This is a "core dump" of potential questions for Midterm 2. This should give you a good idea of the *types* of questions that we will ask on the exam—in particular, there *will* be a series of True/False questions—but the actual exam questions may or may not appear in this handout. This list intentionally includes a few questions that are too long or difficult for exam conditions.

Questions from Spring 2014 exams are labeled ⟨⟨*S14*⟩⟩. Questions from labs are labeled ⟨⟨*Lab*⟩⟩.

Recursion.

1. ⟨⟨*S14*⟩⟩ Recall that a *palindrome* is any string that is the same as its reversal. For example, `I`, `DAD`, `HANNAH`, `AIBOHPHOBIA` (fear of palindromes), and the empty string are all palindromes.

   (a) Describe and analyze an algorithm to find the length of the longest substring (not *subsequence*!) of a given input string that is a palindrome. For example, **`BASEESAB`** is the longest palindrome substring of `BUB`**`BASEESAB`**`ANANA` ("Bubba sees a banana."). Thus, given the input string `BUBBASEESABANANA`, your algorithm should return the integer 8.

   (b) Any string can be decomposed into a sequence of palindrome substrings. For example, the string `BUBBASEESABANANA` can be broken into palindromes in the following ways (and many others):

   ```
   BUB + BASEESAB + ANANA
   B + U + BB + A + SEES + ABA + NAN + A
   B + U + BB + A + SEES + A + B + ANANA
   B + U + B + B + A + S + E + E + S + A + B + A + N + A + N + A
   ```

   Describe and analyze an algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string `BUBBASEESABANANA`, your algorithm should return the integer 3.

2. ⟨⟨*S14*⟩⟩ Binaria uses coins whose values are $1, 2, 4, \ldots, 2^k$, the first k powers of two, for some integer k. As in most countries, Binarian shopkeepers always make change using the following greedy algorithm:

   ```
   MakeChange(N):
     if N = 0
         say "Thank you, come again!"
     else
         c ← largest coin value such that c ≤ N
         give the customer one c cent coin
         MakeChange(N − c)
   ```

   For example, to make 37 cent in change, the shopkeeper would give the customer a 32 cent coin, a 4 cent coin, and a 1 cent coin, and then say "Thank you, come again!" (For purposes of this problem, assume that every shopkeeper has an unlimited supply of each type of coin.) Prove that this greedy algorithm always uses the smallest possible number of coins.

3. ⟨⟨*Lab*⟩⟩ Here are several problems that are easy to solve in $O(n)$ time, essentially by brute force. Your task is to design algorithms for these problems that are significantly faster, and ***prove*** that your algorithm is correct.

(a) i. Suppose $A[1..n]$ is an array of n distinct integers, sorted so that $A[1] < A[2] < \cdots < A[n]$. Each integer $A[i]$ could be positive, negative, or zero. Describe a fast algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists..

ii. Now suppose $A[1..n]$ is a sorted array of n distinct ***positive*** integers. Describe an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists. *[Hint: This is **really** easy.]*

(b) Suppose we are given an array $A[1..n]$ such that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a ***local minimum*** if both $A[x-1] \geq A[x]$ and $A[x] \leq A[x+1]$. For example, there are exactly six local minima in the following array:

9	**7**	7	2	**1**	3	7	5	**4**	7	**3**	**3**	4	8	**6**	9
	▲			▲				▲		▲	▲			▲	

Describe and analyze a fast algorithm that returns the index of one local minimum. For example, given the array above, your algorithm could return the integer 5, because $A[5]$ is a local minimum. *[Hint: With the given boundary conditions, any array **must** contain at least one local minimum. Why?]*

(c) i. Suppose you are given two sorted arrays $A[1..n]$ and $B[1..n]$ containing distinct integers. Describe a fast algorithm to find the median (meaning the nth smallest element) of the union $A \cup B$. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1..8] = [2, 4, 5, 8, 17, 19, 21, 23]$$

your algorithm should return the integer 9. *[Hint: What can you learn by comparing one element of A with one element of B?]*

ii. Now suppose you are given two sorted arrays $A[1..m]$ and $B[1..n]$ and an integer k. Describe a fast algorithm to find the kth smallest element in the union $A \cup B$. For example, given the input

$$A[1..8] = [0, 1, 6, 9, 12, 13, 18, 20] \qquad B[1..5] = [2, 5, 7, 17, 19] \qquad k = 6$$

your algorithm should return the integer 7.

4. *⟨⟨Lab⟩⟩* Describe and analyze ***dynamic programming*** algorithms for the following problems. For the first three, use the backtracking algorithms you developed on Wednesday.

(a) Given an array $A[1..n]$ of integers, compute the length of a longest ***increasing*** subsequence of A. A sequence $B[1..\ell]$ is *increasing* if $B[i] > B[i-1]$ for every index $i \geq 2$.

(b) Given an array $A[1..n]$ of integers, compute the length of a longest ***decreasing*** subsequence of A. A sequence $B[1..\ell]$ is *decreasing* if $B[i] < B[i-1]$ for every index $i \geq 2$.

(c) Given an array $A[1..n]$ of integers, compute the length of a longest ***alternating*** subsequence of A. A sequence $B[1..\ell]$ is *alternating* if $B[i] < B[i-1]$ for every even index $i \geq 2$, and $B[i] > B[i-1]$ for every odd index $i \geq 3$.

(d) Given an array $A[1..n]$ of integers, compute the length of a longest ***convex*** subsequence of A. A sequence $B[1..\ell]$ is *convex* if $B[i] - B[i-1] > B[i-1] - B[i-2]$ for every index $i \geq 3$.

(e) Given an array $A[1..n]$, compute the length of a longest ***palindrome*** subsequence of A. Recall that a sequence $B[1..\ell]$ is a *palindrome* if $B[i] = B[\ell - i + 1]$ for every index i.

5. ⟨⟨***Lab***⟩⟩ It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the the list of n songs that the judges will play during the contest, in chronological order.

   You know all the songs, all the judges, and your own dancing ability extremely well. For each integer k, you know that if you dance to the kth song on the schedule, you will be awarded exactly $Score[k]$ points, but then you will be physically unable to dance for the next $Wait[k]$ songs (that is, you cannot dance to songs $k+1$ through $k + Wait[k]$). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

   Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays $Score[1..n]$ and $Wait[1..n]$.

6. ⟨⟨***Lab***⟩⟩ A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, the string `BANANAANANAS` is a shuffle of the strings `BANANA` and `ANANAS` in several different ways.

$$\mathtt{BANANA}_{\mathtt{ANANAS}} \qquad \mathtt{BAN}_{\mathtt{ANA}}{}^{\mathtt{ANA}}{}_{\mathtt{NAS}} \qquad \mathtt{B}_{\mathtt{AN}}{}^{\mathtt{AN}}{}_{\mathtt{A}}{}^{\mathtt{A}}{}_{\mathtt{NA}}{}^{\mathtt{NA}}{}_{\mathtt{S}}$$

   Similarly, the strings `PRODGYRNAMAMMIINCG` and `DYPRONGARMAMMICING` are both shuffles of `DYNAMIC` and `PROGRAMMING`:

$$\mathtt{PRO}^{\mathtt{D}}{}_{\mathtt{G}}{}^{\mathtt{Y}}{}_{\mathtt{R}}{}^{\mathtt{NAM}}{}_{\mathtt{AMMI}}{}^{\mathtt{I}}{}_{\mathtt{N}}{}^{\mathtt{C}}{}_{\mathtt{G}} \qquad {}^{\mathtt{DY}}{}_{\mathtt{PRO}}{}^{\mathtt{N}}{}_{\mathtt{G}}{}^{\mathtt{A}}{}_{\mathtt{R}}{}^{\mathtt{M}}{}_{\mathtt{AMM}}{}^{\mathtt{IC}}{}_{\mathtt{ING}}$$

   Describe and analyze an efficient algorithm to determine, given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, whether C is a shuffle of A and B.

7. ⟨⟨***Lab***⟩⟩ Suppose you are given a sequence of non-negative integers separated by $+$ and $\times$ signs; for example:

$$2 \times 3 + 0 \times 6 \times 1 + 4 \times 2$$

   You can change the value of this expression by adding parentheses in different places. For example:

$$\begin{aligned}
&2 \times (3 + (0 \times (6 \times (1 + (4 \times 2))))) = 6\\
&(((((2 \times 3) + 0) \times 6) \times 1) + 4) \times 2 = 80\\
&((2 \times 3) + (0 \times 6)) \times (1 + (4 \times 2)) = 108\\
&(((2 \times 3) + 0) \times 6) \times ((1 + 4) \times 2) = 360
\end{aligned}$$

   Describe and analyze an algorithm to compute, given a list of integers separated by $+$ and $\times$ signs, the largest possible value we can obtain by inserting parentheses.

   Your input is an array $A[0..2n]$ where each $A[i]$ is an integer if i is even and $+$ or $\times$ if i is odd. Assume any arithmetic operation in your algorithm takes $O(1)$ time.

8. ⟨⟨*Lab*⟩⟩ Recall the class scheduling problem described in lecture on Tuesday. We are given two arrays $S[1..n]$ and $F[1..n]$, where $S[i] < F[i]$ for each i, representing the start and finish times of n classes. Your goal is to find the largest number of classes you can take without ever taking two classes simultaneously. We showed in class that the following greedy algorithm constructs an optimal schedule:

   Choose the course that *ends first*, discard all conflicting classes, and recurse.

   But this is not the only greedy strategy we could have tried. For each of the following alternative greedy algorithms, either prove that the algorithm always constructs an optimal schedule, or describe a small input example for which the algorithm does not produce an optimal schedule. Assume that all algorithms break ties arbitrarily (that is, in a manner that is completely out of your control).

   [Hint: Exactly three of these greedy strategies actually work.]

   (a) Choose the course x that *ends last*, discard classes that conflict with x, and recurse.

   (b) Choose the course x that *starts first*, discard all classes that conflict with x, and recurse.

   (c) Choose the course x that *starts last*, discard all classes that conflict with x, and recurse.

   (d) Choose the course x with *shortest duration*, discard all classes that conflict with x, and recurse.

   (e) Choose a course x that *conflicts with the fewest other courses*, discard all classes that conflict with x, and recurse.

   (f) If no classes conflict, choose them all. Otherwise, discard the course with *longest duration* and recurse.

   (g) If no classes conflict, choose them all. Otherwise, discard a course that *conflicts with the most other courses* and recurse.

   (h) Let x be the class with the *earliest start time*, and let y be the class with the *second earliest start time*.

   - If x and y are disjoint, choose x and recurse on everything but x.
   - If x completely contains y, discard x and recurse.
   - Otherwise, discard y and recurse.

   (i) If any course x completely contains another course, discard x and recurse. Otherwise, choose the course y that *ends last*, discard all classes that conflict with y, and recurse.

9. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over.

   Describe an algorithm to sort an arbitrary stack of n pancakes using as few flips as possible. *Exactly* how many flips does your algorithm perform in the worst case?

10. You are a visitor at a political convention (or perhaps a faculty meeting) with n delegates; each delegate is a member of exactly one political party. It is impossible to tell which political party

any delegate belongs to; in particular, you will be summarily ejected from the convention if you ask. However, you can determine whether any pair of delegates belong to the *same* party or not simply by introducing them to each other—members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.

(a) Suppose more than half of the delegates belong to the same political party. Describe an efficient algorithm that identifies all members of this majority party.

(b) Now suppose exactly k political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Present a practical procedure to pick out the people from the plurality political party as parsimoniously as possible. (Please.)

11. An array $A[0..n-1]$ of n distinct numbers is ***bitonic*** if there are unique indices i and j such that $A[(i-1) \mod n] < A[i] > A[(i+1) \mod n]$ and $A[(j-1) \mod n] > A[j] < A[(j+1) \mod n]$. In other words, a bitonic sequence either consists of an increasing sequence followed by a decreasing sequence, or can be circularly shifted to become so. For example,

$[4, 6, 9, 8, 7, 5, 1, 2, 3]$ is bitonic, but
$[3, 6, 9, 8, 7, 5, 1, 2, 4]$ is *not* bitonic.

Describe and analyze an algorithm to find the smallest element in an n-element bitonic array in $O(logn)$ time. You may assume that the numbers in the input array are distinct.

12. Suppose you are given an array $A[1..n]$ of numbers, which may be positive, negative, or zero, and which are ***not*** necessarily integers.

(a) Describe and analyze an algorithm that finds the largest sum of of elements in a contiguous subarray $A[i..j]$.

(b) Describe and analyze an algorithm that finds the largest *product* of of elements in a contiguous subarray $A[i..j]$.

For example, given the array $[-6, 12, -7, 0, 14, -7, 5]$ as input, your first algorithm should return the integer 19, and your second algorithm should return the integer 504.

For the sake of analysis, assume that comparing, adding, or multiplying any pair of numbers takes $O(1)$ time.

[Hint: Problem (a) has been a standard computer science interview question since at least the mid-1980s. You can find many correct solutions on the web; the problem even has its own Wikipedia page! But at least in 2013, the few solutions I found on the web for problem (b) were all either slower than necessary or incorrect.]

13. This series of exercises asks you to develop efficient algorithms to find optimal *subsequences* of various kinds. A subsequence is anything obtained from a sequence by extracting a subset of elements, but keeping them in the same order; the elements of the subsequence need not be contiguous in the original sequence. For example, the strings `C`, `DAMN`, `YAIOAI`, and `DYNAMICPROGRAMMING` are all subsequences of the string `DYNAMICPROGRAMMING`.

(a) Let $A[1..m]$ and $B[1..n]$ be two arbitrary arrays. A *common subsequence* of A and B is another sequence that is a subsequence of both A and B. Describe an efficient algorithm to compute the length of the *longest* common subsequence of A and B.

(b) Let $A[1..m]$ and $B[1..n]$ be two arbitrary arrays. A *common supersequence* of A and B is another sequence that contains both A and B as subsequences. Describe an efficient algorithm to compute the length of the *shortest* common supersequence of A and B.

(c) Call a sequence $X[1..n]$ of numbers *bitonic* if there is an index i with $1 < i < n$, such that the prefix $X[1..i]$ is increasing and the suffix $X[i..n]$ is decreasing. Describe an efficient algorithm to compute the length of the longest bitonic subsequence of an arbitrary array A of integers.

(d) Call a sequence $X[1..n]$ of numbers *oscillating* if $X[i] < X[i+1]$ for all even i, and $X[i] > X[i+1]$ for all odd i. Describe an efficient algorithm to compute the length of the longest oscillating subsequence of an arbitrary array A of integers.

(e) Describe an efficient algorithm to compute the length of the shortest oscillating supersequence of an arbitrary array A of integers.

(f) Call a sequence $X[1..n]$ of numbers convex if $2 \cdot X[i] < X[i-1] + X[i+1]$ for all i. Describe an efficient algorithm to compute the length of the longest convex subsequence of an arbitrary array A of integers.

14. (a) Suppose we are given a set L of n line segments in the plane, where each segment has one endpoint on the line $y = 0$ and one endpoint on the line $y = 1$, and all $2n$ endpoints are distinct. Describe and analyze an algorithm to compute the largest subset of L in which no pair of segments intersects.

    (b) Suppose we are given a set L of n line segments in the plane, where each segment has one endpoint on the line $y = 0$ and one endpoint on the line $y = 1$, and all $2n$ endpoints are distinct. Describe and analyze an algorithm to compute the largest subset of L in which ***every*** pair of segments intersects.

15. Suppose you are given an $m \times n$ bitmap, represented by an array $M[1..n, 1..n]$ of 0s and 1s. A *solid block* in M is a subarray of the form $M[i..i', j..j']$ containing only 1-bits. A solid block is square if it has the same number of rows and columns.

    (a) Describe an algorithm to find the maximum area of a solid *square* block in M in $O(n^2)$ time.

    (b) Describe an algorithm to find the maximum area of a solid block in M in $O(n^3)$ time.

16. Let X be a set of n intervals on the real line. We say that a set P of points stabs X if every interval in X contains at least one point in P. Describe and analyze an efficient algorithm to compute the smallest set of points that stabs X. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X.

Graph algorithms.

1. *⟨⟨S14⟩⟩* You just discovered your best friend from elementary school on Twitbook. You both want to meet as soon as possible, but you live in two different cites that are far apart. To minimize travel time, you agree to meet at an intermediate city, and then you simultaneously hop in your cars and start driving toward each other. But where *exactly* should you meet?

   You are given a weighted graph $G = (V, E)$, where the vertices V represent cities and the edges E represent roads that directly connect cities. Each edge e has a weight $w(e)$ equal to the time required to travel between the two cities. You are also given a vertex p, representing your starting location, and a vertex q, representing your friend's starting location.

   Describe and analyze an algorithm to find the target vertex t that allows you and your friend to meet as quickly as possible.

2. *⟨⟨Lab⟩⟩* ***Snakes and Ladders*** is a classic board game, originating in India no later than the 16th century. The board consists of an $n \times n$ grid of squares, numbered consecutively from 1 to n^2, starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares, always in different rows, are connected by either "snakes" (leading down) or "ladders" (leading up). Each square can be an endpoint of at most one snake or ladder.

   ⟨⟨Insert Snakes and Ladders figure here.⟩⟩ ⟸✠

   A typical Snakes and Ladders board.
   Upward straight arrows are ladders; downward wavy arrows are snakes.

   You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to k positions, for some fixed constant k (typically 6). If the token ends the move at the *top* end of a snake, you ***must*** slide the token down to the bottom of that snake. If the token ends the move at the *bottom* end of a ladder, you ***may*** move the token up to the top of that ladder.

   Describe and analyze an algorithm to compute the smallest number of moves required for the token to reach the last square of the grid.

3. *⟨⟨Lab⟩⟩* Let G be a connected undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of G. At every step, each coin *must* move to an adjacent vertex. Describe and analyze an algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph $G = (V, E)$ and two vertices $u, v \in V$ (which may or may not be distinct).

4. *⟨⟨Lab⟩⟩* Inspired by the previous lab, you decided to organize a Snakes and Ladders competition with n participants. In this competition, each game of Snakes and Ladders involves three players. After the game is finished, they are ranked first, second and third. Each player may be involved in any (non-negative) number of games, and the number needs not be equal among players.

   At the end of the competition, m games have been played. You realized that you had forgotten to implement a proper rating system, and therefore decided to produce the overall ranking of all n

players as you see fit. However, to avoid being too suspicious, if player A ranked better than player B in any game, then A must rank better than B in the overall ranking.

You are given the list of players involved and the ranking in each of the m games. Describe and analyze an algorithm to produce an overall ranking of the n players that satisfies the condition, or correctly reports that it is impossible.

5. ⟨⟨*Lab*⟩⟩ There are n galaxies connected by m intergalactic teleport-ways. Each teleport-way joins two galaxies and can be traversed in both directions. Also, each teleport-way e has an associated toll of c_e dollars, where c_e is a positive integer. A teleport-way can be used multiple times, but the toll must be paid every time it is used.

   Judy wants to travel from galaxy u to galaxy v, but teleportation is not very pleasant and she would like to minimize the number of times she needs to teleport. However, she wants the total cost to be a multiple of five dollars, because carrying small bills is not pleasant either.

   (a) Describe and analyze an algorithm to compute the smallest number of times Judy needs to teleport to travel from galaxy u to galaxy v while the total cost is a multiple of five dollars.

   (b) Solve (a), but now assume that Judy has a coupon that allows her to waive the toll once.

6. ⟨⟨*Lab*⟩⟩ Suppose we are given both an undirected graph G with weighted edges and a minimum spanning tree T of G.

   (a) Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge $e \in T$ is decreased.

   (b) Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge $e \notin T$ is increased.

   (c) Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge $e \in T$ is increased.

   (d) Describe an efficient algorithm to update the minimum spanning tree when the weight of one edge $e \notin T$ is decreased.

   In all cases, the input to your algorithm is the edge e and its new weight; your algorithms should modify T so that it is still a minimum spanning tree. Of course, we could just recompute the minimum spanning tree from scratch in $O(E \log V)$ time, but you can do better.

7. ⟨⟨*Lab*⟩⟩ A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has non-negative weight.

   ⟨⟨Insert looped-tree figure here.⟩⟩

   A looped tree.

   (a) How much time would Dijkstra's algorithm require to compute the shortest path between two vertices u and v in a looped tree with n nodes?

   (b) Describe and analyze a faster algorithm.

8. ⟨⟨*Lab*⟩⟩ After graduating you accept a job with Aerophobes-Я-Us, the leading traveling agency for people who hate to fly. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying (and by extension, airports), so any trip you plan needs to be as short as possible. You know all the departure and arrival times of all the flights on the planet.

   Suppose one of your customers wants to fly from city X to city Y. Describe an algorithm to find a sequence of flights that minimizes the *total time in transit*—the length of time from the initial departure to the final arrival, including time at intermediate airports waiting for connecting flights. *[Hint: Build an appropriate graph from the input data and apply Dijkstra's algorithm.]*

9. Prove that any connected acyclic graph with $n \geq 2$ vertices has at least two vertices with degree 1. Do not use the words "tree" or "leaf", or any well-known properties of trees; your proof should follow entirely from the definitions of "connected" and "acyclic".

10. A graph (V, E) is bipartite if the vertices V can be partitioned into two subsets L and R, such that every edge has one vertex in L and the other in R.

    (a) Prove that every tree is a bipartite graph.

    (b) Describe and analyze an efficient algorithm that determines whether a given undirected graph is bipartite.

11. Let G be a directed acyclic graph with a unique source s and a unique sink t.

    (a) A Hamiltonian path in G is a directed path in G that contains every vertex in G. Describe an algorithm to determine whether G has a Hamiltonian path.

    (b) Suppose the vertices of G have weights. Describe an efficient algorithm to find the path from s to t with maximum total weight.

    (c) Suppose we are also given an integer ℓ. Describe an efficient algorithm to find the maximum-weight path from s to t, such that the path contains at most ℓ edges. (Assume there is at least one such path.)

    (d) Suppose the vertices of G have integer labels, where $label(s) = -\infty$ and $label(t) = \infty$. Describe an algorithm to find the path from s to t with the maximum number of edges, such that the vertex labels define an increasing sequence.

    (e) Describe an algorithm to compute the number of distinct paths from s to t in G. (Assume that you can add arbitrarily large integers in $O(1)$ time.)

12. Most classical minimum-spanning-tree algorithms use the notions of "safe" and "useless" edges described in the lecture notes, but there is an alternate formulation. Let G be a weighted undirected graph, where the edge weights are distinct. We say that an edge e is ***dangerous*** if it is the longest edge in some cycle in G, and ***useful*** if it does not lie in any cycle in G.

    (a) Prove that the minimum spanning tree of G contains every useful edge.

    (b) Prove that the minimum spanning tree of G does not contain any dangerous edge.

(c) Describe and analyze an efficient implementation of the "anti-Kruskal" minimum spanning tree algorithm: Examine the edges of G in decreasing order; if an edge is dangerous, remove it from G. *[Hint: It won't be as fast as Kruskal's algorithm.]*

13. For any edge e in any graph G, let $G \setminus e$ denote the graph obtained by deleting e from G. Suppose we are given a directed graph G in which the shortest path σ from vertex s to vertex t passes through every vertex of G. Describe an algorithm to compute the shortest-path distance from s to t in $G \setminus e$, for every edge e of G, in $O(ElogV)$ time. Your algorithm should output a set of E shortest-path distances, one for each edge of the input graph. You may assume that all edge weights are non-negative. *[Hint: If we delete an edge of the original shortest path, how do the old and new shortest paths overlap?]*

14. When there is more than one shortest path from one node s to another node t, it is often convenient to choose a shortest path with the fewest edges; call this the best path from s to t. Suppose we are given a directed graph G with positive edge weights and a source vertex s in G. Describe and analyze an algorithm to compute best paths in G from s to every other vertex.