

# Algorithms

## 3.1 Algorithms

---

# Algorithms

## Definition

An **algorithm** is a finite sequence of precise instructions for performing a computation or for solving a problem.

- My undergrad advisor, Dr. Basor: “A recipe”
- al-Khwārizmī (780-850) was a Persian mathematician, astronomer, geographer and a scholar in the House of Wisdom in Baghdad. He was the most widely read mathematician in Europe in the late Middle Ages.



A statue of Al-Khwarizmi in Uzbekistan.

# Finding the Maximum Element in a Sequence

## Max Element

**Input:** A finite sequence of integers.

**Goal:** Return the maximum value in the sequence.

---

$\text{max\_element}(a_1, a_2, \dots, a_n)$

---

**Input:** A sequence of integers:  $a_1, a_2, \dots, a_n$

**Output:** The maximum value in the sequence.

- 1:  $max = a_1$
  - 2: **for**  $i = 2$  to  $n$  **do**
  - 3:     **if**  $max < a_i$  **then**
  - 4:          $max = a_i$
  - 5: **return**  $max$
-

# Properties of Algorithms

## Properties

- *Input*: An algorithm has input values from a specified set.
- *Output*: From each set of input values an algorithm produces output values from a specified set.
- *Definiteness*: The steps of an algorithm must be defined precisely.
- *Correctness*: An algorithm should produce the correct output values for each set of input values.

# Properties of Algorithms

## Properties (continued)

- *Finiteness*: An algorithm should produce the desired output after a finite (but perhaps large) number of steps for any input in the set.
- *Effectiveness*: It must be possible to perform each step of an algorithm exactly and in a finite amount of time.
- *Generality*: The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

Does `max_element` satisfy all of these properties?

## max\_element is Correct

### Lemma

*Suppose  $\text{max\_element}(a_1, a_2, \dots, a_n)$  returns  $x$ , then  $x \geq a_i$  for  $\forall i$ .*

Proof (on board)

# Search

## Search

**Input:** An integer to search for,  $x$ , and a finite sequence of integers.

**Goal:** Either the position of  $x$  in the sequence or a statement that  $x$  is not in the sequence.

# Linear Search

---

LinearSearch( $x, a_1, a_2, \dots, a_n$ )

---

**Input:** An integer to search for,  $x$ , and a sequence of integers:  $a_1, a_2, \dots, a_n$

**Output:** Either the position of  $x$  in the sequence or a statement that  $x$  is not in the sequence.

```
1:  $i = 1$ 
2: while  $i \leq n$  and  $x \neq a_i$  do
3:    $i = i + 1$ 
4: if  $i \leq n$  then
5:    $location = i$ 
6: else
7:    $location = 0$ 
8: return  $location$ 
```

---

## LinearSearch is Correct

### Lemma

*Suppose  $x \in (a_1, a_2, \dots, a_n)$  and  $a_i$  is the first element of the sequence for which  $x = a_i$ , then  $\text{LinearSearch}(x, a_1, a_2, \dots, a_n)$  returns  $i$ .*

### Lemma

*If  $x \neq a_i$  for all  $a_i \in (a_1, a_2, \dots, a_n)$ , then  $\text{LinearSearch}(x, a_1, a_2, \dots, a_n)$  returns 0.*

# Binary Search

---

BinarySearch( $x, a_1, a_2, \dots, a_n$ )

---

**Input:** An integer to search for,  $x$ , and a sequence of increasing integers:

$a_1, a_2, \dots, a_n$

**Output:** Either the position of  $x$  in the sequence or a statement that  $x$  is not in the sequence.

```
1:  $i = 1, j = n$ 
2: while  $i < j$  do
3:    $m = \lfloor \frac{i+j}{2} \rfloor$ 
4:   if  $x > a_m$  then
5:      $i = m + 1$ 
6:   else
7:      $j = m$ 
8:   if  $x = a_i$  then
9:      $location = i$ 
10: else
11:    $location = 0$ 
12: return  $location$ 
```

# BinarySearch is Correct

## Lemma

*Suppose  $x \in (a_1, a_2, \dots, a_n)$ , then  $\text{BinarySearch}(x, a_1, a_2, \dots, a_n)$  returns some  $i$  such that  $x = a_i$ .*

## Lemma

*If  $x \neq a_i$  for all  $a_i \in (a_1, a_2, \dots, a_n)$ , then  $\text{BinarySearch}(x, a_1, a_2, \dots, a_n)$  returns 0.*

# Sorting

## Sort

**Input:** A finite sequence of integers.

**Goal:** A sequence with the same elements as the input sequence, but sorted in increasing order.

# Bubble Sort

---

BubbleSort( $a_1, \dots, a_n$ )

---

**Input:** A sequence of integers:  $a_1, \dots, a_n$

**Output:** A sequence with the same elements as the input sequence, but sorted in increasing order.

- 1: **for**  $i = 1$  **to**  $n - 1$  **do**
  - 2:     **for**  $j = 1$  **to**  $n - i$  **do**
  - 3:         **if**  $a_j > a_{j+1}$  **then**
  - 4:             interchange  $a_j$  and  $a_{j+1}$
-

# BubbleSort is Correct

## Lemma

*When BubbleSort( $a_1, \dots, a_n$ ) has completed,  $a_i \leq a_{i+1}$  for all  $1 \leq i \leq n - 1$*

# Insertion Sort

---

InsertionSort( $a_1, \dots, a_n$ )

---

**Input:** A sequence of integers:  $a_1, \dots, a_n$

**Output:** A sequence with the same elements as the input sequence, but sorted in increasing order.

```
1: for  $j = 2$  to  $n$  do
2:    $i = 1$ 
3:   while  $a_j > a_i$  do
4:      $i = i + 1$ 
5:    $m = a_j$ 
6:   for  $k = 0$  to  $j - i - 1$  do
7:      $a_{j-k} = a_{j-k-1}$ 
8:    $a_i = m$ 
```

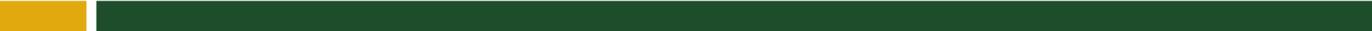
---

# InsertionSort is Correct

## Lemma

*When InsertionSort( $a_1, \dots, a_n$ ) has completed,  $a_i \leq a_{i+1}$  for all  $1 \leq i \leq n - 1$*

## 3.2 The Growth of Functions



# The Growth of Functions

Suppose we have two algorithms to solve the same problem.

- The first algorithm requires  $2n^2$  operations to solve the problem on an input of size  $n$ .
- The second algorithm requires  $57n + 92$  operations to solve the problem on an input of size  $n$ .

Which algorithm should you choose?

- When the input is small
- When the input is large

# Big- $O$ Notation

## Definition

Let  $f$  and  $g$  be functions from the set of integers (or real numbers) to the set of real numbers.

We say that  $f(x)$  is  $O(g(x))$  if there are constants  $C$  and  $k$  such that:

$$|f(x)| \leq C|g(x)|$$

whenever  $x > k$ .

Intuitively,  $f(x) = O(g(x))$  if  $f$  grows slower than some fixed multiple of  $g(x)$  as  $x$  grows without bound.

# Big- $O$ Notation

## Examples:

- Show that  $x^2 + 4x + 1 = O(x^2)$ .
  - Take  $C = 6$  and  $k = 1$ .
- Show that  $9x^2 = O(x^3)$ .
  - Take  $C = 9$  and  $k = 1$ .
- Is  $n^2 = O(n)$ ?
  - Prove using contradiction.

# Big- $O$ Notation

## Theorem

Let  $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  where  $a_0, a_1, \dots, a_n$  are real numbers. Then:  
 $f(x) = O(x^n)$ .

Proof (on board).

Examples:

- $57x + 92 = O(x)$
- $2x^2 = O(x^2)$
- Thus  $57x + 92 = O(2x^2)$  but  $2x^2 \neq O(57x + 92)$

# Big- $O$ Notation

Examples:

What is  $O$  of the following functions?

- $1 + 2 + 3 + \dots + n$
- $n!$

Tips:

- $n^c = O(n^d)$  but  $n^d \neq O(n^c)$  when  $d > c > 1$
- $(\log_b n)^c = O(n^d)$  but  $n^d \neq O((\log_b n)^c)$  when  $b > 1$  and  $c, d > 0$
- $n^d = O(b^n)$  but  $b^n \neq O(n^d)$  when  $d > 0$  and  $b > 1$
- $b^n = O(c^n)$  but  $c^n \neq O(b^n)$  when  $c > b > 1$

# Growth of Combinations of Functions

## Theorem

Suppose  $f_1(x) = O(g_1(x))$  and that  $f_2(x) = O(g_2(x))$ .  
Then  $(f_1 + f_2)(x) = O(\max\{g_1(x), g_2(x)\})$ .

## Theorem

Suppose  $f_1(x) = O(g_1(x))$  and that  $f_2(x) = O(g_2(x))$ .  
Then  $(f_1 f_2)(x) = O(g_1(x)g_2(x))$ .

Examples:

What is  $O$  for the following functions?

- $x^7 + 2^4 - 72x + 15 + \log_2(x)$
- $(x^2 + 4)(\log_2(x - 12))$

# Big Omega

## Definition

Let  $f$  and  $g$  be functions from the set of integers (or real numbers) to the set of real numbers.

We say that  $f(x)$  is  $\Omega(g(x))$  (“big omega”) if there are positive constants  $C$  and  $k$  such that:

$$|f(x)| \geq C|g(x)|$$

whenever  $x > k$ .

Examples:

- $x^4 - 17x = \Omega(x^2)$
- $\log_2 x + 12 = \Omega(1)$

# Big Theta

## Definition

Let  $f$  and  $g$  be functions from the set of integers (or real numbers) to the set of real numbers.

We say that  $f(x)$  is  $\Theta(g(x))$  ("big theta") if  $f(x) = O(g(x))$  AND  $f(x) = \Omega(g(x))$ .

We say that  $f(x)$  and  $g(x)$  have the same **order**.

Example:

$$\square 45x^2 + 72x = \Theta(x^2)$$

## Theorem

Let  $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  where  $a_0, a_1, \dots, a_n$  are real numbers. Then:

$$f(x) = \Theta(x^n).$$

## 3.3 Complexity of Algorithms

---

# Complexity of Algorithms

There are always two (sometimes three) questions to ask about a given algorithm:

- Is it correct?
- How much time does it take to run?
- How much space does it take?

We will focus our attention in this section to time complexity.

# Time Complexity

## Examples:

- Analyze the time complexity for all of the algorithms presented in Section 3.1.
- Our analysis will be a **worst-case** analysis.
- Why?
  - Determining the average-case can often be difficult.
  - A worst-case analysis guarantees that an algorithm will terminate within a given time span—it's a pessimistic and safe analysis.

# Tractability

## Definition

An algorithm is said to have **polynomial time complexity** if it has complexity  $\Theta(n^b)$  where  $b \in \mathbb{Z}$ ,  $b \geq 1$ .

An algorithm is said to have **exponential time complexity** if it has complexity  $\Theta(b^n)$  where  $b > 1$ .

An algorithm is said to have **factorial time complexity** if it has complexity  $\Theta(n!)$ .

## Definition

A problem that is solvable using an algorithm with polynomial time complexity is called **tractable**.

A problem that cannot be solved using an algorithm with polynomial time complexity is called **intractable**.

A problem for which there exists no algorithm to solve it is called **unsolvable**.

## $P$ and $NP$ and $NP$ – Complete

### Definition

Problems for which a proposed *solution* can be checked in polynomial time belong to the class **NP**.

Problems that have a polynomial time *algorithm* (or tractable problems) belong to the class **P**.

Problems that are at least as hard as any problem in  $NP$  belong to the class **NP-Complete**.

In class discussion about the status of  $P = NP$  or  $P \neq NP$ .