Complexity & Computability

- lower bounds on problems
  - brute force, decision trees, adversary arguments, problem reduction
- complexity theory
  - tractability, decidability
  - P vs. NP, NP-complete
  - NP-complete proofs & reductions

Lower bounds

when studying a problem, may wish to establish a lower bound on efficiency

- binary search is $O(\log N)$ – can we do better?
- merge/quick/heap sorts are $O(N \log N)$ – can we do better?

establishing a lower bound can tell us

- when a particular algorithm is as good as possible
- when the problem is intractable (by showing that best possible algorithm is BAD)

methods for establishing lower bounds:

- brute force
- information-theoretic arguments (decision trees)
- adversary arguments
- problem reduction
Brute force arguments

sometimes, a problem-specific approach works

example: Towers of Hanoi puzzle

- can prove, by induction, that moving a tower of size N requires $\Omega(2^N)$ steps

Information-theoretic arguments

can sometimes establish a lower bound based on the amount of information the problem must produce

example: guess a randomly selected number between 1 and N
- with possible responses of "correct", "too low", or "too high"

  - the amount of uncertainty is $\lceil \log_2 N \rceil$, the number of bits needed to specify the selected number
  - each answer to a question yields at most 1 bit of information
  - thus, $\lceil \log_2 N \rceil$ is a lower bound on the number of questions

a useful structure for information-theoretic arguments is a decision tree
Decision trees

A decision tree is a model of algorithms involving comparisons
- Internal nodes represent comparisons
- Leaves represent outcomes

E.g., decision tree for 3-element insertion sort:

Decision trees & sorting

Note that any comparison-based sorting algorithm can be represented by a decision tree
- Number of leaves (outcomes) $\geq N!$
- Height of binary tree with $N!$ leaves $\geq \lceil \log_2 N! \rceil$

Therefore, the minimum number of comparisons required by any comparison-based sorting algorithm $\geq \lceil \log_2 N! \rceil$
- Since $\lceil \log_2 N! \rceil \approx N \log_2 N$, the lower bound $\Omega(N \log N)$ is tight

Thus, merge/quick/heap sorts are as good as it gets
Decision trees & searching

similarly, we can use a decision tree to show that binary search is as good as it gets (assuming the list is sorted)

decision tree for binary search of 4-element list:
- internal nodes are found elements
- leaves are ranges if not found

- number of leaves (ranges where not found) = N + 1
- height of binary tree with N+1 leaves $\geq \lceil \log_2 (N+1) \rceil$
- therefore, the minimum number of comparisons required by any comparison-based searching algorithm $\geq \lceil \log_2 (N+1) \rceil$
- lower bound $\Omega(\log N)$ is tight

Exercise

consider finding the median of a 3-element list of numbers $[x_1, x_2, x_3]$

- information-theoretic lower bound?
- decision tree?
Adversary arguments

using an adversary argument, you repeatedly adjust the input to make an algorithm work the hardest

example: dishonest hangman
  - adversary always puts the word in a larger of the subset generated by last guess
  - for a given dictionary, can determine a lower bound on guesses

example: merging two sorted lists of size N (as in merge sort)
  - adversary makes it so that no list "runs out" of values (e.g., \(a_i < b_j\) iff \(i < j\))
  - forces 2N-1 comparisons to produce \(b_1 < a_1 < b_2 < a_2 < \ldots < b_N < a_N\)

Problem reduction

problem reduction uses a transform & conquer approach
  - if we can show that problem \(P\) is at least as hard as problem \(Q\), then a lower bound for \(Q\) is also a lower bound for \(P\).

example: multiplication can be reduced to the complexity of squaring

\[
a \times b = \frac{(a + b)^2 - a^2 - b^2}{2}
\]

in general:
1. find problem \(Q\) with a known lower bound
2. reduce that problem to problem \(P\) (i.e., show that can solve \(Q\) by solving an instance of \(P\))
3. then \(P\) is at least as hard as \(Q\), so same lower bound applies
Problem reduction example

CLOSEST NUMBERS (CN) PROBLEM: given N numbers, find the two closest numbers

1. consider a different problem: ELEMENT UNIQUENESS (EU) PROBLEM
   - given a list of N numbers, determine if all are unique (no dupes)
   - this problem has been shown to have a lower bound of \( \Omega(N \log N) \)

2. consider an instance of EU: given numbers \( e_1, \ldots, e_N \), determine if all are unique
   - find the two closest numbers (this is an instance of CN)
   - if the distance between them is > 0, then \( e_1, \ldots, e_N \) are unique

3. this shows that CN is at least as hard as EU
   - can solve an instance of EU by performing a transformation & solving CN
   - since transformation is \( O(N) \), CN must also have a lower-bound of \( \Omega(N \log N) \)
   - (proof by contradiction) assume CN could be solved in \( O(X) \) where \( X < N \log N \)
     then, could solve EU by transforming & solving CN \( \rightarrow O(N) + O(X) < O(N \log N) \)
     this contradicts what we know about EU, so CN must be \( \Omega(N \log N) \)

Another example

CLOSEST POINTS (CP) PROBLEM: given N points in the plane, find the two closest points

1. consider a different problem: CLOSEST NUMBER (CN) PROBLEM
   - we just showed that CN has a lower bound of \( \Omega(N \log N) \)

2. consider an instance of CN: given numbers \( e_1, \ldots, e_N \), determine closest numbers
   - from these N numbers, construct N points: \( (e_1, 0), \ldots, (e_N, 0) \)
   - find the two closest points (this is an instance of CP)
   - if \( (e_i, 0) \) and \( (e_j, 0) \) are closest points, then \( e_i \) and \( e_j \) are closest numbers

3. this shows that CP is at least as hard as CN
   - can solve an instance of CN by performing a transformation & solving CP
   - since transformation is \( O(N) \), CP must also have a lower-bound of \( \Omega(N \log N) \)
   - (proof by contradiction) assume CP could be solved in \( O(X) \) where \( X < N \log N \)
     then, could solve EU by transforming & solving CP \( \rightarrow O(N) + O(X) < O(N \log N) \)
     this contradicts what we know about EU, so CP must be \( \Omega(N \log N) \)
Tightness

Are the $\Omega(N \log N)$ lower bounds tight for CLOSEST NUMBERS and CLOSEST POINTS problems?

- Can you devise $O(N \log N)$ algorithm for CLOSEST NUMBERS?
- Can you devise $O(N \log N)$ algorithm for CLOSEST POINTS?

Classifying problem complexity

Throughout this class, we have considered problems, designed algorithms, and classified their efficiency

- E.g., sorting a list – could use $O(N^2)$ selection sort or $O(N \log N)$ quick sort
- Big-Oh provides for direct comparison between two algorithms

When is a problem too difficult?

Extremely low bar: We say that a problem is intractable if there does not exist a polynomial time $O(p(n))$ algorithm that solves it

$\Theta(2^n)$ is definitely intractable

- Note: $N = 20 \rightarrow$ millions of steps
- $2^{60} > \#$ of seconds since Big Bang
- $2^{273} > \#$ of atoms in the universe

But $\Theta(N^{100})$ is tractable?!?

In reality, anything worse than $N^3$ is not practical.
Beyond intractable

Alan Turing showed that there is a class of problems beyond intractable
- there are problems that have been shown to be unsolvable (regardless of efficiency)

THE HALTING PROBLEM: Given a computer program and an input to it, determine whether the program will halt on the input.
Assume that there is an algorithm $A$ that solves the Halting Problem. That is, for any program $P$ and input $I$:

$A(P, I)$ returns true if $P$ halts on $I$; otherwise, returns false

*note: a program is represented as bits, so a program can be input to a program*

Construct the following program $Q$:

```
Q(P): if ( A(P, P) ) {  // if P halts on input P
    while (true) { }  // infinite loop
}                          // otherwise
return;                   // halt
```

If you call $Q$ with itself as input:
$Q(Q)$ halts if and only if $A(Q, Q)$ returns false if and only if $Q(Q)$ does not halt

CONTRADICTION

Problem types: decision & optimization

the Halting Problem is an example of a **decision problem**
- solving the problem requires answering a yes/no question

another common type of problems is an **optimization problem**
- solving the problem requires finding the best/largest/shortest answer
- e.g., shortest path, minimal spanning tree

many problems have decision and optimization versions
- find the shortest path between vertices $v_1$ and $v_2$ in a graph
- is there a path between $v_1$ and $v_2$ whose length is $\leq d$

decision problems are more convenient for formal investigation of their complexity
Class P

P: the class of decision problems that are solvable in polynomial time $O(p(n))$
- i.e., the class of tractable decision problems

Interestingly, there are many important problems for which no polynomial-time algorithm has been devised
- Hamiltonian Circuit Problem: determine whether a graph has a path that starts and ends at the same vertex and passes through every other vertex once
- Traveling Salesman Problem: find the shortest Hamiltonian circuit in a complete graph
- Graph Coloring Problem: determine the smallest number of colors needed so that adjacent vertices are different colors
- Partition Problem: Given N positive integers, determine whether it is possible to partition them into two disjoint subsets with the same sum.
- Knapsack Problem: Given a set of N items with integer weights and values, determine the most valuable subset that fits in a knapsack with limited capacity.
- Bin-packing Problem: Given N items with varying sizes, determine the smallest number of uniform-capacity bins required to contain them.

Class NP

However, many of these problems fit into a (potentially) broader class

A nondeterministic polynomial algorithm is a two-stage procedure that:
1. generates a random string purported to solve the problem (guessing stage)
2. checks whether this solution is correct in polynomial time (verification stage)

NP: class of decision problems that can be solved by a nondeterministic polynomial algorithm
- i.e., whose proposed solutions can be verified in polynomial time

Example: Hamiltonian Circuit Problem is in NP
- given a path, can verify that it is a Hamiltonian circuit in $O(N)$

Example: Partition Problem is in NP
- given two partitions, can verify that their sums are equal in $O(N)$
P vs. NP

decision versions of Traveling Salesman, Knapsack, Graph Coloring, and many other optimization problems are also in NP

note that problems in P can also be solved using the 2-stage procedure
- the guessing stage is unnecessary
- the verification stage generates and verifies in polynomial time

so, \( P \subseteq NP \)

big question: does \( P = NP \) ?
- considerable effort has gone into trying to find polynomial-time solutions to NP problems (without success)
- most researchers believe they are not equal (i.e., P is a proper subset), but we don't know for sure
NP-complete

while we don't know whether \( P = NP \), we can identify extremes within NP

given decision problems \( D_1 \) and \( D_2 \), we say that \( D_1 \) is polynomial-time reducible to \( D_2 \) if there exists transformation \( t \) such that:
1. \( t \) maps all yes-instances of \( D_1 \) to yes-instances of \( D_2 \)
2. \( t \) maps all no-instances of \( D_1 \) to no-instances of \( D_2 \)
3. \( t \) is computable by a polynomial time algorithm

we say that decision problem \( D \) is \( NP \)-complete if:
1. \( D \) belongs to \( NP \)
2. every problem in \( NP \) is polynomial-time reducible to \( D \)

in short, an \( NP \)-complete problem is as hard as any problem in \( NP \)

NP-complete example

the first problem proven to be \( NP \)-complete was Boolean Satisfiability (SAT)

- given a Boolean expression, determine if satisfiable
  e.g., \( (A \lor B) \land (\neg B \lor C) \) is true if \( A & C \) are true

- SAT is clearly in \( NP \)
given true/false assignments to the propositions, can evaluate the truth of the expression in polynomial time

- to be \( NP \)-complete, every other \( NP \) problem must be reducible to it
proof idea (Cook, 1971):
  - if a problem is in \( NP \), can construct a non-deterministic (Turing) machine to solve it
  - for each input to that machine, can construct a Boolean expression that evaluates to true if the machine halts and answers "yes" on input
  - thus, original problem is reduced to determining whether the corresponding Boolean expression is satisfiable
NP-complete reductions

if we can reduce SAT to another NP problem, then it is also NP-complete

CLIQUE: given a graph with N vertices, is there a fully connected subgraph of C vertices?

SAT $\rightarrow$ CLIQUE

can reduce the SAT problem to CLIQUE

- given an instance of SAT, e.g., $(A \lor B) \land (\neg B \lor C) \land (B \lor \neg C)$
  
  note: any Boolean expression can be transformed into conjunctive normal form
  here, there are 3 OR-groups, joined together with AND

- construct a graph with vertices grouped by each OR-group
  there is an edge between two vertices if
  1. the vertices are in different OR-groups, and
  2. they are not negations of each other
  
  note: edge implies endpoints can be simultaneously true

- the expression is satisfiable if can have vertex from each OR-group simultaneously true
  in other words, is a clique of size C (where C is the number of OR-groups)
  since example expression has 3 OR-groups, need a clique of size 3

so, CLIQUE is also NP-complete
Implications of NP-completeness

an NP-complete problem is as hard as any problem in NP
- i.e., all problems in NP reduce to it
- discovering a polynomial solution to any NP-complete problem
  - would imply a polynomial solution to all problems in NP
  - would show P = NP

if P = NP, many problems currently thought to be intractable would be tractable
- e.g., PRIME FACTORIZATION PROBLEM: factor a number into its prime factors
- the RSA encryption algorithm relies on the fact that factoring large numbers is intractable
- if an efficient factorization algorithm were discovered, modern encryption could break

QUESTION: would it necessarily break?