Introduction to Computation Theory

Paige Zhu and Ronni Chang

(Mentored by Zoe Xi)

Table of Contents

0

Preliminaries

What is Computation Theory? How does it apply to the world?

Definitions

Alphabet - a finite non-empty set

ex. {A}, {1, 2, A}, {A, B, C, 0, 23}

String - finite sequence of symbols from an alphabet

ex. “0000”, “111”, “1ee3”, “aaa”

Language - a set of strings

ex. {A}, {A, B, C}, {1, 2, A}

01

Introduction

What is Computation Theory? How does it apply to the world?

What is Computation Theory?

- The study of the fundamental principles underlying computation and the analysis of algorithms
- Three major parts we'll talk about:
  - Automata theory
  - Computability theory
  - Complexity theory
- Aims to answer the question:
  - *What are the fundamental capabilities and limitations of computers?*

Parts of Computation Theory

Automata Theory

Explores the capabilities and limitations of computational models such as finite automata, and Turing machines

Computability Theory

Investigates what problems can be solved and cannot be solved via an algorithm

Complexity Theory

Studies the resources required to solve computational problems, including time and space, and aims to identify efficient algorithms for solving them

02

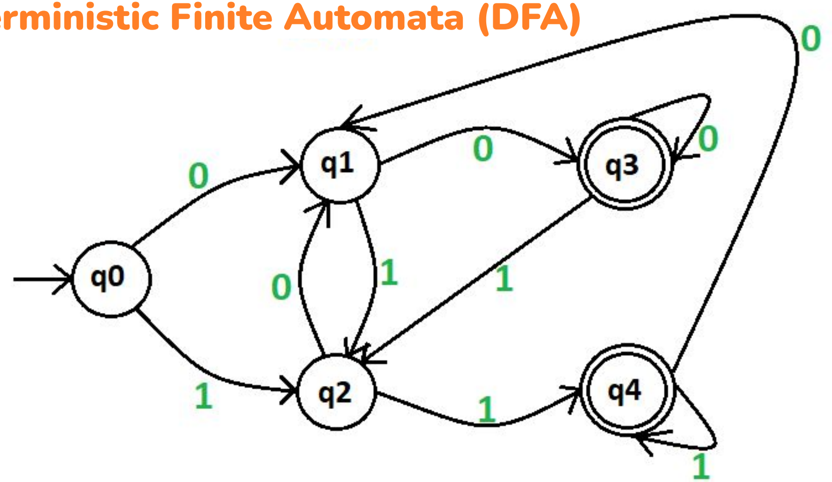
Finite Automata

What is finite automata? How can these be used to solve problems?

What is Finite Automata?

- A set of states (Q)
- A set of input symbols (alphabet, Σ)
- A transition function (δ)
- A start state (q_0)
- A set of accept states (F)

Deterministic Finite Automata (DFA)



Simple Example: Switching Light Bulb

States:

- On and Off

Input Symbols:

- Toggle, switching the bulb on or off

Transitions:

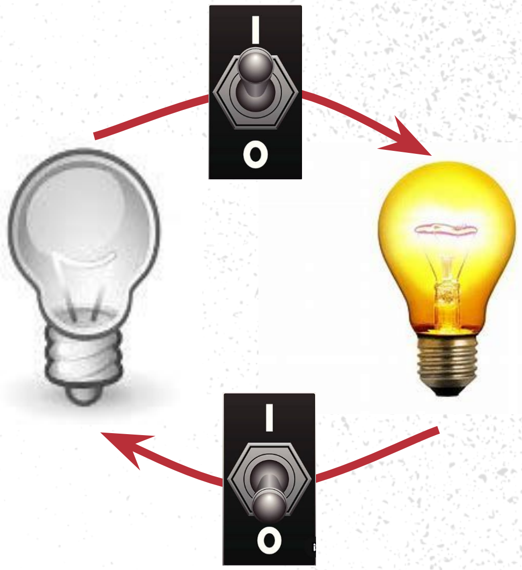
- From Off to On when the input is Toggle.
- From On to Off when the input is Toggle.

Start State:

- Off: Assuming the bulb starts in the Off state.

Accept States:

- Both On and Off can be considered accept states depending on the task. Both states are valid end conditions after an input.



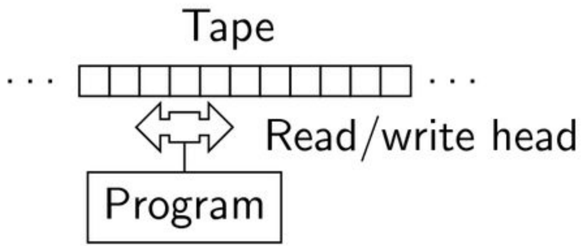
03

Turing Machines

What are Turing Machines? What can you discover from them?

What are Turing Machines?

- One of the first models of our modern computer
  - A precise model for the definition of an algorithm
- Very similar to finite automata, with one major difference: uses an infinite tape with an unlimited memory
- The construction: a TM uses a tape head to read and write symbols and move on a tape until entering an accept or reject state



Parts of a TM

States

These are the different 'modes' the function could be in.

Input Alphabet

All of the possible inputs.

Tape Alphabet

Contains all symbols that can be written onto the tape.

Transition Function

How the TM knows what to do at every iteration.

Start State

The 'mode' in which the TM begins.

Accept/Reject States

When you reach these states, your TM will return either accept/reject.

Example of a TM

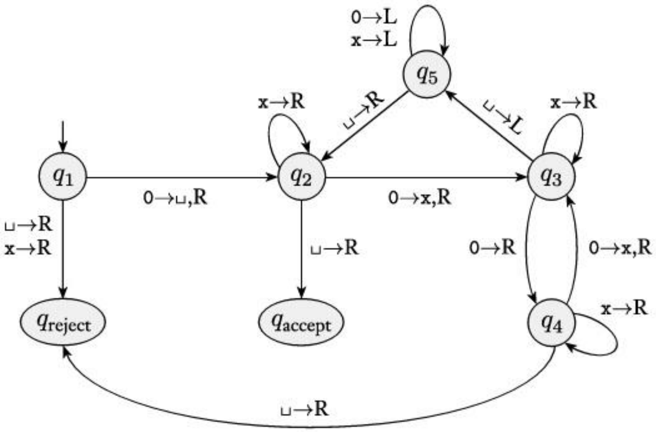
We construct as follows a TM M that decides the language $A = \{0^{(2^n)} \mid n \geq 0\}$. Describe M:

M = "On input string ω:

- Go from left to right and cross off every other 0.
- If there was only 1 zero in the first step, *accept.*
- If there were more than 1 zero in the first step and there were an odd number of zeros, *reject.*
- Go back to the very left of the tape.
- Go back to the first step.

M:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$
- $\Sigma = \{0\}$
- $\Gamma = \{0, \text{x}, \sqcup\}$
- δ as pictured below
- The start state is q_1, the accept state is q_{accept}, and the reject state is q_{reject}



04

Time Complexity

How do you evaluate how much time a function takes to compute?

What is Time Complexity?

The **time complexity** or **running time** of a Turing machine M is the function $f : N \to N$, where $f(n)$ is the maximum number of steps that M uses on any input of length n.

Big-O vs Small-o

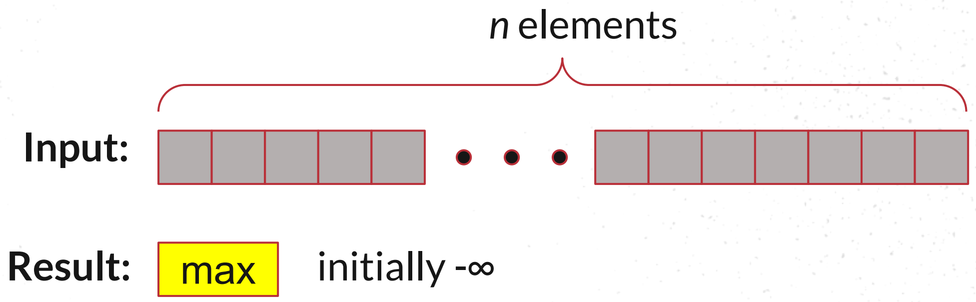
- Used to estimate the running time of an algorithm
- Asymptotic analysis
- Considering only highest order term (disregarding coefficient and lower order terms)

Big-O	Small-o
Inclusive upper bound (≤)	Strict upper bound (<)
$n = O(n)$	$2n = o(n^2)$
$16n = O(n)$	$2^n = o(3^n)$

*n is all positive real numbers

Example: Max Number in List

Problem: find the largest number in a list

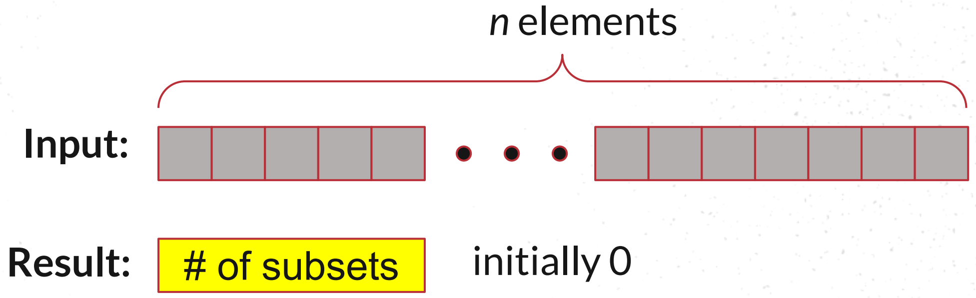


Algorithm: Loop over the elements of the list A. For every element $A[k]$, compare it with the current result R, and let $R = A[k]$ if $R < A[k]$.

Running Time: Every element needs 1 operation, so total time is n operations, denoted as $O(n)$.

Example: Subsets in a Set

Problem: find all possible subsets of a set of length n

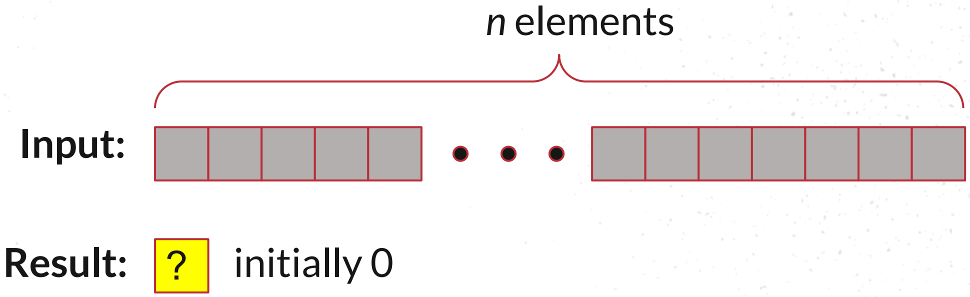


Algorithm: Loop over the elements in list A, we have two choices for each element: include it in the current subset or exclude it, and loop over A again until all subsets are found.

Running Time: There are 2^n subsets in a set so the running time is 2^n, denoted as $O(2^n)$ or $o(3^n)$.

Example: All Pairs Sum

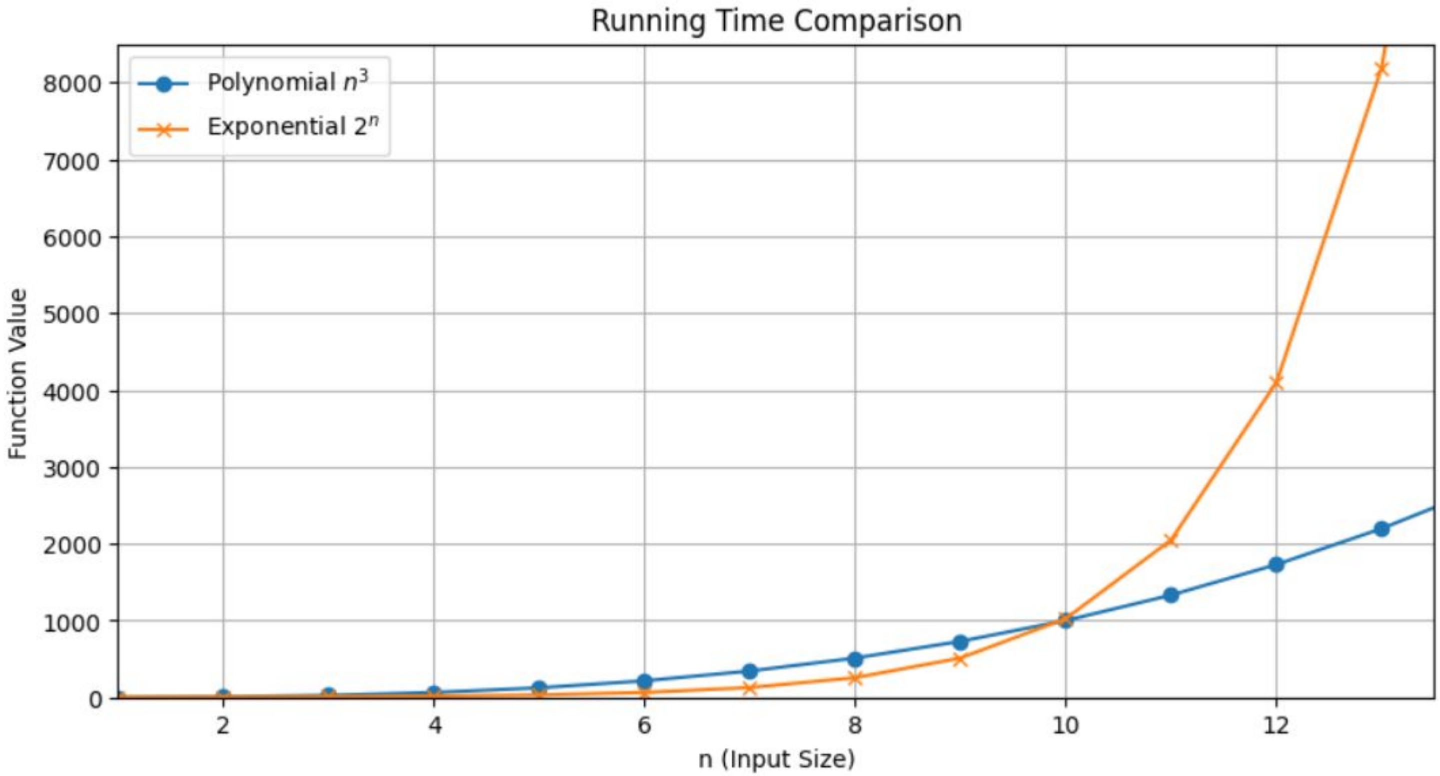
Problem: find the amount of pairs of numbers from a list



Algorithm: Loop over the elements of the array A. For every element $A[k]$, pair it with other elements $\{A[p] \mid p = k+1, \ldots, n\}$ to perform additions.

Running Time: $n(n\text{-}1)/2$ operations, but $< n^2$; i.e., upper bounded by n^2. Hence, denoted by $O(n^2)$.

Polynomial Time Preferred!



The Class P

- The class of algorithms that can be solved in polynomial time on a deterministic single-tape Turing machine

$$P = \bigcup_k \mathrm{TIME}(n^k).$$

- Problems that are realistically solvable on a computer

Acknowledgements

- MIT Math Department
- MIT PRIMES Program
- Marisa R Gaetz
- Mary Stelow
- Zoe Xi
- And all program sponsors!

Sources

“Practice Problems on Finite Automata.” GeeksforGeeks, GeeksforGeeks, 28 Aug. 2019, www.geeksforgeeks.org/practice-problems-finite-automata/.

Sipser, Michael. Introduction to the Theory of Computation. Cengage Learning, 2021.