Final Exam Review v2

CPSC 320 | 2024 W1

References

Content/Problem/Solutions adapted from:

- Course Material
- https://courses.csail.mit.edu/6.006/fall09/lecture_notes/lecture18.pdf

Part I

Summary of Topics

Main Topics

Introductory Topics:

1. **Stable Matching Problem *(ws: 01-introduction)***
    a. Motivating Problem: SMP
    b. Your introduction to algorithm design
    c. Brute force solutions
2. **Reductions *(ws: 02-smp-reductions)***
    a. Motivating Problem: Resident Matching (RHP)
    b. Reduction from RHP to SMP
    c. (Intro to) proof of correctness
3. **Asymptotic Analysis *(ws: 03-asymptotic-analysis)***
    a. Compare orders of growth for functions
    b. Orders of growth for code (i.e., while- and for- loops)
4. **Graphs (ws: 04-graphs)**
    a. Graph terminology
    b. More formalized problem representations
    c. Simple algorithms (Motivating Problem: Diameter Algorithm)

Main Topics

Algorithm Paradigms:

1. **Greedy Algorithms *(ws: 05-clustering, 06-clustering-completed)***
    a. Design of greedy algorithms
    b. Greedy Optimality (i.e., (1) greedy-stays-ahead, (2) exchange argument)
    c. Motivating Problem: (photo) clustering algorithm (i.e., categorization)
2. **Divide and Conquer Algorithms *(ws: 07-divide-and-conquer)***
    a. Complexity Analysis for Recursions
        i. The Master Theorem
        ii. Recursion Trees (Motivating Problem: Quicksort runtime analysis)
    b. D&C Algorithm Design
        i. Motivating Problem: Stock Market (Dividends and Risks)
3. **(D&C) Prune and Search Algorithms (ws: 08-prune-and-search)**
    a. Motivating Example: Finding median (Quicksort modification)

Main Topics

Algorithm Paradigms (Continued):

4. Dynamic Programming *(ws: 09-dp-change, 10-dp-lcs)*

a. 1-D:
   i. Motivating Problem: coin change
   ii. Recurrences and DP
   iii. Memoization (Top Down)
   iv. DP Algorithm Design (Tabulation, Bottom up)

b. 2-D:
   i. Motivating Problem: longest common subsequence of two strs
   ii. Essentially DP on grids.

Main Topics

Computational Complexity Theory / Theoretical Computer Science

1. **Reductions Revisited *(ws: 11-satisfiability, 12-steiner)***
    a. Reductions as a method of proving complexity class membership
        i. Complexity Classes: P, NP, and NP-completeness
        ii. Motivating Example: Boolean Satisfiability (SAT) & 3-SAT
    b. Reductions to perform asymptotic runtime analysis
    c. Complex Reductions & Reduction Correctness
        i. Motivating Example: Steiner Tree (and 3-SAT)

Part II

Dynamic Programming

Dynamic Programming and Memoization

Naive Approach: brute-force top-down (RECURSIVE) approach

Memoization: top-down (RECURSIVE) approach where we cache the results of function calls and return the cached result if the function is called again with the same inputs.

Tabulation (Dynamic Programming): bottom-up (ITERATIVE) approach where we store results of subproblems in a table and use these results to solve larger subproblems until we solve the entire problem.

(Naive Approach)

Running Example:

It's the *Fibonacci sequence,* described by the recursive formula:

$$F_0 := 0;\ F_1 := 1;$$
$$F_n = F_{n-1} + F_{n-2},\ \text{for all } n \geq 2.$$

Naive Approach: brute-force top-down approach

Trivial algorithm for computing F_n:

naive_fibo(n):
    if $n = 0$: return 0
    else if $n = 1$: return 1
    else: return naive_fibo($n - 1$) + naive_fibo($n - 2$).

(Naive Approach)

It's the *Fibonacci sequence,* described by the recursive formula:

$$F_0 := 0;\ F_1 := 1;$$
$$F_n = F_{n-1} + F_{n-2},\ \text{for all } n \geq 2.$$

Naive Approach: brute-force top-down approach

Trivial algorithm for computing F_n:

naive_fibo(n):
    if $n = 0$: return 0
    else if $n = 1$: return 1
    else: return naive_fibo($n - 1$) + naive_fibo($n - 2$).

What's the problem with this approach?

(Naive Approach)

Naive Approach: brute-force top-down approach

What's the problem with this approach?

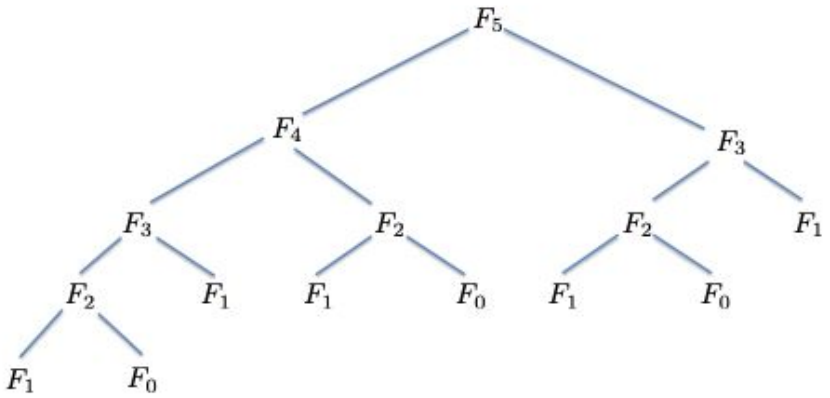


Figure 1: Unraveling the Recursion of the Naive Fibonacci Algorithm.

(Naive Approach)

Naive Approach: brute-force top-down approach

What's the problem with this approach?

Runtime Analysis

Suppose we store all intermediate results in n-bit registers. (Optimizing the space needed for intermediate results is not going to change much.)

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + c \\ &\geq 2T(n-2) + c \\ &\geq \dots \\ &\geq 2^k T(n - 2 \cdot k) + c(2^{k-1} + 2^{k-2} + \dots + 2 + 1) = \Omega(c2^{n/2}), \end{aligned}$$

where c is the time needed to add n-bit numbers. Hence $T(n) = \Omega(n2^{n/2})$.

EXPONENTIAL - BAD!

Problem with recursive algorithm:

Computes $F(n-2)$ twice, $F(n-3)$ three times, etc., each time from scratch.

(Memoization)

Running Example:

It's the *Fibonacci sequence,* described by the recursive formula:

$$F_0 := 0;\ F_1 := 1;$$
$$F_n = F_{n-1} + F_{n-2},\ \text{for all } n \geq 2.$$

Memoization: top-down (RECURSIVE) approach where we cache the results of function calls and return the cached result if the function is called again with the same inputs.

(Memoization)

Memoization: top-down (RECURSIVE) approach where we cache the results of function calls and return the cached result if the function is called again with the same inputs.

Recursive Formulation of Algorithm:

```
memo = { }
fib(n):
        if n in memo: return memo[n]
        else if n = 0: return 0
            else if n = 1: return 1
            else: f = fib(n − 1) + fib(n − 2)
                                   free of charge!
            memo[n] = f
            return f
```

(Memoization)

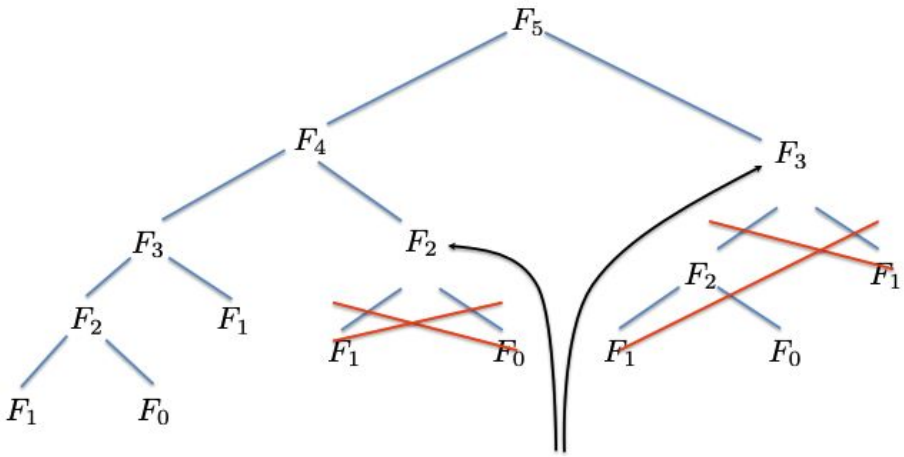


Figure 2: Unraveling the Recursion of the Clever Fibonacci Algorithm.

```
memo = { }
fib(n):
        if n in memo: return memo[n]
        else if n = 0: return 0
             else if n = 1: return 1
             else: f = fib(n − 1) + fib(n − 2)
                                    free of charge!
             memo[n] = f
             return f
```

(Tabulation)

Running Example:

It's the *Fibonacci sequence,* described by the recursive formula:

$$F_0 := 0;\ F_1 := 1;$$
$$F_n = F_{n-1} + F_{n-2},\ \text{for all } n \geq 2.$$

Tabulation (Dynamic Programming): bottom-up (ITERATIVE) approach where we store results of subproblems in a table and use these results to solve larger subproblems until we solve the entire problem.

(Tabulation)

Tabulation (Dynamic Programming): bottom-up (ITERATIVE) approach where we store results of subproblems in a table and use these results to solve larger subproblems until we solve the entire problem.

```
fib(n):
    dp = [1...n]

    dp[0] = 0
    if n > 0:
        dp[1] = 1

    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]

    return dp[n]
```

How could we improve this algorithm?

(Tabulation)

Tabulation (Dynamic Programming): bottom-up (ITERATIVE) approach where we store results of subproblems in a table and use these results to solve larger subproblems until we solve the entire problem.

How could we improve this algorithm?
Only store certain values

```
fib(n):
    if n == 0: return 0
    if n == 1: return 1

    prev2 = 0
    prev1 = 1

    for i in range(2, n + 1):
        curr = prev1 + prev2
        prev2 = prev1
        prev1 = curr

    return prev1
```

Past Final Question (2019 S2)

7 A Golden Opportunity [14 marks]

You're in sales at a mining company and you have an n-centimeter-long bar of 24 karat gold. For each value of i for $1 \leq i \leq n$ (where i is an integer), you have the price you would get for selling an i-centimeter piece of the bar. Your goal is to determine the maximum value obtainable by cutting up your bar of gold and selling the pieces.

For example, if the bar of gold is 8 centimetres long, and the different lengths have the following prices, then the maximum value is 1900 (by cutting into two pieces of length 3 and 5):

length	1	2	3	4	5	6	7	8
price	100	200	800	900	1100	1200	1200	1100

If the prices were listed as below, then the maximum obtainable value would be 1600 (by cutting into eight pieces of length 1):

length	1	2	3	4	5	6	7	8
price	200	300	400	500	600	700	800	900

Past Final Question (2019 S2), Q1

7 A Golden Opportunity [14 marks]

You're in sales at a mining company and you have an n-centimeter-long bar of 24 karat gold. For each value of i for $1 \leq i \leq n$ (where i is an integer), you have the price you would get for selling an i-centimeter piece of the bar. Your goal is to determine the maximum value obtainable by cutting up your bar of gold and selling the pieces.

For example, if the bar of gold is 8 centimetres long, and the different lengths have the following prices, then the maximum value is 1900 (by cutting into two pieces of length 3 and 5):

length	1	2	3	4	5	6	7	8
price	100	200	800	900	1100	1200	1200	1100

If the prices were listed as below, then the maximum obtainable value would be 1600 (by cutting into eight pieces of length 1):

length	1	2	3	4	5	6	7	8
price	200	300	400	500	600	700	800	900

1. **[4 marks]** Complete the following recurrence to compute the maximum obtainable value of a bar of gold of length n, given the prices in an array P (assuming 1-based indexing):

$$\text{MaxValue}(n) = \begin{cases} \square & \text{when } n = 0 \\ \max\limits_{j=1,\ldots,n} \square & \text{when } n \geq 1 \end{cases}$$

Past Final Question (2019 S2), Q1 (Solution)

1. **[4 marks]** Complete the following recurrence to compute the maximum obtainable value of a bar of gold of length n, given the prices in an array P (assuming 1-based indexing):

$$\text{MaxValue}(n) = \begin{cases} 0 & \text{when } n = 0 \\ \max_{j=1,\ldots,n} \text{P[j] + MaxValue(n-j)} & \text{when } n \geq 1 \end{cases}$$

Note: MaxValue(j) + P[n-j] is incorrect (because P[0] is undefined), and MaxValue(j) + MaxValue(n-j) is incorrect (never looks at P!)

Past Final Question (2019 S2), Q2

You're in sales at a mining company and you have an n-centimeter-long bar of 24 karat gold. For each value of i for $1 \leq i \leq n$ (where i is an integer), you have the price you would get for selling an i-centimeter piece of the bar. Your goal is to determine the maximum value obtainable by cutting up your bar of gold and selling the pieces.

For example, if the bar of gold is 8 centimetres long, and the different lengths have the following prices, then the maximum value is 1900 (by cutting into two pieces of length 3 and 5):

length	1	2	3	4	5	6	7	8
price	100	200	800	900	1100	1200	1200	1100

If the prices were listed as below, then the maximum obtainable value would be 1600 (by cutting into eight pieces of length 1):

length	1	2	3	4	5	6	7	8
price	200	300	400	500	600	700	800	900

1. **[4 marks]** Complete the following recurrence to compute the maximum obtainable value of a bar of gold of length n, given the prices in an array P (assuming 1-based indexing):

$$\text{MaxValue}(n) = \begin{cases} \boxed{0} & \text{when } n = 0 \\ \max\limits_{j=1,\ldots,n} \boxed{\text{P[j] + MaxValue(n-j)}} & \text{when } n \geq 1 \end{cases}$$

2. **[8 marks]** Complete the following dynamic programming algorithm to compute the maximum obtainable value in dividing a bar of gold of length n, given the prices in an array P:

```
function ComputeMaxValue(P):
   n = length(P)

   // Set up the array Table (Table[i] should contain MaxValue(i))

   let Table be an array with indexes 0 .. __________        // FILL IN THE BLANK

   Table[0] = ____________                                   // FILL IN THE BLANK

   for i = ____________________________:                     // FILL IN THE BLANK

      // FILL IN THE LOOP BODY



   return _________________                                  // FILL IN THE BLANK
```

Past Final Question (2019 S2), Q2 (Solution)

```
function ComputeMaxValue(P):
   n = length(P)

   // Set up the array Table (Table[i] should contain MaxValue(i))

   let Table be an array with indexes 0 .. __n_____        // FILL IN THE BLANK

   Table[0] = __0_______                                   // FILL IN THE BLANK

   for i = __1___to___n___________:                        // FILL IN THE BLANK

      // FILL IN THE LOOP BODY
      best_i = -infinity  (or 0 - reasonable to assume that all values in P will be positive!)
      for   j = 1 to i:
          val_j = P[j] + Table[i-j]
          if  val_j > best_i:
              best_i = val_j
      Table[i] = ~~val_j~~ best_i


   return Table[n]                                         // FILL IN THE BLANK
```

Past Final Question (2019 S2), Q3

3. **[2 marks]** Give and briefly justify a good asymptotic bound on the runtime and memory usage of your dynamic programming algorithm for question 2.

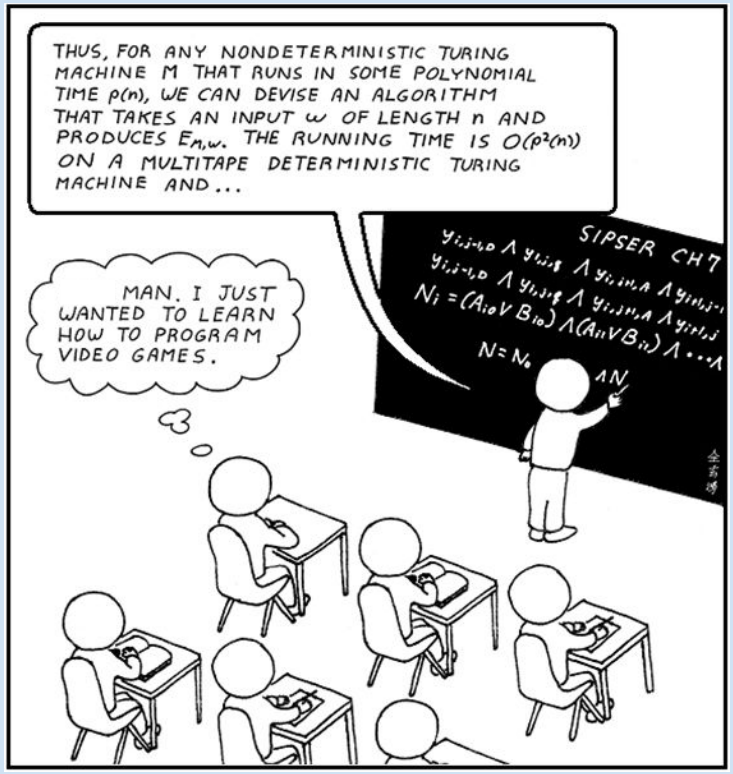
Past Final Question (2019 S2), Q3 (Solution)

3. **[2 marks]** Give and briefly justify a good asymptotic bound on the runtime and memory usage of your dynamic programming algorithm for question 2.

Memory: $\theta(n)$
Runtime: $\theta(n^2)$

Part III

Reductions Revisited (and Computational Complexity Theory)



Optimization and Decision Problems

Optimization Problems: we want to find solution *s* that maximizes or minimizes some function *f(s)*

Decision Problems: given a parameter k, we want to know if there exists a solution *s* for which $f(s) \geq k$ (maximization) or $f(s) \leq k$ (minimization)

If we can solve the decision problem efficienciently, then we can use binary search on *k* to find the answer to optimization problem.

Question: why do we care about this?

Optimization and Decision Problems (Solution)

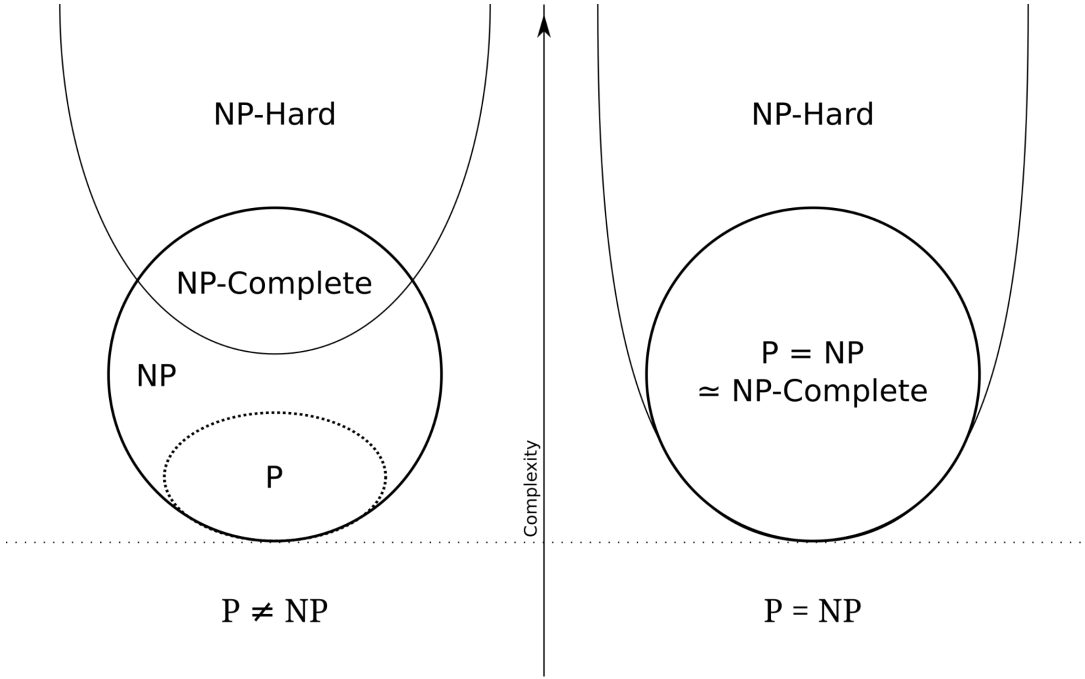
Optimization Problems: we want to find solution *s* that maximizes or minimizes some function *f(s)*

Decision Problems: given a parameter k, we want to know if there exists a solution *s* for which $f(s) \geq k$ (maximization) or $f(s) \leq k$ (minimization)

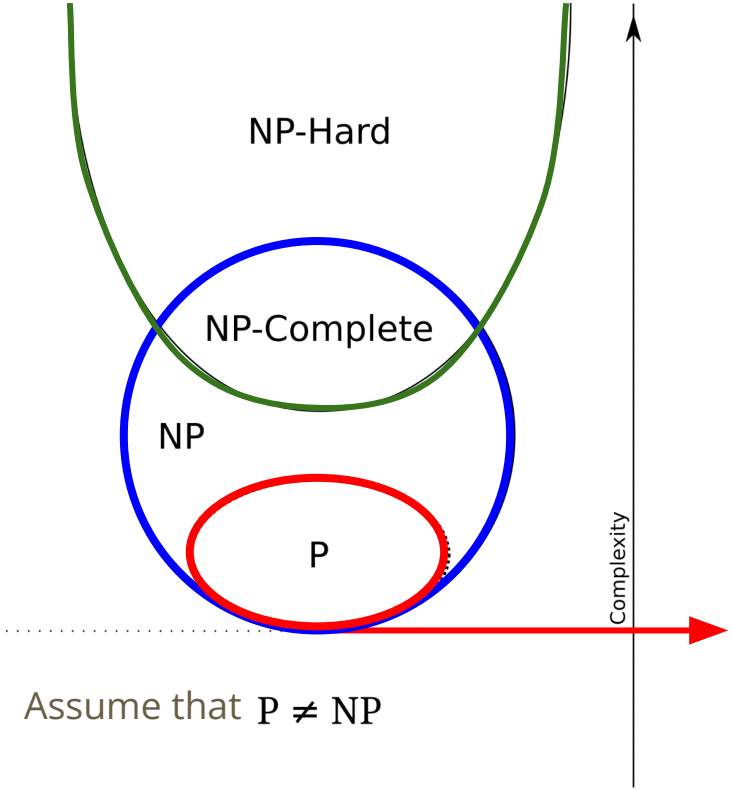
If we can solve the decision problem efficienciently, then we can use binary search on *k* to find the answer to optimization problem.

Question: why do we care about this? Complexity classes are defined in terms decision problems (i.e., yes or no answers)

P, NP, NP-Completeness, and NP-Hard



P, NP, NP-Completeness, and NP-Hard



Assume that $\mathrm{P} \neq \mathrm{NP}$

P: set of all decision problems that can be solved in polynomial time (*efficient solver*)

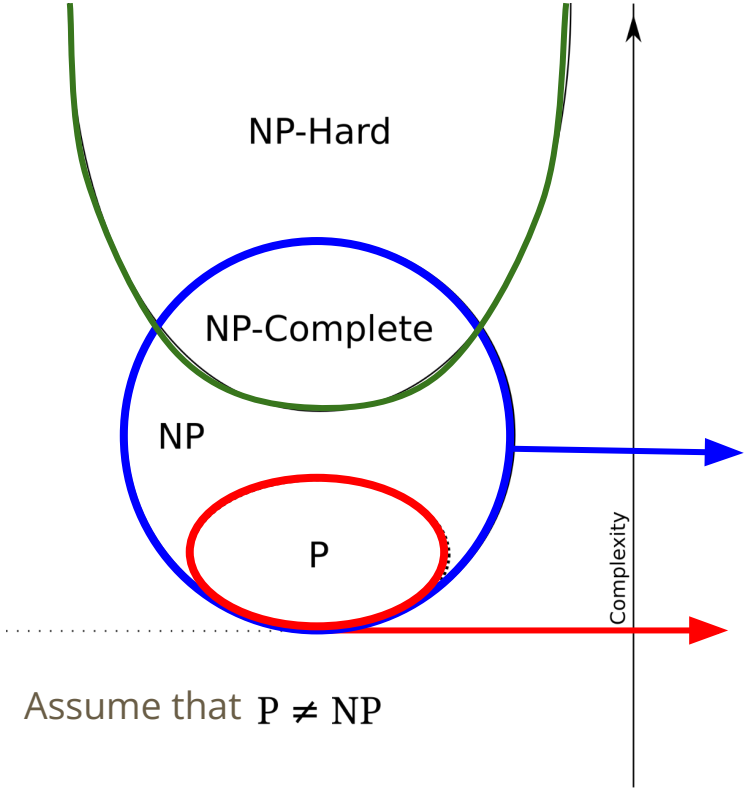
$Y \leq_p X$: Y is polynomial-time reducible to X, or X is at least as hard as Y.

- "At least as hard": solving X efficiently implies we can solve Y efficiently (but not necessarily vice versa)
- Conversely, if Y cannot be solved efficiently (i.e., not in poly-time), then X cannot be solved efficiently (if Y is hard, and we show $Y \leq_p X$, then the hardness spreads to X)

P, NP, NP-Completeness, and NP-Hard

$Y \leq_p X$: Y is polynomial-time reducible to X, or X is at least as hard as Y.

- "At least as hard": solving X efficiently implies we can solve Y efficiently (but not necessarily vice versa)
- Conversely, if Y cannot be solved efficiently (i.e., not in poly-time), then X cannot be solved efficiently (if Y is hard, and we show $Y \leq_p X$, then the hardness spreads to X)



Assume that $P \neq NP$

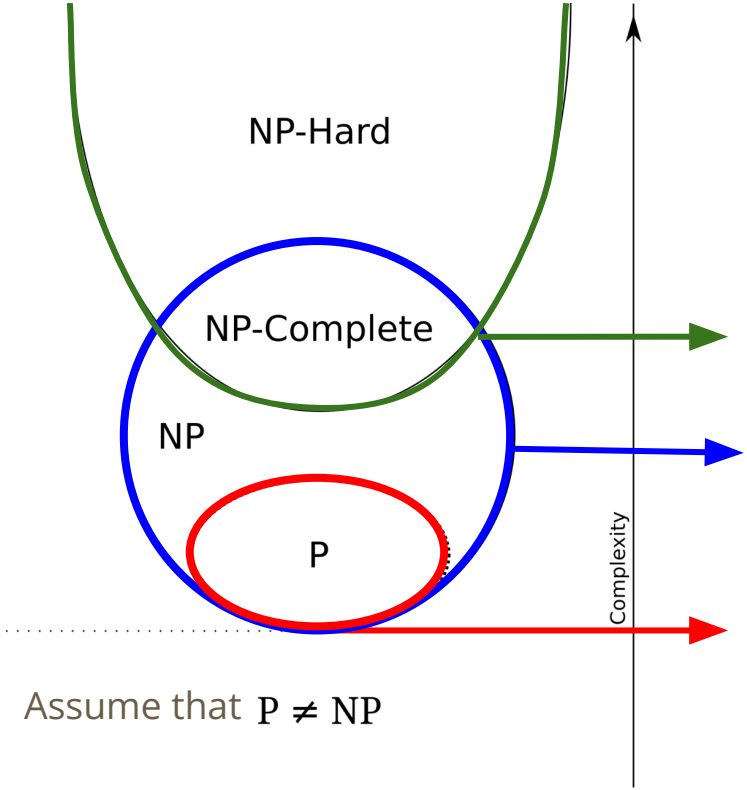
NP: set of all decision problems for which the instances where the answer is "yes" have proofs that can be verified in polynomial time *(efficient verifier)*

P: set of all decision problems that can be solved in polynomial time (*efficient solver*)

P, NP, NP-Completeness, and NP-Hard

$Y \leq_p X$: Y is polynomial-time reducible to X, or X is at least as hard as Y.

- "At least as hard": solving X efficiently implies we can solve Y efficiently (but not necessarily vice versa)
- Conversely, if Y cannot be solved efficiently (i.e., not in poly-time), then X cannot be solved efficiently (if Y is hard, and we show $Y \leq_p X$, then the hardness spreads to X)



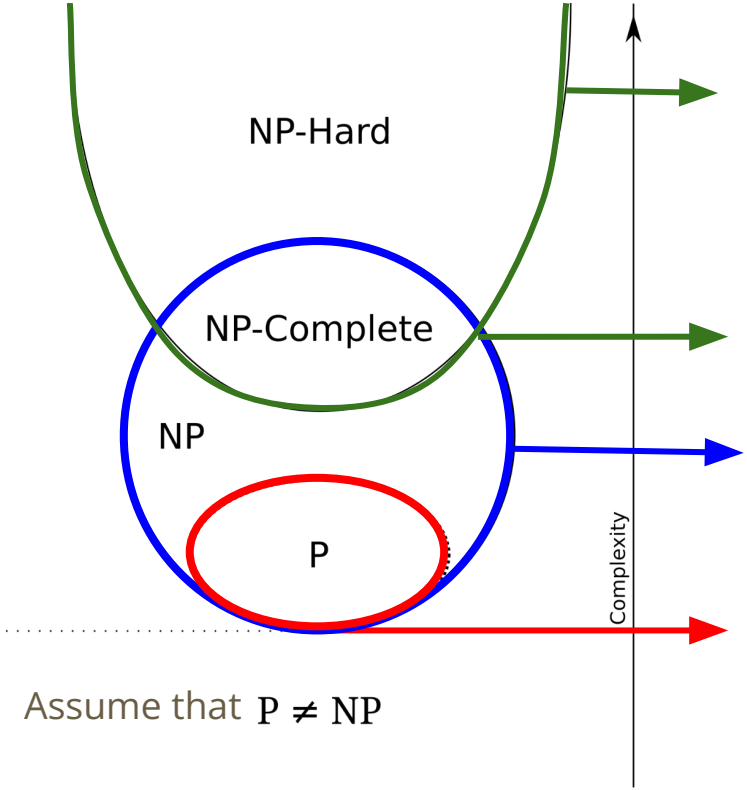
Assume that $P \neq NP$

NP-Complete: set of all decision problems $X \in NP$ (i.e., with efficient verifiers) for which we can reduce all other NP Problems Y to X in polynomial time. I.e., $\forall\ x \in$ NP-Complete, $\forall\ y \in NP$, $Y \leq_p X$

NP: set of all decision problems for which the instances where the answer is "yes" have proofs that can be verified in polynomial time *(efficient verifier)*

P: set of all decision problems that can be solved in polynomial time (*efficient solver*)

P, NP, NP-Completeness, and NP-Hard



Assume that $P \neq NP$

NP-Hard: set of all problems X for which we can reduce all NP-Complete problems Y to X in polynomial time. I.e., $\forall x \in$ NP-Hard, $\forall y \in$ NP-Complete, $Y \leq_p X$

NP-Complete: set of all decision problems $X \in$ NP (i.e., with efficient verifiers) for which we can reduce all other NP Problems Y to X in polynomial time. I.e., $\forall x \in$ NP-Complete, $\forall y \in$ NP, $Y \leq_p X$

NP: set of all decision problems for which the instances where the answer is "yes" have proofs that can be verified in polynomial time *(efficient verifier)*

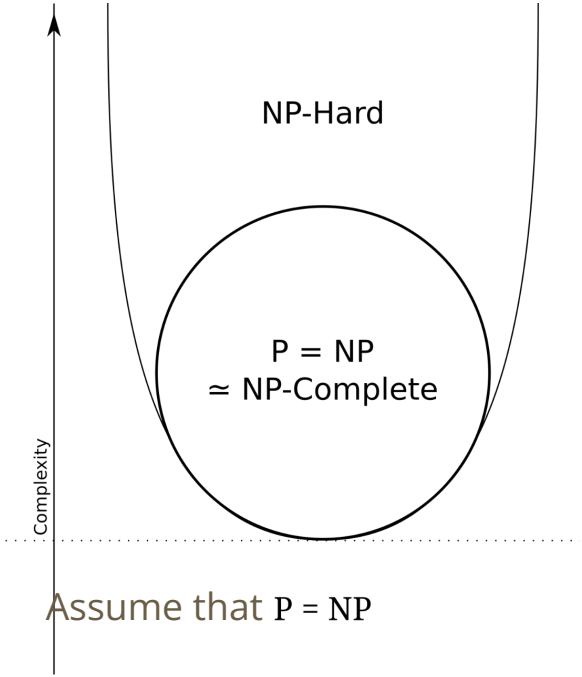
P: set of all decision problems that can be solved in polynomial time (*efficient solver*)

$Y \leq_p X$: Y is polynomial-time reducible to X, or X is at least as hard as Y.

- "At least as hard": solving X efficiently implies we can solve Y efficiently
- Conversely, if Y cannot be solved efficiently (i.e., not in poly-time), then X cannot be solved efficiently (if Y is hard, and we show $Y \leq_p X$, then the hardness spreads to X)

V2 Update: broader **NP-hard** definition (i.e., v2 was "$\exists y \in$ NP-Complete", in v3 we have "$\forall y \in$ NP-Complete")

P, NP, NP-Completeness, and NP-Hard



Assume that $P = NP$

Practice Problem 1

5.1 NP Truths

1. Select from the following list of statements those that are true. (Recall that the 3SAT problem is NP-complete.)

   ◯ 3SAT does not have a polynomial-time algorithm if and only if P $\neq$ NP.

   ◯ If P = NP then the problem of determining whether an undirected, unweighted graph is connected is NP-complete.

   ◯ If there is a polynomial-time reduction from decision problem X to 3SAT (i.e., $X \leq_p 3SAT$), then X must be in NP.

   ◯ If there is a polynomial-time reduction from decision problem 3SAT to X (i.e., $3SAT \leq_p X$), then X must be NP-complete.

   ◯ If there is no polynomial-time reduction from decision problem X to 3SAT (i.e., it is not the case that $X \leq_p 3SAT$), then X cannot be in NP.

Practice Problem 1 (Solution)

5.1 NP Truths

1. Select from the following list of statements those that are true. (Recall that the 3SAT problem is NP-complete.)

- [x] 3SAT does not have a polynomial-time algorithm if and only if P $\neq$ NP.
- [x] If P = NP then the problem of determining whether an undirected, unweighted graph is connected is NP-complete.
- [x] If there is a polynomial-time reduction from decision problem X to 3SAT (i.e., $X \leq_p 3SAT$), then X must be in NP.
- [ ] If there is a polynomial-time reduction from decision problem 3SAT to X (i.e., $3SAT \leq_p X$), then X must be NP-complete.
- [x] If there is no polynomial-time reduction from decision problem X to 3SAT (i.e., it is not the case that $X \leq_p 3SAT$), then X cannot be in NP.

Practice Problem 2.0

5.2 Super Mario Brothers is Hard!

Determining whether or not a generalized level of the Super Mario Brothers game can be successfully completed (Mario reaches the flag) is NP-hard. That is, if we refer to this decision problem as MARIO, then it is a fact that for any problem X in NP,

[ ] $X \leq_p$ MARIO
[ ] MARIO $\leq_p$ X

(Select all that apply)

Practice Problem 2.0 (Solution)

5.2 Super Mario Brothers is Hard!

Determining whether or not a generalized level of the Super Mario Brothers game can be successfully completed (Mario reaches the flag) is NP-hard. That is, if we refer to this decision problem as MARIO, then it is a fact that for any problem X in NP,

[X] $X \leq_p$ MARIO
[ ] MARIO $\leq_p$ X

(Select all that apply)

Practice Problem 2.1

Determining whether or not a generalized level of the Super Mario Brothers game can be successfully completed (Mario reaches the flag) is NP-hard. That is, if we refer to this decision problem as MARIO, then it is a fact that for any problem X in NP,

$$X \leq_p \text{MARIO}.$$

(If you're not familiar with the Super Mario Brothers game, no worries, this fact is all you need. Details of the game are irrelevant for completing this question.)

1. Let's assume that P $\neq$ NP. What can we conclude from the fact provided above? Check all that apply.

[ ] 3D-Matching $\leq_p$ MARIO
[ ] MARIO $\leq_p$ 3D-Matching

(Select all that apply)

Practice Problem 2.1 (Solution)

Determining whether or not a generalized level of the Super Mario Brothers game can be successfully completed (Mario reaches the flag) is NP-hard. That is, if we refer to this decision problem as MARIO, then it is a fact that for any problem X in NP,

$$X \leq_p \text{MARIO}.$$

(If you're not familiar with the Super Mario Brothers game, no worries, this fact is all you need. Details of the game are irrelevant for completing this question.)

1. Let's assume that P $\neq$ NP. What can we conclude from the fact provided above? Check all that apply.

[X] 3D-Matching $\leq_p$ MARIO
[ ] MARIO $\leq_p$ 3D-Matching

(Select all that apply)

Practice Problem 2.2

Determining whether or not a generalized level of the Super Mario Brothers game can be successfully completed (Mario reaches the flag) is NP-hard. That is, if we refer to this decision problem as MARIO, then it is a fact that for any problem X in NP,

$$X \leq_p \text{MARIO}.$$

(If you're not familiar with the Super Mario Brothers game, no worries, this fact is all you need. Details of the game are irrelevant for completing this question.)

2. Again assume that P $\neq$ NP. What can we conclude? Check all that apply. Note that the 2D matching problem (which is a decision version of the bipartite matching problem) is in P.

[ ] 2D-Matching $\leq_p$ MARIO
[ ] MARIO $\leq_p$ 2D-Matching

(Select all that apply)

Practice Problem 2.2 (Solution)

Determining whether or not a generalized level of the Super Mario Brothers game can be successfully completed (Mario reaches the flag) is NP-hard. That is, if we refer to this decision problem as MARIO, then it is a fact that for any problem X in NP,

$$X \leq_p \text{MARIO}.$$

(If you're not familiar with the Super Mario Brothers game, no worries, this fact is all you need. Details of the game are irrelevant for completing this question.)

2. Again assume that P $\neq$ NP. What can we conclude? Check all that apply. Note that the 2D matching problem (which is a decision version of the bipartite matching problem) is in P.

[X] 2D-Matching $\leq_p$ MARIO
[ ] MARIO $\leq_p$ 2D-Matching

(Select all that apply)

Practice Problem 2.3

Determining whether or not a generalized level of the Super Mario Brothers game can be successfully completed (Mario reaches the flag) is NP-hard. That is, if we refer to this decision problem as MARIO, then it is a fact that for any problem X in NP,

$$X \leq_p \text{MARIO}.$$

(If you're not familiar with the Super Mario Brothers game, no worries, this fact is all you need. Details of the game are irrelevant for completing this question.)

3. Finally, let's consider what happens if P = NP. Again, based on the fact above, what can we conclude? Select all that apply.

[ ] 2D-Matching $\leq_p$ MARIO
[ ] MARIO $\leq_p$ 2D-Matching
[ ] 3D-Matching $\leq_p$ MARIO
[ ] MARIO $\leq_p$ 3D-Matching
(Select all that apply)

Practice Problem 2.3 (Solution)

Determining whether or not a generalized level of the Super Mario Brothers game can be successfully completed (Mario reaches the flag) is NP-hard. That is, if we refer to this decision problem as MARIO, then it is a fact that for any problem X in NP,

$$X \leq_p \text{MARIO}.$$

(If you're not familiar with the Super Mario Brothers game, no worries, this fact is all you need. Details of the game are irrelevant for completing this question.)

3. Finally, let's consider what happens if P = NP. Again, based on the fact above, what can we conclude? Select all that apply.

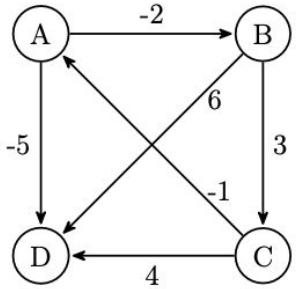
[X] 2D-Matching $\leq_p$ MARIO
[ ] MARIO $\leq_p$ 2D-Matching
[X] 3D-Matching $\leq_p$ MARIO
[ ] MARIO $\leq_p$ 3D-Matching
(Select all that apply)

Practice Problem 3.1 -2024 Summer

7 Cycling Back to Hamiltonian Paths

You are given a directed graph $G = (V, E)$ with weights w_e on its edges $e \in E$. The weights can be positive or negative. The **Zero-Weight Cycle Problem** asks: is there a simple cycle in G so that the sum of the edge weights on this cycle is zero?

For example, in the graph below the cycle $A \to B \to C \to A$ has a total weight of $-2 + 3 + (-1) = 0$, forming a zero-weight cycle.



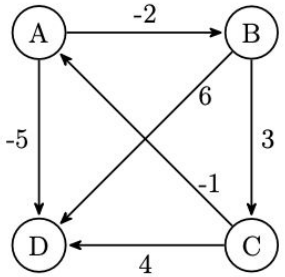
1. [3 points] Prove that Zero-Weight Cycle is in NP.

Practice Problem 3.1 (Solution)

7 Cycling Back to Hamiltonian Paths

You are given a directed graph $G = (V, E)$ with weights w_e on its edges $e \in E$. The weights can be positive or negative. The **Zero-Weight Cycle Problem** asks: is there a simple cycle in G so that the sum of the edge weights on this cycle is zero?

For example, in the graph below the cycle $A \to B \to C \to A$ has a total weight of $-2 + 3 + (-1) = 0$, forming a zero-weight cycle.



1. [3 points] Prove that Zero-Weight Cycle is in NP.

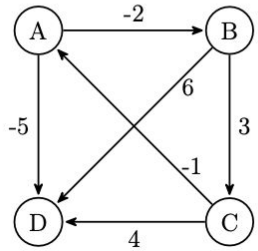
   Given a subset of edges in E, we can verify in linear time (or maybe quadratic with a very simple implementation) whether those edges form a simple cycle, and whether the sums of the edge weights sum to zero. Therefore, Zero-Weight Cycle is in NP.

Practice Problem 3.2

7 Cycling Back to Hamiltonian Paths

You are given a directed graph $G = (V, E)$ with weights w_e on its edges $e \in E$. The weights can be positive or negative. The **Zero-Weight Cycle Problem** asks: is there a simple cycle in G so that the sum of the edge weights on this cycle is zero?

For example, in the graph below the cycle $A \to B \to C \to A$ has a total weight of $-2 + 3 + (-1) = 0$, forming a zero-weight cycle.



2. [9 points] Recall the (directed) Hamiltonian Path problem: given a directed graph $G = (V, E)$, a start node s, and an end node t, does there exist a path from s to t that goes through every vertex in V?

   Complete the proof that Zero-Weight Cycle is NP-complete by providing a reduction from Hamiltonian Path to Zero-Weight Cycle. You do not need to prove the correctness of your reduction, but you should clearly explain the key components of your reduction and why they are there. (Hint: my reduction adds exactly one new edge to G.)

Practice Problem 3.2 (Solution)

2. [9 points] Recall the (directed) Hamiltonian Path problem: given a directed graph $G = (V, E)$, a start node s, and an end node t, does there exist a path from s to t that goes through every vertex in V?

   Complete the proof that Zero-Weight Cycle is NP-complete by providing a reduction from Hamiltonian Path to Zero-Weight Cycle. You do not need to prove the correctness of your reduction, but you should clearly explain the key components of your reduction and why they are there. (Hint: my reduction adds exactly one new edge to G.)

   Create a graph G' by taking all vertices in V, and taking all edges in E and assigning an edge weight of 1. Then, add an edge from t to s with weight $-(|V| - 1)$.

   If G has a Hamiltonian Path, this creates a zero-weight cycle in the G' by taking the same Hamiltonian path from s to t and adding the edge from t back to s.