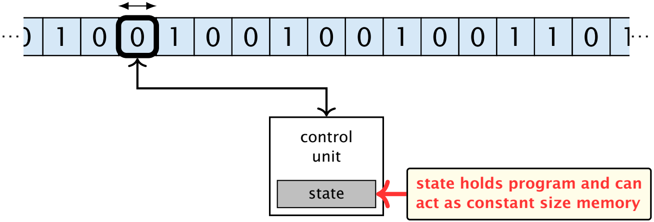
How to measure performance

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.

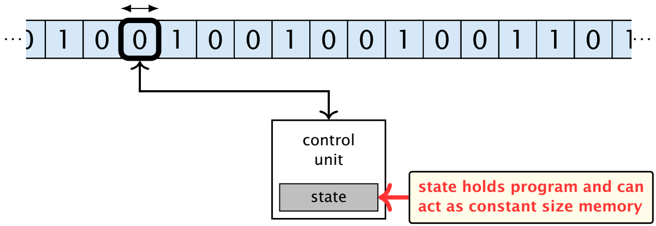
Turing Machine

- Very simple model of computation.



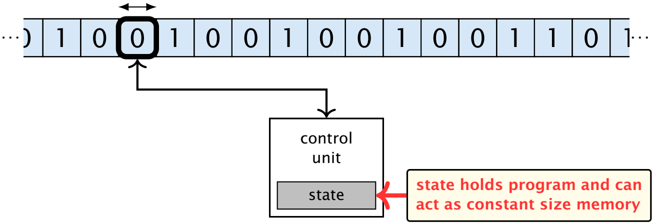
Turing Machine

- Very simple model of computation.
- Only the “current” memory location can be altered.



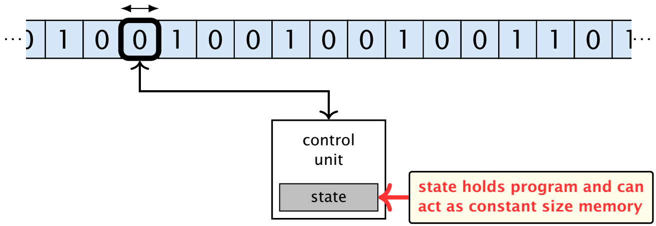
Turing Machine

- Very simple model of computation.
- Only the "current" memory location can be altered.
- Very good model for discussing computabiliy, or polynomial vs. exponential time.



Turing Machine

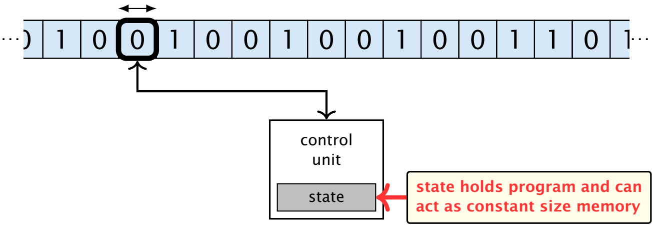
- Very simple model of computation.
- Only the “current” memory location can be altered.
- Very good model for discussing computabiliy, or polynomial vs. exponential time.
- Some simple problems like recognizing whether input is of the form xx, where x is a string, have quadratic lower bound.



Turing Machine

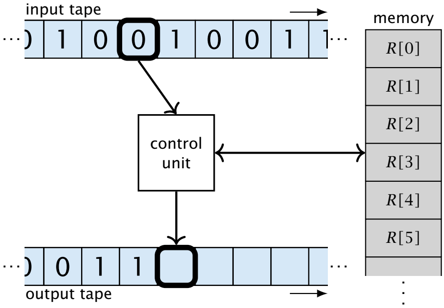
- Very simple model of computation.
- Only the “current” memory location can be altered.
- Very good model for discussing computabiliy, or polynomial vs. exponential time.
- Some simple problems like recognizing whether input is of the form xx, where x is a string, have quadratic lower bound.

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



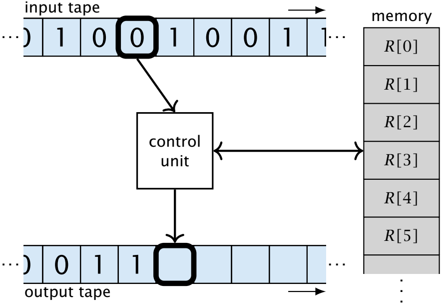
Random Access Machine (RAM)

- Input tape and output tape (sequences of zeros and ones; unbounded length).



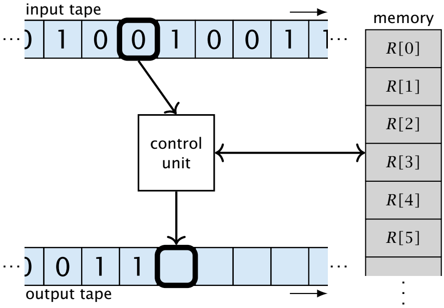
Random Access Machine (RAM)

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



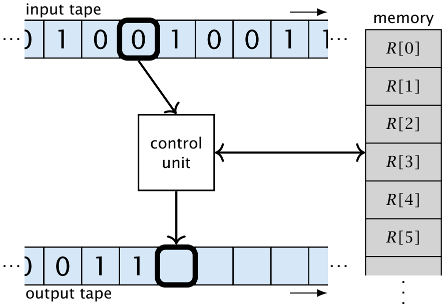
Random Access Machine (RAM)

- 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.



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 $\rightarrow R[i]$)

Random Access Machine (RAM)

Operations

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

Random Access Machine (RAM)

Operations

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

Random Access Machine (RAM)

Operations

- input operations (input tape $\rightarrow R[i]$)
  - `READ` i
- output operations ($R[i] \rightarrow$ output tape)
  - `WRITE` i

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

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
  - $R[j]$ `:=` $R[i]$

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

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

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

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

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 $R[x]$

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;

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);

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: $+$, $-$, $\times$, $/$

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.

Model of Computation

- uniform cost model
  Every operation takes time 1.
- logarithmic cost model
  The cost depends on the content of memory cells:

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;

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.

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)

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

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 (for Line 3):

4 Modelling Issues

Example 2

Algorithm 1 RepeatedSquaring(n)

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

- running time (for Line 3):
  - uniform model: n steps

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

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 (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:

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 (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: $\mathcal{O}(1)$

4 Modelling Issues

Example 2

Algorithm 1 RepeatedSquaring(n)

1: $r \leftarrow 2$;
2: **for** $i = 1 \rightarrow n$ **do**
3: $\quad r \leftarrow r^2$
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.

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.

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:

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

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

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

There are different types of complexity bounds:

- amortized complexity:
  The average cost of data structure operations over a worst case sequence of operations.

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 $|x| = n$.