Success stories of algorithms:

Shortest path (Google maps)

Pattern matching (Text editors, genome)

Fast-fourier transform (Audio/video processing)

This class:

General techniques:	Divide-and-conquer,
	dynamic programming,
	data structures
	amortized analysis
Various topics:	Sorting
	Matrixes
	Graphs
	Polynomials

MATHEMATICAL BACKGROUND

- What is theory?

- Theory is when you make claims
  that are either True or False,
  but not both

Example of claims:

1+1 = 2

there is a graph with > 56 edges

all prime numbers are between 57 and 59

all regular languages are context-free

- More complicated claims are made up with logical connectives

Logical connectives

- not A also written !A, $\overline{A}$, $\neg$ A , $\cancel{A}$
- A or B also written $A \vee B$, $A \cup B$, …
- A and B also written $A \wedge B$, A & B, ...
- A implies B also written $A \Rightarrow B$, if A then B, B if A

- You should be familiar with these, but let's clear some doubts

Or

A or B means A or B, possibly both

- Different use in everyday language:

“We shall triumph or perish”

- Intended meaning is:

“We shall triumph **exclusive-or** perish”

- Do not confuse or with exclusive or!

Implication

A	B	A implies B
False	False	True
False	True	True
True	False	False
True	True	True

$A \Rightarrow B$ means if A then B

Only False when A True and B False, True otherwise

"$1 = 0 \Rightarrow$ the earth is flat" is True (False $\Rightarrow$ False)

- $A \Rightarrow B$ same as: (not A) or B

  (not B) $\Rightarrow$ (not A) (contrapositive)

Different meaning in everyday language:

- "You go out if you finish your homework"

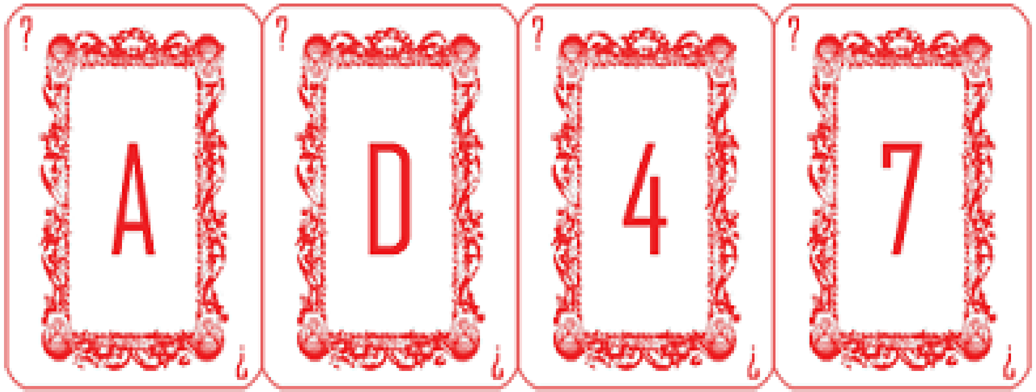
  A B

- Logically means B $\Rightarrow$A, can go out and not having finished homework!
- Intended meaning: A $\Rightarrow$B

  "You go out **only if** you finish your homework"

- Do not confuse A $\Rightarrow$B with B $\Rightarrow$A !

Do you understand implication?



- Know for true: Each card has a number on one side and a letter on the other.
- Suppose I claim: If a card has a vowel on one side, then it has an even number on the other side

- Which cards **must** you turn to know if I lie or not?

De Morgan's Laws:

$\neg(A \wedge B)$ is equivalent to $(\neg A) \vee (\neg B)$

$\neg(A \vee B)$ is equivalent to $(\neg A) \wedge (\neg B)$

There are two quantifiers:

$\exists$	there exists	same thing as OR
$\forall$	for all	same thing as AND

Usually:	OR, AND	few things
	$\exists$, $\forall$	many (infinite) things

Example:

$\exists$ a prime x > 5

same as

6 is prime OR 7 is prime OR 8 is prime OR ...

$\forall x,\ x < y$

same as

$1 < y$ AND $2 < y$ AND $3 < y$ AND …

De Morgan's Laws for quantifiers:

$\neg \exists x\, A(x)$ is equivalent to $\forall x\, \neg A(x)$

$\neg \forall x\, A(x)$ is equivalent to $\exists x\, \neg A(x)$

Sets, Functions:

• Sets are just different notation to express the same claims we construct using logical connectives and quantifiers.

This redundant notation turns out to be useful.

$$(x = 1) \lor (x = 16) \lor (x = 23) \quad \Leftrightarrow \quad x \in \{1, 16, 23\}$$

$$x \text{ is even} \quad \Leftrightarrow \quad x \in \{x \mid x \text{ is even}\}$$

$$A(x) \quad \Leftrightarrow \quad x \in \{x \mid A(x)\}$$

With this in mind, sets become straightforward.

- When are two sets equal?

  When the defining claims are equivalent:

  $\{x \mid A(x)\} = \{x \mid B(x)\}$ same as $A(x) \Leftrightarrow B(x)$

This shows that order and repetitions do not matter,

for example $\{b, a, a\} = \{a, b\}$,

because $(x = b) \vee (x = a) \vee (x = a)$ and

$(x = a) \vee (x = b)$ are equivalent claims

- When is a set contained in another?

When its defining claim implies the defining claim of the latter:

$$\{x|A(x)\} \subseteq \{x|B(x)\} \Leftrightarrow \ A(x) \Rightarrow B(x)$$

$$\{x|A(x)\} \supseteq \{x|B(x)\} \Leftrightarrow \ B(x) \Rightarrow A(x)$$

$$\{x \mid A(x)\} \cup \{x \mid B(x)\} = \{x \mid A(x) \vee B(x)\}$$

$$\{x \mid A(x)\} \cap \{x \mid B(x)\} = \{x \mid A(x) \wedge B(x)\}$$

$$\overline{\{x \mid A(x)\}} = \{x \mid \neg A(x)\}$$

$$\cup_i \{x \mid A_i(x)\} = \{x \mid \exists i \ A_i(x)\}$$

$$\cap_i \{x \mid A_i(x)\} = \{x \mid \forall i \ A_i(x)\}$$

The empty set is denoted Ø

It can be defined as $\emptyset = \{x : 1+1= 3\}$

The empty set is a subset of any set:

$$\emptyset \subseteq \{ x : A(x) \} \qquad \text{always}$$

because $1+1=3 \Rightarrow A$ for any A

Powerset(A): Set of all subsets of A.

Example:

Powerset({1,2,3}) = {$\emptyset$,{1},{2},{3},{1,2},{2,3},{1, 3},{1,2,3}}

Size of a set A: |A| = number of elements in it

Example: | { 1,2,3 } | = 3

Fact: | Powerset(A) | = $2^{|A|}$

Example: |Powerset({1,2,3})| = 2^3 = 8

Important sets:

$\mathbb{N} = \{0,1,2,3,...\}$ Natural numbers

$\mathbb{Z} = \{..., -3, -2, -1, 0, 1, 2, 3, \dots\}$ Integer numbers

$\mathbb{R} = \{0, 2.5748954, \pi, \sqrt{2}, -17, \dots\}$ Real numbers

These are all infinite sets: contain an infinite number of elements

A **function** f from set A to set B is written $f : A \to B$ is a way to associate to EVERY element $a \in A$ ONE element $f(a) \in B$

A is called domain, B range

Example: $f : \{0,1\} \to \{a,b,c\}$ defined as $f(0)=a$, $f(1)=c$

$f : N \to N$ defined as $f(n) = n+1$

$f : Z \to Z$ defined as $f(n) = n^2$

Some $b \in B$ may not be `touched,' but every $a \in A$ must be

Tuples (arrays) : Ordered sequences of elements

Example: (5, 2) 2-tuple, or pair

(7, 8, -1) 3-tuple, or triple

(Ø, {4,5}, 8, 21) 4-tuple

Order matters: $(a, b) \neq (b, a)$

By contrast, $\{a, b\} = \{b, a\}$

Construct tuples from sets via Cartesian product

$A \times B$ = set of pairs (a, b) : $a \in A$ and $b \in B$

$= \{(a, b) : a \in A \text{ and } b \in B\}$

$A \times B \times C = \{(a, b, c) : a \in A \text{ and } b \in B \text{ and } c \in C\}$

$A^k = A \times A \times \ldots \times A$ (k times)

Example

$\{q,r,s\} \times \{0,1\} = \{ (q,0), (q,1), (r,0), (r,1), (s,0), (s,1) \}$

$\{a,b\}^3 = \{ (a,a,a), (a,a,b), (a,b,a), (a,b,b),$

$(b,a,a), (b,a,b), (b,b,a), (b,b,b) \}$

Strings

Strings are like tuples,

but written without brackets and commas

Example: (h, e, l, l, o) is written as hello

(0, 1, 0) is written as 010

Strings

An alphabet Σ is a finite, non-empty set.

We call its elements symbols.

Example: $\Sigma = \{0,1\}$ (the binary alphabet)

$\Sigma = \{a,b,..., z\}$ (English language alphabet)

A string over an alphabet Σ is a finite, ordered sequence of symbols from Σ

Example: 010101000 a string over $\Sigma = \{0,1\}$

hello a string over $\Sigma = \{a,b,..., z\}$

A string w is a substring of a string x if the symbols in w appears consecutively in x

Example: aba is a substring of aaabbaaaababbb

00 is a substring of 111100010010100

The length of a string w is the number of symbols in it

Length is denoted $|w|$

Example: $|hello| = 5$ $|001|=3$

We denote by Σ^i the set of strings of length i

Example: $\{0,1\}^2 = \{00, 01, 10, 11\}$

$hello \in \{a,b,..., z\}^5$

$001 \in \{0,1\}^3$

The empty string is denoted ε (never in Σ)

Its length is 0: $|\varepsilon| = 0$

We denote by Σ^* the set of all strings over Σ of any length, including ε

Example: $\{0,1\}^* = \{\varepsilon, 0, 1, 001, 10101010, \dots \}$

$= $ all binary strings

$\{a\}^* = \{\varepsilon, a, aa, aaa, aaaa, \dots \}$

$=$ all strings containing only a

$\varnothing^* = \{\varepsilon\}$

Note: $\Sigma^* = \{ \varepsilon \} \cup (\cup_i \Sigma^i) = \{ \varepsilon \} \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$

Σ^* is an infinite set

What is an algorithm?

- Informally,

  an algorithm for a function $f : A \to B$ (the problem) is a simple, step-by-step, procedure

  that computes $f(x)$ on every input x

- Example: $A = N \times N$ $B = N$, $f(x,y) = x+y$

- Algorithm: Kindergarten addition

$$\begin{array}{r} 112 \\ +\,999 \\ \hline 1111 \end{array}$$

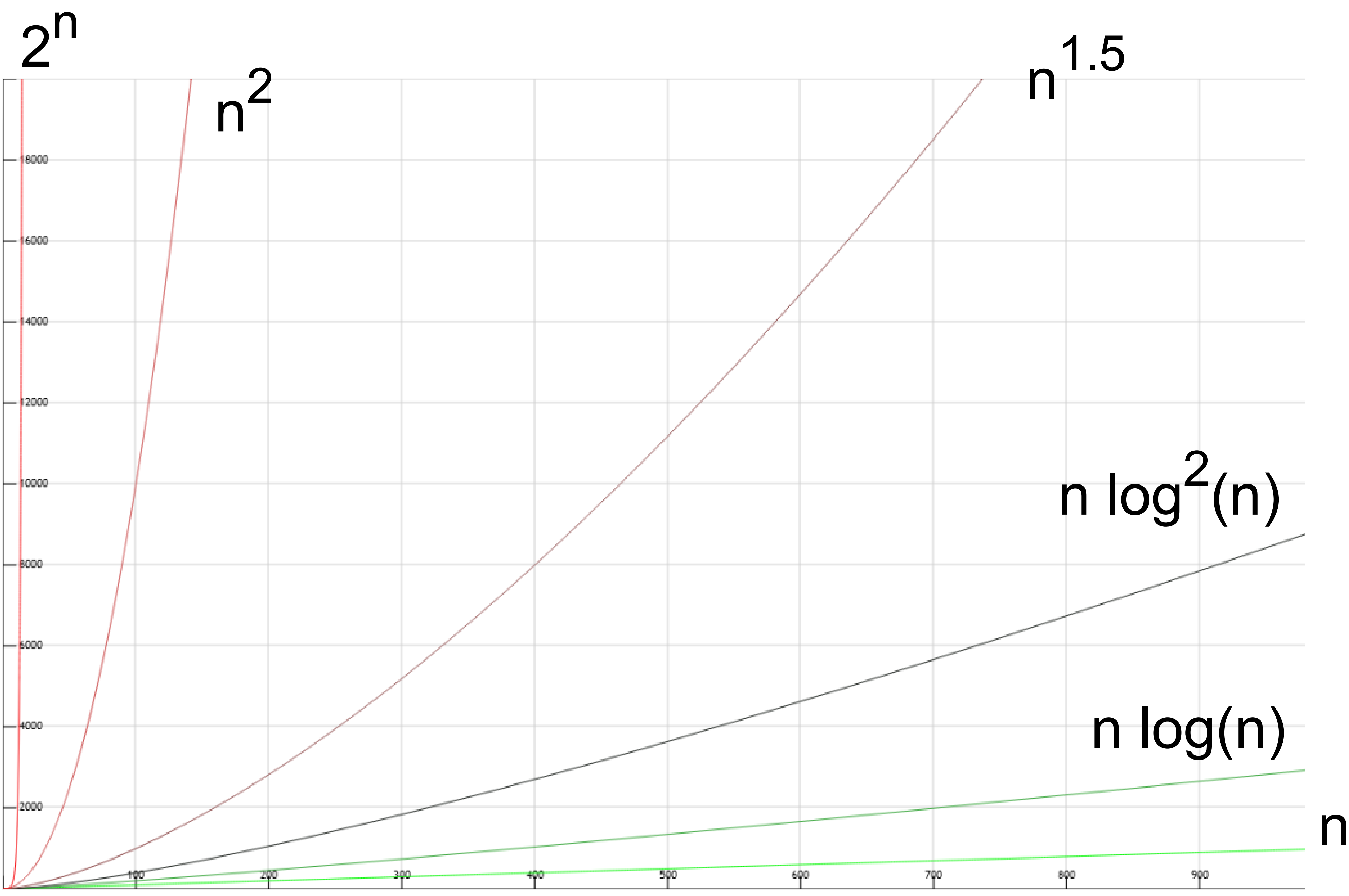
Abstraction 1

- Computers actually only handle strings $\in \{0,1\}^*$
- However, we often abstract from this and consider more structured objects,

  such as numbers, tuples, graphs
- We implicitly assume a suitable encoding in $\{0,1\}^*$
- Example: $30 \in N \rightarrow 11110 \in \{0,1\}^*$

Measuring performance of algorithms

- Time, space, etc. of algorithms are measured as a function of the input length.
- Makes sense: need to at least read the input!
- The input length is usually denoted n
- We are interested in which functions of n grow faster



2^n
n^2
n^1.5
n log^2(n)
n log(n)
n
18000
16000
14000
12000
10000
8000
6000
4000
2000
100
200
300
400
500
600
700
800
900

Abstraction 2: Time

- The exact time depends on the actual machine

- We ignore constant factors, to have more robust theory that applies to most computer

- Example:

  on my computer it takes 67 n + 15 operations,

  on yours 58 n – 15, but that's about the same

- We now give definitions that make this precise