CSE 431/531: Algorithm Analysis and Design (Fall 2024)

Introduction II: Algorithm and Asymptotic Analysis

Lecturer: Kelin Luo

Department of Computer Science and Engineering
University at Buffalo

Outline

1. Introduction
   - What is an Algorithm?
   - More Computation Problems

Outline

What is an Algorithm?

- Donald Knuth: An algorithm is a finite, definite effective procedure, with some input and some output.

What is an Algorithm?

- Donald Knuth: An algorithm is a finite, definite effective procedure, with some input and some output.
- Computational problem: specifies the input/output relationship.
- An algorithm solves a computational problem if it produces the correct output for any given input.

What is an Algorithm?

- Computational problem

1. What is the shortest route from classroom Nsc 225 to classroom Knox 110?

What is an Algorithm?

- Computational problem

1. What is the shortest route from classroom Nsc 225 to classroom Knox 110?
2. Which restaurant has the highest rating?

What is an Algorithm?

- Computational problem

1. What is the shortest route from classroom Nsc 225 to classroom Knox 110?
2. Which restaurant has the highest rating?
3. True or False? 2^{10} is greater than 5^2

What is an Algorithm?

- Computational problem

1. What is the shortest route from classroom Nsc 225 to classroom Knox 110?
2. Which restaurant has the highest rating?
3. True or False? 2^{10} is greater than 5^2

- Computational problem: specifies the input/output relationship.

What is an Algorithm?

- Computational problem

1. What is the shortest route from classroom Nsc 225 to classroom Knox 110?
2. Which restaurant has the highest rating?
3. True or False? 2^{10} is greater than 5^2

- Computational problem: specifies the input/output relationship.
- An algorithm solves a computational problem if it produces the correct output for any given input.

Examples

Greatest Common Divisor

Input: two integers $a, b > 0$

Output: the greatest common divisor of a and b

Examples

Greatest Common Divisor

Input: two integers $a, b > 0$

Output: the greatest common divisor of a and b

Example:

- Input: 210, 270
- Output: 30

Examples

Greatest Common Divisor

Input: two integers $a, b > 0$

Output: the greatest common divisor of a and b

Example:

- Input: 210, 270
- Output: 30

- Algorithm: Euclidean algorithm

Examples

Greatest Common Divisor

Input: two integers $a, b > 0$

Output: the greatest common divisor of a and b

Example:

- Input: 210, 270
- Output: 30

- Algorithm: Euclidean algorithm
- $\gcd(270, 210) = \gcd(210, 270 \bmod 210) = \gcd(210, 60)$

Examples

Greatest Common Divisor

Input: two integers $a, b > 0$

Output: the greatest common divisor of a and b

Example:

- Input: 210, 270
- Output: 30

- Algorithm: Euclidean algorithm
- $\gcd(270, 210) = \gcd(210, 270 \bmod 210) = \gcd(210, 60)$
- $(270, 210) \to (210, 60) \to (60, 30) \to (30, 0)$

Algorithm = Computer Program?

- Algorithm: "abstract", can be specified using computer program, English, pseudo-codes or flow charts.
- Computer program: "concrete", implementation of algorithm, using a particular programming language

Pseudo-Code

Pseudo-Code:

Euclidean(a, b)

1: **while** $b > 0$ **do**
2: $(a, b) \leftarrow (b, a \bmod b)$
3: **return** a

Python program:

- def euclidean(a: int, b: int):
-     c = 0
-     while b > 0:
-         c = b
-         b = a % b
-         a = c
-     return a

Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)

Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage

Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g, GUI)
  - ...

Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g, GUI)
  - ...
- Why is it important to study the running time (efficiency) of an algorithm?

Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g, GUI)
  - ...
- Why is it important to study the running time (efficiency) of an algorithm?

1. feasible vs. infeasible

Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g, GUI)
  - ...
- Why is it important to study the running time (efficiency) of an algorithm?

1. feasible vs. infeasible
2. efficient algorithms: less engineering tricks needed, can use languages aiming for easy programming (e.g, python)

Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g, GUI)
  - ...
- Why is it important to study the running time (efficiency) of an algorithm?

1. feasible vs. infeasible
2. efficient algorithms: less engineering tricks needed, can use languages aiming for easy programming (e.g, python)
3. fundamental

Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g, GUI)
  - ...
- Why is it important to study the running time (efficiency) of an algorithm?

1. feasible vs. infeasible
2. efficient algorithms: less engineering tricks needed, can use languages aiming for easy programming (e.g, python)
3. fundamental
4. it is fun!

Outline

1. Introduction
   - What is an Algorithm?
   - More Computation Problems

Examples

Sorting

Input: sequence of n numbers $(a_1, a_2, \cdots, a_n)$

Output: a permutation $(a'_1, a'_2, \cdots, a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Examples

Sorting

Input: sequence of n numbers $(a_1, a_2, \cdots, a_n)$

Output: a permutation $(a'_1, a'_2, \cdots, a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

Examples

Sorting

Input: sequence of n numbers $(a_1, a_2, \cdots, a_n)$

Output: a permutation $(a'_1, a'_2, \cdots, a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

- Algorithms: insertion sort, merge sort, quicksort, ...

Insertion-Sort

- At the end of j-th iteration, the first j numbers are sorted.

iteration 1: $53, 12, 35, 21, 59, 15$

iteration 2: $12, 53, 35, 21, 59, 15$

iteration 3: $12, 35, 53, 21, 59, 15$

iteration 4: $12, 21, 35, 53, 59, 15$

iteration 5: $12, 21, 35, 53, 59, 15$

iteration 6: $12, 15, 21, 35, 53, 59$

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

for $j \leftarrow 2$ to n **do**
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ **do**
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

```
for j ← 2 to n do
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key do
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← key
```

- $j = 6$
- $key = 15$

12	21	35	53	59	15
				$\uparrow$ i	

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

```
for j ← 2 to n do
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key do
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← key
```

- $j = 6$
- $key = 15$

12	21	35	53	59	59
				$\uparrow$	
				i	

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

```
for j ← 2 to n do
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key do
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← key
```

- $j = 6$
- $key = 15$

$$\begin{array}{cccccc} 12 & 21 & 35 & 53 & 59 & 59 \\ & & & \uparrow & & \\ & & & i & & \end{array}$$

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

for $j \leftarrow 2$ to n **do**
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ **do**
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$

- $j = 6$
- $key = 15$

$$\begin{array}{cccccc} 12 & 21 & 35 & 53 & \color{red}{53} & 59 \\ & & & \uparrow & & \\ & & & i & & \end{array}$$

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

1: **for** $j \leftarrow 2$ to n **do**
2: $\quad key \leftarrow A[j]$
3: $\quad i \leftarrow j - 1$
4: $\quad$ **while** $i > 0$ and $A[i] > key$ **do**
5: $\qquad A[i+1] \leftarrow A[i]$
6: $\qquad i \leftarrow i - 1$
7: $\quad A[i+1] \leftarrow key$

- $j = 6$
- $key = 15$

12	21	35	53	53	59
		$\uparrow$			
		i			

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

```
for j ← 2 to n do
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key do
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← key
```

- $j = 6$
- $key = 15$

12	21	35	35	53	59
		$\uparrow$			
		i			

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

for $j \leftarrow 2$ to n **do**
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ **do**
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$

- $j = 6$
- $key = 15$

12	21	35	35	53	59
	$\uparrow$				
	i				

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

1: **for** $j \leftarrow 2$ to n **do**
2: $\quad key \leftarrow A[j]$
3: $\quad i \leftarrow j - 1$
4: $\quad$ **while** $i > 0$ and $A[i] > key$ **do**
5: $\quad\quad A[i+1] \leftarrow A[i]$
6: $\quad\quad i \leftarrow i - 1$
7: $\quad A[i+1] \leftarrow key$

- $j = 6$
- $key = 15$

12	21	21	35	53	59
	$\uparrow$				
	i				

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

> **for** $j \leftarrow 2$ to n **do**
> $key \leftarrow A[j]$
> $i \leftarrow j - 1$
>   **while** $i > 0$ and $A[i] > key$ **do**
> $A[i+1] \leftarrow A[i]$
> $i \leftarrow i - 1$
> $A[i+1] \leftarrow key$

- $j = 6$
- $key = 15$

12	21	21	35	53	59
$\uparrow$					
i					

Example:

- Input: $53, 12, 35, 21, 59, 15$
- Output: $12, 15, 21, 35, 53, 59$

insertion-sort(A, n)

```
for j ← 2 to n do
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key do
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← key
```

- $j = 6$
- $key = 15$

12 15 21 35 53 59

$\uparrow$

i

Analysis of Insertion Sort

- Correctness
- Running time

Correctness of Insertion Sort

- Invariant: after iteration j of outer loop, $A[1..j]$ is the sorted array for the original $A[1..j]$.

$$\text{after } j = 1 : {\color{red}53}, 12, 35, 21, 59, 15$$

$$\text{after } j = 2 : {\color{red}12, 53}, 35, 21, 59, 15$$

$$\text{after } j = 3 : {\color{red}12, 35, 53}, 21, 59, 15$$

$$\text{after } j = 4 : {\color{red}12, 21, 35, 53}, 59, 15$$

$$\text{after } j = 5 : {\color{red}12, 21, 35, 53, 59}, 15$$

$$\text{after } j = 6 : {\color{red}12, 15, 21, 35, 53, 59}$$

Analyzing Running Time of Insertion Sort

- Q1: what is the size of input?

Analyzing Running Time of Insertion Sort

- Q1: what is the size of input?
- A1: Running time as the function of size

Analyzing Running Time of Insertion Sort

- Q1: what is the size of input?
- A1: Running time as the function of size
- possible definition of size :
  - Sorting problem: # integers,
  - Greatest common divisor: total length of two integers
  - Shortest path in a graph: # edges in graph

Analyzing Running Time of Insertion Sort

- Q1: what is the size of input?
- A1: Running time as the function of size
- possible definition of size :
  - Sorting problem: # integers,
  - Greatest common divisor: total length of two integers
  - Shortest path in a graph: # edges in graph
- Q2: Which input?
  - For the insertion sort algorithm: if input array is already sorted in ascending order, then algorithm runs much faster than when it is sorted in descending order.

Analyzing Running Time of Insertion Sort

- Q1: what is the size of input?
- A1: Running time as the function of size
- possible definition of size :
  - Sorting problem: # integers,
  - Greatest common divisor: total length of two integers
  - Shortest path in a graph: # edges in graph
- Q2: Which input?
  - For the insertion sort algorithm: if input array is already sorted in ascending order, then algorithm runs much faster than when it is sorted in descending order.
- A2: Worst-case analysis:
  - Running time for size n = worst running time over all possible arrays of length n

Analyzing Running Time of Insertion Sort

- Q3: How fast is the computer?
- Q4: Programming language?

Analyzing Running Time of Insertion Sort

- Q3: How fast is the computer?
- Q4: Programming language?
- A: They do not matter!

Analyzing Running Time of Insertion Sort

- Q3: How fast is the computer?
- Q4: Programming language?
- A: They do not matter!

Important idea: asymptotic analysis

- Focus on growth of running-time as a function, not any particular value.

Exercise

- Ex 1: What is the worst-case runtime for the Insertion Sort algorithm when the input sequence A is already sorted?
  - A is sorted in ascending order
  - A is sorted in descending order
- Ex 2: What is the worst-case runtime for the Insertion Sort algorithm with any sequence input A?