Algorithms and Data Structures for Mathematicians

Lecture 1: An Introduction

Peter Kostolányi
kostolanyi at fmph and so on
Room M-258

28 September 2017

Some Not Really Formal Definitions

Computational problems:

Some Not Really Formal Definitions

Computational problems:

- Mappings $F : \mathbb{I} \to \mathbb{O}$

Some Not Really Formal Definitions

Computational problems:

- Mappings $F : \mathbb{I} \to \mathbb{O}$
- $\mathbb{I}$ is the set of inputs, $\mathbb{O}$ is the set of outputs

Some Not Really Formal Definitions

Computational problems:

- Mappings $F : \mathbb{I} \to \mathbb{O}$
- $\mathbb{I}$ is the set of inputs, $\mathbb{O}$ is the set of outputs
- Example 1: Given n in $\mathbb{N}$, find out if n is prime

Some Not Really Formal Definitions

Computational problems:

- Mappings $F : \mathbb{I} \to \mathbb{O}$
- $\mathbb{I}$ is the set of inputs, $\mathbb{O}$ is the set of outputs
- Example 1: Given n in $\mathbb{N}$, find out if n is prime
- Example 2: Given n in $\mathbb{N}$ and $a_1, \ldots, a_n$ from a totally ordered set $(S, \preceq)$, find a permutation $\varphi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $a_{\varphi(1)} \preceq \ldots \preceq a_{\varphi(n)}$ (sorting)

Some Not Really Formal Definitions

Computational problems:

- Mappings $F : \mathbb{I} \to \mathbb{O}$
- $\mathbb{I}$ is the set of inputs, $\mathbb{O}$ is the set of outputs
- Example 1: Given n in $\mathbb{N}$, find out if n is prime
- Example 2: Given n in $\mathbb{N}$ and $a_1, \ldots, a_n$ from a totally ordered set $(S, \preceq)$, find a permutation $\varphi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $a_{\varphi(1)} \preceq \ldots \preceq a_{\varphi(n)}$ (sorting)

Algorithms:

Some Not Really Formal Definitions

Computational problems:

- Mappings $F : \mathbb{I} \to \mathbb{O}$
- $\mathbb{I}$ is the set of inputs, $\mathbb{O}$ is the set of outputs
- Example 1: Given n in $\mathbb{N}$, find out if n is prime
- Example 2: Given n in $\mathbb{N}$ and $a_1, \ldots, a_n$ from a totally ordered set $(S, \preceq)$, find a permutation $\varphi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $a_{\varphi(1)} \preceq \ldots \preceq a_{\varphi(n)}$ (sorting)

Algorithms:

- Well defined and always halting sequences of elementary operations solving a given computational problem

Some Not Really Formal Definitions

Computational problems:

- Mappings $F : \mathbb{I} \to \mathbb{O}$
- $\mathbb{I}$ is the set of inputs, $\mathbb{O}$ is the set of outputs
- Example 1: Given n in $\mathbb{N}$, find out if n is prime
- Example 2: Given n in $\mathbb{N}$ and $a_1, \ldots, a_n$ from a totally ordered set $(S, \preceq)$, find a permutation $\varphi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $a_{\varphi(1)} \preceq \ldots \preceq a_{\varphi(n)}$ (sorting)

Algorithms:

- Well defined and always halting sequences of elementary operations solving a given computational problem
- Each I in $\mathbb{I}$ is transformed to $F(I)$ in $\mathbb{O}$

Some Not Really Formal Definitions

Computational problems:

- Mappings $F : \mathbb{I} \to \mathbb{O}$
- $\mathbb{I}$ is the set of inputs, $\mathbb{O}$ is the set of outputs
- Example 1: Given n in $\mathbb{N}$, find out if n is prime
- Example 2: Given n in $\mathbb{N}$ and $a_1, \ldots, a_n$ from a totally ordered set $(S, \preceq)$, find a permutation $\varphi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $a_{\varphi(1)} \preceq \ldots \preceq a_{\varphi(n)}$ (sorting)

Algorithms:

- Well defined and always halting sequences of elementary operations solving a given computational problem
- Each I in $\mathbb{I}$ is transformed to $F(I)$ in $\mathbb{O}$
- Might or might not be implemented on a computer

Some Not Really Formal Definitions

Computational problems:

- Mappings $F : \mathbb{I} \to \mathbb{O}$
- $\mathbb{I}$ is the set of inputs, $\mathbb{O}$ is the set of outputs
- Example 1: Given n in $\mathbb{N}$, find out if n is prime
- Example 2: Given n in $\mathbb{N}$ and $a_1, \ldots, a_n$ from a totally ordered set $(S, \preceq)$, find a permutation $\varphi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $a_{\varphi(1)} \preceq \ldots \preceq a_{\varphi(n)}$ (sorting)

Algorithms:

- Well defined and always halting sequences of elementary operations solving a given computational problem
- Each I in $\mathbb{I}$ is transformed to $F(I)$ in $\mathbb{O}$
- Might or might not be implemented on a computer
- We shall be particularly interested in efficient algorithms

Some Not Really Formal Definitions

Data Structures:

Some Not Really Formal Definitions

Data Structures:

- Representations of data in memory (e.g., arrays, linked lists, . . .)

Some Not Really Formal Definitions

Data Structures:

- Representations of data in memory (e.g., arrays, linked lists, . . .)
- Aim: to access and/or modify data efficiently

Some Not Really Formal Definitions

Data Structures:

- Representations of data in memory (e.g., arrays, linked lists, . . .)
- Aim: to access and/or modify data efficiently

Design and analysis of algorithms (and data structures):

Some Not Really Formal Definitions

Data Structures:

- Representations of data in memory (e.g., arrays, linked lists, . . .)
- Aim: to access and/or modify data efficiently

Design and analysis of algorithms (and data structures):

- Can make programming efficient, but is not programming

Some Not Really Formal Definitions

Data Structures:

- Representations of data in memory (e.g., arrays, linked lists, . . .)
- Aim: to access and/or modify data efficiently

Design and analysis of algorithms (and data structures):

- Can make programming efficient, but is not programming
- Uses some elementary mathematics, but is not mathematics

Some Not Really Formal Definitions

Data Structures:

- Representations of data in memory (e.g., arrays, linked lists, . . .)
- Aim: to access and/or modify data efficiently

Design and analysis of algorithms (and data structures):

- Can make programming efficient, but is not programming
- Uses some elementary mathematics, but is not mathematics
- A truly mathematical approach: computation theory

Some Not Really Formal Definitions

Data Structures:

- Representations of data in memory (e.g., arrays, linked lists, . . .)
- Aim: to access and/or modify data efficiently

Design and analysis of algorithms (and data structures):

- Can make programming efficient, but is not programming
- Uses some elementary mathematics, but is not mathematics
- A truly mathematical approach: computation theory
  - `2-MPG-218` Complexity theory (this summer)

Course Organisation

Web page for the first half of the semester (or so):

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Lectures in the second half of the semester:

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Lectures in the second half of the semester:

- Dana Pardubská (Room M-250)

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Lectures in the second half of the semester:

- Dana Pardubská (Room M-250)
- `pardubska@dcs.fmph.uniba.sk`

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Lectures in the second half of the semester:

- Dana Pardubská (Room M-250)
- `pardubska@dcs.fmph.uniba.sk`

Lectures interleaved with exercises when needed

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Lectures in the second half of the semester:

- Dana Pardubská (Room M-250)
- `pardubska@dcs.fmph.uniba.sk`

Lectures interleaved with exercises when needed

Grading:

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Lectures in the second half of the semester:

- Dana Pardubská (Room M-250)
- `pardubska@dcs.fmph.uniba.sk`

Lectures interleaved with exercises when needed

Grading:

- 100 points in total

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Lectures in the second half of the semester:

- Dana Pardubská (Room M-250)
- `pardubska@dcs.fmph.uniba.sk`

Lectures interleaved with exercises when needed

Grading:

- 100 points in total
- Mid-term exam: 40 points

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Lectures in the second half of the semester:

- Dana Pardubská (Room M-250)
- `pardubska@dcs.fmph.uniba.sk`

Lectures interleaved with exercises when needed

Grading:

- 100 points in total
- Mid-term exam: 40 points
- Final examination: 60 points

Course Organisation

Web page for the first half of the semester (or so):

- `http://www.dcs.fmph.uniba.sk/~kostolanyi/ads/`

Lectures in the second half of the semester:

- Dana Pardubská (Room M-250)
- `pardubska@dcs.fmph.uniba.sk`

Lectures interleaved with exercises when needed

Grading:

- 100 points in total
- Mid-term exam: 40 points
- Final examination: 60 points
- A: 90+, B: 80 – 89, C: 70 – 79, D: 60 – 69, E: 50 – 59, FX: 0 – 49

Suggested Textbooks

Principal Sources:

Suggested Textbooks

Principal Sources:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: *Introduction to Algorithms*, 3rd edition. Cambridge : MIT Press, 2009.

Suggested Textbooks

Principal Sources:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.:
  Introduction to Algorithms, 3rd edition.
  Cambridge : MIT Press, 2009.
- Aho, A. V., Hopcroft, J. E., Ullman, J. D.:
  The Design and Analysis of Computer Algorithms.
  Reading : Addison-Wesley, 1974.

Suggested Textbooks

Principal Sources:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.:
  Introduction to Algorithms, 3rd edition.
  Cambridge : MIT Press, 2009.
- Aho, A. V., Hopcroft, J. E., Ullman, J. D.:
  The Design and Analysis of Computer Algorithms.
  Reading : Addison-Wesley, 1974.

A Book Including Implementations (in Java):

Suggested Textbooks

Principal Sources:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.:
  Introduction to Algorithms, 3rd edition.
  Cambridge : MIT Press, 2009.
- Aho, A. V., Hopcroft, J. E., Ullman, J. D.:
  The Design and Analysis of Computer Algorithms.
  Reading : Addison-Wesley, 1974.

A Book Including Implementations (in Java):

- Sedgewick, R., Wayne, K.:
  Algorithms, 4th edition.
  Upper Saddle River : Addison-Wesley, 2011.

Suggested Textbooks

Principal Sources:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.:
  Introduction to Algorithms, 3rd edition.
  Cambridge : MIT Press, 2009.
- Aho, A. V., Hopcroft, J. E., Ullman, J. D.:
  The Design and Analysis of Computer Algorithms.
  Reading : Addison-Wesley, 1974.

A Book Including Implementations (in Java):

- Sedgewick, R., Wayne, K.:
  Algorithms, 4th edition.
  Upper Saddle River : Addison-Wesley, 2011.

A More Gentle Introduction:

Suggested Textbooks

Principal Sources:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.:
  Introduction to Algorithms, 3rd edition.
  Cambridge : MIT Press, 2009.
- Aho, A. V., Hopcroft, J. E., Ullman, J. D.:
  The Design and Analysis of Computer Algorithms.
  Reading : Addison-Wesley, 1974.

A Book Including Implementations (in Java):

- Sedgewick, R., Wayne, K.:
  Algorithms, 4th edition.
  Upper Saddle River : Addison-Wesley, 2011.

A More Gentle Introduction:

- Cormen, T. H.:
  Algorithms Unlocked.
  Cambridge : MIT Press, 2013.

First Example: Finding the Maximum

- Let $(S, \preceq)$ be a totally ordered set

First Example: Finding the Maximum

- Let $(S, \preceq)$ be a totally ordered set
- Assume that $\bot \prec x$ for all x in S

First Example: Finding the Maximum

- Let $(S, \preceq)$ be a totally ordered set
- Assume that $\bot \prec x$ for all x in S
- Given n elements of S, we want to find the greatest one

First Example: Finding the Maximum

- Let $(S, \preceq)$ be a totally ordered set
- Assume that $\bot \prec x$ for all x in S
- Given n elements of S, we want to find the greatest one

Algorithm:

First Example: Finding the Maximum

- Let $(S, \preceq)$ be a totally ordered set
- Assume that $\bot \prec x$ for all x in S
- Given n elements of S, we want to find the greatest one

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ⪰ max then
        max ← a[i];
    end
end
return max;
```

First Example: Finding the Maximum

- Let $(S, \preceq)$ be a totally ordered set
- Assume that $\bot \prec x$ for all x in S
- Given n elements of S, we want to find the greatest one

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ⪰ max then
        max ← a[i];
    end
end
return max;
```

How fast is the above algorithm?

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
|   if a[i] ⪰ max then
|   |   max ← a[i];
|   end
end
return max;
```

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ⪰ max then
        max ← a[i];
    end
end
return max;
```

- How many elementary operations on an input of a given size?

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ⪰ max then
        max ← a[i];
    end
end
return max;
```

- How many elementary operations on an input of a given size?
- Size of the input can be measured by n

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ⪰ max then
        max ← a[i];
    end
end
return max;
```

- How many elementary operations on an input of a given size?
- Size of the input can be measured by n
- Elementary operations: perhaps $x \leftarrow y$ and **if** $y \succeq x$ **then** $x \leftarrow y \ldots$

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
|   if a[i] ⪰ max then
|   |   max ← a[i];
|   end
end
return max;
```

- How many elementary operations on an input of a given size?
- Size of the input can be measured by n
- Elementary operations: perhaps $x \leftarrow y$ and **if** $y \succeq x$ **then** $x \leftarrow y \ldots$
- Exactly $n + 1$ elementary operations on each input of size n

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ≽ max then
        max ← a[i];
    end
end
return max;
```

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ≽ max then
        max ← a[i];
    end
end
return max;
```

- What about elementary "operations" $x \leftarrow y$ and $x \preceq y$?

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ⪰ max then
        max ← a[i];
    end
end
return max;
```

- What about elementary "operations" $x \leftarrow y$ and $x \preceq y$?
- Worst case: $2n + 1$ operations on input of size n

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ≽ max then
        max ← a[i];
    end
end
return max;
```

- What about elementary "operations" $x \leftarrow y$ and $x \preceq y$?
- Worst case: $2n + 1$ operations on input of size n
- Best case: $n + 2$ operations on input of size n

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ⪰ max then
        max ← a[i];
    end
end
return max;
```

- What about elementary "operations" $x \leftarrow y$ and $x \preceq y$?
- Worst case: $2n + 1$ operations on input of size n
- Best case: $n + 2$ operations on input of size n
- Or $3n + 1$ and $2n + 2$???

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ⪰ max then
        max ← a[i];
    end
end
return max;
```

- What about elementary "operations" $x \leftarrow y$ and $x \preceq y$?
- Worst case: $2n + 1$ operations on input of size n
- Best case: $n + 2$ operations on input of size n
- Or $3n + 1$ and $2n + 2$???
- Does not really matter, in each case the number is linear in n

Time Complexity of an Algorithm

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Output: $\max\{a[i] \mid i \in \{1, \ldots, n\}\}$

```
max ← ⊥;
for i ← 1 to n do
    if a[i] ⪰ max then
        max ← a[i];
    end
end
return max;
```

- What about elementary "operations" $x \leftarrow y$ and $x \preceq y$?
- Worst case: $2n + 1$ operations on input of size n
- Best case: $n + 2$ operations on input of size n
- Or $3n + 1$ and $2n + 2$???
- Does not really matter, in each case the number is linear in n
- Time complexity can only be given with respect to some underlying model (e.g., set of elementary operations)

Time Complexity of an Algorithm

Need not be the same for all inputs of size n

Time Complexity of an Algorithm

Need not be the same for all inputs of size n

- Worst-case complexity

Time Complexity of an Algorithm

Need not be the same for all inputs of size n

- Worst-case complexity
- Expected complexity (w.r.t. some probability distribution of inputs)

Time Complexity of an Algorithm

Need not be the same for all inputs of size n

- Worst-case complexity
- Expected complexity (w.r.t. some probability distribution of inputs)
- ~~Best-case complexity~~

Time Complexity of an Algorithm

Need not be the same for all inputs of size n

- Worst-case complexity
- Expected complexity (w.r.t. some probability distribution of inputs)
- ~~Best-case complexity~~

We have seen that there is an algorithm for finding a maximum in linear worst-case time

Time Complexity of an Algorithm

Need not be the same for all inputs of size n

- Worst-case complexity
- Expected complexity (w.r.t. some probability distribution of inputs)
- ~~Best-case complexity~~

We have seen that there is an algorithm for finding a maximum in linear worst-case time

- There definitely is an algorithm that is slower in worst case

Time Complexity of an Algorithm

Need not be the same for all inputs of size n

- Worst-case complexity
- Expected complexity (w.r.t. some probability distribution of inputs)
- ~~Best-case complexity~~

We have seen that there is an algorithm for finding a maximum in linear worst-case time

- There definitely is an algorithm that is slower in worst case
- And there also might be a substantially faster algorithm...

Time Complexity of an Algorithm

Need not be the same for all inputs of size n

- Worst-case complexity
- Expected complexity (w.r.t. some probability distribution of inputs)
- ~~Best-case complexity~~

We have seen that there is an algorithm for finding a maximum in linear worst-case time

- There definitely is an algorithm that is slower in worst case
- And there also might be a substantially faster algorithm...
- ...But there is no such algorithm (proof?)

Time Complexity of an Algorithm

Need not be the same for all inputs of size n

- **Worst-case complexity**
- Expected complexity (w.r.t. some probability distribution of inputs)
- ~~Best-case complexity~~

We have seen that there is an algorithm for finding a maximum in linear worst-case time

- There definitely is an algorithm that is slower in worst case
- And there also might be a substantially faster algorithm...
- ...But there is no such algorithm (proof?)

Second Example: Insertion Sort

- Let $(S, \preceq)$ be a totally ordered set

Second Example: Insertion Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given n elements of S, we wish to sort them in increasing order

Second Example: Insertion Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given n elements of S, we wish to sort them in increasing order

Algorithm:

Second Example: Insertion Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given n elements of S, we wish to sort them in increasing order

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
    key $\leftarrow a[i]$;
    $j \leftarrow i$;
    while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
        $A[j] \leftarrow A[j-1]$;
        $j \leftarrow j - 1$;
    end
    $A[j] \leftarrow$ key
end

Second Example: Insertion Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given n elements of S, we wish to sort them in increasing order

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
  key $\leftarrow a[i]$;
  $j \leftarrow i$;
  while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
    $A[j] \leftarrow A[j-1]$;
    $j \leftarrow j - 1$;
  end
  $A[j] \leftarrow$ key
end

- Worst-case time complexity?

Second Example: Insertion Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given n elements of S, we wish to sort them in increasing order

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
  key $\leftarrow a[i]$;
  $j \leftarrow i$;
  while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
    $A[j] \leftarrow A[j-1]$;
    $j \leftarrow j - 1$;
  end
  $A[j] \leftarrow$ key
end

- Worst-case time complexity?
- It will get much more complicated later

Second Example: Insertion Sort

- Let $(S, \preceq)$ be a totally ordered set
- Given n elements of S, we wish to sort them in increasing order

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
    key $\leftarrow a[i]$;
    $j \leftarrow i$;
    while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
        $A[j] \leftarrow A[j-1]$;
        $j \leftarrow j - 1$;
    end
    $A[j] \leftarrow$ key
end

- Worst-case time complexity?
- It will get much more complicated later
- Seems that we need some techniques that would help us forget about unimportant details...

Motivation for Asymptotic Analysis

Consider the following two pieces of information:

Motivation for Asymptotic Analysis

Consider the following two pieces of information:

- The time complexity of an algorithm is

$$T(n) = 3n\left(1 + \left\lfloor\sqrt{n}\right\rfloor + 9n\left\lceil\sqrt{n}\right\rceil\right) + \frac{1}{6}n\left(2n^2 + 9n + 7\right) + \\ + 11\left\lceil\log n\right\rceil(n+1)^2 - 2\left\lceil\log n\right\rceil + 42$$

Motivation for Asymptotic Analysis

Consider the following two pieces of information:

- The time complexity of an algorithm is

$$T(n) = 3n\left(1 + \left\lfloor\sqrt{n}\right\rfloor + 9n\left\lceil\sqrt{n}\right\rceil\right) + \frac{1}{6}n\left(2n^2 + 9n + 7\right) + \\ + 11\left\lceil\log n\right\rceil(n+1)^2 - 2\left\lceil\log n\right\rceil + 42$$

- The time complexity of an algorithm grows "similarly" to n^3 as n tends to ∞

Motivation for Asymptotic Analysis

Consider the following two pieces of information:

- The time complexity of an algorithm is

$$T(n) = 3n\left(1 + \left\lfloor\sqrt{n}\right\rfloor + 9n\left\lceil\sqrt{n}\right\rceil\right) + \frac{1}{6}n\left(2n^2 + 9n + 7\right) + \\ + 11\left\lceil\log n\right\rceil(n+1)^2 - 2\left\lceil\log n\right\rceil + 42$$

- The time complexity of an algorithm grows "similarly" to n^3 as n tends to ∞

Which one is more useful?

Motivation for Asymptotic Analysis

Consider the following two pieces of information:

- The time complexity of an algorithm is

$$T(n) = 3n\left(1 + \left\lfloor\sqrt{n}\right\rfloor + 9n\left\lceil\sqrt{n}\right\rceil\right) + \frac{1}{6}n\left(2n^2 + 9n + 7\right) + \\ + 11\left\lceil\log n\right\rceil(n+1)^2 - 2\left\lceil\log n\right\rceil + 42$$

- The time complexity of an algorithm grows "similarly" to n^3 as n tends to ∞

Which one is more useful?

Exact time complexity is not only hard to compute, but may also be hard to comprehend:

Motivation for Asymptotic Analysis

Consider the following two pieces of information:

- The time complexity of an algorithm is

$$T(n) = 3n\left(1 + \left\lfloor\sqrt{n}\right\rfloor + 9n\left\lceil\sqrt{n}\right\rceil\right) + \frac{1}{6}n\left(2n^2 + 9n + 7\right) + \\ + 11\left\lceil\log n\right\rceil(n+1)^2 - 2\left\lceil\log n\right\rceil + 42$$

- The time complexity of an algorithm grows "similarly" to n^3 as n tends to ∞

Which one is more useful?

Exact time complexity is not only hard to compute, but may also be hard to comprehend:

- Solution: asymptotic analysis

Motivation for Asymptotic Analysis

Consider the following two pieces of information:

- The time complexity of an algorithm is

$$T(n) = 3n\left(1 + \left\lfloor\sqrt{n}\right\rfloor + 9n\left\lceil\sqrt{n}\right\rceil\right) + \frac{1}{6}n\left(2n^2 + 9n + 7\right) + \\ + 11\left\lceil\log n\right\rceil(n+1)^2 - 2\left\lceil\log n\right\rceil + 42$$

- The time complexity of an algorithm grows "similarly" to n^3 as n tends to ∞

Which one is more useful?

Exact time complexity is not only hard to compute, but may also be hard to comprehend:

- Solution: asymptotic analysis
- We shall be primarily interested in time complexity for large inputs

Motivation for Asymptotic Analysis

Consider the following two pieces of information:

- The time complexity of an algorithm is

$$T(n) = 3n\left(1 + \left\lfloor\sqrt{n}\right\rfloor + 9n\left\lceil\sqrt{n}\right\rceil\right) + \frac{1}{6}n\left(2n^2 + 9n + 7\right) + \\ + 11\left\lceil\log n\right\rceil(n+1)^2 - 2\left\lceil\log n\right\rceil + 42$$

- The time complexity of an algorithm grows "similarly" to n^3 as n tends to ∞

Which one is more useful?

Exact time complexity is not only hard to compute, but may also be hard to comprehend:

- Solution: asymptotic analysis
- We shall be primarily interested in time complexity for large inputs
- That is, when $n \to \infty$

How Large is This Number? (Think of Money)

42808518994895608486 91

And How Large is This Number? (Think of Money)

84795551818733482928358989704011965523586391960169353123841499275161716541614130248079686518893062743528028270661354798029493263073584985095562975639018998806567092693677608011234478419587070503835005599718728588150686533243684009181797426171883222991245962132902198193449147350817134122866534527139324266014275038885469315531344270843365472877851040028341343446812975361588038115962323696276213633010227723117346742793809486832344936918539522695019005474402586729448774658329488313043282804390925188410810300110289559989160868665250433758583040150144399344168406565330785174160961264728256705619645503580555958532651067869506317081480329379589924149250096656021238118034770836265089287436131069459108907243619617600703335393461805670822339941641799267514128970212804731682385052490576588699315287877053377030140307710569671543281014266131997196768761443229245013195360210771335676036158397648726277623505349100091556495121531765813088806487142102519821442076626922948555738959708550893125767319559649460468338138640046315396268687600039129751982852028462608855212630469177757531610682716389540632435940123841033387630698907593474195191184795551818733482928358989704011965523586391960169353123841499275161716541614130248079686518893062743528028270661354798029493263073584985095562975639018998806567092693677608011234478419587070503835005599718728588150686533243684009181797426171883222991245962132902198193449147350817134122866534527139324266014275038885469315531344270843365472877851040028341343446812975361588038115962323696276213633010227723117346742793809486832344936918539522695019005474402586729448774658329488313043282804390925188410810300110289559989160868665250433758583040150144399344168406565330785174160961264728256705619645503580555958532651067869506317081480329379589924149250096656021238118034770836265089287436131069459108907243619617600703335393461805670822339941641799267514128970212804731682385052490576588699315287877053377030140307710569671543281014266131997196768761443229245013195360210771335676036158397648726277623505349100091556495121531765813088806487142102519821442076626922948555738959708550893125767319559649460468338138640046315396268687600039129751982852028462608855212630469177757531610682716389540632435940123841033387630698907593474195191184

Asymptotic Analysis

- Number of digits $\rightsquigarrow$ error up to $10\times$

Asymptotic Analysis

- Number of digits $\rightsquigarrow$ error up to $10\times$
- Number of slides $\rightsquigarrow$ error 10^6 makes little difference

Asymptotic Analysis

- Number of digits $\rightsquigarrow$ error up to $10\times$
- Number of slides $\rightsquigarrow$ error 10^6 makes little difference
- Each constant factor $c > 0$ seems to be a reasonable error for large enough n

Asymptotic Analysis

- Number of digits $\rightsquigarrow$ error up to $10\times$
- Number of slides $\rightsquigarrow$ error 10^6 makes little difference
- Each constant factor $c > 0$ seems to be a reasonable error for large enough n
- We shall say that $f : \mathbb{N} \to \mathbb{N}$ grows "similarly" to $g : \mathbb{N} \to \mathbb{N}$ if there is such constant factor c

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ *be functions. Then we shall write:*

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ be functions. Then we shall write:

(*i*) $f(n) = O(g(n))$ *if* $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$.

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ *be functions. Then we shall write:*

(*i*) $f(n) = O(g(n))$ *if* $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$.

(*ii*) $f(n) = \Omega(g(n))$ *if* $g(n) = O(f(n))$.

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ *be functions. Then we shall write:*

(*i*) $f(n) = O(g(n))$ *if* $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$.

(*ii*) $f(n) = \Omega(g(n))$ *if* $g(n) = O(f(n))$.

(*iii*) $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $g(n) = O(f(n))$.

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ *be functions. Then we shall write:*

(*i*) $f(n) = O(g(n))$ *if* $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$.

(*ii*) $f(n) = \Omega(g(n))$ *if* $g(n) = O(f(n))$.

(*iii*) $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $g(n) = O(f(n))$.

Some stronger notation:

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ *be functions. Then we shall write:*

(*i*) $f(n) = O(g(n))$ *if* $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$.

(*ii*) $f(n) = \Omega(g(n))$ *if* $g(n) = O(f(n))$.

(*iii*) $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $g(n) = O(f(n))$.

Some stronger notation:

(*iv*) $f(n) = o(g(n))$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ *be functions. Then we shall write:*

(*i*) $f(n) = O(g(n))$ *if* $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$.

(*ii*) $f(n) = \Omega(g(n))$ *if* $g(n) = O(f(n))$.

(*iii*) $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $g(n) = O(f(n))$.

Some stronger notation:

(*iv*) $f(n) = o(g(n))$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

(*v*) $f(n) = \omega(g(n))$ *if* $g(n) = o(f(n))$.

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ *be functions. Then we shall write:*

(*i*) $f(n) = O(g(n))$ *if* $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$.

(*ii*) $f(n) = \Omega(g(n))$ *if* $g(n) = O(f(n))$.

(*iii*) $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $g(n) = O(f(n))$.

Some stronger notation:

(*iv*) $f(n) = o(g(n))$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

(*v*) $f(n) = \omega(g(n))$ *if* $g(n) = o(f(n))$.

(*vi*) $f(n) \sim g(n)$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$.

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ *be functions. Then we shall write:*

(*i*) $f(n) = O(g(n))$ *if* $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$.

(*ii*) $f(n) = \Omega(g(n))$ *if* $g(n) = O(f(n))$.

(*iii*) $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $g(n) = O(f(n))$.

Some stronger notation:

(*iv*) $f(n) = o(g(n))$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

(*v*) $f(n) = \omega(g(n))$ *if* $g(n) = o(f(n))$.

(*vi*) $f(n) \sim g(n)$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$.

Asymptotic Analysis

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ *be functions. Then we shall write:*

(*i*) $f(n) = O(g(n))$ *if* $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$.

(*ii*) $f(n) = \Omega(g(n))$ *if* $g(n) = O(f(n))$.

(*iii*) $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $g(n) = O(f(n))$.

Some stronger notation:

(*iv*) $f(n) = o(g(n))$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

(*v*) $f(n) = \omega(g(n))$ *if* $g(n) = o(f(n))$.

(*vi*) $f(n) \sim g(n)$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$.

Asymptotic Analysis

Example

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

In calculus, you used to write:

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

In calculus, you used to write:

- $f(x) = 1 + x + x^2 + O(x^3)$, or so

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

In calculus, you used to write:

- $f(x) = 1 + x + x^2 + O(x^3)$, or so
- Thus x^3 is negligible compared to x^2, we have $x^4 = O(x^3)$, etc.

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

In calculus, you used to write:

- $f(x) = 1 + x + x^2 + O(x^3)$, or so
- Thus x^3 is negligible compared to x^2, we have $x^4 = O(x^3)$, etc.

For us:

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

In calculus, you used to write:

- $f(x) = 1 + x + x^2 + O(x^3)$, or so
- Thus x^3 is negligible compared to x^2, we have $x^4 = O(x^3)$, etc.

For us:

- n^2 is negligible compared to n^3, we have $n^3 = O(n^4)$, etc.

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

In calculus, you used to write:

- $f(x) = 1 + x + x^2 + O(x^3)$, or so
- Thus x^3 is negligible compared to x^2, we have $x^4 = O(x^3)$, etc.

For us:

- n^2 is negligible compared to n^3, we have $n^3 = O(n^4)$, etc.
- Reason: $n \to \infty$ instead of $x \to 0$

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

In calculus, you used to write:

- $f(x) = 1 + x + x^2 + O(x^3)$, or so
- Thus x^3 is negligible compared to x^2, we have $x^4 = O(x^3)$, etc.

For us:

- n^2 is negligible compared to n^3, we have $n^3 = O(n^4)$, etc.
- Reason: $n \to \infty$ instead of $x \to 0$

Two important properties of Θ-notation:

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

In calculus, you used to write:

- $f(x) = 1 + x + x^2 + O(x^3)$, or so
- Thus x^3 is negligible compared to x^2, we have $x^4 = O(x^3)$, etc.

For us:

- n^2 is negligible compared to n^3, we have $n^3 = O(n^4)$, etc.
- Reason: $n \to \infty$ instead of $x \to 0$

Two important properties of Θ-notation:

- If $f_1(n) = \Theta(f_2(n))$ and $g_1(n) = \Theta(g_2(n))$, then $f_1(n) + g_1(n) = \Theta(f_2(n) + g_2(n))$

Asymptotic Analysis

Example

- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^3)$ *and* $f(n) = \Theta(n^3)$
- *If* $f(n) = 2n^3 + n^2 + 10$, *then* $f(n) = O(n^4)$, *but not* $f(n) = \Theta(n^4)$

In calculus, you used to write:

- $f(x) = 1 + x + x^2 + O(x^3)$, or so
- Thus x^3 is negligible compared to x^2, we have $x^4 = O(x^3)$, etc.

For us:

- n^2 is negligible compared to n^3, we have $n^3 = O(n^4)$, etc.
- Reason: $n \to \infty$ instead of $x \to 0$

Two important properties of Θ-notation:

- If $f_1(n) = \Theta(f_2(n))$ and $g_1(n) = \Theta(g_2(n))$, then $f_1(n) + g_1(n) = \Theta(f_2(n) + g_2(n))$
- If $f_1(n) = \Theta(f_2(n))$ and $g_1(n) = \Theta(g_2(n))$, then $f_1(n) \cdot g_1(n) = \Theta(f_2(n) \cdot g_2(n))$

Insertion Sort: Worst-Case Time Complexity

Algorithm:

Insertion Sort: Worst-Case Time Complexity

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
  key $\leftarrow a[i]$;
  $j \leftarrow i$;
  while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
    $A[j] \leftarrow A[j-1]$;
    $j \leftarrow j - 1$;
  end
  $A[j] \leftarrow$ key
end

Insertion Sort: Worst-Case Time Complexity

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
    key $\leftarrow a[i]$;
    $j \leftarrow i$;
    while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
        $A[j] \leftarrow A[j-1]$;
        $j \leftarrow j - 1$;
    end
    $A[j] \leftarrow$ key
end

- Let $T(n)$ be the worst-case time complexity of insertion sort

Insertion Sort: Worst-Case Time Complexity

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
    key $\leftarrow a[i]$;
    $j \leftarrow i$;
    while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
        $A[j] \leftarrow A[j-1]$;
        $j \leftarrow j - 1$;
    end
    $A[j] \leftarrow$ key
end

- Let $T(n)$ be the worst-case time complexity of insertion sort
- The **for** loop executes $\leq n$ times on each input

Insertion Sort: Worst-Case Time Complexity

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
  key $\leftarrow a[i]$;
  $j \leftarrow i$;
  while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
    $A[j] \leftarrow A[j-1]$;
    $j \leftarrow j - 1$;
  end
  $A[j] \leftarrow$ key
end

- Let $T(n)$ be the worst-case time complexity of insertion sort
- The **for** loop executes $\leq n$ times on each input
- The **while** loop executes $\leq n$ times for each i

Insertion Sort: Worst-Case Time Complexity

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
    key $\leftarrow a[i]$;
    $j \leftarrow i$;
    while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
        $A[j] \leftarrow A[j-1]$;
        $j \leftarrow j - 1$;
    end
    $A[j] \leftarrow$ key
end

- Let $T(n)$ be the worst-case time complexity of insertion sort
- The **for** loop executes $\leq n$ times on each input
- The **while** loop executes $\leq n$ times for each i
- Hence, $T(n) = O(n^2)$

Insertion Sort: Worst-Case Time Complexity

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
  key $\leftarrow a[i]$;
  $j \leftarrow i$;
  while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
    $A[j] \leftarrow A[j-1]$;
    $j \leftarrow j - 1$;
  end
  $A[j] \leftarrow$ key
end

- Let $T(n)$ be the worst-case time complexity of insertion sort
- The **for** loop executes $\leq n$ times on each input
- The **while** loop executes $\leq n$ times for each i
- Hence, $T(n) = O(n^2)$
- Considering inputs sorted in decreasing order: $T(n) = \Omega(n^2)$

Insertion Sort: Worst-Case Time Complexity

Algorithm:

Input : Integer $n \geq 0$, array $a = \langle a[1] \ldots, a[n] \rangle$ of elements of $(S, \preceq)$
Behaviour: Sorts a in increasing order

for $i \leftarrow 2$ **to** n **do**
  key $\leftarrow a[i]$;
  $j \leftarrow i$;
  while $j \geq 2$ *and* $a[j-1] \succ$ key **do**
    $A[j] \leftarrow A[j-1]$;
    $j \leftarrow j - 1$;
  end
  $A[j] \leftarrow$ key
end

- Let $T(n)$ be the worst-case time complexity of insertion sort
- The **for** loop executes $\leq n$ times on each input
- The **while** loop executes $\leq n$ times for each i
- Hence, $T(n) = O(n^2)$
- Considering inputs sorted in decreasing order: $T(n) = \Omega(n^2)$
- $T(n) = \Theta(n^2)$

When Model Matters. . .

Algorithm:

When Model Matters. . .

Algorithm:

Input : Integer $n \geq 0$
Output: n^n

$k \leftarrow 1$;
for $i \leftarrow 1$ **to** n **do**
  $k \leftarrow k \cdot n$;
end
return k;

When Model Matters...

Algorithm:

Input : Integer $n \geq 0$
Output: n^n

$k \leftarrow 1$;
for $i \leftarrow 1$ **to** n **do**
  $k \leftarrow k \cdot n$;
end
return k;

- Worst-case time complexity: $\Theta(n)$?

When Model Matters...

Algorithm:

Input : Integer $n \geq 0$
Output: n^n

$k \leftarrow 1$;
for $i \leftarrow 1$ **to** n **do**
  $k \leftarrow k \cdot n$;
end
return k;

- Worst-case time complexity: $\Theta(n)$?
- $n^n = 2^{n \log n}$ – we need at least $n \log n$ bits to store n^n

When Model Matters...

Algorithm:

Input : Integer $n \geq 0$
Output: n^n

$k \leftarrow 1$;
for $i \leftarrow 1$ **to** n **do**
  $k \leftarrow k \cdot n$;
end
return k;

- Worst-case time complexity: $\Theta(n)$?
- $n^n = 2^{n \log n}$ – we need at least $n \log n$ bits to store n^n
- At least $n \log n$ bit operations, and this is not $\Theta(n)$

When Model Matters...

Algorithm:

Input : Integer $n \geq 0$
Output: n^n

$k \leftarrow 1$;
for $i \leftarrow 1$ **to** n **do**
  $k \leftarrow k \cdot n$;
end
return k;

- Worst-case time complexity: $\Theta(n)$?
- $n^n = 2^{n \log n}$ – we need at least $n \log n$ bits to store n^n
- At least $n \log n$ bit operations, and this is not $\Theta(n)$
- Even worse if we take $\log n$ as the size of the input