

Famous Complexity Classes

P – decision problems solvable by a deterministic Turing machine in polynomial time

NP – decision problems verifiable by a deterministic Turing machine in polynomial time

FP – function problems solvable by a deterministic Turing machine in polynomial time

FNP – function problems verifiable by a deterministic Turing machine in polynomial time

Decision Problems vs Function Problems

Decision problems are problems where the result is a yes/no answer.

- e.g. is there a solution to the 0-1 knapsack problem with total weight $\leq W$ and total value $\geq V$?

Function problems compute the result of a function.

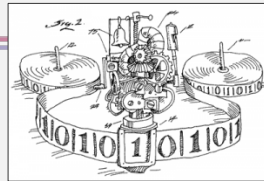
- e.g. 0-1 knapsack problem: maximize the total value such that the total weight $\leq W$

Observation –

- a function problem can be solved efficiently given a black box for the corresponding decision problem
 - “efficiently” = logarithmic number of steps
 - use one-sided binary search

(one-sided binary search for knapsack means trying $V = 2^i$ for $i = 0, 1, 2, \dots$ until the answers are different for successive values of i , then repeating the process within the interval found to find a smaller interval, and so forth)

Turing Machines



<http://www.nikoloplakis.gr/>

A *Turing machine* is a theoretical machine consisting of:

- an infinite tape divided into cells
- a head that can read and write symbols on the tape, and move one cell left or right
- a current state, which is one of a finite number of possible states
- a finite table which, given a current state and symbol on the tape, specifies an action (erase or write symbol), a movement (left, right, or stay), and a new state

| Tape symbol | Current state A | | | Current state B | | | Current state C | | |
|-------------|-----------------|-----------|------------|-----------------|-----------|------------|-----------------|-----------|------------|
| | Write symbol | Move tape | Next state | Write symbol | Move tape | Next state | Write symbol | Move tape | Next state |
| 0 | 1 | R | B | 1 | L | A | 1 | L | B |
| 1 | 1 | L | C | 1 | R | B | 1 | R | HALT |

Deterministic vs Nondeterministic

A *deterministic* Turing machine has at most one rule that applies to a given state and symbol.

A *nondeterministic* Turing machine may have multiple rules that apply to a given state and symbol.

Famous Complexity Classes

P – solvable by a deterministic Turing machine in polynomial time

NP – verifiable by a deterministic Turing machine in polynomial time

- alternatively, solvable by a nondeterministic Turing machine in polynomial time

Key points –

- for NP, technically it is only “yes” solutions that are polynomial-time verifiable
 - a “yes” answer requires only a single instance that works (and is checkable in polynomial time)
 - a “no” answer requires showing that no instance works
- in both cases, there are at most a polynomial number of choices to make in order to generate the solution
 - for each choice –
 - deterministic has rules to pick the right alternative
 - nondeterministic can be thought of as correctly guessing the right alternative

7

Famous Complexity Classes

- does NP include P? that is, is every problem in P also in NP?
 - yes – if you can solve a problem in polynomial time, you can verify a possible solution by computing the solution yourself and comparing
- are there problems in NP that aren't in P?
 - probably
 - (proving this one way or the other will get you fame and a million dollars)
- are there problems that aren't in NP?
 - yes e.g. function problems (NP is only decision problems), the halting problem (undecidable)

CPSC 327: Data Structures and Algorithms • Spring 2026

28

Famous Complexity Classes

P – decision problems solvable by a deterministic Turing machine in polynomial time

NP – decision problems verifiable by a deterministic Turing machine in polynomial time

FP – function problems solvable by a deterministic Turing machine in polynomial time

FNP – function problems verifiable by a deterministic Turing machine in polynomial time

CPSC 327: Data Structures and Algorithms • Spring 2026

29

Famous Complexity Classes

- does FNP contain FP?
 - yes
- are there problems in FNP that aren't in FP?
 - probably (for the same reason as there are probably problems in NP not in P)
- are there problems that aren't in FNP?
 - yes – e.g. enumeration tasks (solution size can be exponential)

CPSC 327: Data Structures and Algorithms • Spring 2026

30

Famous Complexity Classes

- is FP easier or harder than P?
 - no – each can be used a black box to efficiently solve the other problem
 - the solution to the FP version can be used directly to answer the P version's question
 - the P version can be used as a black box to find the FP solution in polynomial time using one-sided binary search
- is FNP easier or harder than NP?
 - [Bellare, Goldwasser 1994] under certain assumptions, there are FNP problems that are harder than their corresponding NP problems
 - i.e. there seem to be problems in FNP where a solution to the NP version can't be used to efficiently solve the FNP version

Determining Complexity

Reductions are useful for making arguments about complexity.

Let A be a problem with a polynomial-time reduction to B.

- i.e. polynomial time to turn an instance of A into an instance of B, and polynomial time to turn a solution for B into a solution for A

Then B is at least as hard as A.

easy/hard has to do with efficiency of solution

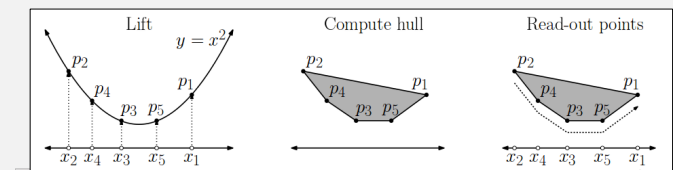
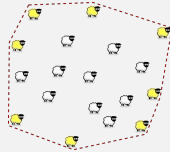
Why?

- if B has an efficient algorithm, A can be solved efficiently via the reduction
- if B doesn't have an efficient algorithm, it may still be possible to solve A efficiently using a different approach – we don't know

Reductions for Lower Bounds

- Sorting can be reduced to convex hull –
- for each element i to be sorted, create a point (i, i^2)
 - compute the convex hull of the points
 - (using an algorithm that outputs the hull points in cyclic order)
 - read points on the hull from left to right, starting with the leftmost point in the hull
 - this is the sorted order of the elements

the convex hull of a set of points is the shape of a rubber band stretched around those points



Reductions for Lower Bounds

Sorting can be reduced to convex hull –

- for each element i , create a point (i, i^2) $O(n)$
- compute the convex hull
 - (using an algorithm that outputs the hull points in cyclic order) $O(??)$
- read points on the hull from left to right, starting with the leftmost point in the hull $O(n)$

Since comparison-based sorting is known to take $\Omega(n \log n)$ time, the ?? step cannot be faster than $n \log n$ or else we'd have a better algorithm for sorting using convex hull.

→ convex hull (if the points on the hull are output in cyclic order) is $\Omega(n \log n)$

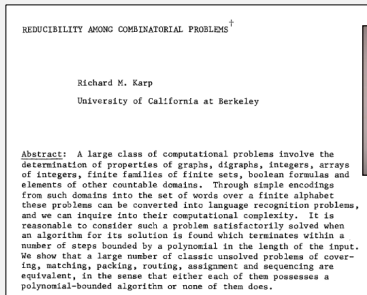
Completeness

Within a class, the *complete* problems are the hardest – if you can solve a complete problem, you can solve every problem in the class.

- **P-complete** – set of problems in P such that every other problem in P is polynomial-time reducible to one in the set
 - these are problems believed to be “inherently sequential” i.e. a parallel computer would not significantly speed them up
- **NP-complete** – set of problems in NP such that every other problem in NP is polynomial-time reducible to one in the set

Karp's 21 NP-Complete Problems

One of the first demonstrations that many common computational problems are computationally intractable. (1972)



Richard Karp, 1935-
American computer scientist

known for work in computer science, combinatorial algorithms, operations research, bioinformatics

- Held-Karp algorithm – TSP
- Edmonds-Karp algorithm – max flow
- 21 NP-complete problems
- Hopcroft-Karp algorithm – matchings in bipartite graphs
- Karp-Lipton theorem – complexity result
- Rabin-Karp string search algorithm

1985 Turing Award for contributions to the theory of NP-completeness

Karp's 21 NP-Complete Problems

| | |
|------------------------------|--|
| clique | is there a set of k vertices in the graph such that every vertex in the set is connected to every other vertex in the set? |
| clique cover | can the graph be partitioned into k cliques? |
| vertex cover | is there a set of k vertices in the graph such that every edge has at least one endpoint in the set? |
| chromatic number | can the graph be colored with k colors? |
| feedback node set | is there a set of k vertices in an undirected graph whose removal leaves the graph without cycles? |
| feedback arc set | is there a set of k edges in a directed graph whose removal leaves the graph without directed cycles? |
| directed hamiltonian cycle | is there a directed/undirected cycle which visits every vertex exactly once? |
| undirected hamiltonian cycle | |
| max cut | can the vertices of a graph be split into two sets so that the sum of the weights of the edges between vertices in different sets is at most k ? |
| Steiner tree | version of MST where additional points may be introduced to reduce the overall weight of the tree |

Karp's 21 NP-Complete Problems

| | |
|----------------------------|--|
| CNF SAT | is there an assignment of values to make a boolean expression with only OR and NOT within a clause and clauses joined by AND true? |
| 3-SAT | CNF SAT where there are exactly three variables per clause |
| binary integer programming | linear programming where variables are constrained to the values 0 or 1 |
| set packing | in a collection of sets, is there a group of k that are disjoint? |
| set covering | given a collection of subsets of X , is there a group of k subsets that together contain every element of X ? |
| exact cover | given a collection of subsets of X , is there a group of those subsets such that every element of X is contained in exactly one subset? |
| hitting set | given a collection of subsets of X , is there a subset H of X of size k so that every set in the collection contains at least one element of H ? |
| 3-dimensional matching | given a set of triples (x,y,z) where $x \in X, y \in Y, z \in Z$, is there a collection of triples such that every element of X, Y , and Z occurs exactly once? |
| 0-1 knapsack | is there a set of items with total weight $\leq W$ and total value $\geq V$? |
| partition | can a set of numbers be split into two parts so that the sums of the parts are equal? |
| job sequencing | can a set of jobs be scheduled so that no more than k miss their deadlines? |

Proving NP-Completeness

Most NP-complete problems are proven NP-complete by a reduction to a known NP-complete problem.

If problem A has a polynomial-time reduction to problem B, which of the following can you conclude?

if A is NP-complete, there can't be a polynomial-time algorithm for B

if B is NP-complete, there can't be a polynomial-time algorithm for A

none of the above

Reduction Example

Course scheduling –

Given a set of courses requested by each student, a set of time slots, and an integer k , determine if there is an assignment of courses to time slots with at most k conflicts amongst the students' schedules.

Is course scheduling in NP?

True
False

- yes
 - it is a decision problem (yes/no question)
 - a 'yes' answer is verifiable in polynomial time – given an assignment of courses to time slots, one simply needs to go through each student's schedule and count conflicts

Is course scheduling NP-complete?

Reduction Example

Course scheduling –

Given a set of courses requested by each student, a set of time slots, and an integer k , determine if there is an assignment of courses to time slots with at most k conflicts amongst the students' schedules.

Is course scheduling NP-complete?

- we need a known NP complete problem to reduce to this problem

3-coloring –

Determine if you can color a graph with three colors so that no two adjacent nodes have the same color.

- known to be NP-complete

Reduction Example

Reduce an instance of 3-coloring to an instance of course scheduling:

- vertices → courses
 - each student will have two requested courses, corresponding to the two vertices connected by the edge
- edges → students
 - there will be three timeslots, corresponding to the three colors
- colors → timeslots
 - an illegal 3-coloring means two adjacent nodes with the same color – this corresponds to a student whose courses are scheduled in the same timeslot
 - a schedule with no conflicts thus means a legal 3-coloring

This construction only takes polynomial time.

- traverse the graph – $O(n)$ to create courses for n vertices, $O(m)$ to create students for m edges, $O(1)$ to create three timeslots

remember that reducing A to B means that B is at least as hard as A

- B can't be easier than A, or else we'd have a better algorithm for A
- A could be easier than B, because there may be another algorithm for A

Reduction Example

Thus, course scheduling is also NP-complete.

- if course scheduling could be solved in polynomial time, so could 3-coloring
- if 3-coloring can be solved in polynomial time, so can everything else in NP

Direction of Reductions Matters!

A sudoku puzzle is an instance of an exact cover problem, meaning that sudoku can be reduced to exact cover.

- exact cover: given a collection of subsets of X , is there a group of those subsets such that every element of X is contained in exactly one subset?

| | | | | | | | |
|---|---|---|---|---|--|---|---|
| 8 | | | 4 | 6 | | | 7 |
| | 1 | | | | | 4 | |
| 5 | | 9 | | 3 | | 7 | 8 |
| | 4 | 8 | | 2 | | 1 | 3 |
| | 5 | 2 | | | | | 9 |
| | | 1 | | | | | |
| 3 | | | 9 | 2 | | | 5 |

Exact cover is NP complete.

Does this mean sudoku is NP complete too?

- not necessarily – the hard problem (exact cover) can be used to solve sudoku, but maybe there's some other way to solve sudoku more efficiently

Proving NP-Completeness

But how do you get a known NP-complete problem in the first place?

- need a direct proof!

[1971] Cook-Levin theorem proved SAT is NP-complete.

$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$ SAT: is there an assignment of values to variables in a boolean expression so that the expression evaluates to TRUE?

SAT is in NP because

- it is a decision problem
- whether a given assignment of values actually satisfies the expression can be checked in polynomial time

Idea:

- start with a nondeterministic Turing machine which solves some problem in NP
- for each possible input, build a boolean expression which is satisfiable if and only if the machine accepts the input (a “yes” response)

Cook-Levin Details

Ingredients – building a boolean expression to represent the Turing machine's task

- boolean variables represent the state of the Turing machine as it executes its program

| Variables | Intended interpretation | How many? |
|-----------|--|-------------|
| T_{jk} | True if tape cell j contains symbol j at step k of the computation. | $O(p(n)^2)$ |
| H_{ik} | True if the M 's read/write head is at tape cell i at step k of the computation. | $O(p(n)^2)$ |
| Q_{qk} | True if M is in state q at step k of the computation. | $O(p(n))$ |

http://en.wikipedia.org/wiki/Cook-Levin_theorem

- $p(n)$ = number of steps required for the Turing machine to accept or reject an input
 - polynomial function because it solves a problem in NP

Cook-Levin Details

- expressions involving those variables represent the machine's input, goal, program, and rules of operation

| Expression | Conditions | Interpretation | How many? |
|---|---|---|-------------|
| T_{i0} | Tape cell i initially contains symbol j | Initial contents of the tape. For $i > n-1$ and $i < 0$, outside of the actual input i , the initial symbol is the special default/blank symbol. | $O(p(n))$ |
| Q_{s0} | | Initial state of M . | 1 |
| H_{00} | | Initial position of read/write head. | 1 |
| $T_{jk} \rightarrow \neg T_{jk}$ | $j \neq f$ | One symbol per tape cell. | $O(p(n)^2)$ |
| $T_{jk} \wedge T_{j(k+1)} \rightarrow H_{ik}$ | $j \neq f$ | Tape remains unchanged unless written. | $O(p(n)^2)$ |
| $Q_{qk} \rightarrow \neg Q_{qk}$ | $q \neq q'$ | Only one state at a time. | $O(p(n))$ |
| $H_{ik} \rightarrow \neg H_{ik}$ | $i \neq i'$ | Only one head position at a time. | $O(p(n)^3)$ |
| $(H_{ik} \wedge Q_{qk} \wedge T_{i\alpha k}) \rightarrow \bigvee_{(q, \sigma, q', \sigma', d) \in \delta} (H_{i+d(k+1)} \wedge Q_{q'(k+1)} \wedge T_{i\sigma'(k+1)})$ | $k < p(n)$ | Possible transitions at computation step k when head is at position i . | $O(p(n)^2)$ |
| $\bigvee_{f \in F} Q_{fp(n)}$ | | Must finish in an accepting state. | 1 |

http://en.wikipedia.org/wiki/Cook-Levin_theorem

Cook-Levin Details

The transformation from Turing machine to instance of SAT is a polynomial-time reduction.

- $O(p(n)^2)$ variables
- $O(p(n)^3)$ clauses

Transforming the SAT solution to a solution for the NP problem is $O(1)$.

Thus, if SAT could be solved in polynomial time, so could any problem in NP.