Analysis of Algorithms
Comprehensive Exams Problems

Computer and Information Science (CIS) Department, Broolyn College

Amotz Bar-Noy Devorah Kletenik Dina Sokol Noson Yanofsky

April 15, 2020

- This document contains all the 90 problems that appeared in the 18 comprehensive exams from the Spring 2011 semester to the Fall 2019 semester.
- The problems are partitioned into five categories where each category is associated with one of the five problems that appeared in an exams.
  1. Discrete Math
  2. Growth of Functions and Recursion
  3. Sorting and Analysis of Algorithms
  4. Graphs
  5. NP-Completeness and NP-Hardness
- **Disclaimers:**
  - Knowing the solutions to all of these problems does not guarantee passing the comprehensive exam.
  - The list of topics that appears at the beginning of each section is only a partial list that represent the most important topics.
- **Study Resources:**
  - "Introduction to Algorithms" by Cormen Leiserson, Rivest, Stein (3rd Edition), chapters 1,2,3,4,6,7,8,9,10,22,23,34 and appendices A,B,C(without C.5).
  - "Problems on Algorithms" by Ian Parberri, Chapters: 2,3,4,5,6,7,12
    http://www.inf.ufes.br/~raulh/ufes/teaching/courses/ds/texts/poa.pdf
  - "Mathematics for Computer Science" video lectures.
    https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2010/video-lectures/

1 Problem 1: Discrete Math

Topics:

- Basic Combinatorics and enumerations.
- Binomial coefficients.
- Arithmetic and geometric progressions.
- Proofs in general.
- Proof by induction.
- Evaluating sums and other expressions.
- The Fibonacci sequence.
- Solving recursive formulas.

Spring 2011

Let $n \geq 1$ be an integer, and let $a, b > 0$ be positive real numbers. Find a closed form for the following expression:

$$\sum_{k=1}^{n} (ak + b) \ .$$

In other words, you are to eliminate the summation and write the expression as a function of a, b, and n. Explain how you computed your answer.

Fall 2011

Given a string S of length n, $S = s_1 \ldots s_n$. A *substring* of S is a contiguous sequence of characters in S.

1. How many substrings of length 3 are there in S?
2. Let m be an integer such that $1 \leq m \leq n$. How many substrings of length m are there in S?
3. How many substrings of S are there? (Hint: One way is to sum the number of substrings for all values of m.)

Spring 2012

Solve the following recursive formula. Prove the correctness of your solution.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 1 + 2T(n-1) \end{aligned}$$

Fall 2012

Prove the following identity for positive integers n and k where $k \leq n$:

$$k\binom{n}{k} = n\binom{n-1}{k-1}$$

Spring 2013

Given a string S of length n, $S = s_1 \ldots s_n$. A *prefix* of S is a substring of S beginning at s_1.

1. How many distinct prefixes of S are there?
2. Write pseudocode that will print all prefixes of S, each on its own line.
3. Count the number of characters that your algorithm prints.
4. Using big-Oh notation, state the time complexity of your algorithm in terms of n, where the basic operation is printing a single character.

Fall 2013

Assume $n = 3^k$ is a power of 3 for $k \geq 0$. **Accurately** solve the following recursion.

If you cannot find the exact solution, use the big-O notation.

$$\begin{aligned} T(1) &= 0 \\ T(n) &= T(n/3) + 2 \end{aligned}$$

Spring 2014

Given a sequence over a four letter alphabet $\Sigma = \{a, g, c, t\}$. A *trigram* is a contiguous subsequence of length 3, and an n-gram is a contiguous subsequence of length n.

1. How many different possible trigrams are there?
2. Given an integer n, state in terms of n how many different possible n-grams there are. Is this function polynomial or exponential?
3. How many different n-grams are there for all values of n, ranging from 3 to some value k?

Fall 2014

Let c be a constant. Solve the following recursion **accurately** by finding a closed form for $T(n)$ as a function of n and c. Prove that your solution is correct.

If you use induction, note that it has to be done twice since the recursion is different for odd and even n.

$$\begin{aligned} T(1) &= c \\ T(2k) &= 2T(k) \\ T(2k+1) &= 2T(k) + c \end{aligned}$$

Spring 2015

The Fibonacci sequence is a sequence of numbers in which each number is the sum of the two preceding numbers. More formally, $\{F_k \mid k = 0, 1, 2, \ldots\}$ is defined as follows

$$F_0 = 0, \quad F_1 = 1, \qquad \text{and} \qquad \forall \mathrm{k} \geq 2, \quad \mathrm{F_k} = \mathrm{F_{k-1}} + \mathrm{F_{k-2}} \ .$$

The infinite sequence is: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$

Based on the definition, prove that $2F_k = F_{k+1} + F_{k-2}$ for $k \geq 2$.

Note: it is not necessary to use induction.

Fall 2015

Prove (by induction or by a direct proof) the following identity for any $n \geq 1$.

$$\sum_{i=1}^{n} (i \cdot i!) \;=\; (n+1)! - 1$$

Spring 2016

Prove the following identity:

$$\sum_{i=0}^{k} 3^i = \frac{3^{k+1} - 1}{2}$$

Fall 2016

Prove by induction:

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2} \right)^2$$

Spring 2017

Find the sum of the following n terms. Note that the last term is n and not $2n$.

$$\left(\sum_{i=1}^{i=n-1} 2i \right) + n \;=\; 2 + 4 + 6 + \cdots + (2n-4) + (2n-2) + n \;.$$

Hint: You may use the identity $1 + 2 + \cdots + n = n(n+1)/2$.

Fall 2017

Let $a_n = q a_{n-1}$ for some constant q, and let $a_0 = 4$.

Prove by induction that $a_n = 4q^n$ for all $n \geq 1$.

Spring 2018

Prove the following identity for an integer $n \geq 1$.

$$\sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3} \;.$$

Fall 2018

Prove the triangle inequality: $|x + y| \leq |x| + |y|$ for any real numbers x and y.

Recall that $|z| = z$ for a positive z and that $|z| = -z$ for a negative z.

Spring 2019

1. For $n \geq 1$, give a formula in terms of n for the sum of the first n odd numbers; i.e., what is the value of $\sum_{i=1}^{n}(2i - 1)$?

   Hint: Evaluate the sum for $n = 1, 2, 3, 4, \ldots$

2. Use mathematical induction to prove the formula that you gave in part (a).

Fall 2019

Prove the following identity for any positive integer $n \geq 1$.

$$\sum_{i=1}^{n} \sum_{j=1}^{i} 2 = n(n + 1)$$

2 Problem 2: Growth of Functions and Recursive formulas

Topics:

- Growth of functions.
- The O, Ω, and Θ notation.
- The hierarchy of functions and the o amd ω notation.
- Expressing the complexity of algorithms with recursive formulas.
- Solving recursive formulas.
- The master theorem.
- Asymptotic complexity.
- Worst case and average case upper, lower, amd tight bounds.

Spring 2011

Let A be an un-sorted array with n distinct integers and let B be a sorted array with n^7 distinct integers. Let x be a number that appears in both A and B. Alice is checking if $x \in A$ and Bob is checking if $x \in B$. It costs one dollar per comparison between x and an integer in an array. Both Alice and Bob attempt to spend the least possible.

Who will asymptotically pay less? Justify your answer.

Fall 2011

In each of the following, state whether the statement is true or false. If true, provide justification, and if false, give the correct relation.

1. $f(n) = \log n$, $g(n) = \sqrt{n}$ and $f = O(g)$.
2. $f(n) = 2^n$, $g(n) = 4^n$ and $f = \Omega(g)$.
3. $f(n) = n^3 - n^2$, $g(n) = n^2$ and $f = O(g)$.
4. $f(n) = 2n + \log n$, $g(n) = 2n \log n$ and $f = \Theta(g)$.
5. $f(n) = \log_2 n$, $g(n) = log_{10} n$ and $f = \Theta(g)$.

Spring 2012

True or false? Justify your answers.

1. Is $2^{n+1} = O(2^n)$?
2. Is $2^{2n} = O(2^n)$?
3. Is $\log(n^2) = O(\log(n))$?
4. Is $\log(2^n) = O(\log(n))$?

Fall 2012

A problem P has an upper bound complexity $O(n^2)$ and a lower bound complexity $\Omega(n \log n)$.

1. Could someone design an algorithm that solves the problem with complexity n^2? Justify your answer.
2. Could someone design an algorithm that solves the problem with complexity $0.5n \log n$? Justify your answer.
3. Could someone design an algorithm that solves the problem with complexity $100n$? Justify your answer.

Spring 2013

Given the following recurrence formula:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(n/2) + \sqrt{n} \end{aligned}$$

1. Iterate three levels of the recursion either by using the substitution method, or by drawing the top three levels of the recursion tree.
2. Use the Master Theorem to find a closed form for the above formula.

Fall 2013

Suppose that you have two different algorithms to solve a given problem. Algorithm A has worst-case time complexity $\Theta(n^2)$ and Algorithm B has worst-case time complexity $\Theta(n \log n)$. Which of the following statements are true and which are false? Justify your answers.

1. Algorithm B runs faster than algorithm A on all possible inputs of length n, for $n \geq 1$.
2. There exists n_0 such that for any $n > n_0$, algorithm B runs faster than algorithm A on all inputs of length n.
3. It is possible that for $n = 100$, algorithm A runs faster than algorithm B on all possible inputs of length 100.

Spring 2014

Suppose that an algorithm divides a file into three equal size pieces, using time n for a file of size n. It then processes (recursively) each piece with size greater than 1. You may assume that the size of the file is such that each time it is possible to divide into three equal size pieces.

1. Write a recurrence relation for the running time of the algorithm.
2. Solve the recurrence relation.

Fall 2014

Let A be an algorithm that solves problem P whose worst-case time complexity is $\Theta(n^2)$.

1. Is it possible that for some instances of P the running time of A is n?
2. Is it possible that for some instances of P the running time of A is n^3?

Justify your answers.

Spring 2015

Match the following 10 functions into 5 pairs. If $f(n)$ is paired with $g(n)$ then $f(n) = \Theta(g(n))$,

$\underline{2^{n+1}}$; $\underline{n}$; $\underline{\log_2(n^2)}$; $\underline{\log_2^2(n)}$; $\underline{n^2/100}$; $\underline{2^n}$; $\underline{\log_2(n)}$; $\underline{100n^2 - 500n}$; $\underline{\log(2^n)}$; $\underline{\log_{10}^2(n)}$

Fall 2015

For two integers k and n ($1 \leq k \leq n$), let X be an **un-sorted** array containing k distinct positive integers and let Y be a **sorted** array containing n distinct positive integers. The goal is to determine if **all** the k numbers from X appear in Y.

Algorithm A performs k binary searches in array Y one per each number from array X.

Algorithm B first sorts array X. Then it performs a sequential search of $X[1], X[2], \ldots, X[k]$ in array Y but starts the sequential search of $X[i]$ at the index at which the sequential search of $X[i-1]$ stopped.

Which algorithm is asymptotically better for $k = \sqrt{n}$? Justify your answer.

Spring 2016

Order the following 10 functions such that if $f(n)$ appears before $g(n)$ then $f(n) = O(g(n))$.

$\underline{2^n}$; $\underline{n}$; $\underline{\log(n)}$; $\underline{\log^2(n)}$; $\underline{n^2}$; $\underline{3^n}$; $\underline{n!}$; $\underline{\sqrt{n}}$; $\underline{n^{100}}$; $\underline{n/\log(n)}$

Fall 2016

For each of the following questions, give an example of a function that satisfies the criteria, or state that none exist. There is no need to justify your answers.

1. A function that is $O(\log n)$.
2. A function that is both $\Omega(n \log n)$ and $O(n^2)$.
3. A function that is both $O(2^n)$ and $\Omega(3^n)$.
4. A function that is $O(n)$ but not $\Theta(n)$.

Spring 2017

Let $f(n) = 1000n$ and $g(n) = n^2/1000$. For each one of the following 4 parts, find a function $h(n)$ that satisfies the conditions of this part or explain why such a function does not exist.

1. $h = O(f)$ and $h = O(g)$.
2. $h = O(f)$ and $h = \Omega(g)$.
3. $h = \Omega(f)$ and $h = O(g)$.
4. $h = \Omega(f)$ and $h = \Omega(g)$.

Fall 2017

Express each of the following functions in Θ notation. For example, if the function is $7n^2$, it should be expressed as $\Theta(n^2)$.

1. $100 + 5\log n + n^2/47$
2. $n/3 + 3^n + 3n + n^3 + 3/n$
3. $\log_2 8$
4. $\log n + \sqrt{n}$

Spring 2018

Let f, g, and h be three functions. It is known that $f(n) > g(n)/2$ for all integers $n > 2018$ and that $f(n) < 3h(n)$ for all integers $n > 8102$. Which of the following statements is **always** true? Justify your answer.

1. $g = O(h)$.
2. $g = \Omega(h)$.
3. $g = \Theta(h)$.

Fall 2018

Suppose an array of size n can be divided *exactly* in half recursively until each part has size 1.

1. Let k be the number of recursive iterations. Give a formula for n in terms of k.
2. Now suppose algorithm A works by dividing the given array in half recursively and processes each part in $O(f(n))$-time. Write TRUE or FALSE for each of the following statements. *Justify* your answers.
   (a) Algorithm A must have an exponential time complexity.
   (b) It is possible that the time complexity of algorithm A is $\Theta(f(n)\log n)$.
   (c) It is possible that the time complexity of algorithm A is constant.

Spring 2019

Consider the following pairs of functions

1. $f(n) = \sqrt{n}$, $g(n) = \log n$
2. $f(n) = 5000$, $g(n) = \log_2 16$
3. $f(n) = 2n + 100$, $g(n) = 5n - 30$
4. $f(n) = 2^n$, $g(n) = n!$

For each pair of functions in (a), (b), (c), and (d), indicate which of the following is true, and *justify* your answers.

- $f(n) = \Theta(g(n))$
- $f(n) = O(g(n))$ and $f(n) \neq \Theta(g(n))$
- $f(n) = \Omega(g(n))$ and $f(n) \neq \Theta(g(n))$

Fall 2019

For each of the following functions, give an example of a function that is $\Theta(f)$, a function that is $O(f)$ but not $\Theta(f)$, and a function that is $\Omega(f)$ but not $\Theta(f)$.

1. $f(n) = n^2 + n + 1$.
2. $f(n) = \log(n) + \log\log(n) + 8$.
3. $f(n) = 2^n + \sqrt{n}$.

3 Problem 3: Sorting and analysis of algorithms

Topics:

- Binary search.
- Order statistics: finding the minimum and maximum, finding the median, ...
- Sorting algorithms: Bubble Sort, Insert Sort, Merge Sort, Quick Sort, Heap Sort, ...
- Analyzing the running time of algorithms.
- Algorithms on arrays and matrices.
- Divide and Conquer algorithms.
- Greedy algorithms

Spring 2011

Let A be an un-sorted array with $n \geq 2$ distinct positive integers where at least one of them is odd and one of them is even. The numbers are very large and therefore it requires complexity $\Omega(n \log n)$ to sort A.

Design a linear time algorithm to determine if **ALL** the odd numbers in A are smaller than **ALL** the even numbers in A.

Justify your answer.

Fall 2011

You are given a sorted array A with k elements, and an unsorted array B with $\log k$ elements. The problem is to combine the arrays into one sorted array (with n elements, where $n = k + \log k$).

1. Give an efficient algorithm to solve this problem.
2. Analyze the time complexity of your algorithm, in terms of the *number of comparisons* performed.
3. How much space does your algorithm use?

Spring 2012

Let $A = A[1] < \cdots < A[n]$ be a sorted array of n distinct integers, in which each integer is in the range $[1..n+1]$. That is, exactly one integer out of $\{1, \ldots, n+1\}$ is missing from A. Describe an efficient algorithm to find the missing integer. Analyze the worst case complexity (number of accesses to array A) of your algorithm.

Fall 2012

Suppose $A_1, \ldots, A_k$ are k ($k > 1$) sorted arrays each of size n containing a total of kn distinct numbers. Describe an efficient algorithm to find the smallest and largest numbers among all the kn numbers. What is the complexity of your algorithm? Justify your answer.

Spring 2013

Suppose you are given an array A of n sorted numbers that has been circularly shifted k positions to the right. For example, $[35; 42; 5; 15; 27; 29]$ is a sorted array that has been circularly shifted $k = 2$ positions, while $[27; 29; 35; 42; 5; 15]$ has been shifted $k = 4$ positions.

1. Suppose you know the value of k. Give an $O(1)$ algorithm to find the largest number in A.
2. Suppose you do not know the value of k. Give an $O(\log n)$ algorithm to find the largest number in A. For partial credit, you may give an $O(n)$ algorithm. Justify the time complexity of your algorithm.

Fall 2013

Let $A = [A[1] < A[2] < \cdots < A[n]]$ be a *sorted* array containing n *distinct* positive integers. Let x be a positive integer such that both x and $2x$ are not in A. Describe an efficient algorithm to find the number of integers in A that are larger than x and smaller than $2x$. What is the complexity of your algorithm? Justify your answer.

Spring 2014

The *predecessor query* is defined as follows. Given a sorted array A of n distinct positive real numbers, and a real number x (not necessarily in A), find the largest element in the array that is strictly less than x.

1. Write pseudocode for an efficient algorithm to answer a predecessor query. You can define the predecessor of the first element in the array to be zero.
2. What is the time complexity of your algorithm?

Fall 2014

Let A and B be two unsorted arrays of numbers. We say that array A is greater than array B, if **all** the numbers in A are larger than **all** the numbers in B.

To clarify the definition we give the following examples. The array $[20, 18, 14, 9, 10]$ is greater than the array $[3, 8, 2, 5]$ since all of the numbers in the first array are greater than all the numbers in the second array. However, the array $[7, 4, 6, 9]$ is not greater than the array $[3, 5, 1]$ since the 4 in the first array is smaller than the 5 in the second array.

Given two arrays, X and Y, both of size n, describe an efficient algorithm that decides whether X is greater than Y. Explain why your algorithm is correct, and provide an analysis of the time complexity of your algorithm.

Spring 2015

For $k \leq n$, let A be an $k \times n$ matrix with k rows and n columns containing $k \cdot n$ distinct integers. For $1 \leq i \leq k$ and $1 \leq j \leq n$, denote by $A[i, j]$ the value of A at row i and column j.

Assume that each row is sorted. That is, $A[i, j] < A[i, j']$ for $1 \leq i \leq k$ and $1 \leq j < j' \leq n$.

Describe an efficient algorithm to find the second smallest number in A. Explain why your algorithm is correct.

What is the worst-case number of comparisons among entries from A that are performed by your algorithm? Justify your answer.

Fall 2015

The *range* of an array X is defined as $[min(X)..max(X)]$ where $min(X)$ is the minimum number in the array and $max(X)$ is the maximum number in the array.

Define $Y \preceq Z$ for two arrays Y and Z if the range of Z contains the range of Y. That is, both inequalities $min(Z) \leq min(Y)$ and $max(Z) \geq max(Y)$ hold.

Let Y and Z be two **unsorted** arrays each containing n ($n \geq 2$) distinct numbers. Describe an efficient algorithm that determines if $Y \preceq Z$.

What is the worst-case number of comparisons made by your algorithm as a function of n? Justify your answer.

Spring 2016

Let $A = A[1] < A[2] < \cdots < A[n]$ and $B = B[1] < B[2] < \cdots < B[m]$ be two sorted arrays containing $n+m$ distinct integers. B **separates** A if for any two entries $A[i] < A[j]$ in A there exists an entry $B[k]$ in B such that $A[i] < B[k] < A[j]$.

Describe an efficient algorithm that checks if B separates A. What is the time complexity of your algorithm as a function of n and m? Justify your answer.

Fall 2016

Let $A = A[1] < A[2] < \cdots A[n]$ be a sorted array of n distinct integers. Describe a $\Theta(\log n)$ algorithm that determines if there exists an index $1 \leq i \leq n$ such that $A[i] = i$. If such an index exists, the algorithm should return it. If not, it should return the number -1.

What is the time complexity of your algorithm as a function of n?

You will earn partial credit for a correct but non-efficient algorithm.

Spring 2017

Bob selects an integer x in the range $[1..n]$ for $n = 2^k$ and $k > 1$. Alice is trying to find x using the binary search procedure. After exactly k questions of the type: "is x less than i?" for some $1 < i \leq n$, she announces that Bob has selected y. Unfortunately, $x \neq y$ because Bob lied in one of his answers while giving the correct answers to all the other $k-1$ questions.

Justify your answers to the following three questions.

1. If the first answer was a lie, what are the possible values for y?
2. If the first answer was a lie, how many more questions does Alice need to find x assuming Bob will not lie again?
3. If the last answer was a lie, how many more questions does Alice need to find x assuming Bob will not lie again?

Fall 2017

Let A and B be two arrays of integers, both of size n, and let x be an integer. Describe a $\Theta(n \log n)$ algorithm that determines if there exist two elements, one from each array, that add up to x. That is, the algorithm should return a pair of indices (j, k) such that $A[j] + B[k] = x$ if such a pair exists.

You do not need to formally prove the running time of your algorithm but you will receive only partial credit for a $\Theta(n^2)$ algorithm.

Spring 2018

Let $A = [A[1], A[2], \ldots, A[n]]$ be an array of n distinct integers. For $1 \leq j \leq n$, the index j is a *happy* index if $A[i] < A[j]$ for all $1 \leq i < j$.

Describe an $O(n)$-time algorithm that finds **all** the happy indices in the array A.

Partial credit will be given for an $O(n \log(n))$-time algorithm and a minimal credit will be given for an $O(n^2)$-time algorithm.

What is the running time of your algorithm?

Justify the correctness of your algorithm and your complexity claim.

Fall 2018

Suppose you have an array A of n integers with many duplicates, and you are told that there are at most 10 distinct integers in A.

Give a linear time algorithm to sort array A using the comparison-based model for sorting.

You may not use Hashing or any other value-based linear time sorting algorithm like Bucket Sort or Radix Sort.

Spring 2019

Let $A = (A[1], A[2], \ldots, A[n])$ be an array containing n integers that denote the prices of a particular stock over the course of n days. Specifically, $A[i]$ denotes the price of this stock on the ith day. You would like to determine, retroactively, the days to have bought and sold the stock to maximize the profit earned. In other words, find the buy date i and sell date j such that $i < j$ and $A[j] - A[i]$ is maximized. (Note that there is no requirement that i and j be consecutive numbers.) The algorithm should return the profit $A[j] - A[i]$.

For simplicity, you may assume that n is a power of 2.

For example, let $A =$

5	6	9	4	3	10	8	1

Then the algorithm should select 5 as the buy date and 6 as the sell date for a profit of $A[6] - A[5] = 10 - 3 = 7$.

The following divide-and-conquer algorithm with the initial call `maxProfit`(A, 1, n) is **incorrect** (though it works on our example above). Give an example of an input array A on which this algorithm will return an incorrect solution, and explain in general the type of inputs for which the algorithm would *not* work.

```
int maxProfit(Array A, int s, int t)
if s = t then
    return 0
end
if t − s = 1 then
    if A[t] > A[s] then
        return A[t] − A[s]
    end
    return 0
end
int mid = (s+t−1)/2                    // calculates the midpoint of the array
int m1 = maxProfit(A, s, mid)
int m2 = maxProfit(A, mid + 1, t)
return max{m1, m2}
```

Fall 2019

Let X be a sorted array containing $n \geq 1$ distinct integers and let Y be an arbitrary (unsorted) array containing $k \geq 1$ distinct integers. The goal is to find an integer that appears in both arrays.

Algorithm A first sorts Y, then merges X with Y into an array Z of length $n + k$, and finally scans Z to find the first integer, if one exists, that appears twice in Z.

Algorithm B applies a binary search on X for each of the k integers from Y until it finds an integer from Y in X or fails with all the integers in Y.

1. What are the time complexities (number of comparisons between array integers) of each algorithm A and B, as a function of n and k? Justify your answer.
2. Which algorithm is more efficient when $k = n$?
3. Which algorithm is more efficient when $k = \log_2(n)$?

4 Problem 4: Graphs

Topics:

- Graph representations: adjacent matrix and adjacent lists.
- Analyzing graph algorithms.
- Directed and undirected graphs.
- Weighted graphs.
- Labeled and unlabeled graphs.
- Simple graphs and graphs with parallel edges and self loops.
- Classes of graphs: complete graphs (cliques), bipartite graphs, trees, forests, cycles, ...
- Connected graphs and connected components.
- Degrees and neighborhhod of vertices.
- Sub-graphs.
- Graph isomorphism.
- Special sets in graphs: independent sets, cliques, vertex cover sets, dominating sets, ...
- BFS and DFS traversals.
- Minimum spanning trees.
- Paths in graphs, Hamiltonian paths, and Euler paths.
- Graph coloring.
- Tournaments.

Spring 2011

Let $G = (X, Y, E)$ be a complete bipartite graph. X and Y are sets of vertices and E is the set of all possible edges (x, y) where $x \in X$ and $y \in Y$. Assume that X has k vertices and that Y has h vertices for $k > h > 0$.

A set of vertices I is an *independent set* of G if for any two vertices $u, v \in I$, the edge (u, v) does not exist.

Identify the largest possible independent set in the complete bipartite graph G? What is the size of this set?

Justify your answer.

Fall 2011

Let G be a *complete* undirected graph (all possible edges exist) with n vertices. Assume that you run both a Breadth-first (BFS) and a Depth-first (DFS) search on G.

1. What is the height of the BFS-tree? Justify.
2. What is the height of the DFS-tree? Justify. Note that the height of the DFS-tree is equivalent to the depth of recursion of the algorithm.
3. For a general connected graph (i.e. not necessarily complete) what would be the maximum possible height for the BFS-tree?
4. For a general connected graph (i.e. not necessarily complete) what would be the maximum possible height for the DFS-tree?

Spring 2012

A *lonely* edge in a simple undirected graph is an edge $e = (u, v)$ for which the edge e is the only edge adjacent to the vertices u and v. For a given graph $G = (V, E)$, describe an efficient algorithm to identify a lonely edge if such exists. What is the complexity of your algorithm? Justify your answer.

Fall 2012

The following algorithm is applied to a simple undirected graph G with n vertices.

let S be a set containing an arbitrary vertex u of G
let R be the set off all the neighbors of u:
while R is not empty **do**
    let $v \in R$ be an arbitrary vertex from R
    add v to S
    omit v from R
    omit from R all the vertices that are not neighbors of v
end-while

What could you say about set S when the algorithm terminates?

Spring 2013

Let G be a simple and undirected graph with n vertices. Assume G is represented by the adjacency matrix A. A set of vertices D is a *dominating set* if for every vertex u not in D there exists a vertex v in D such that the edge (u, v) belongs to G. Let F be a set of vertices of size k.

1. Describe an efficient algorithm to determine if F is a dominating set in G.
2. What is the time-complexity of your algorithm as a function of n and k?

Fall 2013

The *union* of two graphs with the same set of vertices is a graph with the same vertex set, and all of the edges of both graphs. A graph *contains* another graph if its set of edges includes all of the edges of the other graph.

Let F, G, and H be three graphs on the same set of n vertices. Assume these graphs are represented by their adjacency matrices. Describe an $O(n^2)$ algorithm to decide whether G contains the union of F and H ($F \cup H \subseteq G$).

Spring 2014

A bipartite graph is a graph whose vertices can be partitioned into two subsets such that there is no edge between any two vertices in the same subset. Give an efficient algorithm to determine if a graph G is bipartite.

1. What is the time complexity of your algorithm given the *adjacency list* representation of the graph?
2. What is the time complexity of your algorithm given the *adjacency matrix* representation of the graph?

Fall 2014

Let G be an undirected simple graph with n vertices. For each of the following graphs determine the exact number of edges in G as a function of n and/or other parameters (e.g. d, r, and s). Justify your answers.

1. G is a clique.
2. G is a tree.
3. G is a cycle.
4. G is a d-regular graph (the degree of each vertex is exactly d).
5. G is a complete bipartite graph that has r vertices in one side and s vertices in the other side such that $r + s = n$. (In a complete bipartite graph, each vertex on one side is adjacent to every vertex on the other side.)

Spring 2015

Let H be a sub-graph of a graph G that contains all the vertices of G. That is, every edge in H is also an edge in G but there might be edges in G that are not in H. Which of the following statements are **always TRUE**. Justify your answers.

1. The number of edges in G is greater than or equal to the number of edges in H.
2. The maximum degree in G is greater than or equal to the maximum degree in H.
3. If G has a cycle than H has a cycle.
4. If G is connected then H is connected.
5. If G is bipartite then H is bipartite.

Fall 2015

Let $\mathcal{G}$ be the family of all undirected simple graphs G with n vertices for which it is possible to color the vertices of G with the three colors Yellow, Red, and Blue as follows:

- All the neighbors of any Blue vertex are Yellow.
- All the neighbors of any Red vertex are Yellow.
- All the neighbors of any Yellow vertex are either Blue or Red (but not Yellow).

For each of the following three statements determine if for all graphs $G \in \mathcal{G}$ (i) the statement is always true, (ii) the statement is sometimes true and sometimes false, or (iii) the statement is always false. Justify your answers.

1. G has a triangle: there exist three vertices u, v, w such that the edges $(u, v), (v, w), (w, u)$ exist.
2. G is a tree.
3. G is a bipartite graph.

Spring 2016

Let G be an undirected graph with n vertices and m edges. Let C be a coloring of the vertices of G with the k colors $1, 2, \ldots, k$ (if $C(v) = j$ then the color of vertex v is j). C is a **legal** coloring of G if $C(u) \neq C(v)$ for every edge (u, v) of G.

Describe an efficient algorithm that checks if a coloring C is a legal coloring. What is the time complexity of your algorithm as a function of n and m? Justify your answer. Specify the data structure you use to represent the graph.

Fall 2016

The *reverse* of a directed graph $G = (V, E)$ is another directed graph, $G^R = (V, E^R)$ where $E^R = \{(v \to u) | (u \to v) \in E\}$. That is, the set of vertices is the same but the edges are reversed.

Describe an efficient algorithm that, given a representation of G with adjacency lists, outputs the adjacency list representation of G^R. Your algorithm should run in time linear of n and m, where n denotes the number of vertices and m denotes the number of edges. Be precise in your answer.

Spring 2017

Two simple (no self loops and no parallel edges) undirected graphs G and H are **isomorphic** if there exists a 1-1 function f from the vertices of G to the vertices of H such that an edge (u, v) exists in G **if and only if** the edge $(f(u), f(v))$ exists in H.

For each one of the following 4 parts say if G and H (i) are always isomorphic, (ii) could never be isomorphic, (iii) sometimes isomorphic. Justify your answers.

1. Both graphs have the same number of edges. However, G has n vertices while H has $n + 1$ vertices for $n \geq 1$.
2. Both graphs have $n \geq 2$ vertices and exactly one edge.
3. Both graphs have $n \geq 2$ vertices and exactly two edges.
4. All the vertices have exactly one neighbor.
5. All the vertices have exactly two neighbors.

Fall 2017

Let $G = (V, E)$ be a directed graph with n vertices and m edges. For a vertex v, the $indegree(v)$ is the number of edges $(u \to v)$ into v and the $outdegree(v)$ is the number of edges $(v \to u)$ exiting v.

For the following two representations of G, give an efficient algorithm to compute $indegree(v)$ of a given vertex v. State the running time of each of your algorithms as a function of n and m.

1. G is represented by adjacency lists, where each vertex is represented by a linked list of its outgoing edges.
2. G is represented by an $n \times n$ adjacency matrix A, where $A[i, j] = 1$ if and only if the directed edge $(i \to j)$ is in G.

Spring 2018

Let G be a simple graph with $n > 1$ vertices. Assume that the degree of each vertex is $1 < \Delta < n$. The following algorithm outputs a simple path of length k edges.

1. Round 0: Let v_0 be an arbitrary vertex.
2. Round 1: Let v_1 be an arbitrary neighbor of v_0.
3. Round $k > 1$:
   (a) Let $P = v_0, v_1, \ldots, v_{k-1}$ be the simple path of length k found so far.
   (b) If all of the neighbors of v_{k-1} are in P then **Return(P)**.
   (c) Else let v_k be one of the neighbors of v_{k-1} that is not in P: append v_k to P.
   (d) Go to the next round.

Explain why the length of the output path is **at least** Δ.

Fall 2018

Let G be a simple, connected, undirected graph with n vertices. The height of a spanning tree of G is the distance in edges from the root of the tree to its furthest leaf.

Both the DFS and the BFS traversals when applied to G generate spanning trees.

In the following you need to find the height of the spanning trees generated by each of these two graph traversals when applied to the complete graph and the cycle graph. Justify your answers.

1. What is the height of a DFS-tree of a complete graph?
2. What is the height of a BFS-tree of a complete graph?
3. What is the height of a DFS-tree of a cycle graph?
4. What is the height of a BFS-tree of a cycle graph?

Spring 2019

An *articulation point* of a graph is a vertex whose removal disconnects the graph. Given an undirected connected graph with n vertices and m edges, describe an algorithm to find an articulation point (if one exists) in time $O(n(m+n))$.

Explain why your algorithm is correct and why its running time is $O(n(m+n))$

Hint: What is the structure of the DFS tree if its root is an articulation point?

Fall 2019

Let G be a simple graph (no parallel edges and no self loops) with the $n \geq 1$ vertices $v_1, v_2, \ldots, v_n$. If v_i has d_i neighbors (equivalently, the degree of v_i is d_i), then $D_G = (d_1, d_2, \ldots, d_n)$ is the degree sequence of G.

Let $D = (3, 3, 3, 3, 1)$ be a sequence of length 5. Prove that there is no graph with 5 vertices for which D is its degree sequence.

5 Problem 5: NP-Completeness and NP-Hardness

Topics:

- NP vs. P.
- NP-Completeness and NP-Hardness.
- Reductions.
- Known NP-Complete problems: SAT, 3-SAT, graph coloring, subsetsum, the Travelling Salesperson Problem, Hamiltonian Path, ...
- Exponential time algorithms.

Spring 2011

The Travelling Salesperson Problem (TSP) is a well-known NP-Complete problem. If you prove either that $TSP \in P$ or that $TSP \notin P$ you will get the equivalence of the Nobel prize in Computer Science. Why?

Fall 2011

Let G be a directed graph with n vertices. A *Hamiltonian path* in the graph is a path of length n that includes each vertex exactly once. Deciding whether a graph has a Hamiltonian path is an NP-Complete problem.

Consider a variation of this problem in which the start and end vertices are told to you. That is, v and w are given, and the question is whether G has a Hamiltonian path from v to w.

1. Prove that this variation of the problem is in the class NP.
2. Is this problem NP-Complete?
3. If you answered 'yes,' prove it by doing a reduction, and if you answered 'no' outline an algorithm for the problem.

Spring 2012

Assume that there are proofs for the following four statements. Can you infer anything about A, B, C, and D? Explain your answer.

1. X is a polynomially solvable problem and A is polynomially reducible to X.
2. X is an NP-Hard problem and B is polynomially reducible to X.
3. X is a polynomially solvable problem and X is polynomially reducible to C.
4. X is an NP-Hard problem and X is polynomially reducible to D.

Fall 2012

Let G be a simple undirected graph. An *independent set* is a set of vertices with no edges among them. A *clique* is a set of vertices that contains all the possible edges among them. Let $\mathcal{A}$ be an algorithm whose complexity is $T(n)$ that finds the maximum independent set in a graph

Describe an efficient algorithm to find the maximum clique in the graph based on $\mathcal{A}$. What is the complexity of your algorithm as a function of n and $T(n)$? Explain your answer. The complexity should be polynomial with both n and $T(n)$.

Spring 2013

The *longest-simple-cycle problem* is the problem of finding a simple cycle (i.e. no repeated vertices) of maximum length in a graph.

1. State the decision problem of the longest-simple-cycle problem.
2. Prove that the decision problem in part (a) is in NP.
3. Assume you know how to reduce the Hamiltonian Cycle Problem the longest-simple-cycle problem. What can you conclude?

Fall 2013

Consider the sets P, NP, and NPC. Mark each of the following six statements as true, false, or unknown.

$P \subseteq NP$	$NP \subseteq P$
$P \subseteq NPC$	$NPC \subseteq P$
$NP \subseteq NPC$	$NPC \subseteq NP$

Spring 2014

If I would describe a new problem Q to you, explain how you would prove whether Q belongs to each of the following classes: (i) P, (ii) NP, and (iii) NP-Complete.

Fall 2014

Alice claims that she can solve the SAT problem with an algorithm whose running time is $O(n^{12})$. Bob claims that he has a proof that the Travelling Saleperson Problem (TSP) cannot be solved with a polynomial time algorithm. Can both claims be correct? Can both claims be incorrect? Explain.

Spring 2015

Let A, B, and C be three decision problems. Suppose that there exist polynomial time reductions from A to B ($A \leq_P B$) and from B to C ($B \leq_P C$). Let P be the class of all polynomial time problems and let NPH be the class of all NP-Hard problems.

Which of the following eight combinations are **impossible**? Justify your answer.

1. $A \in$ P, $B \in$ P, $C \in$ P.
2. $A \in$ P, $B \in$ P, $C \in$ NPH.
3. $A \in$ P, $B \in$ NPH, $C \in$ P.
4. $A \in$ P, $B \in$ NPH, $C \in$ NPH.
5. $A \in$ NPH, $B \in$ P, $C \in$ P.
6. $A \in$ NPH, $B \in$ P, $C \in$ NPH.
7. $A \in$ NPH, $B \in$ NPH, $C \in$ P.
8. $A \in$ NPH, $B \in$ NPH, $C \in$ NPH.

Fall 2015

Justify your answers for both parts of the problem.

1. Problem A has a polynomial time solution. Can you learn something about Problem B by finding a polynomial-time reduction from A to B or from B to A?
2. Problem C is an NP-Hard problem. Can you learn something about Problem D by finding a polynomial-time reduction from C to D or from D to C?

Spring 2016

Let P and Q be two NP-Complete problems.

1. Is it possible to find a polynomial time algorithm for P and at the same time prove that it is *impossible* to solve Problem Q with a polynomial time algorithm? Justify your answer.
2. Is it possible to prove that there is no polynomial time reduction from Problem P to Problem Q or from Problem Q to Problem P? Justify your answer.

Fall 2016

1. Circle all that we know to be true about the Traveling Salesman problem.

   in P in NP NP-complete

2. Suppose that someone give you a polynomial-time (correct) algorithm for the SAT problem. What do we now know about the Traveling Salesman problem? Circle all that apply.

   in P in NP NP-complete

3. Suppose that (instead) someone gives you a polynomial-time (correct) algorithm for the Minimum Spanning Tree problem. What do we now know about the Traveling Salesman problem? Circle all that apply.

   in P in NP NP-complete

Justify your answers.

Spring 2017

Problem A is an NP-Complete problem and problem B can be solved with a polynomial time algorithm. Alice and Bob do not know these facts. Instead, Alice is trying to find a polynomial time reduction from problem A to problem B while Bob is trying to find a polynomial time reduction from problem B to problem A. Who has a fair chance to succeed and who will probably fail? Justify your answers.

Fall 2017

You can win a million dollars if you resolve the open problem "is $\mathsf{P} = \mathsf{NP}$ or not?"

In a few sentences describe two strategies to win the prize. (Think about both of the possible answers to the question.)

Spring 2018

There exists a polynomial time reduction from Problem A to Problem B. Which of the following would prove that P=NP? Justify your answer.

1. (A is in P) **and** (B is in P).
2. (A is in P) **and** (B is in NP-Complete).
3. (A is in NP-Complete) **and** (B is in P).
4. (A is in NP-Complete) **and** (B is in NP-Complete).

Fall 2018

A *coloring* of a graph is an assignment of colors to the vertices of the graph such that no two adjacent vertices have the same color. The minimum number of colors required for coloring a graph is called the *chromatic number* of the graph. Finding the chromatic number of a graph is a known NP-Complete problem.

Suppose you are given a **black box** that does the following. Given an undirected graph $G = (V, E)$, it instantly and correctly answers the yes or no question: "Is there a coloring of G that uses no more than k colors?"

1. Propose an algorithm that uses the black box to find the chromatic number of a given graph. Ananlyze the time complexity of your algorithm assuming the black box takes constant time.
2. In your opinion, does such a black box exist? Explain why or why not.

Spring 2019

Let X be a problem that is in the class NP. Which of the following **must** be true? Select **all** that apply. Justify your answer.

1. Solutions to X can be verified with a polynomial-time algorithm.
2. There is no polynomial-time algorithm for X.
3. If there exists a polynomial-time algorithm for X, then $\mathsf{P} = \mathsf{NP}$.
4. If there exists a polynomial-time algorithm for X, then there must exist a polynomial-time algorithm for the Traveling Salesman Problem.
5. If there exists a polynomial-time algorithm for the Traveling Salesman Problem, then there must exist a polynomial-time algorithm for X.

Fall 2019

For each of the following statements write if it is "True," "False," or "We don't know."

Justify your answers. Correct guesses with no explanation would receive partial credit while correct guesses with a wrong explanation would get no credit.

1. A problem Q can be in P and in NP.
2. A problem Q can be in P and in NP-Complete.
3. A problem Q can be in NP and in NP-Complete.