CSE 421: Introduction to Algorithms

Terminology: Complexity

Yin-Tat Lee
Administrativia Stuffs

HW1 is out.
Please submit to Gradescope

Guidelines:
• You can collaborate, but you must write solutions on your own
• You CANNOT search the solution online
• See Ed for more guidelines.

Tips:
• Rewrite your proof.
• Make sure you use assumptions of the problem
• Make sure it is easy to understand
More

Algorithm:

Exceptions (not limited to):
- If possible, reduce the question into a solved problem, For example, for stable matching, explain
  - What “man” and “woman” are corresponding to
  - What their preference are.
  - How convert the stable matching to what we asked in the question.

Runtime:

- If the algorithm is similar to one in the class, Simply explain the difference.
  Make sure your description is not ambiguous.

Correctness:
Definition for Efficiency in This Course

Worst case complexity:
The worst case running time $T(n)$ of an algorithm is max # steps algorithm takes on any input of size $n$.

Definition of 1 step in this course:
- only simple operations (+, *, -, =, if, call, …).
- each operation takes one time step.
- each memory access takes one time step.
- no fancy stuff (add two matrices, copy long string, …).

Definition of efficiency in this course:
An algorithm is efficient if it has polynomial worst case runtime.
Time Complexity on Worst Case Inputs

\[ T(N) = 2N \log_2 N \]

\[ N \log_2 N \]
O-Notation

Given two positive functions $f$ and $g$

- $f(n) = O(g(n))$ if there is a constant $C > 0$ and $N$ st
  \[ f(n) \leq Cg(n) \text{ for all } n > N \]

- $f(n) = \Omega(g(n))$ if there is a constant $C > 0$ and $N$ st
  \[ f(n) \geq Cg(n) \text{ for all } n > N \]

- $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, namely
  There is a constant $C_1, C_2 > 0$ and $N$ st
  \[ C_1g(n) \leq f(n) \leq C_2g(n) \text{ for all } n > N \]
Common Asymptotic Bounds

- **Polynomials**: 
  \[ a_0 + a_1 n + \cdots + a_d n^d \text{ is } O(n^d) \]

- **Logarithms**: 
  \[ \log_a n = O(\log_b n) \text{ for all constants } a, b > 0 \]

- **Logarithms**: log grows slower than every polynomial
  For all \( x > 0 \), \( \log n = O(n^x) \)
**Running Time**

An algorithm runs in polynomial time if $T(n) = n^{O(1)}$.

Equivalently, $T(n) = O(n^d)$ for some constant $d$.

<table>
<thead>
<tr>
<th>Name</th>
<th>Complexity class</th>
<th>Running time $T(n)$</th>
<th>Examples of running times</th>
<th>Example algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant time</td>
<td></td>
<td>$O(1)$</td>
<td>10</td>
<td>Finding the median value in a sorted array of numbers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Calculating $(-1)^n$</td>
</tr>
<tr>
<td>inverse Ackermann time</td>
<td></td>
<td>$O(n(n))$</td>
<td></td>
<td>Amortized time per operation using a disjoint set</td>
</tr>
<tr>
<td>iterated logarithmic time</td>
<td></td>
<td>$O((\log n)^n)$</td>
<td></td>
<td>Distributed coloring of cycles</td>
</tr>
<tr>
<td>log-logarithmic</td>
<td>DLOGTIME</td>
<td>$O(\log n)$</td>
<td>$\log n, \log(n^2)$</td>
<td>Amortized time per operation using a bounded priority queue[2]</td>
</tr>
<tr>
<td>logarithmic time</td>
<td></td>
<td>$O(n \log n)$</td>
<td></td>
<td>Binary search</td>
</tr>
<tr>
<td>polylogarithmic time</td>
<td></td>
<td>$\text{poly}(n)$</td>
<td>$(\log n)^2$</td>
<td></td>
</tr>
<tr>
<td>fractional power</td>
<td></td>
<td>$O(n^c)$ where $0 &lt; c &lt; 1$</td>
<td>$n^{1/2}, n^{2/3}$</td>
<td>Searching in a kd-tree</td>
</tr>
<tr>
<td>linear time</td>
<td></td>
<td>$O(n)$</td>
<td>$n, 2n + 5$</td>
<td>Finding the smallest or largest item in an unsorted array, Kadane's algorithm, linear search</td>
</tr>
<tr>
<td>&quot;n log-star n&quot; time</td>
<td></td>
<td>$O(n \log^* n)$</td>
<td></td>
<td>Seidel’s polygon triangulation algorithm.</td>
</tr>
<tr>
<td>linearithmic time</td>
<td></td>
<td>$O(n \log n)$</td>
<td>$n \log n, \log n!$</td>
<td>Fastest possible comparison sort; Fast Fourier transform.</td>
</tr>
<tr>
<td>quasilinear time</td>
<td></td>
<td>$n \text{poly}(n)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quadratic time</td>
<td></td>
<td>$O(n^2)$</td>
<td>$n^2$</td>
<td>Bubble sort; Insertion sort; Direct convolution</td>
</tr>
<tr>
<td>cubic time</td>
<td></td>
<td>$O(n^3)$</td>
<td>$n^3$</td>
<td>Naive multiplication of two $n \times n$ matrices. Calculating partial correlation.</td>
</tr>
<tr>
<td>polynomial time</td>
<td>P</td>
<td>$2^{O(\log n)} = \text{poly}(n)$</td>
<td>$n^2 + n, n^{10}$</td>
<td>Karmarkar’s algorithm for linear programming; AKS primality test[3][4]</td>
</tr>
<tr>
<td>quasi-polynomial time</td>
<td>QP</td>
<td>$2^{\text{poly}(n)}$</td>
<td>$n\log\log n, n\log n$</td>
<td>Best-known $O(\log^2 n)$-approximation algorithm for the directed Steiner tree problem.</td>
</tr>
<tr>
<td>sub-exponential time (first definition)</td>
<td>SUBEXP</td>
<td>$O(2^n)$ for all $\varepsilon &gt; 0$</td>
<td></td>
<td>Contains BPP unless EXPTIME (see below) equals MA [5]</td>
</tr>
<tr>
<td>sub-exponential time (second definition)</td>
<td></td>
<td>$2^{o(n)}$</td>
<td>$2^{n^{1/3}}$</td>
<td>Best-known algorithm for integer factorization; formerly-best algorithm for graph isomorphism</td>
</tr>
<tr>
<td>exponential time (with linear exponent)</td>
<td>E</td>
<td>$2^{O(n)}$</td>
<td>$1.1^n, 10^n$</td>
<td>Solving the traveling salesman problem using dynamic programming</td>
</tr>
<tr>
<td>exponential time</td>
<td>EXPTIME</td>
<td>$2^{\text{poly}(n)}$</td>
<td>$2^n, 2^{2^n}$</td>
<td>Solving matrix chain multiplication via brute-force search</td>
</tr>
<tr>
<td>factorial time</td>
<td></td>
<td>$O(n!)$</td>
<td>$n!$</td>
<td>Solving the traveling salesman problem via brute-force search</td>
</tr>
<tr>
<td>double exponential time</td>
<td>2-EXPTIME</td>
<td>$2^{\text{exp}(n)}$</td>
<td>$2^{2^n}$</td>
<td>Deciding the truth of a given statement in Presburger arithmetic</td>
</tr>
</tbody>
</table>
Why “Polynomial”?

Point is not that $n^{2000}$ is a practical bound, or that the differences among $n$ and $2n$ and $n^2$ are negligible.

- “My problem is in P” is a starting point for a more detailed analysis

- “My problem is not in P” may suggest that you need to shift to a more tractable variant

<table>
<thead>
<tr>
<th>Year/Authors</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1989/Dyer-Frieze-Kannan</td>
<td>$n^{23}$</td>
</tr>
<tr>
<td>1990/Lovász-Simonovits</td>
<td>$n^{18}$</td>
</tr>
<tr>
<td>1990/Lovász</td>
<td>$n^{10}$</td>
</tr>
<tr>
<td>1991/Applegate-Kannan</td>
<td>$n^{10}$</td>
</tr>
<tr>
<td>1990/Dyer-Frieze</td>
<td>$n^8$</td>
</tr>
<tr>
<td>1993/Lovász-Simonovits</td>
<td>$n^7$</td>
</tr>
<tr>
<td>1997/Kannan-Lovász-Simonovits</td>
<td>$n^5$</td>
</tr>
<tr>
<td>2003/Lovász-Vempala</td>
<td>$n^4$</td>
</tr>
<tr>
<td>2020/YinTat,..., Github Repo</td>
<td>$n^3$, $n^2$</td>
</tr>
</tbody>
</table>
Other Complexities

Average Case Complexity:
\[ \text{avg} \] # steps algorithm takes

Communication Complexity:
\[ \text{max} \] # communication algorithm send between servers

Space Complexity:
\[ \text{max} \] # space algorithm needs

Parallel Complexity:
\[ \text{max} \] length of the longest series of operations that have to be performed sequentially due to data dependencies
CSE 351 Quiz

What is the cost of following operations? (in terms of cycle for CPU with 1 core)

- Compute \(a\times b+c\) where \(a, b, c\) are float (throughput)
  \(~1/16\) cycles

- Cost of unpredictable if (latency)
  \(~20\) cycles

- Cost of reading 1 byte from a random location in memory (latency)
  \(~300\) cycles

- Cost of reading 1 byte from a random location in a M.2 SSD (latency)
  \(~100k\) cycles

- Cost of reading 1 byte from a random location in a 7200RPM harddisk (latency)
  \(~10M\) cycles

- Cost of Elon posting a Twitter from Mars (latency)
  \(~10T\) cycles

Life is simple in 421. Everything is \(O(1)\)
**Warning**

In real world, not all operations take same amount of time.

---

### Not all CPU operations are created equal

| Operation                                      | 
|------------------------------------------------|------------------------------------------------|
| "Simple" register-register op (ADD, OR, etc.) | $10^0$ |
| Memory write                                   | $10^0$ |
| Bypass delay: switch between                  | $10^0$ |
| integer and floating-point units               | $10^0$ |
| "Right" branch of "if"                        | $10^0$ |
| Floating-point/vector addition                 | $10^0$ |
| Multiplication (integer/float/vector)          | $10^0$ |
| Return error and check                         | $10^0$ |
| L1 read                                        | $10^0$ |
| TLB miss                                       | $10^0$ |
| L2 read                                        | $10^0$ |
| "Wrong" branch of "if" (branch misprediction)  | $10^0$ |
| Floating-point division                        | $10^0$ |
| 128-bit vector division                        | $10^0$ |
| Atomics/CAS                                     | $10^0$ |
| C function direct call                         | $10^0$ |
| Integer division                               | $10^0$ |
| C function indirect call                       | $10^0$ |
| C++ virtual function call                      | $10^0$ |
| L3 read                                        | $10^0$ |
| Main RAM read                                   | $10^0$ |
| NUMA: different-socket atomics/CAS (guesstimate)| $10^0$ |
| NUMA: different-socket L3 read                 | $10^0$ |
| Allocation+deallocation pair (small objects)   | $10^0$ |
| NUMA: different-socket main RAM read           | $10^0$ |
| Kernel call                                    | $10^0$ |
| Thread context switch (direct costs)           | $10^0$ |
| C++ Exception thrown+caught                    | $10^0$ |
| Thread context switch (total costs,            | $10^0$ |
| including cache invalidation                   | $10^0$ |

---

Distance which light travels while the operation is performed:

- 30 cm
- 3 m
- 30 m
- 300 m
- 3 km
- 30 km

---

13
Warning

In real world, not all memory accesses take same amount of time.

<table>
<thead>
<tr>
<th>Latency Comparison Numbers (~2012)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
</tr>
<tr>
<td>Branch mispredict</td>
</tr>
<tr>
<td>L2 cache reference</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
</tr>
<tr>
<td>Main memory reference</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
</tr>
<tr>
<td>Send 1K bytes over 1 Gbps network</td>
</tr>
<tr>
<td>Read 4K randomly from SSD*</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
</tr>
<tr>
<td>Read 1 MB sequentially from SSD*</td>
</tr>
<tr>
<td>Disk seek</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
</tr>
</tbody>
</table>

Example:
Improving Google CPU usage by 0.5% via a better hash table
https://www.youtube.com/watch?v=ncHmEUmJZf4
CSE 421: Introduction to Algorithms

Terminology: Graph

Yin-Tat Lee
Graphs

Examples:
• Transportation networks
• Communication networks
• Internet
• Social networks
• Dependency networks
Undirected Graphs $G=(V,E)$

Disconnected graph
Multi edges
Isolated vertices
Self loop
Graphs don’t Live in Flat Land

Geometrical drawing is mentally convenient, but mathematically irrelevant:

4 drawings of a single graph:
Directed Graphs
Terminology

• **Degree of a vertex**: \# edges that touch that vertex

\[
\text{deg}(6) = 3
\]

• **Connected**: Graph is connected if there is a path between every two vertices

• **Connected component**: Maximal set of connected vertices
• **Path**: A sequence of distinct vertices s.t. each vertex is connected to the next vertex with an edge.

• **Cycle**: Path of length > 2 that has the same start and end.

• **Tree**: A connected graph with no cycles.
Exercise: Degree 1 vertices

Claim: If G has no cycle, then it has a vertex of degree $\leq 1$
(Every tree has a leaf)

Proof: (By contradiction)
Suppose every vertex has degree $\geq 2$.
Start from a vertex $v_1$ and follow a path, $v_1, \ldots, v_i$ when we are at $v_i$ we choose the next vertex to be different from $v_{i-1}$. We can do so because $\text{deg}(v_i) \geq 2$.
The first time that we see a repeated vertex ($v_j = v_i$) we get a cycle.
We always get a repeated vertex because $G$ has finitely many vertices
Exercise: Trees and Induction

Claim: Every tree with \( n \) vertices has \( n - 1 \) edges.

Proof: (Induction on \( n \).)

Base: \( n = 1 \), the tree has no edge

Induction: Let \( T \) be a tree with \( n \) vertices.

So, \( T \) has a vertex \( v \) of degree 1.

Remove \( v \) and the neighboring edge, and let \( T' \) be the new graph.

We claim \( T' \) is a tree: It has no cycle, and it must be connected.

So, \( T' \) has \( n - 2 \) edges and \( T \) has \( n - 1 \) edges.
Exercise: Degree Sum

Claim: In any undirected graph, the number of edges is equal to \((1/2) \sum_{\text{vertex } v} \deg(v)\)

Pf: \(\sum_{\text{vertex } v} \deg(v)\) counts every edge of the graph exactly twice; once from each end of the edge.

\(|E| = 8\)

\[\sum_{\text{vertex } v} \deg(v) = 2 + 2 + 1 + 1 + 3 + 2 + 3 + 2 = 16\]
Exercise: Odd Degree Vertices

**Claim**: In any undirected graph, the number of odd degree vertices is even.

**Pf**: In previous claim we showed sum of all vertex degrees is even. So there must be even number of odd degree vertices, because sum of odd number of odd numbers is odd.

4 odd degree vertices: 3, 4, 5, 6
Exercise: #edges

Let $G = (V, E)$ be a graph with $n = |V|$ vertices and $m = |E|$ edges.

Claim: $0 \leq m \leq \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$

Pf: Since every edge connects two distinct vertices (i.e., $G$ has no loops) and no two edges connect the same pair of vertices (i.e., $G$ has no multi-edges) it has at most $\binom{n}{2}$ edges.
Sparse Graphs

A graph is called **sparse** if $m \ll n^2$ and it is called **dense** otherwise.

Sparse graphs are very common in practice:
- Friendships in social network
- Planar graphs
- Web graph

$O(n + m)$ is usually much better runtime than $O(n^2)$. 
Storing Graphs

Vertex set $V = \{v_1, ..., v_n\}$.

**Adjacency Matrix:** $A$
- For all, $i, j, A[i, j] = 1$ iff $(v_i, v_j) \in E$
- Storage: $n^2$ bits

**Advantage:**
- $O(1)$ test for presence or absence of edges

**Disadvantage:**
- Inefficient for sparse graphs both in storage and edge-access
Storing Graphs

Adjacency List:
$O(n + m)$ words

Advantage
• Compact for sparse
• Easily see all edges

Disadvantage
• Bad memory access
• Not good for parallel algorithms.
Storing Graphs

Adjacency Array:
$O(n + m)$ words

Advantage
- Compact for sparse
- Easily see all edges
- Better for memory access
- Better for parallel algorithms.

Disadvantage
- Difficult to update the graph
Storing Graphs

Implicit Representation:

\( f(v) \) outputs an iterator of neighbor of \( v \).
Aka, \( f(v) \rightarrow \text{next()} \rightarrow \text{next()} \rightarrow \text{next()} \rightarrow \text{next()} \)

Advantage

• No space is required

Disadvantage

• Mainly work for abstractly defined graph

2,125,922,464,947,725,402,112,000 states.
Storing Graphs

In practice, pick the representation according to the algorithm (depends how we want to access the graph).

In this course, we focus on asymptotic runtime.

We can simply do this:

- For each vertex, use a hash table to store its neighbors
- This gives $O(1)$ time for many operations
  
  Insert, Delete, Find, Next, …