Theory of Computation
Time Complexity

Arjun Chandrasekhar

Introduction to complexity theory

- So far we've studied what problems can (and can't) be solved by computers with theoretically unlimited resources
- In the real world, we have limited resources
  - time
  - memory
  - parallelism (i.e. number of processors)
  - randomness
- **Complexity theory:** what problems can (and can't) be solved *within specific resource constraints*

Worst case analysis

- We measure resources (e.g. time) using the Turing machine model of computation
- Resources are measured as a function $f : \mathbb{N} \to \mathbb{N}$ of the input length
  - $f(n)$ tells us the *maximum* number of resources the machine could use on *all possible* inputs of size n
  - "Worst case analysis"
- Input length n is the number of symbols in the input string on the tape
  - The input string may encode an object with a different size (e.g. graph with n vertices vs. adjacency matrix with n^2 elements)

Algorithm running time

$$L = \{0^k 1^k | k \geq 0\}$$

How "fast" is the following machine to decide L?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Repeat the following while both 0s and 1s are on the tape:
   2.1 Scan across the tape, erasing a single 0 and a single 1
3. If the tape is empty, accept. Otherwise, reject.

The machine runs in 5 seconds. Is that "fast"?

Algorithm running time

Physical running time

The physical running time of a machine is important! But it depends on...

- Hardware
- Input size/structure
- Perhaps the temperature of the room on that particular day?

None of these are properties of the actual *algorithm!*

Time Complexity

- Let M be a Turing machine
- **Def:** The **time complexity** of M is a function

$$T : \mathbb{N} \to \mathbb{N}$$

  where $T(n)$ is the maximum number of steps that M runs for on an input of length n
- We say "M runs in time $T(n)$"
- The running time of an algorithm is the running time of a TM that implements the algorithm

Time Complexity

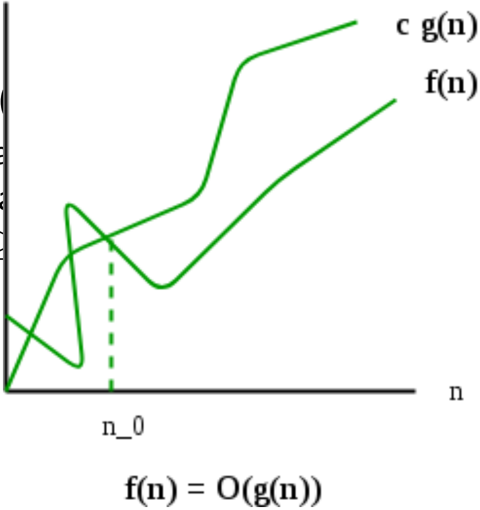
- Generally we don't care about the *exact* number of steps that the machine takes
- Instead, we ask: what is the relationship between the size of the input and the number of steps that the algorithm takes?
- What is the "order of magnitude" for the algorithm runtime?
- How does the algorithm "scale"?
  - As the input gets bigger, how many extra steps will the algorithm require?

Big-O Notation

- Let $f(n)$ and $g(n)$ be functions
- We say $f(n)$ is $O(g(n))$ if there exists a constant c, and a cutoff point n_0, such that for all $n \geq n_0$
$$f(n) \leq c \cdot g(n)$$

Big-O Notation

- Let $f($
- We sa … s a consta … ch that for all $n \geq$



Big-O Runtime

- Let $T(n)$ be the runtime for a machine M
- To convert $T(n)$ to Big-O notation:
  1. Remove all "lower order" terms
  2. Remove any constant factors
- **Example:**

$$
\begin{aligned}
T(n) &= 5n^3 + 17n^2\log(n) + 3.2n^{1.5} + 19747487584 \\
&\rightarrow 5n^3 \\
&\rightarrow O(n^3)
\end{aligned}
$$

Runtime Analysis Example

What is the time complexity of the following TM to decide $L = \{0^k 1^k | k \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Repeat the following while both 0s and 1s are on the tape:
   2.1 Scan across the tape, erasing a single 0 and a single 1
3. If the tape is empty, accept. Otherwise, reject.

Runtime Analysis Example

What is the time complexity of the following TM to decide $L = \{0^k1^k | k \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Repeat the following while both 0s and 1s are on the tape:
    2.1 Scan across the tape, erasing a single 0 and a single 1
3. If the tape is empty, accept. Otherwise, reject.

$O(n)$ to check the input format

Runtime Analysis Example

What is the time complexity of the following TM to decide $L = \{0^k 1^k | k \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Repeat the following while both 0s and 1s are on the tape:
   2.1 Scan across the tape, erasing a single 0 and a single 1
3. If the tape is empty, accept. Otherwise, reject.

$O(n)$ to check the input format

$O(n)$ loop iterations

Runtime Analysis Example

What is the time complexity of the following TM to decide $L = \{0^k 1^k | k \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Repeat the following while both 0s and 1s are on the tape:
   2.1 Scan across the tape, erasing a single 0 and a single 1
3. If the tape is empty, accept. Otherwise, reject.

$O(n)$ to check the input format

$O(n)$ loop iterations

$O(n)$ per loop iteration

Runtime Analysis Example

What is the time complexity of the following TM to decide $L = \{0^k 1^k | k \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Repeat the following while both 0s and 1s are on the tape:
   2.1 Scan across the tape, erasing a single 0 and a single 1
3. If the tape is empty, accept. Otherwise, reject.

$O(n)$ to check the input format
$O(n)$ loop iterations
$O(n)$ per loop iteration
$O(n) + O(n) \cdot O(n) = O(n^2)$

Complexity of 0^k1^k

- The language $L = \{0^k1^k | k \geq 0\}$ can be recognized in $O(n^2)$ time
- In fact, it an be recognized in $O(n \log n)$ time (Sipser)
- Can we do better?
  - It turns out, we cannot!
  - ...on a single-tape TM

Runtime Analysis Example

What is the time complexity of the following 2-tape TM to decide $L = \{0^n 1^n | n \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Read the 0's on tape 1, copy them onto tape 2
3. Read the 1's on tape 1, cross off 0's on tape 2
4. If the 0's and 1's run out at the same time, accept; otherwise reject.

Runtime Analysis Example

What is the time complexity of the following 2-tape TM to decide $L = \{0^n 1^n | n \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Read the 0's on tape 1, copy them onto tape 2
3. Read the 1's on tape 1, cross off 0's on tape 2
4. If the 0's and 1's run out at the same time, accept; otherwise reject.

$O(n)$ to check the input format

Runtime Analysis Example

What is the time complexity of the following 2-tape TM to decide $L = \{0^n 1^n | n \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Read the 0's on tape 1, copy them onto tape 2
3. Read the 1's on tape 1, cross off 0's on tape 2
4. If the 0's and 1's run out at the same time, accept; otherwise reject.

$O(n)$ to check the input format
$O(n)$ to read the 0's

Runtime Analysis Example

What is the time complexity of the following 2-tape TM to decide $L = \{0^n 1^n | n \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Read the 0's on tape 1, copy them onto tape 2
3. Read the 1's on tape 1, cross off 0's on tape 2
4. If the 0's and 1's run out at the same time, accept; otherwise reject.

$O(n)$ to check the input format
$O(n)$ to read the 0's
$O(n)$ to read the 1's

Runtime Analysis Example

What is the time complexity of the following 2-tape TM to decide $L = \{0^n 1^n | n \geq 0\}$?

1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Read the 0's on tape 1, copy them onto tape 2
3. Read the 1's on tape 1, cross off 0's on tape 2
4. If the 0's and 1's run out at the same time, accept; otherwise reject.

$O(n)$ to check the input format
$O(n)$ to read the 0's
$O(n)$ to read the 1's
$O(n) + O(n) + O(n) = O(n)$

Common Runtimes

- $O(1)$ – "constant"
- $O(\log(n))$ – "logarithmic"
- $O(n)$ – "linear"
- $O(n^2)$ – "quadratic"
- $O(n^c) = n^{O(1)} = c^{O(\log(n))}$ – "polynomial"
- $O(2^{n^c})$ – "exponential"

Models of computation in complexity

- Our choice of model of computation did *not* affect our computability results
  - A single-tape Turing machine is just as *robust* as any other model
- The previous example shows that our choice of model *does* affect complexity results
  - A single-tape Turing machine isn't as *fast* as some other models
- For the rest of this course, a single-tape TM will still suffice (but we need to justify this)
- For an algorithms course, we typically analyze complexity using models that are more expressive than a single-tape TM

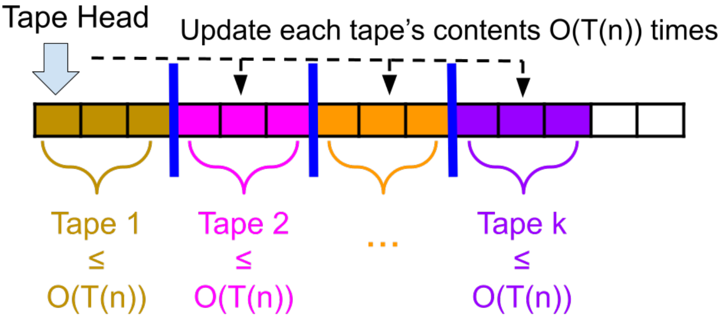
Complexity of multi-tape TMs

Theorem: Any language that can be recognized by a k-tape TM in $O(T(n))$ time can be recognized by a single-tape TM in $O(T(n)^2)$ time

Proof Idea:

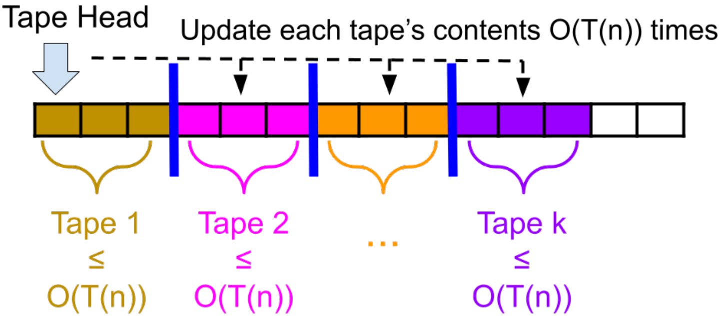
- Simulate the original k tapes on k separate sections of the single tape
- $O(T(n))$ simulation rounds
- $O(T(n))$ steps per round
- **Remark:** If a TM runs in $O(T(n))$ time, it touches at most $O(T(n))$ tape squares

Complexity of multi-tape TMs



1. Repeat the following $O(T(n))$ times:
   1.1 Scan across the tape, and update each tape's contents

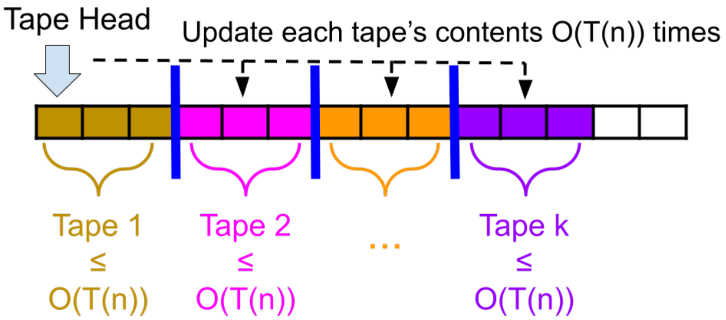
Complexity of multi-tape TMs



1. Repeat the following $O(T(n))$ times:
   1.1 Scan across the tape, and update each tape's contents

$O(T(n))$ rounds

Complexity of multi-tape TMs

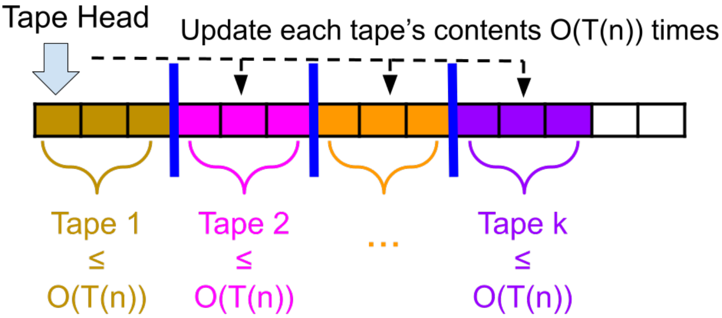


1. Repeat the following $O(T(n))$ times:
   1.1 Scan across the tape, and update each tape's contents

$O(T(n))$ rounds

$k \cdot O(T(n)) = O(T(n))$ to scan the k sections

Complexity of multi-tape TMs



1. Repeat the following $O(T(n))$ times:
   1.1 Scan across the tape, and update each tape's contents

$O(T(n))$ rounds
$k \cdot O(T(n)) = O(T(n))$ to scan the k sections
$O(T(n)) \cdot O(T(n)) = O(T(n)^2)$

Complexity of multi-tape TMs

- It is often more convenient to describe our algorithm with a multi-tape TM
- We only incur a polynomial slowdown when we convert the algorithm to a single-tape TM
- We will see that this is good enough for the problems we are exploring in this course

Extended Church-Turing Thesis

Anything that can be computed in time $O(T(n))$ on a "physical computer" can be computed in time $O(T(n)^c)$ on a Turing machine

- An algorithm on any type of machine can be converted to a TM algorithm with only a polynomial-time slowdown
- **TMs formalize our intuitive notion of (efficient) algorithms**
- Quantum computers may prove to be an exception