Objects of this area of study:

- Develop good algorithms
- Analyze the complexity of algorithms
- Provide 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
- (sequential) running time
- storage space
- parallel running time
- number of processors

function only of the problem

function of the input
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.
<table>
<thead>
<tr>
<th>Complexity</th>
<th>20</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 $N$</td>
<td>.02 sec</td>
<td>.05 sec</td>
<td>.1 sec</td>
<td>.2 sec</td>
<td>.5 sec</td>
<td>1 sec</td>
</tr>
<tr>
<td>$100 \log_2 N$</td>
<td>.09 sec</td>
<td>.3 sec</td>
<td>.6 sec</td>
<td>1.5 sec</td>
<td>4.5 sec</td>
<td>10 sec</td>
</tr>
<tr>
<td>100 $N^2$</td>
<td>.04 sec</td>
<td>.25 sec</td>
<td>1 sec</td>
<td>4 sec</td>
<td>25 sec</td>
<td>2 min</td>
</tr>
<tr>
<td>$10 \cdot N^3$</td>
<td>.02 sec</td>
<td>1 sec</td>
<td>10 sec</td>
<td>1 min</td>
<td>21 min</td>
<td>2.7 hr</td>
</tr>
<tr>
<td>$N \log N$</td>
<td>.4 sec</td>
<td>1.1 hr</td>
<td>220 days</td>
<td>125 cent</td>
<td>$5 \cdot 10^6$ cent</td>
<td></td>
</tr>
<tr>
<td>$2^{N/3}$</td>
<td>.0001 sec</td>
<td>.1 sec</td>
<td>2.7 hr</td>
<td>$3 \cdot 10^4$ cent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^N$</td>
<td>1 sec</td>
<td>35 yr</td>
<td>$3 \cdot 10^4$ cent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$3^N$</td>
<td>58 min</td>
<td>$2 \cdot 10^9$ cent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 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
- super-exponential
- exponential
- polynomial
- "good"
- "bad"

- Hillbert's 10th problem
- Presburger arithmetic
- Circularity of attribute grammers
- $N^P$
- $P$

- $n^3$
- $n^2$
- $n \log n$
- $n$

- matrix multiplication
- sorting
- selection
undecidable

exponential space

exponential time

polynomial space

NP

\[ P \overset{n^5}{\longrightarrow} n^3 \overset{n^2}{\longrightarrow} n \log n \overset{n}{\longrightarrow} n \]

\[ P = NP ? \]
High order complexity

(What can’t we do?)

Undecidability

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
<table>
<thead>
<tr>
<th>High-Level Complexity</th>
<th>Low-Level Complexity</th>
</tr>
</thead>
</table>

- Ignorance of e.g. polynomial functions
- Constant Factor (maybe)

- Emphasis on lower bounds
- Emphasis on upper bounds

- Techniques are those of logic, 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 question.

\[ P = \text{NP?} \]
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 any has a polynomial time algorithm, they all do.

Examples

Validity in propositional calculus

Travelling salesman problem

Maximum independent set problem

Graph coloring

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 nonexistent)

Better-and-better polynomial upper bounds
High-level Complexity vs. Low-level Complexity

Ignorance of e.g. polynomial functions

Emphasis on lower bounds

Techniques are those of logic, simulation, diagonalization, quantifier elimination (to get algorithms)

Emphasis on upper bounds

Techniques are eclectic

Undecidable

Exponential space

Exponential time

Polynomial space

P

P = NP?
Low-Level Complexity

Lower bounds are based on problem-specific computation models
count only the relevant or dominant operations
e.g. 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)
  search trees
  dynamic trees
  permutable trees


Graph search

Depth-first (maze traversal (Trenaur), connectivity problems)
Breadth-first (network flow)
Shortest-first (shortest paths)
Oldest-first (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

Quick sort (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 \leq \text{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.  
\[
\lceil \log (n!) \rceil = \lceil \log_2 (n!) \rceil \quad \text{is a lower bound on the worst-case depth.}
\]

Stirling's approximation for \( n! \)  
\[
n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n
\]

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

\[
\Rightarrow \text{sorting takes } n \lg n - n/\ln 2 \text{ comparisons (or binary decisions)}
\]

\[
\text{(worst-case)}
\]

Information-Theoretic Lower Bound