MIDTERM PRACTICE PROBLEMS CMSC 451 (Section 0101, Fall 2020)

The midterm exam will be on Tuesday, October 13, during our regular class time (2:00–3:15 pm eastern time). The exam will be available on Canvas at 1:30 pm (giving you half an hour to get set up before you begin working at 2 pm) and will be due on Gradescope by 3:45 pm (giving you half an hour to scan and submit the exam after you stop working by 3:15 pm). You may consult textbooks and notes during the exams, but you cannot use any other resources (in particular, internet access is not allowed). You will take the exams at home, working alone. You will be asked to sign an honor pledge indicating that you have followed the exam guidelines.

The following problems may help you prepare for the midterm. They are not to be handed in. Solutions are not available, but of course you are welcome to discuss these problems during office hours or on Piazza.

Disclaimer: Many of these problems are taken from exams in previous offerings of CMSC 451 when the emphasis of the course may have been different. These problems do not necessarily reflect the difficulty level or distribution of material to be covered on the exam.

1. *Short answer questions.*

   For the following questions, explanations are not required, but can be given for partial credit.

   (a) Recall that in the Gale-Shapley algorithm as discussed in class, companies make offers to students. True or false: once a *company* makes an offer that is accepted, that company is guaranteed to remain engaged (to some student) for the remainder of the algorithm.

   (b) Recall that in the Gale-Shapley algorithm as discussed in class, companies make offers to students. Which of the following describes how the sequence of students engaged to a given company changes over time? The company's preference for the student to whom it is engaged...

      i. ... strictly increases in each iteration.
      ii. ... may either increase or stay the same from one iteration to the next.
      iii. ... may either decrease or stay the same from one iteration to the next.
      iv. ... strictly decreases in each iteration.

   (c) In the stable matching problem, suppose that students are allowed to leave companies out of their ranking and that they can only be assigned to companies they have ranked. Show that even if each student is allowed to leave out at most one company, then a stable perfect matching might not exist. (Hint: You can give a counterexample with a very small number of students and companies.)

   (d) Consider the following greedy algorithm for the stable matching problem (as formulated in class): go through all companies in an arbitrary order and assign each to its highest-preference available student. Give a minimal counterexample showing that this algorithm does not necessarily produce a stable matching. Explain why the matching in your example is unstable.

   (e) What is the asymptotic running time of the following program as a function of n? (Express your running time using Θ notation.)

```
f(n):
  set i = n
```

```
print "the output goes on"
while i > 0
  for j = 1 to i
    print " and on"
  endfor
  set i = i/2
endwhile
```

(f) Consider the following algorithm that takes as input the adjacency list representation of an n-vertex undirected graph G.

```
ExploreTriangles(G):
  for every vertex v of G
    for every neighbor u of v
      for every neighbor w of u
        if w is a neighbor v then print "found a triangle!"
      endfor
    endfor
  endfor
```

i. Suppose every vertex of G has degree at most 100. What is the running time of this algorithm as a function of n? Express your answer using Θ notation.

ii. Suppose every vertex of G has degree $\Theta(\sqrt{n})$. What is the running time of this algorithm as a function of n? Express your answer using Θ notation.

(g) For each of the following functions, indicate whether $f(n)$ is in o, ω, or Θ of $g(n)$. In this problem, log denotes the base-2 logarithm.

i. $f(n) = 10^{n/2}$, $g(n) = 5^n$

ii. $f(n) = n^2 + 10^{100}\, n \log n$, $g(n) = 10^{100}\, n^2 + n \log n$

iii. $f(n) = \log \sqrt{n}$, $g(n) = \sqrt{\log n}$

(h) List the following functions in increasing asymptotic order. If two functions are asymptotically equivalent, then indicate this.

(i) $n^{3/2} + n^{2/3}$ (ii) $n(\log n)^2$ (iii) $4^{\log n}$ (iv) $\max\{2000n^2, n^3\}$

(In this problem, log represents the base-2 logarithm.)

(i) For each of the following functions, indicate whether $f(n)$ is in o, ω, or Θ of $g(n)$. In this problem, log denotes the base-2 logarithm.

i. $f(n) = n^3 - 5n$, $g(n) = 3n^2$

ii. $f(n) = 2^{\sqrt{n}}$, $g(n) = \sqrt{2^n}$

iii. $f(n) = n^{(\log n)^2}$, $g(n) = 2^n$

(j) For each of the following functions, indicate whether $f(n)$ is in o, ω, or Θ of $g(n)$. In this problem, log denotes the base-2 logarithm.

i. $f(n) = 27n^3$, $g(n) = \sum_{i=1}^{n} \sum_{j=1}^{i} (3j + 5)$

ii. $f(n) = n \log \log n$, $g(n) = (\log n)^2$

iii. $f(n) = 2^n$, $g(n) = 5^{n/2}$

(k) List the following functions of n in increasing asymptotic order. If two functions are asymptotically equivalent, then indicate this.

(i) $n \log_2(n^2)$ (ii) $1.01^{\log_2 n}$ (iii) $n^{1.01}\sqrt{\log_2 n}$ (iv) $\min\{n^2, 101\, n \log_2 n\}$.

(l) What is the maximum number of edges in an undirected graph with n vertices, in which each vertex has degree at most k?

(m) How many edges are there in an n-vertex forest with c components?

(n) Give an example of a graph for which the breadth-first search algorithm (as presented in class) can output different BFS trees starting from some particular initial vertex, depending on the order in which the neighbors of vertices are considered. Your example should involve as few vertices as possible.

(o) True or false: Every directed acyclic graph has a unique topological ordering. If true, explain why; if false, give a minimal counterexample.

(p) Consider the following algorithm that takes as input a directed graph G and outputs a list of vertices of G:

```
TS(G):
    if G is empty then
        return an empty list
    else
        let v be a vertex of G with outdegree 0
        return a list with v appended to the end of TS(G \ v)
    endif
```

Here $G \setminus v$ denotes the graph G with vertex v and all associated edges removed.

True or false: If G is a DAG, then TS(G) outputs a topological ordering of G.

(q) A *maximum* spanning tree of a weighted, connected graph is a spanning tree with the largest possible total weight of its edges. True or False: If we modify Prim's algorithm to select the non-tree vertex with the *highest* attachment cost, it will produce a maximum spanning tree.

(r) Recall that in the *interval scheduling problem*, we are given a list of start times s_i and finish times f_i for all $i \in \{1, 2, \ldots, n\}$, and our goal is to find a largest possible subset of the intervals $[s_i, f_i]$ such that no two intervals overlap. Consider the following greedy algorithm for interval scheduling:

```
Sort the intervals by starting times
While there are any remaining intervals
    Add an interval with the latest starting time to the schedule
    Delete all conflicting intervals
Endwhile
```

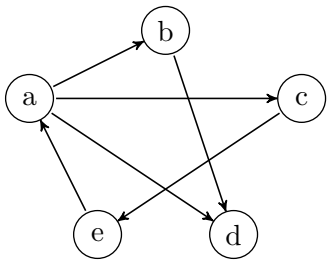
True or false: This algorithm outputs an optimal schedule.

2. *Degree sums in undirected graphs.*

Suppose $G = (V, E)$ is an undirected graph given in *adjacency list* format. Recall that this format provides a linked list of the neighbors of each vertex (you should assume that the format does not directly provide the degree of a vertex). For any vertex v, let sumdegree[v] denote the sum of the degrees of the neighbors of v: e.g., if v has neighbors w_1, w_2, and w_3 with respective degrees $3, 6, 2$, then sumdegree[v] $= 3 + 6 + 2 = 11$. Show how to construct the array of all sumdegree[v] values (one value for each $v \in V$), in $O(n + m)$ time. (As usual, we let $n = |V|$ and $m = |E|$.) Explain why your algorithm has the desired running time and why it is correct.

3. *DFS.*

   Show the result of running DFS on the digraph shown below, starting from vertex a, using the algorithm described in class. Whenever you have a choice of which vertex to visit next, take the lowest vertex in alphabetical order. Indicate tree edges with solid lines and non-tree edges with dashed lines.



4. *Strong connectivity.*

   Let $n \geq 2$.

   (a) Prove that any *strongly-connected* n-vertex directed graph G has at least n edges. (Hint: Start your proof by making a useful observation about the out-degree of each vertex.)

   (b) For each $n \geq 2$, give an example of a strongly-connected directed graph G with n vertices and exactly n edges.

5. *Computing girth.*

   (a) Suppose you are given as input an undirected graph $G = (V, E)$ in adjacency list format. Describe an efficient algorithm that, for a given edge $\{u, v\} \in E$, computes the length of the shortest cycle in G that includes this edge. Prove that your algorithm is correct and analyze its running time.

   (b) The *girth* of a graph G is the length of a shortest cycle in G. Give an efficient algorithm that computes the girth of a graph given in adjacency list format. Prove that your algorithm is correct and analyze its running time. (Hint: use the algorithm from the previous part as a subroutine.)

6. *Shortest path and minimum spanning trees under a uniform shift.*

   Let G be a weighted undirected graph in which all edge weights are distinct. Construct a new graph G' by adding a fixed positive constant k to the weight of every edge in G.

   (a) Prove or disprove: The shortest path tree in G rooted at a vertex s of G (i.e., the tree produced when Dijkstra's algorithm is run on G starting from s) is the same as the shortest path tree in G' rooted at s.

   (b) Prove or disprove: The minimum spanning tree of G is the same as the minimum spanning tree of G'.

7. *Maximum-cost path in a weighted DAG.*

   You are given a directed acyclic graph $G = (V, E)$. Each edge $(u, v) \in E$ of this graph is associated with a weight $w(u, v)$, which can be any positive real number. The cost of a path is defined to be the sum of the weights of the edges along the path. Present an efficient algorithm

that computes the maximum cost of any path in G. Justify your algorithm's correctness and establish its running time.

8. *Tail-maximal paths.*

   Given a DAG $G = (V, E)$, a path of G is said to be tail-maximal if it ends at a vertex with outdegree zero. (If u is a vertex of outdegree zero then the path consisting of just u itself is a tail-maximal path.) Describe an algorithm with running time $O(n + m)$ that, given a DAG $G = (V, E)$, computes for each vertex u the number of distinct tail-maximal paths that originate at u. (Your algorithm can compute an array P such that $P[u]$ contains the desired number of paths for vertex u.)

9. *Minimum spanning tree of a near-tree.*

   An n-vertex graph is a *near-tree* if it is connected and has n edges. Give an algorithm with running time $O(n)$ for finding a minimum spanning tree of a weighted near-tree. Prove that your algorithm is correct and has the desired running time.

10. *Borůvka's algorithm.*

    In 1926, Otakar Borůvka published a method for constructing an efficient electrical network for the country of Moravia. This minimum spanning tree algorithm predates both the Kruskal and Prim algorithms. Given a connected, undirected input graph G with positive edge weights w such that no two edges have the same weight, it works as follows:

    ```
    Borůvka(G, w):
        Let F be a forest with the same vertices as G, but no edges
        While F has more than one component
            Let S be an empty set
            For each component C of F
                Let {u, v} be the lowest-weight edge with u in C and v not in C
                Add edge {u, v} to S
            Endfor
            Add the edges in S to the forest F
        Endwhile
    ```

    In both parts of this problem, you should assume that G is a connected, undirected graph and that no two edges of G have the same weight.

    (a) Prove that Borůvka's algorithm outputs a minimum spanning tree of G. (You are *not* asked to analyze its running time.)

    (b) True or false: Borůvka's algorithm and Prim's algorithm always output the same minimum spanning tree.

11. *Making change.*

    Describe a greedy algorithm for making change using quarters, dimes, nickels, and pennies. Assume that the input is given as the number of cents. Prove that your algorithm yields the minimum number of coins. (Hint: For partial credit, prove this for the simpler binary sequence of denominations: 1, 2, 4, 8, 16.)

12. *Dominating sets.*

Given a graph $G = (V, E)$, a vertex subset $V' \subseteq V$ is called a *dominating set* if every vertex of G is either in the set V' or is a neighbor of a vertex in V'. In the *dominating set problem* you are given a graph G and the objective is to compute a dominating set of smallest size.

(a) Describe a greedy algorithm for computing a small dominating set. Your algorithm should run in time that is polynomial in $n = |V|$.

(b) Present an example to show that your greedy algorithm is not optimal.

13. *Road trip.*

Suppose you are going on a road trip with a car that can travel for 100 miles on a full tank. There are gas stations at distances $0, x_1, x_2, \ldots, x_n$ along the way. Present an algorithm that determines the fewest number of gas stations you must stop at to make it to the station at distance x_n without running out of gas along the way, or outputs "not possible" if it is impossible to reach the last station without running out of gas. Prove that your algorithm is correct.

14. *Bottle filling.*

A pharmacist has m pills and n empty bottles. Let b_i denote the capacity of bottle i, that is, the number of pills it can hold. Let v_i denote the cost of purchasing bottle i. The objective is find the least expensive combination of bottles that will hold all m pills.

Describe a greedy algorithm that, given the number of pills m, the bottle capacities b_i, and the bottle costs v_i, determines the least expensive set of bottles needed to store all the pills. Assume that you pay only for the fraction of the bottle that is used. For example, if the ith bottle is half filled with pills, you pay only $v_i/2$. (This assumption is very important.) Prove the correctness of your algorithm.

15. *Scheduling to minimize lateness revisited.*

Recall that in the problem of scheduling to minimize lateness, we are given n jobs with durations t_i and deadlines d_i, and our goal is to schedule the jobs (i.e., choose their nonnegative starting times) such that no two overlap, minimizing the lateness (the longest time after its deadline that it takes for some job to finish). Let

$$\tau_j = \sum_{i:\, d_i \le d_j} t_i$$

denote the sum of the durations of all jobs with deadlines no later than the deadline of job j.

(a) Prove that the lateness of any feasible schedule (i.e., any schedule in which no two jobs overlap) is at least $\tau_j - d_j$ (for any $j \in \{1, \ldots, n\}$).

(b) Recall that the greedy algorithm we discussed in class simply schedules the jobs in order of increasing deadline. Use the previous part to prove that this is optimal. (You cannot appeal to the proof from class; you have to use the above fact to give a new proof.)

16. *Greedy interval scheduling.*

Recall the following problem, called the Interval Scheduling Problem. We are given a set of n activity requests, each of which has a specified start time s_i and finish time f_i, corresponding to the interval $[s_i, f_i]$. The objective is to compute the maximum number of activities whose corresponding intervals do not overlap.

(a) In the Earliest Activity First (EAF) greedy algorithm, we schedule the activity with the earliest start time, remove all activities that overlap it, and repeat until no more activities remain. Give an example to show that EAF is not optimal.

(b) In the Shortest Activity First (SAF) greedy algorithm, we schedule the activity with the smallest duration ($f_i - s_i$), remove all activities that overlap it, and repeat until no more activities remain. Give an example to show that SAF is not optimal.

17. *Scheduling to minimize total waiting time.*

Suppose n students are waiting for individual meetings during an instructor's office hours, and we are told in advance that student i will need to spend $t[i]$ minutes having her questions answered. Design an efficient algorithm that finds an order for the students to see the instructor such that the total time all the students spend waiting is minimized. (Hint: Try a greedy strategy.) Analyze the running time of your algorithm and prove that it is correct.

18. *Bomb scheduling.*

You are given a set of m time intervals $[s_i, f_i]$, where $1 \leq i \leq m$. You are also given a set of n possible bomb times $T = t_1, ..., t_n$. We say that bomb j *hits* interval i if this bomb time lies within the interval, that is, $t_j \in [s_i, f_i]$. Your objective is to determine the minimum number of bombs from T to hit every one of the intervals. You may assume that every interval is hit by at least one bomb, so a solution exists.

Present a polynomial-time algorithm to determine a minimum set of bombs to hit all the intervals. (Hint: Try a greedy strategy.) Justify your algorithm's correctness.

19. *Greedy point covering.*

You are given real numbers $x_1, x_2, \ldots, x_n$ representing positions on the real line. We say that a unit interval $[t, t+1]$ *covers* x if $t \leq x \leq t+1$. Your goal is to cover all n given points using as few unit intervals as possible. In other words, you would like to find $t_1, t_2, \ldots, t_k$, with k as small as possible, so that for all $i \in \{1, 2, \ldots, n\}$ there exists a $j \in \{1, 2, \ldots, k\}$ such that $[t_j, t_j + 1]$ covers x_i.

(a) Describe a greedy algorithm for this problem.

(b) Prove that your algorithm is correct. (Hint: You might give an exchange argument.)

(c) Analyze the running time of your algorithm.

20. *Maximum ordered ratio.*

Suppose you are given as input a sequence of numbers $a_1, a_2, \ldots, a_n$ with $n \geq 2$. Your goal is to find the largest ratio between two of these numbers where the numerator occurs after the denominator in the sequence. In other words, you would like to compute

$$\max \left\{ \frac{a_i}{a_j} : i, j \in \{1, 2, \ldots, n\} \text{ with } i > j \right\}.$$

Describe a divide-and-conquer algorithm to solve this problem. Prove its correctness and analyze its running time. For full credit, your algorithm should run in time at most $O(n \log n)$.

21. *Maximum partial sum.*

Suppose you are given as input a sequence of integers $a_1, a_2, \ldots, a_n$. Your goal is to find the largest partial sum of these numbers, of the form

$$\sum_{i=s}^{t} a_i$$

for some $s, t \in \{1, 2, \ldots, n\}$ with $s \leq t$. Describe a divide-and-conquer algorithm to solve this problem. Prove its correctness and analyze its running time. For full credit, your algorithm should run in time at most $O(n \log n)$.

22. *Counting mixed-parity inversions.*

Given a list $A = (a_1, \ldots, a_n)$ of integers, a *mixed-parity inversion* is a pair a_i and a_j such that $i < j$, $a_i > a_j$, and a_i and a_j are of different parities. (In other words, it is an inversion in which one number is even and the other is odd.)

Design an $O(n \log n)$-time algorithm that counts the number of mixed-parity inversions in a list A containing n elements. Justify your algorithm's correctness and derive its running time.

23. *Counting significant inversions.*

Given a list of numbers $(a_1, a_2, \ldots, a_n)$, a *significant inversion* is a pair of indices $i < j$ such that $a_i > 2a_j$. Design an algorithm to count the number of significant inversions in a list of length n. Prove that your algorithm is correct and analyze its running time. For full credit, your algorithm should run in time $O(n \log n)$. (Hint: Try a variant of MergeSort.)

24. *Buy low, sell high.*

You are given as input a list $p_1, \ldots, p_n$ of daily prices of a certain stock for n consecutive days. Your goal is to determine the highest possible profit you can make by buy the stock on some day and selling it on some later day. Give an algorithm that solves this problem as efficiently as possible. Prove that your algorithm is correct and analyze its running time. (Hint: Use divide and conquer.)