Chapter 14

Time Complexity
Time Complexity

- The study of *complexity of a problem* is the study of the *complexity of the algorithm* that solves the problem.

- The computational complexity of an algorithm is measured by the amount of *resources* required to carry it out, i.e., *time* and *space*.

- The *time complexity* of a computation $C$ is determined by the amount of time required to perform $C$, whereas the *space complexity* of $C$ is determined by the amount of *storage space* required by $C$. 
14.2 Rates of Growth

- The rate of growth of a function, which measures the increase of the function values as the input gets arbitrarily large, is determined by the most significant contributor to the growth of the function.

- Table 14.2 (Growth of functions)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>25</th>
<th>50</th>
<th>100</th>
<th>1,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$20n + 500$</td>
<td>500</td>
<td>600</td>
<td>700</td>
<td>1,000</td>
<td>1,500</td>
<td>2,500</td>
<td>20,500</td>
</tr>
<tr>
<td>$n^2$</td>
<td>0</td>
<td>25</td>
<td>100</td>
<td>625</td>
<td>2,500</td>
<td>10,000</td>
<td>1,000,000</td>
</tr>
<tr>
<td>$n^2 + 2n + 5$</td>
<td>5</td>
<td>40</td>
<td>125</td>
<td>680</td>
<td>2,605</td>
<td>10,205</td>
<td>1,002,005</td>
</tr>
<tr>
<td>$n^2 / (n^2 + 2n + 5)$</td>
<td>0</td>
<td>0.625</td>
<td>0.800</td>
<td>0.919</td>
<td>0.960</td>
<td>0.980</td>
<td>0.998</td>
</tr>
</tbody>
</table>

- the linear and constant terms of $n^2 + 2n + 5$ are called the lower-order terms, which do not significantly contribute to the growth of the function values.
Defn 14.2.1  Let \( f : \mathbb{N} \rightarrow \mathbb{N} \) and \( g : \mathbb{N} \rightarrow \mathbb{N} \) be one-variable number-theoretic functions.

i. \( f \) is said to be of order \( g \), written \( f \in O(g) \), if there is a positive constant \( c \) and natural number \( n_0 \) such that

\[
f(n) \leq c \times g(n), \quad \forall n \geq n_0
\]

ii. \( O(g) = \{ f \mid f \text{ is of order } g \} \), i.e., the set of all functions of order \( g \), is called the “big oh of \( g \)”

\( f \) is of order \( g \), written \( f \in O(g) \), if the growth of \( f \) is bounded by a constant multiple of the values of \( g \)
14.2 Rates of Growth

- The rate of growth is determined by the most significant contributor to the growth of the function.

- If \( f \in O(g) \) and \( g \in O(f) \), then given two positive constants \( C_1 \) and \( C_2 \),

\[
\begin{align*}
  f(n) &\leq C_1 \times g(n), \quad \forall n \geq n_1; \\
g(n) &\leq C_2 \times f(n), \quad \forall n \geq n_2
\end{align*}
\]

- \( f \) and \( g \) have the same rate of growth, i.e., neither \( f \) nor \( g \) grow faster than the other.
14.2 Rates of Growth

- A function $f$ is said to **exponentially** (polynomially, respectively) bounded if $f \in O(2^n)$ ($f \in O(n^r)$, respectively).

- **Example 14.2.2** Let $f(n) = n^2 + 2n + 5$ and $g(n) = n^2$. Then
  
  $g \in O(f)$, since $n^2 \leq n^2 + 2n + 5$, $\forall n \in \mathbb{N}$, $n_0 = 0$, and $C = 1$.
  
  $f \in O(g)$, since $2n \leq 2n^2$ and $5 \leq 5n^2$, $\forall n \geq 1$.
  
  Then $f(n) = n^2 + 2n + 5 \leq n^2 + 2n^2 + 5n^2 = 8n^2 = 8 \times g(n)$, $\forall n \geq 1$.
  
  Thus $f \in O(n^2)$

- **Example 14.2.1** Let $f(n) = n^2$ and $g(n) = n^3$. $f \in O(g)$, but $g \notin O(f)$.

  Clearly, $n^2 \in O(n^3)$, since $n^2 \leq n^3$, for $\forall n \in \mathbb{N}$, $n_0 = 0$, and $C = 1$.

  Suppose that $n^3 \in O(n^2)$. Then there exists constants $C$ and $n_0$ such that
  
  $n^3 \leq C \times n^2$, $\forall n \geq n_0$.

  Let $n_1 = \max(n_0 + 1, C + 1)$. Then
  
  $n_1^3 = n_1 \times n_1^2 > C \times n_1^2$, since $n_1 > C$,

  contradicting the inequality that $n_1^3 \leq C \times n_1^2$. Thus $n^3 \notin O(n^2)$. 
14.2 Rates of Growth

- Using *limits* to determine the asymptotic complexity of two functions

- Let \( f \) & \( g \) be two number-theoretic functions

  - If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \), then \( f \in O(g) \), but \( g \notin O(f) \)

  - If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = c \) with \( 0 < c < \infty \), then \( f \in \Theta(g) \), and \( g \in \Theta(f) \)

    where \( \Theta(g) = \{ f \mid f \in O(g) \text{ and } g \in O(f) \} \)

  - If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \), then \( f \notin O(g) \) and \( g \in O(f) \)

- **Example.** Let \( f(n) = n^2 \) and \( g(n) = n^3 \). \( f \in O(g) \), but \( g \notin O(f) \).

  Since \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{1}{n} = 0 \), \( f \in O(g) \), but \( g \notin O(f) \).
# 14.2 Rates of Growth

<table>
<thead>
<tr>
<th>Big Oh (Big O)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>constant</td>
</tr>
<tr>
<td>$O(\log_a(n))$</td>
<td>logarithmic</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>linear</td>
</tr>
<tr>
<td>$O(n \log_a(n))$</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>quadratic</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>cubic</td>
</tr>
<tr>
<td>$O(n^r)$</td>
<td>polynomial $r \geq 0$</td>
</tr>
<tr>
<td>$O(b^n)$</td>
<td>exponential $b &gt; 1$</td>
</tr>
<tr>
<td>$O(n!)$</td>
<td>factorial</td>
</tr>
</tbody>
</table>
14.2 Rates of Growth

**TABLE 14.4** Number of Transitions of Machine with Time Complexity $t_{C_M}$ with Input of Length $n$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\log_2(n)$</th>
<th>$n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$2^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>25</td>
<td>125</td>
<td>32</td>
<td>120</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>10</td>
<td>100</td>
<td>1,000</td>
<td>1,024</td>
<td>3,628,800</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>20</td>
<td>400</td>
<td>8,000</td>
<td>1048576</td>
<td>2.4×10^{18}</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>30</td>
<td>900</td>
<td>27,000</td>
<td>1.0×10^{9}</td>
<td>2.6×10^{32}</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>40</td>
<td>1,600</td>
<td>64,000</td>
<td>1.1×10^{12}</td>
<td>8.1×10^{47}</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>50</td>
<td>2,500</td>
<td>125,000</td>
<td>1.1×10^{15}</td>
<td>3.0×10^{64}</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>100</td>
<td>10,000</td>
<td>1,000,000</td>
<td>1.2×10^{30}</td>
<td>&gt; 10^{157}</td>
</tr>
<tr>
<td>200</td>
<td>7</td>
<td>200</td>
<td>40,000</td>
<td>8,000,000</td>
<td>1.6×10^{60}</td>
<td>&gt; 10^{374}</td>
</tr>
</tbody>
</table>
14.3 Time Complexity

Computational Complexity of TMs

- Given a TM \( M \), \( M \) accepts the language \( L \) in \textit{polynomial-time} if \( L = L(M) \) and there exists \( P(n) \), a polynomial expression, such that \( M \) accepts any \( w \in L(M) \) in \( \leq P(|w|) \) steps.

- A problem is \textbf{tractable} if there is a DTM that solves the problem whose computations are \textit{polynomially bounded}.

- A language \( L \) is \textbf{decidable} in \textit{polynomial time} if there exists a TM \( M \) that accepts \( L \) with \( tc_M \in O(n^r) \), where \( r \in N \) is independent of \( n \).

- The family of languages decidable in polynomial time is denoted \( P \), e.g.,

  Q: Is the language of palindromes over \{a, b\} \textit{decidable} in \textit{polynomial time}?

  A: yes, there exists TM \( A \) that accepts the language in \( O(n^2) \) time as shown on P.444 (a TM that accepts palindromes), where \( tc_M(n) = (n^2 + 3n + 2) / 2 \in O(n^2) \).

- A language \( L \) is said to be accepted in \textit{non-deterministic polynomial time} if there is a NDTM \( M \) that accepts \( L \) with \( tc_M \in O(n^r) \), \( r \in N \) and \( r \) is independent of \( n \). This family of language is denoted \( NP \).
14.3 Time Complexity of a TM

**Example:** Let the transition diagram of the TM $M$ that accepts the language of palindromes over \{a, b\} be

(* When evaluating the time complexity of a TM, we assume that the computations terminates for every input since it makes no sense to discuss the efficiency of a computation that continue indefinitely *)

Consider Table 14.5. Hence, $tc_M(0) = 1$; $tc_M(1) = 3$; $tc_M(2) = 6$; $tc_M(3) = 10$
Example. Consider the actions of $M$ when processing an even-length (palindrome) input string. The computation alternates between sequences of right (RM) and left (LM) movements.

- RM: requires $k+1$ transitions, where $k$ is the length of the non-blank portion of the tape
- LM: this requires $k$ transitions
- A pair of RM and LM reduces the length of nonblank portion of the tape by 2.
- It requires $n/2$ iterations of the RM-LM for accepting a (palindrome) string of $|n|$

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Direction</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
<td>$n+1$</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>$n$</td>
</tr>
<tr>
<td>2</td>
<td>R</td>
<td>$n-1$</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>$n-2$</td>
</tr>
<tr>
<td>3</td>
<td>R</td>
<td>$n-3$</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>$n-4$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$(n/2) + 1$</td>
<td>R</td>
<td>1</td>
</tr>
</tbody>
</table>

The time complexity of $M$ is (same as odd-length): $\sum_{i=1}^{n+1} i = (n+2)(n+1)/2 \in O(n^2)$
14.3 Time Complexity of a TM

- Time complexity of a TM is determined by the maximum number of steps (i.e., transitions) executed during the computation, whereas the space complexity of a TM computation is defined to be number of tape cells required by the computation.

- If the time complexity of a TM computation is $n$, then the space complexity of that computation is $\leq n + 1$.
  
  - The space complexity of a TM computation may be less than the number of cells to hold the processing string as the TM may not have to process the entire string in order to accept it.

- We measure the time complexity of a TM $M$ by the worst-case performance of $M$ (occurred when $M$ accepts the input string) with input string of length $n$, denoted $tc_M(n)$. 
14.3 Analysis of the complexity of TMs

- **Example 14.3.1.** Given a two-tape TM $M'$ that accepts the set of palindromes over \{a, b\}, $tc_{M'}(n) = 3(n + 1) + 1$.
  - there is a tradeoff between the complexity of a transition and the number of steps between $M$ (in previous example) and $M'$

- **Complexity of Algorithms**
  - The *time* required to execute an algorithm tends to be a function of the *length of the input*.
  
  - **Average-case complexity**: computed by first multiplying the complexity of each possible computation ($C_i$) by the probability of that computation occurring ($P_i$) and then adding these product, i.e., $\sum_{i=1}^{m} P_i \cdot C_i$
  
  - The analysis requires the identification of the dominating steps in the algorithm and estimates the number of times these steps will be performed.
14.4 Analysis of the complexity of TMs

- Complexity of Algorithms

  - **Example (Insertion Sort Algorithm).** The time complexity of the algorithm is proportional to the *number of times* the body of the *WHILE*-loop is executed.

  ![Insertion Sort Diagram]

  Consider again our example unsorted list of keys:
  
  14  3  22  9  10  14  2  7  25  6

  The first pass considers the first key, which is 14, and results in:

<table>
<thead>
<tr>
<th>3  22  9  10  14  2  7  25  6</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsorted list</td>
<td>sorted list</td>
</tr>
</tbody>
</table>

  After the second pass:

<table>
<thead>
<tr>
<th>22  9  10  14  2  7  25  9</th>
<th>3  14</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsorted list</td>
<td>sorted list</td>
</tr>
</tbody>
</table>

  After the sixth pass:

<table>
<thead>
<tr>
<th>2  7  25  6</th>
<th>3  9  10  14  14  22</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsorted list</td>
<td>sorted list</td>
</tr>
</tbody>
</table>
Insertion Sort Algorithm

Program InsertSort(inout: List; in: ListLength)
var PivotPosition, I: integer;
Pivot: ListEntryType;
begin
  if (ListLength ≥ 2) then
  begin
    PivotPosition := 2;
    repeat
      Pivot := List[PivotPosition];
      I := PivotPosition;
      while (I > 1 and List[I - 1] > Pivot) do
      begin
        List[I] := List[I - 1];
        I := I - 1;
      end;
      List[I] := Pivot;
      PivotPosition := PivotPosition + 1;
    until (PivotPosition > ListLength)
  end.
Example (Insertion Sort Algorithm) (Continued).

Worst-case: when the original list is in the reverse order, since

When the pivot’s entry is the 2\text{nd} entry, the loop body is executed 1
When the pivot’s entry is the 3\text{rd} entry, the loop body is executed 2
When the pivot’s entry is the \text{m}^{\text{th}} entry, the loop body is executed \text{m} - 1

Hence if |list| = n, the worst case complexity of the algorithm is

\[ 1 + 2 + \ldots + (n - 1) = \frac{n(n - 1)}{2} = \frac{1}{2}(n^2 - n) \in O(n^2) \]