Easy and Hard Problems

• Working definition of easy:
  an algorithm is **efficient** if it runs in **polynomial** time;
  easy $\equiv$ efficient.
• Polynomial time means worst case performance of $O(n^k)$ for input of size $n$.
• We also use the term **tractable** to refer to easy problems.

• Alternative: hard $\equiv$ exponential (or worse)
  - E.g. $O(2^n)$, $O(n!)$

Growth of Various Functions

<table>
<thead>
<tr>
<th>$n$</th>
<th>1</th>
<th>10</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log(n)$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>1</td>
<td>3.16</td>
<td>10</td>
<td>31.62</td>
<td>1000</td>
</tr>
<tr>
<td>$n\log(n)$</td>
<td>1</td>
<td>10</td>
<td>200</td>
<td>3000</td>
<td>$6 \times 10^6$</td>
</tr>
<tr>
<td>$n^2$</td>
<td>1</td>
<td>100</td>
<td>$10^6$</td>
<td>$10^{12}$</td>
<td>$10^{24}$</td>
</tr>
<tr>
<td>$2^n$</td>
<td>2</td>
<td>1024</td>
<td>$\approx 10^{10}$</td>
<td>$\approx 10^{20}$</td>
<td>Forget it!</td>
</tr>
</tbody>
</table>

Computation Time

Assuming $2 \times 10^{10}$ operations/second
(approximately the FP performance of a modern Intel desktop chip)

<table>
<thead>
<tr>
<th>$n$</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>&lt; 1 ns</td>
<td>&lt; 1 ns</td>
<td>&lt; 1 ns</td>
<td>50 μs</td>
<td>50 ms</td>
</tr>
<tr>
<td>$n\log(n)$</td>
<td>&lt; 1 ms</td>
<td>&lt; 1 ms</td>
<td>&lt; 1 ms</td>
<td>1 ms</td>
<td>300 ms</td>
</tr>
<tr>
<td>$n^2$</td>
<td>&lt; 1 s</td>
<td>125 ms</td>
<td>500 ms</td>
<td>50 s</td>
<td>1.6 years</td>
</tr>
<tr>
<td>$2^n$</td>
<td>50 ms</td>
<td>16 hours</td>
<td>1.5 trillion years</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Datasets of size $10^9$ and above are now commonplace!

# of unique URLs seen by Google indexer

Exponential Solution Spaces

• Some problems we’ve looked at:
  - Shortest paths
  - Minimum spanning trees
  - Max flows

Common denominator: find one valid solution from exponentially many possibilities.

Example

• Shortest path from $s$ to $t$ in a graph:
  - Worst case: $s$ has $|V| - 1$ neighbors
  - Neighbors have $|V| - 2$ neighbors (excluding $s$), etc.
  - Roughly $(|V|)!$ Possibilities!

• Solvable in $O(|V|\log |V| + |E|)$ using Dijkstra’s algorithm with Fibonacci heap
Another Example

- Traveling salesman problem (TSP)
  - Weighted, undirected graph
  - Find shortest path visiting every vertex once returning to start city (tour)

---

Define Search Problems

- Define in terms of checking algorithm \( C \)
  \( C(I,S) \rightarrow true \) if \( S \) is a valid solution for instance \( I \)
  \( C \) runs in time polynomial in size of \( I \).

Search problem: given \( C, I \):
  - find a solution \( S \) such that \( C(I,S) \rightarrow true \)
  - or -
  state (correctly) that no solution exists.

---

Search vs. Optimization

- TSP doesn’t match our framework: cannot verify optimality in polynomial time.
- Search problem formulation:
  - Find a tour with distance less than \( k \)
  - \( C(I,S) \) checks in poly time distance and validity of \( S \)

---

Search = Optimization

- Transform optimization problem to search:
  - Find max/min at least/no more than \( k \)
  - Recover optimal solution using binary search on \( k \)
- Transform search problem to optimization:
  - Find optimal solution
  - Check if optimal solution is bounded by \( k \)

---

Solving TSP

- No known polynomial time algorithms
- Naive approach: try all \((n-1)!\) tours
- \( O(n^2n) \) solution via dynamic programming
- Can we do better?
  Nobody knows (or they aren’t telling)

---

Complexity Classes

- \( NP \) ("hard" problems)
  - Includes search problems as we’ve defined them
  - Solutions can be verified in polynomial time
  - Solutions can be found in exponential time*
- \( P \) ("easy" problems)
  - Includes all search problems with a known polynomial time algorithm
  - Most of the problems we’ve looked at

* Technically, solutions must be findable by a non-deterministic Turing machine in polynomial time. This is a topic in CompSci 140.
### P vs. NP

- \( P \subseteq NP \)
- Most famous open problem in computer science: is \( P = NP? \)
  - Believed false by most theorists
  - No proof has been revealed either way

### Reductions

- Problems \( P, Q \)
  - \( P \) can be reduced to \( Q \) if:
    - We can transform an instance of \( P \) into an instance of \( Q \) in polynomial time
    - We can transform solutions of \( Q \) into (correct) solutions for \( P \) in polynomial time
  - Result: we can build an algorithm which solves \( P \) via transformation to \( Q \) and back

### NP-Hard/NP-Complete

- A problem \( Q \) is \textit{NP-Hard} if every problem in NP can be reduced to \( Q \)
  - NP-Hard includes problems not in NP – there are harder problems!

- A problem is \textit{NP-Complete} if it is in NP and is NP-Hard

### Easy and Hard Problems

<table>
<thead>
<tr>
<th>Easy (in P)</th>
<th>Hard (NP-hard)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Spanning Tree</td>
<td>Traveling Salesman</td>
</tr>
<tr>
<td>Shortest Path</td>
<td>Longest Path</td>
</tr>
<tr>
<td>Bipartite Matching</td>
<td>3D Matching</td>
</tr>
<tr>
<td>Linear Programming</td>
<td>Integer Linear Programming</td>
</tr>
<tr>
<td>Euler Path</td>
<td>Rudrata (or Hamiltonian) Path</td>
</tr>
<tr>
<td>Horn SAT</td>
<td>SAT</td>
</tr>
</tbody>
</table>