Chapter 7: Time Complexity

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

- \( A = \{0^k1^k \mid k \geq 0\} \)
  - \( A \) is decidable (since any CFL is decidable Theorem 4.9)
  - \( M_1 \) can decide \( A \)
    \( M_1 = \) “On input string \( w \):
    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 across the tape, crossing off a single 0 and a single 1
    4. If 0s still remain or if 1s still remain reject; otherwise accept
  - How much time does \( M_1 \) take?
  - For this course, just compute this in terms of the length of the input
Time Complexity Definition

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

- If $f(n)$ is the running time of $M$
  - we say that $M$ runs in time $f(n)$
  - we say that $M$ is an $f(n)$ time TM

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

- 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 $n^3$
Big O

**Definition 7.2:** Let $f$ and $g$ be functions $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$. We say that $f(n) = O(g(n))$ if positive integers $c, n_0$ exist such that for every integer $n \geq n_0$

$$f(n) \leq cg(n)$$

When $f(n) = O(g(n))$ we say that $g(n)$ is an **upper bound** for $f(n)$, or more precisely, that $g(n)$ is an **asymptotic upper bound** for $f(n)$, to emphasize we are suppressing constant factors.

- Intuitively, $f(n) = O(g(n))$ means that $f$ is less than or equal to $g$ if we disregard differences up to a constant factor.
- In most cases, it will be obvious what the highest order term is.

Examples

- Example: $f_1(n) = 5n^3 + 2n^2 + 22n + 6$

- Example: $f_2(n) = \log_2 n$

- Example: $f_3(n) = 3n \log_2 n + 5n \log_2 \log_2 n + 2$
More on Big-O

- Example: \( f(n) = O(n^2) + O(n) \)

- Example: \( f(n) = 2^O(n) \)
  - An upper bound of \( 2^n \) for some constant \( c \)

- Example: \( f(n) = 2^O(\log n) \)
  - \( n = 2^{\log_2 n} \) and thus \( n^c = 2^{c \log_2 n} \)
  - So \( f(n) \) has an upper bound of \( n^c \) for some \( c \)

- Example: \( f(n) = n^O(1) \)
  - Also says that \( f(n) \) has an upper bound \( f n^c \) for some \( c \)

- **Polynomial Bounds**: \( O(n^c) \) for some \( c > 1 \)

- **Exponential bounds**: \( O(2^{n^\delta}) \) for some \( \delta > 1 \)

Overview

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

• $A = \{0^k1^k | k \geq 0\}$

  $M_1 =$ “On input string $w$:
  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 (noted from prev pass):
  3. Scan across the tape, crossing off a single 0 and a single 1
  4. If 0s still remain or if 1s still remain reject; otherwise accept

• Time Analysis:
  - Step 1 takes $n$ steps to scan, then $n$ steps to reposition to start of tape
  + We didn’t mention repositioning, but does change big-O anyways $O(n)$
  - Step 2 & 3: Each pass takes $O(n)$ steps and $n/2$ scans at most: $O(n^2)$
  - Step 4: another pass $O(n)$ time
  - Total time $O(n) + O(n^2) + O(n)$ or $O(n^2)$

Time Complexity Class

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

• $A = \{0^k1^k | k \geq 0\}$
  - $A \in$ TIME($n^2$) because $M_1$ decides $A$ in time $O(n^2)$

• Is there a TM that decides $A$ asymptotically more quickly?
  - Is $A$ in a time class that is faster than $O(n^2)$?
  - Only if we can find a TM that gives more than a linear speedup
Two Tape Solution

- If you have a 2-tape machine
  
  \[ M_2 = \text{"On input string } w:\}
  
  \begin{enumerate}
  \item Scan across both tapes and reject if a 0 is found to the right of a 1.
  \item If there is an odd number of 0s on one tape and 1s on the other, cross off 0s and copy 1s onto the other tape.
  \item Scan across both tapes and reject if an 1 is found on tape 1 and a 0 on tape 2.
  \item For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, accept.
  \item If 0s still remain, reject; otherwise, accept.
  \end{enumerate}

- Time of this machine is \( O(n) \) time.

- Can decide \( A \) in \( O(n) \) time if TM has a second tape.

- Time complexity depends on the model of computation.
Computability versus Complexity

- **Computability**
  - Church-Turing thesis says that all reasonable models of computation are equivalent
  - So, doesn’t really matter which you use

- **Complexity**
  - Choice of models does affect complexity
  - For most deterministic models, it does not differ greatly, so choice is not crucial

Overview

- Chapter Introduction
- Measuring Complexity
- Analyzing Algorithms
  $\Rightarrow$ Complexity Relationships among Models
Single versus Multi-Tape

**Theorem 7.8:** Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape TM has an equivalent $O(t^2(n))$ time single-tape TM.

- Recall from Chapter 3
  - Say that $M$ has $k$ tapes. Construct $S$ to work as follows
  - Store contents of $k$ tapes on single tape with ‘#’ between each
    - Place a marker where each of $M$’s tape heads would be
  - For each move of $M$, $S$ does the following
    + Scans the entire tape to find out what is under the $k$ tape heads of $M$
    + Updates its state, and for each of the $k$ tapes, updates it and the tape position
    + If reaches the end of a tape, will need to move the subsequent tapes one character to the right

Time Analysis

- How much time?
- Say $M$ takes time $O(t(n))$
  - $S$’s tape can be of length at most $t(n) \times k = O(t(n))$
  - Each simulated step of $M$ takes $S$ time $O(t(n))$
    - Traverse tape to read contents of $k$ tapes
    - Traverse tape to update contents and tape head
    - Optionally traverse tape $k$ times to insert space at each of each tape
- Overall time is $O(t(n)) \times O(t(n)) = O(t^2(n))$
Nondeterminism and Complexity

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

**Definition 7.9:** Let \( \mathcal{N} \) be a nondeterministic TM that is a decider. The **running time** of \( \mathcal{N} \) is the function \( f : \mathcal{N} \rightarrow \mathbb{N} \), where \( f(n) \) is the maximum number of steps that \( \mathcal{N} \) uses on any branch of its computation on any input of length \( n \) (e.g. longest branch).

• 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

• Note: running time is just defined for TMs that are deciders (deterministic or nondeterministic)

Complexity of a Nondeterministic Machine

**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
  - Use 3 tapes: input tape, simulation tape, address tape
    - While
      + Copy input tape to simulation tape
      + Run computation according to choices on address tape
      + If we accept, \textit{accept}
      + Otherwise, if a choice is invalid, or finish the sequence
      + Increment the address tape
**Time Analysis**

- Let $N$ be a nondeterministic TM with time $O(t(n))$
  - Let $b$ be maximum number of choice points for any transition
- Maximum number of leaves in computation tree is $b^{t(n)}$
  - Maximum number of nodes (including leaves) will be at most twice $b^{t(n)}$ (maximum is obtained when it is a binary tree): $O(b^{t(n)})$
  - Each node will be computed in at most $O(t(n))$ steps
- Computation on 3-tape machine takes $O(t(n) \cdot b^{t(n)}) = O(b^{t(n)+1}) = 2^{O(t(n))}$
- Computation on a 1-tape machine takes $(2^{O(t(n))})^2 = 2^{2O(t(n))} = 2^{O(t(n))}$
- Example: if $N$ is polynomial time $O(n^c)$, deterministic version can be done in time $2^{O(n^c)}$ time: exponential
  + This is a worse case. There might be a better implementation

**Prelude**

- 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?