CS 330 - Class 11, Thu Feb 24
Algorithms pt.2

Last time:
- Not all problems are solvable
- Example: The Halting Problem: If we run program P on input x, will P eventually halt? No such program exists (proof by contradiction)

Proof outline
- We'll assume there exists an \( H(P, x) \) solving the halting problem
- We'll use it to build a function \( G(P) \)
- And then find that \( H(G, G) \) can't return the right value

---

Halting Problem:
- So assume (for sake of contradiction) that there's a program \( H \) where \( H(P, x) = \text{true} \) if \( P(x) \) halts; \( H(P, x) = \text{false} \) if \( P(x) \) diverges (= loops forever).
- Define \( G(P) \): if \( H(P, P) = \text{true} \), then diverge, else halt
  - So \( G(P) \) halts iff \( H \) says that \( P(P) \) diverges.
  - But then (using \( P \) for \( G \)), \( G(G) \) halts iff \( H \) says that \( G(G) \) diverges.
  - Contradiction!
- Assuming \( H \) exists implies a contradiction, so \( H \) doesn't exist.

Somewhat weird \( G(G) \):
similar to running a compiler on itself
How Long Does an Algorithm Take to Complete?

Trivial cases — Always the same amount of time
1. Algorithm uses only constants (no variables)
   — Algorithm takes constant runtime
   — Could be big constant: try 1,000,000!
2. Algorithm performs same number of operations regardless of input
   — Example: max(98765432, 987654320)
   — Algorithm again takes constant runtime

(Note we're assuming testing x ≤ y always takes the same amount of time — there's actually a limitation in hardware on the size of numbers.)

How Long Does an Algorithm Take to Complete?

Variable runtime: Depends on
- Size of input
- Number of operations executed
- Speed of execution environment
- Cost per instruction

For theoretical calculations, ignore speed & cost per instructions
- Not as impractical as it looks at first glance
How Long Does an Algorithm Take to Complete?

- Given algorithm with inputs \( x_1, x_2, ..., x_n \).
  - Define function \( T(x_1, x_2, ..., x_n) \) that computes number of operations taken by algorithm on those inputs.
  - \( T \) describes the computational / runtime complexity of the algorithm.

- More generally, we can define memory complexity, ...

---

Time Complexity of max_seq Algorithm

```python
def max_seq( [a_1, ..., a_n] ) :
    m := a_1
    for i := 2 to n
        m := max(m, a_i)
    return m
```

\[ T(n) = 1 + n - 1 + 2(n - 1) + 1 = n + 2n - 2 + 1 = 3n - 1 \]

"Linear time" — proportional to # inputs
**Time Complexity of Insertion Sort**

```python
def insertion_sort(a[1..n]):
    for i := 2 to n
        for k := i to 2 (by -1)
            if a[k-1] > a[k]
                swap a[k-1], a[k]
            else break inner loop
```

<table>
<thead>
<tr>
<th># times executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-1</td>
</tr>
<tr>
<td>1 + 2 + ... + (n-1)</td>
</tr>
<tr>
<td>1 + 2 + ... + (n-1)</td>
</tr>
<tr>
<td>1 + 2 + ... + (n-1)</td>
</tr>
<tr>
<td>(7)</td>
</tr>
</tbody>
</table>

(?) For worst-case analysis, ignore possibility of breaking early

**worst case**

best case? — can fool us into thinking alg is fast

avg case? — hard to define some?

worst case? — what we usually do

---

**Sum of arithmetic series**

\[
1 + 2 + 3 + ... + (n-2) + (n-1) + n
\]

\[
= (1 + n) + (2 + (n-1)) + (3 + (n-2)) + ... + (n/2 + (n/2 + 1)) \quad \text{// if } n \text{ even}
\]

\[
= (n+1) + (n+1) + ... + (n+1) \quad \text{// } n/2 \text{ terms}
\]

\[
= (n+1) \cdot n / 2
\]

If \( n \) is odd, take the sum of 1 ... \( n-1 \) (which is even) and add in \( n \)

\[
= \text{Sum of } (1 + 2 + ... + n-1) + n
\]

\[
= n \cdot (n-1)/2 + n
\]

\[
= n \cdot ((n-1)/2 + 1) = n \cdot (n-1 + 2)/2 = n \cdot (n+1)/2
\]

\[
= (n+1) \cdot n/2 \quad \text{— Same formula as for even numbers}
\]
**Time Complexity of Insertion Sort**

```python
def insertion_sort(a[1..n]) :
    for i := 2 to n
        for k := i to 2 (by -1)
            if a[k-1] > a[k]
                swap a[k-1], a[k]
            else break inner loop

# times executed
n-1
1 + 2 + ... + (n-1)
= n(n-1)/2
1 + 2 + ... + (n-1)
1 + 2 + ... + (n-1)
0 for worst case
```

\[
T(n) = n-1 + 3 \times \frac{n(n-1)}{2} + 0 = n-1 + 3 \times \frac{n(n-1)}{2} = n-1 + 3n^2/2 - 3n/2
\]

\[
= 3n^2/2 - (3n - 2n)/2 - 1
\]

\[
= 3n^2/2 - n/2 + 1
\]

"Quadratic time" — proportional to square of # inputs

---

**Time Complexities of max_seq and insertion_sort**

- \( T_{\text{max_seq}}(n) = 3n - 1 \)
- \( T_{\text{insertion_sort}}(n) = 3n^2/2 - n/2 - 1 \)

- Differences in hardware speed across models and time varies enough that directly comparing \( 3n - 1 \) vs \( 3n - 2 \) or \( 2n \) or even \( 178n \) isn't as useful as we'd like, especially if \( n \) is large.

- But the difference between \( 3n \) and \( 3n^2/2 \) is significant for large \( n \).

- **Asymptotic** runtime complexity — look at behavior for large \( n \)
  - It's ok to ignore slower-growing terms like \( n/2 \) compared to \( n^2 \)

\[ 2n \text{ vs } 3n \]
Big O (= big Oh) notation: Asymptotic Upper Bound

- Saying some function \( f(x) \) is in big Oh of \( g(x) \) means that \( g(x) \) is a rough upper bound for \( f(x) \).
  - (There are also notations for rough <, ≥, >, and =.)
- We say \( f(x) \) is \( O(g(x)) \) or \( f(x) \in O(g(x)) \) [or \( f(x) = O(g(x)) \)]\(^1\)
- Intuitively, \( f(x) \in O(g(x)) \) means \( |f(x)/g(x)| \) is "almost always" ≤ some constant.
  - (\( \exists c \in \mathbb{R} \) (Almost \( \forall x \in \mathbb{R} \) (\( |f(x)/g(x)| \leq c \))

Here, "almost all" means "except for a finite number of cases"

- (Almost \( \forall x \in \mathbb{R} \) (\( P(x) \)) = (\( \exists x_0 \in \mathbb{R} \) (\( \forall x \in \mathbb{R} \) (\( |x| \geq x_0 \rightarrow P(x) \)))
- For runtime complexity, \( f(x) \) and \( g(x) > 0 \), so we drop taking \( |...| \)

\(^1\) Technically, improper/inconsistent notation; informally you hear people say "abuse of notation"
Example of $O(...)$

- Show that $x^2 + 2x + 1 \in O(x^2)$. Let $f(x) = x^2 + 2x + 1$, $g(x) = x^2$
- First check $f(x) / g(x) = \frac{x^2 + 2x + 1}{x^2} = 1 + 2x + 1/x^2$, which approaches 1 as $x \to \infty$ so $f(x) \in O(x^2)$ makes sense
- Raise powers: $f(x) = x^2 + 2x + 1 \leq x^2 + 2x + x^2 = 4x^2$ (once $x > 0$) so $f(x) / g(x) \leq 4 \frac{e^x}{e^x} = 4$
- There's a constant $c = 4$ such that for all $x > x_0 = 0$, $f(x) / x^2 \leq c$, so $f(x) = x^2 + 2x + 1 \in O(x^2)$.
  - Silly comment: We could also say $f(x) / g(x) \leq 10c$ for some base $c$.

---

Polynomial $f(x)$ of degree $n \in O(x^n)$

- We can generalize the argument above.
- For $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ where the $a_i \in \mathbb{R}$ and $a_n > 0$, $f(x) \in O(x^n)$.
  - (We can use $x_0 = 0$; what can we use for $c$?)

Examples

- $n! \in O(n^n)$?
  - $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 \leq n \times n \times n \times \ldots \times n = n^n$
- $\log(n!) \in O(???)$
  - $\log(n!) \leq \log(n^n) = n \times \log n$, so $\log(n!) \in O(n \log n)$
  - $a_0 = 1$, $a_1 = 2$, $a_2 = 1$, $x^2 + 2x + 1 \times 4 \times x^2 + 1 \times 4 \times x^2 (1+2+1)$
Some Useful big-O Relationships

- $\forall b, s \in \mathbb{R}$ where $b > s > 1$ ($b$ = "big", $s$ = "small")
  - $x^b \in O(x^s)$ but $x^b \notin O(x^s)$
  - $s^b \in O(b^s)$ but $b^s \not\in O(s^b)$

- $\forall a, b, c \in \mathbb{R}$ (drop $b$ = "big" here)
  - $x^b \in O(a^b)$ but $a^b \not\in O(x^b)$
  - $(\log_a x)^b \in O(x^c)$ but $x^c \not\in O((\log_a x)^b)$

$O(n)$ Comparing growth rates

Some Combinations of big-O Relationships

- If $f_1(x)$ and $f_2(x)$ both $\in O(g(x))$
  - Then $f_1(x) + f_2(x) \in O(g(x))$

- If $f_1(x) \in O(g_1(x))$ and $f_2(x) \in O(g_2(x))$
  - Then $f_1(x) + f_2(x) \in O(\max(g_1(x), g_2(x)))$
  - And $f_1(x) \times f_2(x) \in O(g_1(x) \times g_2(x))$

$$x \times x^2 \in O(x^2) \times x^4$$

$$x^3 \in O(x^4)$$

$$5x, 3x \in O(x^2)$$

$$5x + 3x \in O(x^2)$$

$$x^2 \in O(x^4)$$

$$x + x^2 \in O(x^4)$$
Random Examples (T or F?)

- $17 \in O(1)$?
- $3x + 27 \in O(x)$?
- $3x + 27 \in O(27^{23})$?
- $5x^2 + 75x + 12345 \in O(x^2)$?
- $5x^2 + 75x + 12345 \in O(x)$?
- $5x^2 + 75x + 12345 \in O(x^2)$?
- $x^2 + 3x \in O(x^2)$
- $x + x\log x \in O(x)$?
- $\log x \in O(x^{10})$?
- $x^2 + \log x \in O(\log(x)^5)$?