Time Complexity

The worst-case time complexity of an algorithm $A$ is

$$T_A : D \rightarrow \mathbb{N}, \text{ for } D \subseteq \mathbb{N}$$

$T_A(n) =$ # of operations performed by algorithm $A$ in worst case on input size $n$.

If known $T_A$ and know # operations/sec, can determine runtime.

---

**Linear Search**

- **Input**: $(a_1, a_2, \ldots, a_n), x$
- **Output**: $j$ if $a_j = x$, 0 otherwise

1) $i = 1$
2) while $(i \leq n$ and $x \neq a_i$) $i = i + 1$
3) $i = i + 1$
4) if $i \leq n$:
   return $i$
5) else:
   return 0

\[ n(\log_2 x + 2\log_2 n) \]

- Adding 1 to $i$ might involve flipping $\log_2 n$ bits
- Checking if $x = a_i$ takes at most $\log_2 x$ operations b/c might need to check $\log_2 x$ bits
- Checking if $i \leq n$ might take at most $\log_2 n$ bits

This is bad! Difficult to count operations even on simplest alg.

Usually consider $i \leq n, a_i = x, \text{ return } i, +, \times$, etc to be constant # of operations.
Issues: (Brainstorm)
- too fine-grained/detailed
  * different computers might do operations differently
  * when \( n \) gets large, don’t care about \( 10000 \) vs \( 100000 \)
- too difficult to count every operation

Big-O to Rescue!

↑ special notation to describe how functions grow

\[ \text{def: Let } f, g: \mathbb{Z} \to \mathbb{R}^{+0} \text{ or } f, g: \mathbb{R} \to \mathbb{R}^{+0}, \quad \mathbb{R}^{+0} = \{ x: x \geq 0 \land x \in \mathbb{R} \} \]

• Then \( f(x) \) is \( O(g(x)) \) if \( \exists k, c \in \mathbb{R}^+: \forall x \geq k, f(x) \leq c g(x) \).

"f of x is big-oh of g of x" → "big omega"

• \( f(x) \) is \( \Omega(g(x)) \) if \( \exists k, c \in \mathbb{R}: \forall x \geq k \)
  \[ f(x) \geq c g(x) \]

• \( f(x) \) is \( \Theta(g(x)) \) if \( f(x) = O(g(x)) \) and \( f(x) = \Omega(g(x)) \)

big theta
For time complexity, we only care about large input sizes, and we only care about the scaling, not the detailed function.

Why do we want these two things for time complexity? How does big-O notation capture these two desiderata? (Which idea corresponds to $C$ and which to $k$?)
Discuss

For time complexity, we only care about large input sizes, and we only care about the scaling, not the detailed function.

Why do we want these two things for time complexity? How does big-O notation capture these two desiderata? (Which idea corresponds to \( c \) and which to \( k \)?)

Slide Problems

\[
\text{A} \\
\text{While } 0 \leq n \leq 100 \text{ do} \\
\quad n = n - 1 \\
\text{end} \\
\text{Print } "\text{All Done}" \\
\]

\[
T_A(n) = O(1) \quad \text{with} \\
\quad k = 101, \quad C = 2 \\
\]

<table>
<thead>
<tr>
<th>Input</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>101</td>
<td>102</td>
</tr>
<tr>
<td>102</td>
<td>103</td>
</tr>
<tr>
<td>103</td>
<td>104</td>
</tr>
</tbody>
</table>
Assume for contradiction that

\[2x^2 + 10 = o(x)\]

This means \(\exists k, c \in \mathbb{R}^+ \text{ s.t. } \forall x \geq k, 2x^2 + 10 \leq cx\)

Since we are only considering \(x = k\), we are only considering positive \(x\), so if we divide both sides by \(x\), we get \(\frac{2x + 10}{x} \leq c\). Now consider a value of \(x\) that is larger than both \(k\) and \(c\).

Then \(\frac{2x + 10}{x} > 2c + 10\). This contradicts that

\[\frac{2x + 10}{x} < c.\]