Chapter 7: Time Complexity

- How do we know when we have such a problem?
- Do certain decidable problems require too much time?
- How can we classify problems according to the amount of time required?
- How do we measure time?
- Let's study time complexity of problems.
- But might not be solvable in practice.
- Whether something is computationally solvable.
- We just looked at decidability.

Overview

- Chapter Introduction
- Complexity Relationships among Models
- Analyzing Algorithms
- Measuring Complexity
- Chapter Introduction
Example

- For this course, just compute this in terms of the length of the input
- How much time does $M_1$ use?

4. If $0$s still remain or if $1$s still remain reject; otherwise accept
3. Scan across the tape, crossing off a single 0 and a single 1
2. Repeat if both 0$s and 1$s remain on the tape:
   1. Scan across tape and reject if a 0 is found to the right of a 1
   2. If 1 is found to the left of a 0, scan across tape and reject.; otherwise accept

- $A$ is decidable (since any CFL is decidable, Theorem 4.9)

$\{0 \geq \gamma | \gamma \geq 0\} = A$

Overview

- Complexity Relationships among Models
- Analyzing Algorithms
- Measuring Complexity
- Chapter Introduction
Asymptotic Analysis

- Understand what the algorithm does on large $n$.
- Only consider the highest order term of the running time.
- Disregard lower order terms.
- Disregard coefficient of the term.
- As $n$ increases, this term will dominate.
- Only consider the highest order term of the running time.

Example

$f(n) = 6n^3 + 2n^2 + 20n + 45$

- Highest order term is $6n^3$.
- Disregarding the coefficient $6$, we say that $f$ is asymptotically at most $n^3$.

Time Complexity Definition

Definition 7.1: Let $M$ be a deterministic Turing machine that halts on all inputs. The running time of $M$ is the function $f : N \to N$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$.

Definition 7.1: Let $M$ be a deterministic Turing machine that halts on all inputs. The running time of $M$ is the function $f : N \to N$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$.

Definition 7.1: Let $M$ be a deterministic Turing machine that halts on all inputs. The running time of $M$ is the function $f : N \to N$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$.
Examples

• Example: \( f_1(n) = 5n^3 + 2n^2 + 22n + 6 \)

• Example: \( u = (u)f \)

• Example: \( f_2(n) = \log_2 n \)

• Example: \( f_3(n) = 3n\log_2 n + 5n\log_2 \log_2 n + 2 \)

In most cases, it will be obvious what the highest order term is. If we disregard differences up to a constant factor, \( f = O(g) \) intuitively means that \( f \) is less than or equal to \( g \) if we disregard differences up to a constant factor. For more precisely, \( f = \Theta(g) \) is an asymptotic upper bound for \( f \). We say that \( (u)g \) is an upper bound for \( f \) when \( (u)g \subseteq (u)f \).

\[ (u)g \subseteq (u)f \quad \text{and} \quad 0u < u^0 \text{ exist such that for } \forall n \in \mathbb{N} : b \cdot f \text{ be functions} \]

Definition 7.2: Let \( f \) and \( g \) be functions. \( f = \Omega(g) \) if there exist positive integers \( c \) and \( n_0 \) such that for all \( n \geq n_0 \), we have \( c \cdot g(n) \leq f(n) \).

\[ \Omega \]

\[ \text{Big} \]
Overview

Complexity Relationships among Models

Analyzing Algorithms

Measuring Complexity

Chapter Introduction

More on Big-O

Exponential bounds: \( O(u^c) \) for some \( c > 0 \)

Polynomial bounds: \( O(u^c) \) for some \( c \)

Also says that \( (u)f \) has an upper bound \( u^c \) for some \( c \)

Example: \( (u^c) = (u)^c \)

So \( (u)f \) has an upper bound \( u^c \) for some \( c \)

Example: \( u^c = u \) and thus \( u^c = u^c \)

Example: \( (u^c) = (u)^c \)

An upper bound of \( (u)^c \) for some constant \( c \)

Example: \( (u^c) = (u)^c \)

Example: \( (u)O + (u)^c = (u)f \)
Time Complexity Class

Definition 7.7: Let \( t : \mathbb{N} \rightarrow \mathbb{R}^+ \) be a function. Define the time complexity class, \( \text{TIME}(t(n)) \), to be the collection of all languages that are decidable by an \( O(t(n)) \) time Turing Machine.

- Is there a machine that decides \( \lambda \) asymptotically more quickly? (\( O((u)) \) \( \forall \in \text{TIME}(u) \) \( O(0) = V \))

Languages that are decidable by an \( O(n^2) \) time Turing Machine.

Analyzing Algorithms

- A = \{0^k1^k \mid k \geq 0\}

- A \( \in \text{TIME}(n^2) \) because \( M_1 \) decides \( A \) in time \( O(n^2) \)

- Is there a machine that decides \( \lambda \) asympotically faster than \( O(n^2) \)?

- Only if we can find something that is more than a linear speedup.

- Time Complexity Class, \( \text{TIME} \), to be the collection of all languages that are decidable by a function, \( \lambda : \mathbb{N} \rightarrow \mathbb{R}^+ \). Define the time complexity class, \( \text{TIME}(t(n)) \), to be the collection of all languages that are decidable by an \( O(t(n)) \) time Turing Machine.

- Time Analyze:

  1. If \( 0 \) is still remain or \( 1 \) is still remain \( \text{reject} \) otherwise accept
  2. Repeat if both 0 and 1 remain on the tape
  3. Scan across the tape, crossing off a single 0 and a single 1
  4. If 0 or 1 remain on the tape, \( \text{reject} \) otherwise accept

- Total time \( \text{time} \) \( O(u) \) + \( O(u) \) + \( O(u) \)

- Step 1: Each pass takes \( O(u) \) scans at most: \( O(u) \) \( \forall \in \text{TIME}(u) \) \( O(0) = V \).

- Step 2: \( u \) steps to scan, then \( u \) steps to resolution to scan or tape

- Time Analyze:

  1. Scan across the tape, crossing off a single 0 and a single 1
  2. Repeat if both 0 and 1 remain on the tape
  3. Scan across the tape, crossing off a single 0 and a single 1
  4. If 0 or 1 remain on the tape, \( \text{reject} \) otherwise accept

  \{0 \leq y \leq n \} = V
Two Tape Solution

If you have a 2-tape machine

1. Scan across tape and reject if a 0 is found to the right of a 1
2. Scan across the 0s on tape 1 until the first 1. At the same time, copy the 0s onto tape 2.
3. Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all 1s are read, reject.
4. If 0s still remain, reject; otherwise accept.

Time of this machine is \( O(n) \)

A Faster Machine

\( A = \{0^k1^k \mid k \geq 0\} \)

1. Scan across tape and reject if a 0 is found to the right of a 1
2. Repeat if both 0s and 1s remain on the tape:
3. Scan tape, check whether total number of Os and 1s remaining is even or odd. If it is odd, reject.
4. Scan tape, crossing off every other 0 starting with first 0. Cross off every other 1 starting with first 1.
5. If 0s still remain or 1s still remain, reject; otherwise accept.

\( M_2 \) has running time of \( O(n \log n) \) time, each iteration taking at most \( O(n) \)

So \( A \in \text{TIME}(n \log n) \)

Loop \( I + \log_2 n \) times, each iteration taking at most \( O(n) \) time

On input string \( w \):

\[ \{0 \geq \lceil \sqrt[3]{|w|} \rceil \} = V \]
Overview

Computability versus Complexity

- Church-Turing thesis says that all reasonable models of computation are equivalent.
- For most deterministic models, it does not differ greatly, so choice is not critical.
- Choice of model does affect complexity.
- So, doesn’t really matter which you use.
Time Analysis

- Overall time is $O((u)O) = O((v)O)$
  - Optionally inverse tape $y$ time to insert space at each tape,
  - Traverses tape to read contents of $y$ tape
  - Each simulated step of $M$ takes $O((u)O)$ time
  - $S$ tape can be of length at most $t(n)$
  - Say $M$ takes time $O((u)O)

- How much time?

---

Theorem 7.8: Let $(u)$ be a function. Where $(u) \geq n$. Then every

Single versus Multi-Tape

Recall from Chapter 3
Theorem 7.11: Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape TM has an equivalent $2^{O(t(n))}$ time deterministic single-tape TM.

Recall from Chapter 3

Nondeterminism and Complexity

- A nondeterministic TM is a decider if it halts on all branches.
- A nondeterministic TM is a decider if one branch accepts.
- Even if one branch accepts, all other branches must halt as well.

Note: Running time is just defined for TMs that are deciders.

Not intended to correspond to any real-world computing device.

Rather, a useful mathematical definition that assists in characterizing the complexity of an important class of computational problems.
Can it actually be done in polynomial time?

So, if a problem can be decided in polynomial time on a nondeterministic single-tape TM, we know it can be decided in exponential time on a deterministic single-tape TM.

Can it be done faster?
Can it actually be done in polynomial time?

Time Analysis

Let $N$ be a nondeterministic TM with time $O(t(n))$.

Let $b$ be the maximum number of choice points for any transition.

The maximum number of leaves in the computation tree is $b \cdot t(n)$.

The maximum number of nodes (including leaves) will be at most twice $b \cdot t(n)$ (a binary tree):

$O(b \cdot t(n))$

Each node will be computed in at most $O((u)Q)$ steps.

The maximum number of levels in the computation tree is $\lfloor \log_b \cdot (u)Q \rfloor$.

Let $N$ be a nondeterministic TM with time $O((u)Q)$.