COMP 3170 - Analysis of Algorithms & Data Structures

Shahin Kamali

Lower Bounds

CLRS 8.1

University of Manitoba
Assume you design an algorithm that solves a given problem $P$ in $\Theta(n^2)$.

Further exploration fails to discover an asymptotically faster algorithm.

How can you know whether it is possible to devise a $o(n^2)$ algorithm for $P$?

Establish a lower bound $f(n)$ showing that every algorithm that solves problem $P$ requires $\Omega(f(n))$ time in the worst-case.

If you can show a lower bound of $\Omega(n^2)$, then every algorithm for solving $P$ requires $\Omega(n^2)$ in the worst-case, and your algorithm's time is asymptotically optimal.

Lower bounds are used to establish limitation of algorithms!
A lower bound $f(n)$ for a **problem** $P$ implies that every algorithm for $P$ runs in time $\Omega(f(n))$ in the worst-case.

- E.g., a lower bound of $n \log n$ for comparison-based sorting problem.

A lower bound $g(n)$ for an **algorithm** $A$ implies that there are inputs for which the running time of $A$ is $\Omega(g(n))$, i.e., in the worst-case $A$ runs in $\Omega(g(n))$.

- E.g., a lower bound of $n^2$ for Bubble-sort (i.e., we show there are inputs for which Bubble-sort runs in $\Omega(n^2)$).

Our focus in this section is on **lower bounds for problems**.
Comparison-based Sorting

- Problem: sort a set of items (e.g., potatoes) by only comparing them (i.e., using a scale to compare two items).
- An array of $n$ distinct items can be ordered in $n!$ ways.
  - This corresponds to the number of permutations of $n$ items.
  - Sorting corresponds to identifying the permutation of a sequence of elements.
  - Once the permutation is known, the correct ordered position of each item can be restored.
A decision tree is a loopless flowchart representing all possible sequences of steps executed by an algorithm while solving a given problem.

- The **height** of the tree corresponds to the **worst-case time** required by the algorithm.
- Each **leaf** indicates one possible input (e.g., a permutation in case of sorting).

For sorting, each internal node is associated with a comparison

- For finding a lower bound for time complexity, we count the number of comparisons in the worst case, i.e., the height of any decision tree.
Decision Trees

- One possible decision tree for determining the correct sorted order of three items \( a, b, c \).
  - Tree has height 3 → the algorithm requires 3 comparisons in the worst case.
  - Every binary tree with \( 3! = 6 \) leaves (possible permutation) has height at least 3.
    - Hence, every algorithm for sorting 3 elements requires at least 3 comparisons in the worst case.
An algorithm that sorts $n$ items corresponds to a decision tree which has $n!$ leaves (each representing one permutation).

The height of a binary tree on $X$ leaves is at least $\log_2(X)$.

The height of a binary tree on $n!$ leaves is at least $\log_2(n!)$. 

(because a binary tree with height $h$ has at most $2^h$ leaves.)

$log n! = log(n \cdot (n-1) \cdot (n-2) \ldots n/2 \cdot (n/2-1) \ldots 2 \cdot 1) > log(\underbrace{n/2 \cdot n/2 \ldots n/2}_{n/2\text{ times}}) = log(n/2)^{n/2} = n/2 \log(n/2) \in \Theta(n \log n)$
Reductions

• Sometimes it is difficult to establish a lower bound directly

• We can use the lower bounds for a different problem using a **reduction**
  
  • Reduction creates a relationship between an easy problem $E$ and a hard problem $H$.
  • It has applications for deriving both lower and upper bounds.

• Steps for reduction (for a lower bound):
  
  I) Assume a lower bound for problem $E$ is known
  II) Show that problem $H$ is as hard as problem $E$
  III) $\rightarrow$ The lower bound for problem $E$ also applies to problem $H$. 
How to show that problem \( H \) is as hard as problem \( E \)?
- Transform any instance of problem \( E \) to an instance of problem \( H \).
- Define a reduction \( f \) such that for any instance \( i \) of problem \( E \), there is an instance \( f(i) \) of problem \( H \)
  - \( x \) is a solution to \( i \) if and only if \( f(x) \) is a solution to \( f(i) \).
Assume reduction requires $O(g(n))$ time and solving problem $E$ requires $\Omega(h(n))$ time.

- If $g(n) \in o(h(n))$, then solving problem $H$ also requires $\Omega(h(n))$ time.
- Proof: consider otherwise, i.e., solving $H$ requires $o(h(n))$. Then, given any instance of $E$, we can transform it to an instance of $H$ (in $g(n) \in o(h(n))$ time) and solve it in $o(h(n))$. This contradicts the lower bound $\Omega(h(n))$ for $E$.

If Problem $E$ is hard, then so is Problem $H$. A reduction allows a lower bound for Problem $E$ to be applied to Problem $H$. 
Assume reduction requires $O(g(n))$ time and there is an algorithm that solves problem $H$ in $O(j(n))$ time.

- If $g(n) \in o(j(n))$, then problem $E$ can also be solved in $O(j(n))$ time.
- Proof: consider otherwise, i.e., assume solving some instances of $E$ requires $\omega(j(n))$. We can transform any of these instances to instances of problem $H$ in $O(j(n))$. Hence, solving the resulting instances of problem $H$ require $\omega(j(n))$, contradicting that any instance of $H$ can be done in $O(j(n))$.

If Problem $H$ is easy, then so is Problem $E$. A reduction allows an upper bound (algorithm) for Problem $H$ to be applied to solve Problem $E$. 
Reduction Summary, Applications

- Reduce any instance $i$ of an easy problem $E$ to an instance $f(i)$ of a hard problem $H$ so that $x$ is a solution for $i$ iff $f(x)$ is a solution for $f(i)$.

- **Negative Results (lower bounds):** If Problem $E$ is hard, then so is Problem $H$. A reduction allows a lower bound for Problem $E$ to be applied to Problem $H$.

- **Algorithm Design:** If Problem $H$ is easy, then so is Problem $E$. A reduction allows an algorithm for Problem $H$ to solve Problem $E$.

- **Complexity Classes:** Group problems into equivalence classes by algorithmic difficulty (complexity zoo).
3Sum and Collinearity

- Problem $E$: 3SUM
  - Instance: a set $S$ of $n$ distinct real numbers
  - Question: Is there a subset $\{a, b, c\} \subset S$ such that $a + b + c = 0$?

- Problem $H$: Collinearity
  - Instance: a set $P$ of $n$ distinct points in the plane
  - Question: Are any three of these points collinear?
Decision Problems

- 3Sum and Collinearity are instances of decision problems which ask questions whose answers are either ‘yes’ or ‘no’.

- Many algorithmic problem can be formulated as decision problems to derive lower bounds on their complexity.
  - E.g., solving the problem “find a set \( \{a, b, c\} \subseteq S \) so that \( a + b + c = 0 \)” is at least as difficult as answering the question “Does there exist a subset \( \{a, b, c\} \subseteq S \) so that \( a + b + c = 0 \)”?
  - A lower bound on the decision problem applies to the original problem.

- When establishing lower bounds, we often consider decision versions of problems.
  - Original Problem: find the median of \( A[0 : n - 1] \)
  - Decision Variant: Is the median of \( A[0 : n - 1] \) equal to \( x \)?
  - Both have lower bound of \( \Omega(n) \).
Reducing from 3Sum to Collinearity

- Choose any instance $S = \{s_1, s_2, \ldots, s_n\}$ for 3Sum.
  - The answer is yes if 3 of these numbers sum to 0.
- Create an instance $P = \{(s_1, s_1^3), (s_2, s_2^3), \ldots, (s_n, s_n^3)\}$ of the Collinearity problem (i.e., $P = f(S)$).
  - The answer is yes if 3 of these points lie on the same line.
- **We have to show the answer to instance $S$ of 3Sum is yes if and only if the answer to $P = f(S)$ of collinearity is yes.**
  - Specifically, we need to show $a + b + c = 0$ iff points $A = (a, a^3), B = (b, b^3)$, and $C = (c, c^3)$ are collinear.
    - $A, B$, and $C$ are collinear iff the line segments $\overline{AB}$ and $\overline{BC}$ have equal slopes.
    - we need to show $a + b + c = 0$ iff slope of $\overline{AB} = \text{slope of } \overline{BC}$. 
Reducing 3Sum to Collinearity

- we use algebra to show $a + b + c = 0$ iff slope of $\overline{AB} = \text{slope of } \overline{BC}$.

\[
slope \overline{AB} = slope \overline{BC} \\
\iff \frac{a^3 - b^3}{a - b} = \frac{b^3 - c^3}{b - c} \\
\iff \frac{(a - b)(a^2 + ab + b^2)}{a - b} = \frac{(b - c)(b^2 + bc + c^2)}{b - c} \\
\iff a^2 + ab + b^2 = b^2 + bc + c^2 \\
\iff a^2 + ab = bc + c^2 \\
\iff a^2 + ab - bc - c^2 = 0 \\
\iff (a - c)(a + b + c) = 0 \\
\iff a + b + c = 0
\]

- $A = (a, a^3), B = (b, b^3)$, and $C = (c, c^3)$ are collinear if and only if $a + b + c = 0$.
  - The answer to collinearity is yes if and only if the answer to 3Sum is yes.
Reducing 3Sum to Collinearity

- Given any instance \( S = \{s_1, s_2, \ldots, s_n\} \) for \( E = 3\text{Sum} \) we created an instance \( f(S) = \{(s_1, s_1^3), (s_2, s_2^3), \ldots, (s_n, s_n^3)\} \) of \( H = \text{Collinearity} \) problem.
  - We don’t need the other direction, i.e., we don’t need to create an instance of 3Sum from collinearity.
- We showed that the answer for instance \( S \) of 3Sum is yes if and only if the answer for instance \( f(S) \) of collinearity is yes.
  - We need to show both directions.
- We conclude that 3Sum can be reduced to Collinearity.
  - In a sense, 3Sum is easier than collinearity.
- Always have an eye on how long the reduction takes.
  - Here, creating instance \( f(S) \) from \( S \) takes \( g(n) = O(n) \) time.
Assume reduction requires $O(g(n))$ time (here $g(n) = O(n)$) and solving problem $E$ (3Sum) requires $\Omega(h(n))$ time (e.g., $\Omega(n^{1.99})$).

- If $g(n) \in o(h(n))$ (which is the case here), then solving problem $H$ (collinearity) also requires $\Omega(h(n))$ (e.g., $\Omega(n^{1.99})$) time.
- Proof: consider otherwise, i.e., solving $H$ requires $o(h(n))$. Then, given any instance of $E$, we can transform it to an instance of $H$ (in $g(n) \in o(h(n))$ time) and answer it (by a yes or no) in $o(h(n))$. This contradicts the lower bound $\Omega(h(n))$ for $E$.

If Problem $E$ (3Sum) is hard (i.e., requires $\Omega(n^{1.99})$), then so is Problem $H$ (Collinearity). A reduction allows a lower bound for Problem $E$ to be applied to Problem $H$.

In other words, any lower bound of $h(n)$ for 3Sum applies for collinearity as long as $h(n) \in \omega(n)$. 
Assume reduction requires $O(g(n))$ (here $O(n)$) time and there is an algorithm that solves any instance of problem $H$ (collinearity) in $O(j(n))$ (e.g., $\Theta(n^2)$) time.

- If $g(n) \in o(j(n))$ (which is the case here), then problem $E$ can also be solved in $O(j(n))$ time.
- Proof: consider otherwise, i.e., assume answering some instances of $E$ requires $\omega(j(n))$. We can transform any of these instances to instances of problem $H$ in $O(j(n))$. Hence, answering the resulting instances of problem $H$ also require $\omega(j(n))$, contradicting that any instance of $H$ can be done in $O(j(n))$.

If Problem $H$ (collinearity) is easy (can be solved in $\Theta(n^2)$), then so is Problem $E$ (3Sum). A reduction allows an upper bound (algorithm) for Problem $H$ (collinearity) to be applied to solve Problem $E$.

- In other words, an algorithm that solves collinearity in $j(n)$ can be used to solve 3Sum in $j(n)$ assuming $j(n) \in \omega(n)$. 
Recall that any lower bound of $h(n)$ for 3Sum applies for collinearity as long as $h(n) \in \omega(n)$.

- 3Sum-conjecture: 3-Sum requires $\Omega(n^2)$ time, any algorithm for 3Sum runs in $\Omega(n^2)$.
- This conjecture was open for a long time, until it was refuted in 2014 by an algorithm which runs in $O(n^2/(\log n \log \log n)^{2/3})$. [Gronlund and Pettie paper on “Threesomes, Degenerates, and Love Triangles”]
- Modern 3Sum-conjecture: 3-Sum requires $\Omega(n^{2-\epsilon})$ time for any constant $\epsilon > 0$.
  - If this conjecture is true, collinearity also requires $\Omega(n^{2-\epsilon})$. 
In fact, there are many other problems that 3Sum reduces to.

Informally, **3Sum-hard** class of problems are those that 3Sum reduces to. It include collinearity, 3Sum itself, and many geometric problems.

E.g., Given a set $S$ of $n$ points on the plane, what is the area of the smallest triangle formed by any three of these points?

E.g., Given a set $S$ of $n$ triangles and a triangle $t$, does the union of the triangles in $S$ cover $t$?

Most 3Sum-hard problems can be solved in $O(n^2)$. An improvement to $O(n^{2-\epsilon})$ depend on the modern 3Sum-conjecture.

If modern 3Sum-Conjecture is correct, 3Sum and hence all 3Sum-hard problem required $\Omega(n^{2-\epsilon})$. 