



MAKE  
SCHOOL

# ALGORITHM ANALYSIS

*Big O? What's the Big Deal?*

# WHAT IS AN ALGORITHM?

A sequence of steps to accomplish a task

...described precisely enough for a computer  
to perform them

## Algorithm != Code

Algorithms can be written in any language,  
including *pseudocode*, English, Farsi, etc.

*More info: [algorithm definition](#), [algorithm characterizations](#), [pseudocode](#)*

# WHO COINED ALGORITHM?

Al Gore... Rhythm?

After all, he *did*\* invent the Internet *\*not*

**Muhammad ibn Mūsā al-Khwārizmī**

9th century Persian mathematician

Also coined “*al-jabr*” – a.k.a. *algebra*

*More info: word origin, etymology, Muhammad ibn Mūsā al-Khwārizmī*

# ALGORITHM ANALYSIS

Correctness – does it solve the problem?

*Formal verification methods – e.g., NASA*

Resource usage – how efficient is it?

*Time – idealized work steps (CPU cycles)*

*Space – working memory (RAM, disk, etc.)*

*More info: [correctness](#), [formal verification](#), [algorithm analysis](#)*

# WHY IS THIS IMPORTANT?

Cost – computers and electricity aren't free

Comparison – choose best solution based on analysis *before* implementing and running

Tractability – not all problems can be solved in your lifetime, even with moderate size datasets

*Non-deterministic Polynomial-time, NP-hard*

*More info: P versus NP problem, NP-hard, NP-complete*

# RUNTIME IN PRACTICE

Actual runtime depends on many factors:

CPU speed – pace of instruction execution

Language – abstractions on top of CPU instructions

Compiler – optimization of instruction ordering, etc.

Environment – resources available during runtime  
(not used by other programs running concurrently)



# RUNTIME COMPARISON

	Computer A	Computer B
Sorting Algorithm	Bubble Sort	Merge Sort
Time Complexity	$n^2$	$n \cdot \log_2 n$
Execution Speed	10 GHz (billion instr/sec)	10 MHz (million instr/sec)
Code Optimization	High (2 instr/step)	Low (50 instr/step)
Actual Complexity	$2n^2$	$50n \cdot \log_2 n$
Time to Sort 100K #s	2 seconds	8.3 seconds
Time to Sort 1M #s	3.33 minutes	1.67 minutes
Time to Sort 10M #s	5.56 hours!	19.4 minutes
Time to Sort 100M #s	23.1 days!!!	3.69 hours

*Example from Algorithms Unlocked*



# RUNTIME ASSUMPTIONS

Need to make some assumptions to easily describe algorithm runtime in the abstract:

Time to execute basic operations is *constant*

Relative speed of operations is *not important*

Including arithmetic, logic, comparison, variable assignment, array indexing, function calls, etc.

# RUNTIME ANALYSIS

What we really want to analyze is *complexity* – how algorithm runtime grows as the input gets larger

Describe runtime as a function of the size of the input

Input array/list length usually written as ***n***, ***m***, ***k***

Ignore constants and lower-order terms

e.g.,  **$3n \rightarrow n$** ,  **$6n^2 + 15n + 24 \rightarrow n^2$**

# CLASSIC LINEAR SEARCH

Return index of **target** in **histogram**, or **None** if not found

```
1 def linear_search(target, histogram):
2     index = None # not found
3     i = 0
4     for item, count in histogram:
5         if item == target:
6             index = i # found
7             i += 1
8     return index
```

# PYTHONIC LINEAR SEARCH

Return index of **target** in **histogram**, or **None** if not found

```
1 def linear_search(target, histogram):  
2     index = None # not found  
  
4     for i, (item, count) in enumerate(histogram):  
5         if item == target:  
6             index = i # found  
  
8     return index
```

# EARLY EXIT LINEAR SEARCH

Return *first* index of **target** in **histogram**, or **None** if not found

```
1  def linear_search(target, histogram):  
  
4      for i, (item, count) in enumerate(histogram):  
5          if item == target:  
6              return i # found  
  
8      return None # not found
```

# ASYMPTOTIC NOTATION

**Worst case** – upper bound is some function  $f(n)$

Algorithm is  $O(f(n))$  – read as “**big oh** of  $f(n)$ ”

**Best case** – lower bound is another function  $g(n)$

Algorithm is  $\Omega(g(n))$  – read as “**omega** of  $g(n)$ ”

If both bounds are the same (that is,  $f(n) = g(n)$ ), then

Algorithm is  $\Theta(f(n))$  – read as “**theta** of  $f(n)$ ”

# RUNNING TIME SUMMARY

	Best Case	Worst Case	Both Cases
Exhaustive Linear Search (find last match)	$\Omega(n)$	$O(n)$	$\Theta(n)$
Early Exit Linear Search (find first match)	$\Omega(1)$	$O(n)$	N/A



# RESOURCES

Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein – widely considered *the Bible of Algorithms*

Algorithms Unlocked by Thomas Cormen – introductory and more accessible, less technical detail than CLRS