Unit #1: Complexity Theory and Asymptotic Analysis

CPSC 221: Algorithms and Data Structures

Will Evans and Jan Manuch

2016W1

Unit Outline

- Brief proof reminder
- Algorithm Analysis: Counting steps
- Asymptotic Notation
- Runtime Examples
- Problem Complexity

Learning Goals

algorithm

- Given ~~code~~, write a formula that measures the number of steps executed as a function of the size of the input.
- Use asymptotic notation to simplify functions and to express relations between functions.
- Know the asymptotic relations between common functions.
- Understand why to use worst-case, best-case, or average-case complexity measures.
- Give examples of tractable, intractable, and undecidable problems.

Proof by ...

Review

- Counterexample
  - show an example which does not fit with the theorem
  - Thus, the theorem is false.
- Contradiction
  - assume the opposite of the theorem
  - derive a contradiction
  - Thus, the theorem is true.
- Induction
  - prove for a base case (e.g., $n = 1$)
  - assume for all $n \leq k$ (for arbitrary k)
  - prove for the next value ($n = k + 1$)
  - Thus, the theorem is true.

Example: Proof by Induction (worked) 1/4

Theorem:

A positive integer x is divisible by 3 if and only if the sum of its decimal digits is divisible by 3.

12

17

174

Proof:

Let $x_1 x_2 x_3 \ldots x_n$ be the decimal digits of x.

Let the sum of its decimal digits be

$$S(x) = \sum_{i=1}^{n} x_i$$

1200

$S(12) = 3$

$S(174) = 12$

We'll prove the stronger result:

$$\longrightarrow \quad S(x) \bmod 3 = x \bmod 3.$$

How do we use induction?

Example: Proof by Induction (worked) 2/4

Base Case:

Consider any number x with one ($n = 1$) digit (0-9).

$$S(x) = \sum_{i=1}^{n} x_i = x_1 = x.$$

So, it's trivially true that $S(x) \bmod 3 = x \bmod 3$ when $n = 1$.

Example: Proof by Induction (worked) 3/4

Inductive hypothesis:

Assume for an arbitrary integer $k > 0$ that for any number x with $n \leq k$ digits:

$$S(x) \bmod 3 = x \bmod 3.$$

Example

$x = 123$

$k = 2$

Inductive step:

Consider a number x with $n = k + 1$ digits:

$$x = x_1 x_2 \ldots x_k x_{k+1}.$$

Let z be the number $x_1 x_2 \ldots x_k$. It's a k-digit number so the inductive hypothesis applies:

$z = 12$

by I. H.

$$S(z) \bmod 3 = z \bmod 3.$$

Example: Proof by Induction (worked) 4/4

$x = 123$

$10 \cdot 12 + 3$

Inductive step (continued):

$$
\begin{aligned}
x \bmod 3 &= (10z + x_{k+1}) \bmod 3 && (x = 10z + x_{k+1}) \\
&= (\cancel{9z} + z + x_{k+1}) \bmod 3 \\
&= (z + x_{k+1}) \bmod 3 && (9z \text{ is divisible by } 3) \\
\text{I.H.}\quad &\overset{\text{"}}{=} (S(z) + x_{k+1}) \bmod 3 && (\text{induction hypothesis}) \\
&= (x_1 + x_2 + \cdots + x_k + x_{k+1}) \bmod 3 \\
&= S(x) \bmod 3
\end{aligned}
$$

QED (quod erat demonstrandum: "what was to be demonstrated")

A Task to Solve and Analyze

Find a student's name in a class given her student ID

array of objects

Sort by ID

insert? delete?

move to front

more complex structure

Skiplist, balanced search tree,

ID
name

hash map

ID, name

Does it matter?

Analysis of Algorithms

- Analysis of an algorithm gives insight into
  - how long the program runs (time complexity or runtime) and
  - how much memory it uses (space complexity).
- Analysis can provide insight into alternative algorithms
- Input size is indicated by a non-negative integer n (sometimes there are multiple measures of an input's size)
- Running time is a real-valued function of n such as:
  - $T(n) = 4n + 5$
  - $T(n) = 0.5n \log n - 2n + 7$
  - $T(n) = 2^n + n^3 + 3n$

Rates of Growth

10^{-12}

Suppose a computer executes 1op per picosecond (trillionth):

$n =$	10
$\log n$	1ps
n	10ps
$n \log n$	10ps
n^2	100ps
2^n	1ns

ns = nanosecond (billionth) 10^{-9} sec

Rates of Growth

Suppose a computer executes 1op per picosecond (trillionth):

$n =$	10	100
$\log n$	1ps	2ps
n	10ps	100ps
$n \log n$	10ps	200ps
n^2	100ps	10ns
2^n	1ns	1Es

Exasecond(Es) = $\cancel{32}$ billion years
37

Rates of Growth

Suppose a computer executes 1op per picosecond (trillionth):

$n =$	10	100	1,000
$\log n$	1ps	2ps	3ps
n	10ps	100ps	1ns
$n \log n$	10ps	200ps	3ns
n^2	100ps	10ns	1μs
2^n	1ns	1Es	10^{289}s

microsecond
= millionth
1/1000000

Exasecond(Es) $=$ 32 billion years

Rates of Growth

Suppose a computer executes 1op per picosecond (trillionth):

$n =$	10	100	1,000	10,000
$\log n$	1ps	2ps	3ps	4ps
n	10ps	100ps	1ns	10ns
$n \log n$	10ps	200ps	3ns	40ns
n^2	100ps	10ns	1μs	100μs
2^n	1ns	1Es	10^{289}s	

Exasecond(Es) $=$ 32 billion years

Rates of Growth

Suppose a computer executes 1op per picosecond (trillionth):

$n =$	10	100	1,000	10,000	10^5
$\log n$	1ps	2ps	3ps	4ps	5ps
n	10ps	100ps	1ns	10ns	100ns
$n \log n$	10ps	200ps	3ns	40ns	500ns
n^2	100ps	10ns	1μs	100μs	10ms
2^n	1ns	1Es	10^{289}s		

millisecond

Exasecond(Es) $=$ 32 billion years

Rates of Growth

Suppose a computer executes 1op per picosecond (trillionth):

$n =$	10	100	1,000	10,000	10^5	10^6
$\log n$	1ps	2ps	3ps	4ps	5ps	6ps
n	10ps	100ps	1ns	10ns	100ns	1μs
$n \log n$	10ps	200ps	3ns	40ns	500ns	6μs
n^2	100ps	10ns	1μs	100μs	10ms	1s
2^n	1ns	1Es	10^{289}s			

Exasecond(Es) $=$ 32 billion years

Rates of Growth

Suppose a computer executes 1op per picosecond (trillionth):

$n =$	10	100	1,000	10,000	10^5	10^6	10^9
$\log n$	1ps	2ps	3ps	4ps	5ps	6ps	9ps
n	10ps	100ps	1ns	10ns	100ns	1μs	1ms
$n \log n$	10ps	200ps	3ns	40ns	500ns	6μs	9ms
n^2	100ps	10ns	1μs	100μs	10ms	1s	1week
2^n	1ns	1Es	10^{289}s				

Exasecond(Es) = 32 billion years

human genome

pine tree

3×10^9 base pairs

23×10^9 base pairs

Analyzing Code

```
// Linear search
find(key, array)
  for i = 0 to length(array) - 1 do
    if array[i] == key
      return i
  return -1
```

is this expensive
key length would
be another
input size measure

1) What's the input size, n?

n = length of array

Analyzing Code

```
// Linear search
find(key, array)
  for i = 0 to length(array) - 1 do
    if array[i] == key
      return i
  return -1
```

2) Should we assume a worst-case, best-case, or average-case input of size n?

n doesn't tell us enough about the input

Some size n arrays

- have key in array[0]
- don't have key at all

Analyzing Code

```
// Linear search
find(key, array)
  for i = 0 to length(array) - 1 do
    if array[i] == key
      return i
  return -1
```

2 lines/iter.

3) How many lines are executed as a function of n in a worst-case?

$T(n) = 2n + 1$

Are lines the right unit?

maybe

Analyzing Code

The number of lines executed in the worst-case is:

$$T(n) = \underline{2}n + 1.$$

- Does the "1" matter? maybe
- Does the "2" matter? maybe

but as n gets big high order term dominates

as technology changes, time per line changes by constant factor

Big-O Notation

set of functions

Assume that for every integer n, $T(n) \geq 0$ and $f(n) \geq 0$.

$T(n) \in O(f(n))$ if there are positive constants c and n_0 such that

$$T(n) \leq cf(n) \text{ for all } n \geq n_0.$$

Meaning: "$T(n)$ grows no faster than $f(n)$"

$T(n) = 2n+1 \qquad f(n) = n$

Claim $\quad 2n+1 \in O(n)$

proof $\quad 2n+1 \leq 3n \quad$ for $n \geq 1$

c $\quad n_0$

$\Leftrightarrow \quad 1 \leq n \quad$ (subtract $2n$ from each side of "$\leq$")

"if and only if"

for $n \geq 1 \leftarrow$ true trivially.

Asymptotic Notation

Can you prove the equivalence of

big Oh

- $T(n) \in O(f(n))$ if there are positive constants c and n_0 such that $T(n) \leq cf(n)$ for all $n \geq n_0$. "$\leq$"

iff $f(n) \in O(T(n))$

Omega

- $T(n) \in \Omega(f(n))$ if there are positive constants c and n_0 such that $T(n) \geq cf(n)$ for all $n \geq n_0$. "$\geq$"

Theta

- $T(n) \in \Theta(f(n))$ if $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$. "$=$"

- $T(n) \in o(f(n))$ if for **any** positive constant c, there exists n_0 such that $T(n) < cf(n)$ for all $n \geq n_0$. $<$

$\lim_{n \to \infty} \frac{T(n)}{f(n)} = 0$

- $T(n) \in \omega(f(n))$ if for **any** positive constant c, there exists n_0 such that $T(n) > cf(n)$ for all $n \geq n_0$. $>$

$= \infty$

not all pairs of function are related

$n^{1+\sin(n)}$ versus n

Examples

$O: 10000n^2 + 25n \le 10025 n^2$ for $n \geq 1$

c n_0

$\Omega: 10000n^2 + 25n \geq 10000n^2$ for $n \geq 1$

$10,000n^2 + 25n \in \Theta(n^2)$

positive constant

$10^{-10}n^2 \in \Theta(n^2)$

$n \log n \in O(n^2)$

$\Longleftrightarrow$ $n \log n \le c \cdot n^2$

$\log n \le c \cdot n$

yes by "racetrack" principle

$n \log n \in \Omega(n)$

$n^3 + 4 \in o(n^4)$ ✓ $\quad n^3 / n^4 \xrightarrow[n\to\infty]{} 0$

$n^3 + 4 \in \omega(n^2)$ ✓ $\quad \frac{n^3}{n^2} = n \xrightarrow[n\to\infty]{} \infty$

Analyzing Code

```
// Linear search
find(key, array)
  for i = 0 to length(array) - 1 do
    if array[i] == key
      return i
  return -1
```

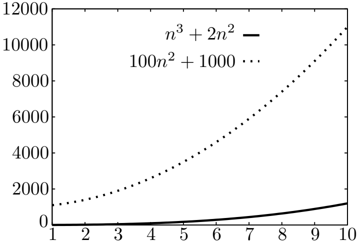
4) How does $T(n) = 2n + 1$ behave asymptotically? What is the appropriate order notation? (O, o, Θ, Ω, ω?)

O my algorithm is fast. It's $O(n)$

Ω my alg. is slow. It's $\Omega(n)$

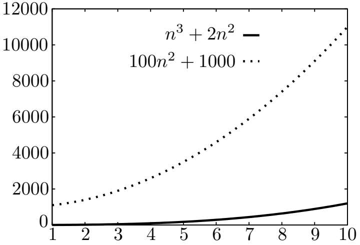
Asymptotically smaller?

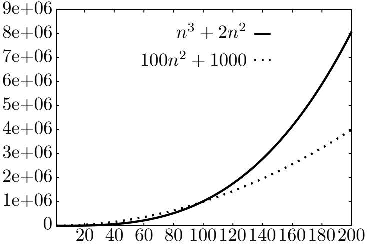
$$n^3 + 2n^2 \qquad \text{versus} \qquad 100n^2 + 1000$$



Asymptotically smaller?

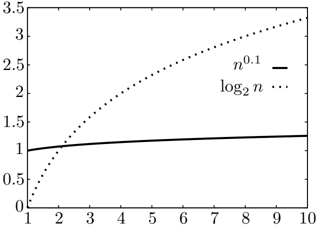
$$n^3 + 2n^2 \qquad \text{versus} \qquad 100n^2 + 1000$$





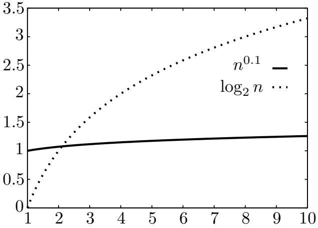
Asymptotically smaller?

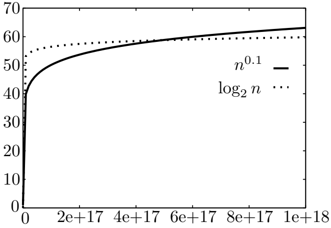
$$n^{0.1} \qquad \text{versus} \qquad \log_2 n$$



Asymptotically smaller?

$n^{0.1}$ versus $\log_2 n$

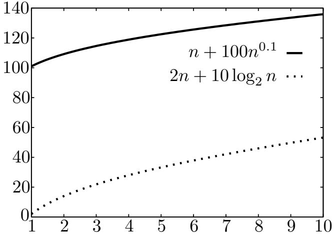




Asymptotically smaller?

A B

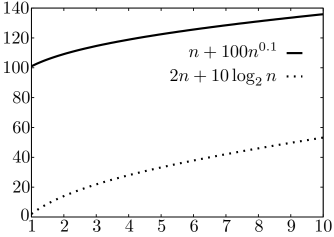
$$n + 100n^{0.1} \quad \text{versus} \quad 2n + 10\log_2 n$$

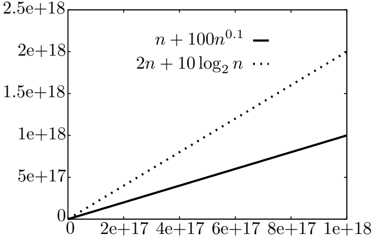


Asymptotically smaller?

A $\quad$ B $\quad$ 1/4

$$n + 100n^{0.1} \quad \text{versus} \quad 2n + 10\log_2 n$$





$A \in O(B)$ ①

$A \in \Omega(B)$ ②

$A \in \Theta(B)$ ③ ✓

Typical asymptotics

$\log_b n = \frac{\log_2 n}{\log_2 b}$

Tractable

- constant: $\Theta(1)$
- logarithmic: $\Theta(\log n)$ ($\log_b n$, $\log n^2 \in \Theta(\log n)$)
- poly-log: $\Theta(log^k n)$ ($\log^k n \equiv (\log n)^k$)
- linear: $\Theta(n)$
- log-linear: $\Theta(n \log n)$
- superlinear: $\Theta(n^{1+c})$ (c is a constant > 0)
- quadratic: $\Theta(n^2)$
- cubic: $\Theta(n^3)$
- polynomial: $\Theta(n^k)$ (k is a constant)

$2^n \notin \Theta(3^n)$

Intractable

- exponential: $\Theta(c^n)$ (c is a constant > 1)

Sample asymptotic relations

- $\{1, \log n, n^{0.9}, n, 100n\} \subset O(n)$
- $\{n, n \log n, n^2, 2^n\} \subset \Omega(n)$
- $\{n, 100n, n + \log n\} \subset \Theta(n)$
- $\{1, \log n, n^{0.9}\} \subset o(n)$
- $\{n \log n, n^2, 2^n\} \subset \omega(n)$

Analyzing Code

- single operations: constant time
- consecutive operations: sum operation times
- conditionals: condition time plus max of branch times
- loops: sum of loop-body times
- function call: time for function

because of worst case

Above all, use your head!

Runtime example #1

Count lines

```
for i = 1 to n do
  for j = 1 to n do
    sum = sum + 1.
```

1

$2n = \sum_{j=1}^{n} 2$

$\sum_{i=1}^{n} (2n+1)$

$= 2n^2 + n$

$\Theta(n^2)$

Runtime example #2

Count sum = sum + 1

```
i = 1
while i < n do
  for j = i to n do
    sum = sum + 1
  i++
```

$$\sum_{j=i}^{n} 1 = n - i + 1$$

arithmetic series

$$T(n) = \sum_{i=1}^{n-1} (n-i+1) = n + n-1 + n-2 + \ldots + 2$$

$$= \frac{n(n+1)}{2} - 1$$

$$= \frac{n^2}{2} + \frac{n}{2} - 1$$

$$\in \Theta(n^2)$$

Runtime example #3

$T(n) = \Theta(n)$

```
i = 1
while i < n do
  for j = 1 to i do
    sum = sum + 1
  i += i
```

$\sum_{j=1}^{i} 1 = i$

$\lg n$

$m = \lceil \log_2 n \rceil - 1$ $\quad n > 1$

$i = 1, 2, 4, 8, 16,$

$2^m < n \leq 2^{m+1}$

$T(n) = 1 + 2 + 4 + \ldots + 2^m$ where

$= \sum_{i=0}^{m} 2^i = 2^{m+1} - 1$ geometric series

$= 2^{\lceil \lg n \rceil} - 1$

$\in \Theta(n)$

$2^{\lceil \lg n \rceil} < 2$

$n-1 = 2^{\lg n} - 1 \leq 2^{\lceil \lg n \rceil} - 1 < 2^{\lg n + 1} = 2n$

$T(n) \geq n-1 \geq n/2$ for $n \geq 2$

$T(n) \leq 2n$.

Runtime example #4

```
int max(A, n)
  if( n == 1 ) return A[0]
  return larger of A[n-1] and max(A, n-1)
```

Recursion almost always yields a recurrence relation:

a constant (like b = 1)

" (like c = 2)

$$T(1) \leq b$$

$$T(n) \leq c + T(n-1) \qquad \text{if } n > 1$$

verify by induction!

Solving recurrence:

$$\begin{aligned} T(n) &\leq c + c + T(n-2) && \text{(substitution)} \\ &\leq c + c + c + T(n-3) && \text{(substitution)} \\ &\leq kc + T(n-k) && \text{(extrapolating } k > 0\text{)} \\ &= (n-1)c + T(1) && \text{(for } k = n-1\text{)} \\ &\leq (n-1)c + b \end{aligned}$$

$T(n) \in O(n)$

only get O from this analysis

Verify claim $T(n) \le (n-1)c + b$

proof (by induction on n)

base $T(1) \le b$ ✓

ind. hyp claim true for $T(n)$

$T(n) \le (n-1)c + b$

ind step $T(n+1) \le c + T(n)$ by def.

$\le c + (n-1)c + b$ by I.H.

$= n \cdot c + b$ ✓

Runtime example #5: Mergesort

$\Theta(n)$ time

Mergesort algorithm:

Split list in half, sort first half, sort second half, merge together

Recurrence relation:

2 subproblems of size $n/2$

merge

$$T(1) \leq b$$
$$T(n) \leq 2T(n/2) + cn \qquad \text{if } n > 1$$

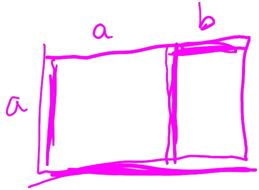
Solving recurrence:

$$\begin{aligned}
T(n) &\leq 2T(n/2) + cn \\
&\leq 2(2T(n/4) + cn/2) + cn && \text{(substitution)} \\
&= 4T(n/4) + 2cn \\
&\leq 4(2T(n/8) + cn/4) + 2cn && \text{(substitution)} \\
&= 8T(n/8) + 3cn \\
&\leq 2^k T(n/2^k) + kcn && \text{(extrapolating } k > 0) \\
&= nT(1) + cn \lg n && \text{(for } 2^k = n)
\end{aligned}$$

$\leq bn + cn\lg n$

$T(n) \in O(n \log n)$

Runtime example #6: Fibonacci 1/2



Recursive Fibonacci:

```
int fib(n)
  if( n == 0 or n == 1 ) return n
  return fib(n-1) + fib(n-2)
```

Recurrence relation: (lower bound)

$$\begin{aligned} T(0) &\geq b \\ T(1) &\geq b \\ T(n) &\geq T(n-1) + T(n-2) + c \quad \text{if } n > 1 \end{aligned}$$

golden ratio

$$\frac{a}{b} = \varphi = \frac{a+b}{a}$$

$\varphi \approx 1.618...$

Claim:

$$T(n) \geq b\varphi^{n-1}$$

where $\varphi = (1+\sqrt{5})/2$.
Note: $\varphi^2 = \varphi + 1$.

Runtime example #6: Fibonacci 2/2

Claim:

$$T(n) \geq b\varphi^{n-1}$$

Proof: (by induction on n)
Base case: $T(0) \geq b > b\varphi^{-1}$ and $T(1) \geq b = b\varphi^0$.
Inductive hyp: Assume $T(n) \geq b\varphi^{n-1}$ for all $n \leq k$.
Inductive step: Show true for $n = k + 1$.

definition

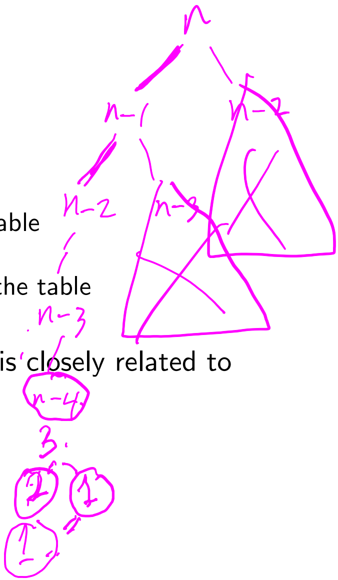
$$\begin{aligned} T(n) &\geq T(n-1) + T(n-2) + c \\ &\geq b\varphi^{n-2} + b\varphi^{n-3} + c \qquad \text{(by inductive hyp.)} \\ &= b\varphi^{n-3}(\varphi + 1) + c \longleftarrow \text{by property of } \varphi \\ &= b\varphi^{n-3}\varphi^2 + c \\ &\geq b\varphi^{n-1} \end{aligned}$$

$T(n) \in \Omega(\varphi^n)$
Why? Same recursive call is made numerous times.

Example #7: Learning from analysis

To avoid recursive calls

- store base case values in a table
- before calculating the value for *n*
  - check if the value for *n* is in the table
  - if so, return it
  - if not, calculate it and store it in the table



This strategy is called memoization and is closely related to dynamic programming.

How much time does this version take?

$\Theta(n)$

Runtime Example #8: Longest Common Subsequence

Problem: Given two strings (A and B), find the longest sequence of characters that appears, in order, in both strings.

Example: $|A| = n$ $|B| = m$

$A =$ `search me` $B =$ `insane method`

1 0 1 0 0 0 1 1

A longest common subsequence is "`same`" (so is "`seme`")

where S is a subseq. of A

Applications:
DNA sequencing, revision control systems, `diff`, ...

for every subsequence of A ← $\Omega(2^n)$
see if it's in B ← greedy. match first occurrence in B of S[0] then first after that of S[1] ...
(take the biggest)

$|S| \leq n$

m

$n \cdot m$

$\Theta(2^n m)$ worst case

Example #9

Find a tight bound on $T(n) = \lg(n!)$.

$\lg(n(n-1)(n-2)\ldots 1)$

$$= \sum_{i=1}^{n} \lg i \leq \sum_{i=1}^{n} \lg n = n \lg n \quad O(n\lg n)$$

$$\geq \sum_{i=n/2}^{n} \lg i \geq \sum_{i=n/2}^{n} \lg(n/2)$$

$$= n/2 \lg n/2$$

$$= n/2 \lg n - n/2$$

$$= n/4 \lg n + n/4 \lg n - n/2$$

$1/4 \lg n \geq 1/2$ for $n \geq 4$

$$\Omega(n \lg n) \quad \geq n/4 \lg n \text{ for } n \geq 4$$

Log Aside

$\log_b x$ is the exponent b must be raised to to equal x.

- $\lg x \equiv \log_2 x$ (base 2 is common in CS)
- $\log x \equiv \log_{10} x$ (base 10 is common for 10 fingered mammals)
- $\ln x \equiv \log_e x$ (the natural log)

Note: $\Theta(\lg n) = \Theta(\log n) = \Theta(\ln n)$ because

$$\log_b n = \frac{\log_c n}{\log_c b}$$

for constants $b, c > 1$.

Asymptotic Analysis Summary

- Determine what is the input size
- Express the resources (time, memory, etc.) an algorithm requires as a function of input size
  - worst case
  - best case
  - average case
- Use asymptotic notation, O, Ω, Θ, to express the function simply

Problem Complexity

The **complexity of a problem** is the complexity of the best algorithm for the problem.

- We can sometimes prove a lower bound on a problem's complexity. (To do so, we must show a lower bound on any possible algorithm.)
- A correct algorithm establishes an upper bound on the problem's complexity.

Searching an unsorted list using comparisons takes $\Omega(n)$ time (lower bound).
Linear search takes $O(n)$ time (matching upper bound).

Sorting a list using comparisons takes $\Omega(n \log n)$ time (lower bound).
Mergesort takes $O(n \log n)$ time (matching upper bound).

Aside: Who Cares About $\Omega(\lg(n!))$?

Can You Beat $\cancel{O}\Theta(n \log n)$ Sort?

Chew these over:

- How many values can you represent with c bits?
- Comparing two values ($x < y$) gives you one bit of information.
- There are $n!$ possible ways to reorder a list. We could number them: $1, 2, \ldots, n!$
- Sorting basically means choosing which of those reorderings/numbers you'll apply to your input.
- How many comparisons does it take to pick among $n!$ numbers?

$\geq \lg(n!)$

Problem Complexity

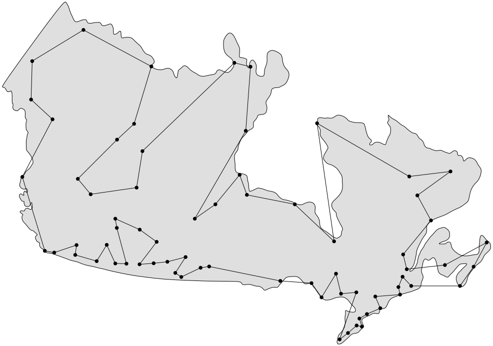
P

Sorting: solvable in polynomial time, tractable

Traveling Salesman Problem (TSP): In 1,290,319km, can I drive to all the cities in Canada and return home? www.math.uwaterloo.ca/tsp/

Checking a solution takes polynomial time. Current fastest way to find a solution takes exponential time in the worst case.

NP



Are problems in NP really in P? $1,000,000 prize

Problem Complexity

Searching and Sorting: P, tractable
Traveling Salesman Problem: NP, intractable?
Kolmogorov Complexity: Uncomputable

Kolmogorov Complexity of a string is the length of the shortest description of it.

Can't be computed. Pithy but hand-wavy proof: What's:

> *The smallest positive integer that cannot be described in fewer than fourteen words*