Algorithms, Complexity, and Combinatorics

Objects of this area of study:

Develop good algorithms

Analyze the complexity of algorithms

Proride lower bounds on the complexity of problem:

(We will deal only with sequential, not parallel, algorithms.)

Historical approach:

Algorithm development

Empirical study

Theoretical analysis rare

Lower bounds non-existent

Algorithm: Step-by-step
problem solving method

Although "algorithms" existed thousands of years ago, the birth of the computer was necessary and sufficient to power their study.

The first questions:

what is an algorithm?

what problems have algorithms?

Computability Theory

Definition of an algorithm:

Turing, Kleene, Markov, Church, Post

Uncomputable functions:

Turing

Techniques are borrowed from

logic:

simulation, diagonalization

This work occurred just before the advent of computers (1930's).

Hilbert, at the turn of the century, was aware of the issue:

Hilbert's 10th problem, proved undecidable in 1970 by Matijasevic, building on work of Davis, Robinson

The next questions:

How efficient is an algorithm?

How efficient can algorithms for a given problem be?

Complexity Theory

Complexity measures:

program length } function only of the problem

(sequential) running time
storage space
} function of the input

parallel running time
number of processors

Possible Complexity measures

Static (data independent)

1. Program size (number of instructions)

Dynamic (data dependent)

1. Running time as a function of data size.

2. Storage space as a function of data size.

Data for dynamic measures

1. Worst case.

2. Representative (average) case.

Special measures for lower bounds

1. Tests in decision tree.

2. Arithmetic operations in straight-line program.

3. Memory accesses.

Program length and programming time: a digression

- Programming, from at least one point of view (Dijkstra's) is a rigorous, logical activity

  akin to theorem-proving and equally demanding of correctness.

Our Complexity Measure

Worst-case running time

as a function of input size.

Constant-time operations:

Accessing a single cell or node.

Performing a single arithmetic or logical operation.

Asymptotic analysis:

We ignore constant factors and concentrate on large problem sizes.

N

Complexity \ size	20	50	100	200	500	1000
$1000N$	.02 sec	.05 sec	.1 sec	.2 sec	.5 sec	1 sec
$100N\log N$	.09 sec	.3 sec	.6 sec	1.5 sec	4.5 sec	10 sec
$100N^2$	.04 sec	.25 sec	1 sec	4 sec	25 sec	2 min
$10N^3$	.02 sec	1 sec	10 sec	1 min	21 min	2.7 hr
$N\log N$	.4 sec	1.1 hr	220 days	125 cent	$5\cdot10^8$ cent	
$2^{N/3}$	.0001 sec	.1 sec	2.7 hr	$3\cdot10^4$ cent		
2^N	1 sec	35 yr	$3\cdot10^4$ cent			
3^N	58 min	$2\cdot10^9$ cent				

RUNNING time estimates:

ONE step = ONE microsecond

logarithms are base two

The Role of Theory in Algorithm Design

Whereas improvements in hardware and in coding can produce constant factor improvements, theoretical insights can lead to asymptotic improvements and gains in simplicity and generality (and correctness)

Key Points

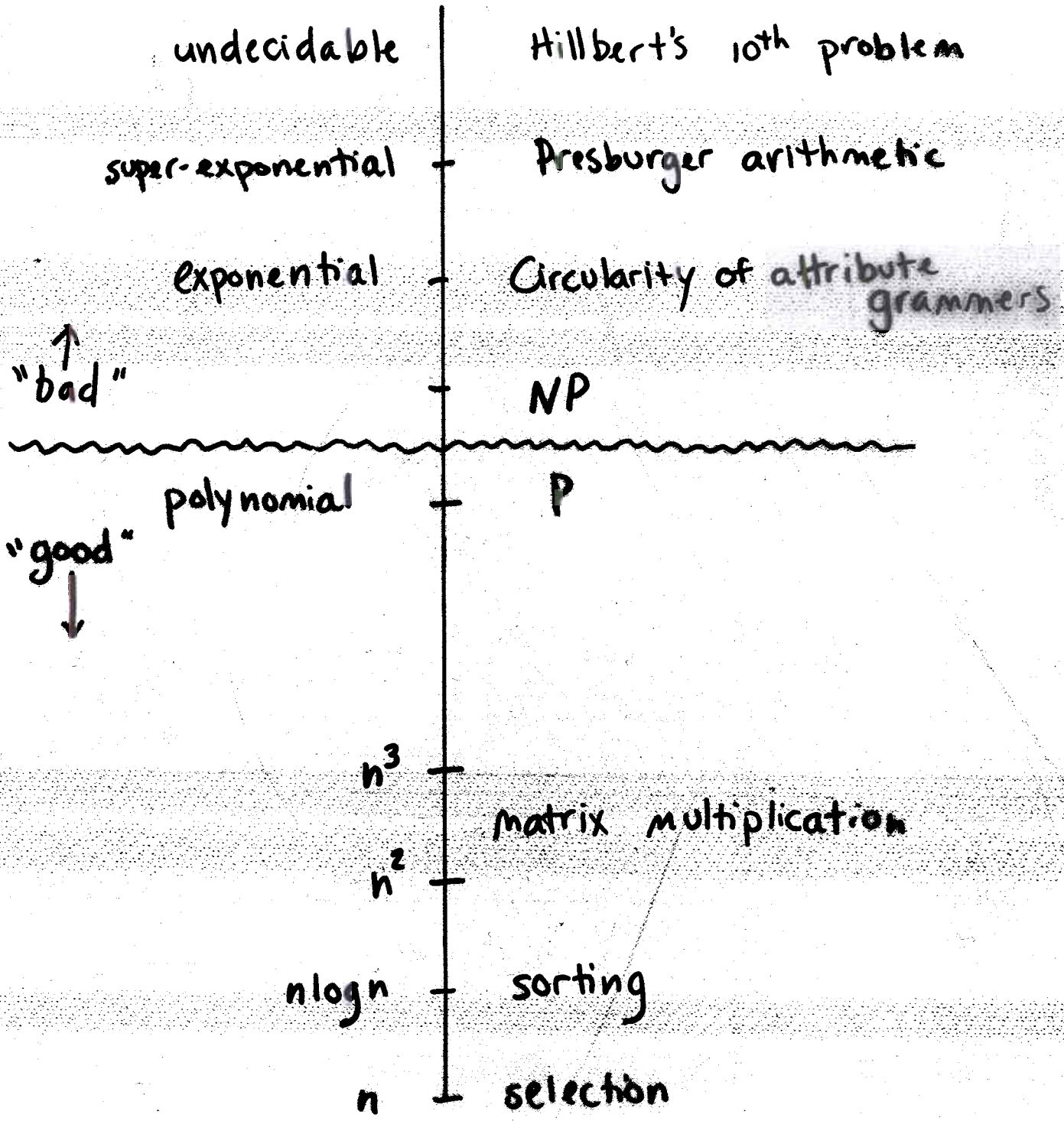
As computers become faster and as computer memories become larger, theoretical analysis yielding asymptotic complexity becomes more, not less important.

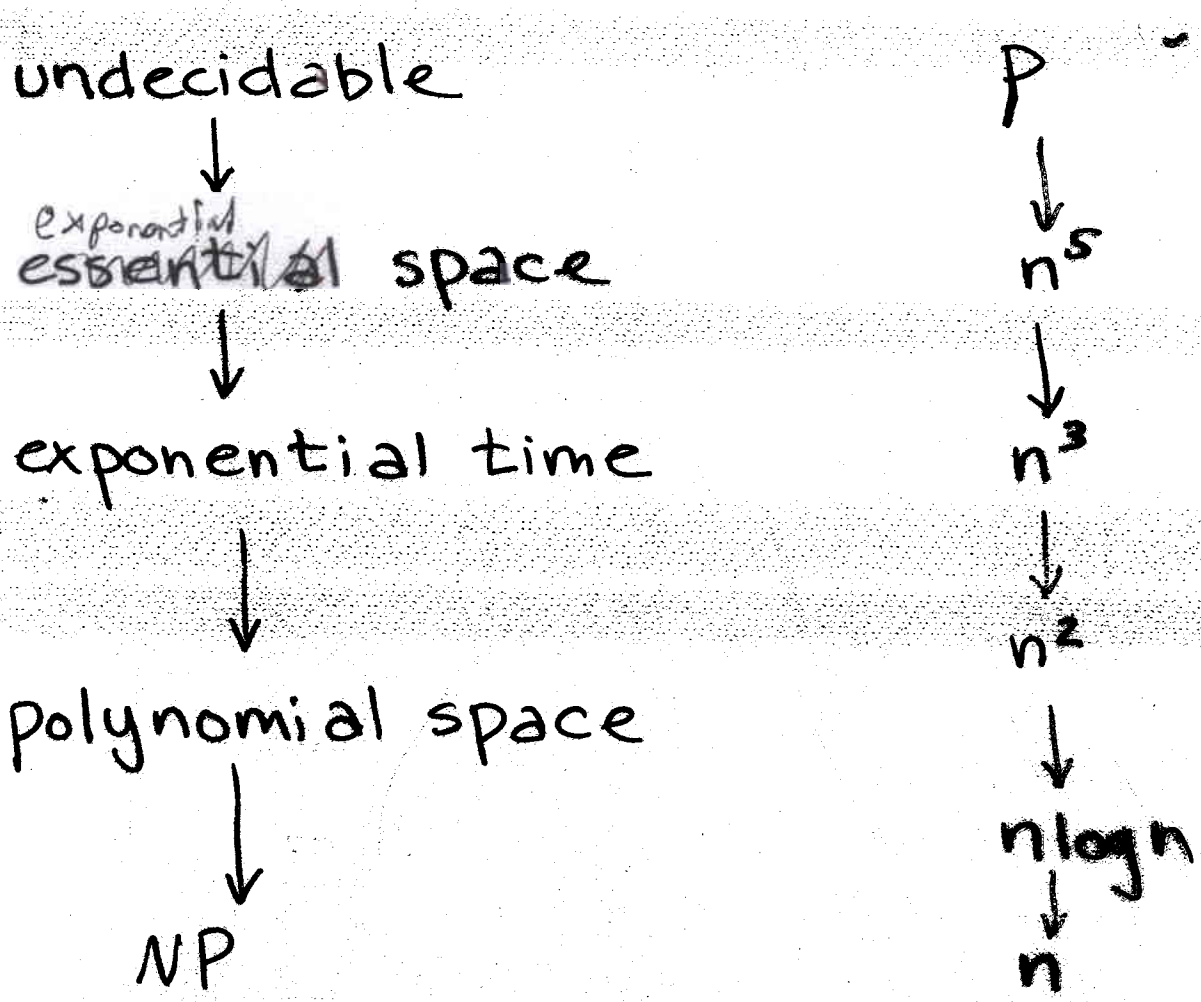
More "room" is available for the efficiency of clever algorithms to show up.

Often, the key to solving a problem efficiently is to use the right data structure.

Algorithmic questions are often at heart data structure questions.

The spectrum of Computational Complexity




undecidable
exponential space
exponential time
polynomial space
NP
P
n^5
n^3
n^2
nlogn
n

$$P = NP?$$

High order complexity

(What can't we do?)

Undecidability

Turing's halting problem

↓

Hilbert's tenth problem
(solution of Diophantine equations)

Good (polynomial time) vs **bad** (exponential time) algorithms

Exponential or **super-exponential** lower bounds

Equivalence of extended regular expressions

2^{2^n} Validity in Presburger arithmetic

Circularity in semantic definitions
for context-free languages

High-Level Complexity	vs.	Low-Level Complexity
Ignorance of e.g. polynomial functions	•	Ignorance of Constant Factors (maybe)
Emphasis on Lower bounds	•	Emphasis on upper bounds
Techniques are those of logic	•	Techniques are eclectic

simulation
diagonalization
quantifier elimination
(to get algorithms)

High order complexity results are machine independent;

complexity in all machine models is polynomially related.

Turing machine is usually used.

Key ideas

simulation (reducibility, transformability)

diagonalization

These are not powerful enough to resolve the

$P = NP?$

question.

NP-complete problems

P is the class of problems solvable in polynomial time.

NP is the class of problems whose solution can be checked in polynomial time.

NP-complete problems

Hardest problems in NP; if <u>any</u> has a polynomial time algorithm, they <u>all</u> do.

Examples

Validity in propositional calculus

Travelling salesman problem

Maximum independent set problem

Graph coloring

Algorithm: Step-by-step problem solving method.

Although "algorithms" existed thousands of years ago, the birth of the computer was necessary and sufficient to power their study.

Low order complexity

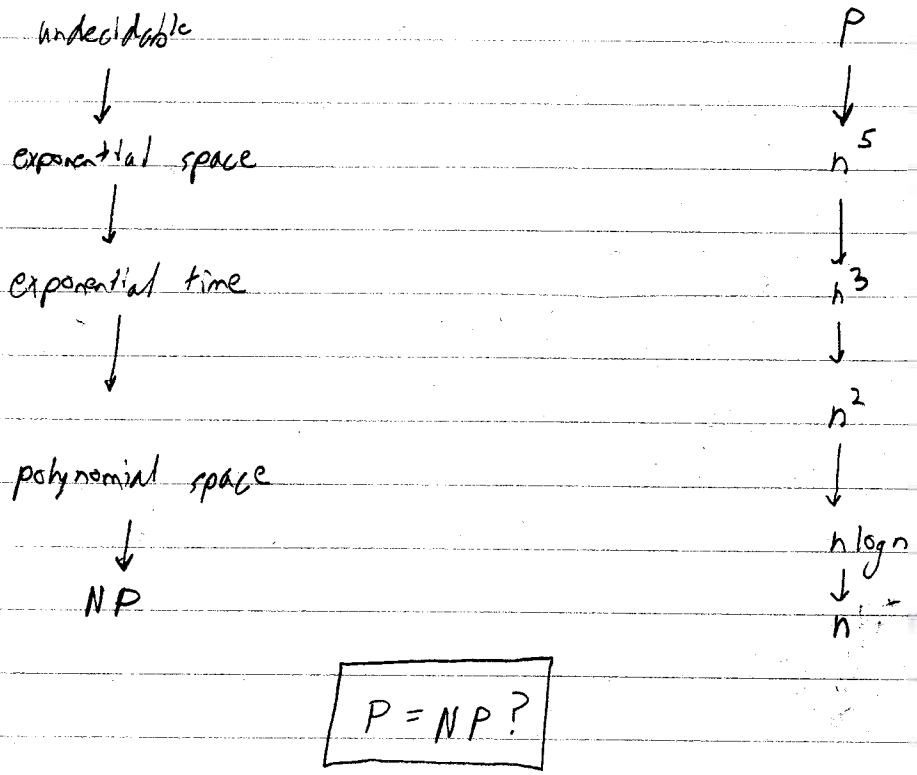
(What can we do?)

Instead of lower bounds, emphasis is on developing fast algorithms.

(Lower bounds almost non-existent)

Better-and-better polynomial upper bounds

High-Level Complexity	vs.	Low-Level Complexity
ignorance of e.g. polynomial functions		ignorance of constant factors (maybe)
emphasis on lower bounds		emphasis on upper bounds
techniques are those of logic simulation diagonalization quantifier elimination (to get algorithms)		techniques are eclectic



Low-Level Complexity

Lower bounds are based on

- problem-specific computation models

  count only the relevant or dominant operations

  eg. comparisons (sorting)

  multiplications (matrix multiplication)

model "natural" algorithms

Fast algorithms rely on

algorithmic techniques:
recursion, dynamic programming, divide-and-conquer, graph search, etc.

data structures: lists, stacks, queues, trees, etc.

Analysis of algorithms and related questions requires eclectic mathematics

Techniques

Recursion

- Dynamic programming
- Divide and Conquer

Data structures

- Linear lists
  - stack, queue, deque
  - list of lists (radix sorting)
  - partitioned stack (planarity testing)
- Trees
  - compressed trees
  - heaps (priority queues) } basic
  - search trees
  - dynamic trees
  - permutable trees } advanced

Graph search

Depth-first (maze traversal (Tremaux):
connectivity problems)
Breadth-first (network flow)
Shortest-first (shortest paths)
Oldest-last (maze traversal (Tarry):
Eulerian cycles)
Maximum cardinality
Lexicographic

Optimization

Greed
Augmentation

Odds + Ends

Sorting, Selecting, Searching

We can sort in $O(n \log n)$ time

Quicksort (average)

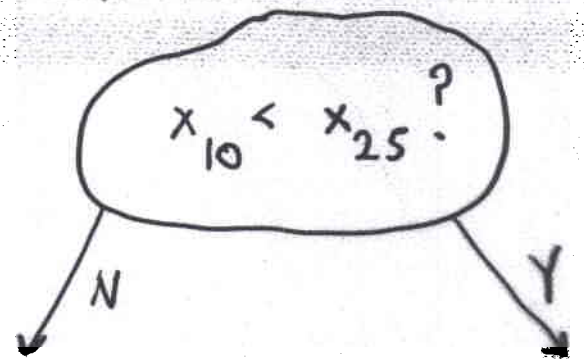
Merge sort (worst-case)

Is this bound tight?

Decision Tree Model

Input: a permutation of n numbers

At each node, ask any yes-no question about the permutation (comparisons a special case)



Each permutation reaches a different leaf

Depth = # questions ≤ time

Too broad: no accounting for deciding what questions to ask: program need not be of fixed size.

Too narrow: only binary decisions

There are $n!$ permutations on n items.

$\lceil \lg(n!) \rceil = \lceil \log_2(n!) \rceil$ is a lower bound on the worst-case depth.

Stirling's approximation for $n!$

$$n! \geq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\Rightarrow \lceil \log_2 n! \rceil \geq n \log_2 n - n/\ln 2$$

$\Rightarrow$ sorting takes $n \lg n - n/\ln 2$ comparisons (or binary decisions) (worst-case)

Information-Theoretic Lower Bound