CMPSCI 311: Introduction to Algorithms
First Midterm Exam

February 28, 2017

Name: ________________________________ ID: ____________________

Instructions:

- Answer the questions directly on the exam pages.
- Show all your work for each question. Providing more detail including comments and explanations can help with assignment of partial credit.
- If you need extra space, use the back of a page.
- No books, notes, calculators or other electronic devices are allowed. Any cheating will result in a grade of 0.
- If you have questions during the exam, raise your hand.

Question	Value	Points Earned
1	10	
2	10	
3	10	
4	10	
5	10	
Total	50	

Question 1. (*10 points*) Indicate whether each of the following statements is TRUE or FALSE. No justification required.

1.1 (*2 points*): $\sum_{i=1}^{n} i = \Theta(n^2)$.

Solution: True. $\sum_{i=1}^{n} i \leq \sum_{i=1}^{n} n = n^2$ And $\sum_{i=1}^{n} i \geq \sum_{i=n/2}^{n} i \geq \sum_{i=n/2}^{n} n/2 = n^2/4$.

1.2 (*2 points*): *If T is a BFS tree for a graph G and (u, v) is an edge in G that is not in T, then u and v are the same distance from the root of T.*

Solution: False. The level of u and v can disagree by at most 1. We proved this in class, but a square graph on 4 nodes has this property. Consider the graph $(a, b), (b, d), (a, c), (c, d)$.

1.3 (*2 points*): *Given n colleges and n applicants where each college has a rank-ordering of all the applicants (no ties) and every applicant has a rank-ordering of all the colleges (no ties), there can be at most one stable matching.*

Solution: False. Suppose A both rank $1, 2$ and B ranks $2, 1$, while 1 ranks B, A and 2 ranks A, B.

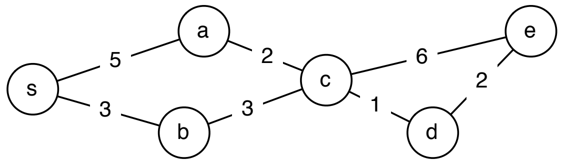
1.4 (*2 points*): *Every directed acyclic graph has a node with no incoming edges.*

Solution: True. Otherwise it must have a cycle, since we could follow reverse edges for n steps.

1.5 (*2 points*): *For every n there exists a directed graph on n vertices with $\Omega(n^2)$ edges that has a topological ordering.*

Solution: True. Consider ordering the vertices $1, \ldots, n$ with edges (i, j) for all $i < j$. This graph clearly has a topological ordering and has $\sum_{i=1}^{n-1} i = \Theta(n^2)$ edges.

Question 2. (*10 points*) In the first two parts of this question we consider the following weighted graph G:



2.1 (*4 points*): *For the above graph:*

What is the length of the shortest path from s to e? **Solution:** 9

How many different spanning trees are there? **Solution:** 12

How many different minimum spanning trees are there? **Solution:** 1

What's the weight of the minimum spanning tree? **Solution:** 11

2.2 (*2 points*): *Recall that Dijkstra's algorithm starts with a set* $S = \{s\}$ *of* explored nodes *and defines* $d[s] = 0$*. In each iteration, the algorithm adds some node* v *to* S *and defines* $d[v]$*.*

What node is added in the first iteration? **Solution:** b

What node is added in the second iteration? **Solution:** a

2.3 (*2 points*): *Your friend claims that the order in which the nodes of an arbitrary graph are added to the set of explored nodes when running Dijkstra's algorithm is unchanged if we increment every edge weight by one. Is this always true? Justify your answer.*

Solution: No it's false. Consider a graph with 4 nodes s, a, b, c with edges $(s, a), (s, b), (b, c)$. Suppose $w((s, a)) = 2.1$ and all other edges have weight one. In the original graph if we run Djikstra's algorithm starting at s, we add nodes in the following order s, b, c, a (The distances are $d[s] = 0, d[b] = 1, d[c] = 2, d[a] = 2.1$.). If we increment each edge by 1, then the order becomes s, b, a, c which is different (The distances are $d[s] = 0, d[b] = 2, d[a] = 3.1, d[c] = 4$).

2.4 (*2 points*): *Assume edge weights are integers. Another friend claims she can find the shortest path between s and every node by building a BFS tree from s if we first replace every edge with an unweighted path of length equal to the weight of the edge. Is this always true? Justify your answer.*

Solution: Yes this is true. In the new graph the depth/level of the node in the BFS is the shortest path distance in the original graph.

Question 3. (*10 points*) In this question, we consider analyzing the performance of a specific company in the stock market. Suppose we tracked the stock over a sequence of n days and let $A[t]$ be the value of the stock at the end of the tth day. For each of the following problems, give a brief description of a *simple* algorithm and state the running time of your algorithm. No proof required.

3.1 (*2 points*): *Compute the maximum value of the stock, i.e.,* $\max_{1 \leq t \leq n} A[t]$.

Running Time: **Solution:** $O(n)$.

Brief Description: **Solution:** Scan the list in order keeping track of the maximum

3.2 (*2 points*): *Compute the average change from the previous day, i.e.,* $\frac{1}{n} \sum_{t=1}^{n-1} (A[t+1] - A[t])$.

Running Time: **Solution:** $O(1)$.

Brief Description: **Solution:** Observe that the sum telescopes to $\frac{1}{n}(A[n] - A[1])$, so we only have to look at two elements.

3.3 (*3 points*): *Determine whether there are at least two (not necessarily consecutive) days that have the same value, i.e., does there exist* $i \neq j$ *such that* $A[i] = A[j]$.

Running Time: **Solution:** $O(n \log_2 n)$.

Brief Description: **Solution:** Sort the list and then see if two adjacent elements are the same in one pass over the list.

3.4 (*3 points*): *Suppose the entire sequence is initially increasing and then decreasing, i.e., for some unknown value* r,

$$A[1] < A[2] < \ldots < A[r] > A[r+1] > A[r+2] > \ldots > A[n] \ .$$

Compute the maximum value of the stock, i.e., $\max_{1 \leq t \leq n} A[t]$.

Running Time: **Solution:** $O(\log_2 n)$.

Brief Description: **Solution:** Look at the middle two elements of the list to see if the list is increasing or decreasing. If it's increasing recurse on the later half, otherwise recurse on the early half.

Question 4. (*10 points*) Alice is planning her course schedule for her time at UMass. There are n courses she must take and each course c_i can have pre-requisites P_i, which is a possibly empty set of courses. However, the department allows students to take a course and its prerequisites in the same semester. In other words, a course c_i can be taken in semester t if for all $c_j \in P_i$ the semester in which Alice takes c_j is at most t.

On the other hand, Alice can take at most 3 courses in a semester.

1. Prove that if the pre-requisite graph has a cycle of length 4, then there is no way for Alice to find a schedule satisfying all the pre-requisites.

   Solution: If there is a cycle in the prerequisite graph, then all courses in the cycle must be taken in the same semester. Suppose that S is such a cycle. If there aren't then there must be some subset of courses $S' \subset S$ that is taken in an earlier semester than the remaining courses $S \setminus S'$. But since S is a cycle, at least one course in S' has a prerequisite in $S \setminus S'$, which is a contradiction.

   Thus a cycle of length 4 cannot be scheduled.

2. Prove that if every course is involved in at most one cycle of length at most 3, then a valid schedule must exist.

   Solution: For any course c, consider the courses C involved in a cycle containing c. We will assign these all to the same semester, but we must assign all other pre-requisites in an earlier semester. This is always possible, since while pre-requisites can be in cycles of their own, they cannot be in cycles containing C. In particular, it cannot be the case that there is a cycle between some pre-requisite p, a course $c' \in C$, and a further course r that has some $c'' \in C$ as a pre-requisite. This implies that c' (and c'') are in multiple cycles.

   Thus there are no reverse edges between the pre-requisites of C and the courses that require C (e.g., edges from a course that depends on C to a course that C is dependent on), which means that we can schedule all of the pre-requisites before C, and then proceed to schedule everything else recursively.

Question 5. (*10 points*) Suppose you have n assignments due in the next 24 hours. You are going to start immediately and not stop until you have finished them all. It is up to you in which *order* you do the assignments. Suppose the ith assignment takes t_i time and has a deadline at d_i.

5.1 (*2 points*): *As an example, suppose you have three assignments and* $t_1 = 5, t_2 = 4,$ *and* $t_3 = 4$. *Is it possible to complete the assignments before their deadlines if their deadlines are:*

$d_1 = 8, d_2 = 5,$ *and* $d_3 = 13$?	**Yes**	<u>**No**</u>
$d_1 = 9, d_2 = 4,$ *and* $d_3 = 13$?	<u>**Yes**</u>	**No**

5.2 (*2 points*): *Suppose there is at least one ordering in which all assignments are completed before their deadlines. Prove that if you complete the assignments in order of increasing deadline then all assignments will be completed before their deadline.*

Solution: Use an exchange argument. Number the jobs in the valid ordering $1, \ldots, n$ (e.g. job 1 is done first, then 2, etc.). If that ordering is in order of deadline, then we're done. Otherwise there is a pair of assignments $i < j$ such that $d_i > d_j$. In fact there must be an adjacent pair of these, since if i is not adjacent to j, then either $d_{i+1} > d_i$ in which case we can use $i+1, j$ for the pair or $d_{i+1} < d_i$ in which case we can use $i, i+1$ for the pair. So if an adjacent pair $i, i+1$ is such that $d_i > d_{i+1}$, then by swapping these two we order them by shortest deadline, but we don't miss any deadlines. We don't miss deadlines since now job $i+1$ is done even earlier (t_i hours earlier) while job i is done before job $i+1$ was in the other schedule. Since in the previous schedule job $i+1$ didn't miss its deadline and $d_{i+1} < d_i$, job i doesn't miss its deadline here. In this way we can flip every pair without missing deadlines to get the schedule that is ordered by deadline.

5.3 (*2 points*): *Suppose the* i*th assignment takes* $2i$ *minutes to complete and has a deadline after* $i(i+1)$ *minutes. Can all the assignments be completed before their deadlines? Justify your answer.*

Solution: Yes. Proof is by induction. The base case is that $2 \times 1 \leq 1(1+1) = 2$. Now,

$$\sum_{j=1}^{i} 2j = \sum_{j=1}^{i-1} 2j + 2i \leq (i-1)(i) + 2i = i^2 - i + 2i = i^2 + i = i(i+1).$$

5.4 (*4 points*): *Suppose there exists an assignment, such that if you skip this assignment (i.e., spend no time on it), you can complete all the other assignments before their deadlines. Consider the following part of an outline of algorithm for finding such an assignment:*

1. Sort assignments such that the ith assignment has the ith smallest deadline.
2. For all i, compute $f(i) = t_1 + \ldots + t_i$, $\ell(i) = f(i) - d_i$, and $m(i) = \max(\ell(i), \ell(i+1), \ldots, \ell(n))$.
3. ???

How fast can step 1 be performed?

Solution: $O(n \log_2 n)$. It requires sorting the list.

How can step 2 be performed in $O(n)$ time?

Solution: $O(n)$. Traverse the list from front to back maintaining the sum $\sum_{j=1}^{i} t_j$ and computing $f(i)$. Simultaneously compute $\ell(i)$. Then walk the list from back to front maintaining $\max\{\ell(j), \ldots, \ell(n)\}$ for $m(i)$.

Suggest a step 3 that returns the answer and can be performed in $O(n)$ time. Justify your answer.

Solution: Find the first index i such that $m(i+1) - t_i \leq 0$, and remove this job.

The problem guarantees the existence of some job i such that removing i allows all jobs to be completed by their deadline, i.e., the lateness of every job is zero. When i is removed, the schedule is $1, 2, \ldots, i-1, i+1, \ldots, n$ is optimal for the new instance, because jobs are sorted by deadline. Note that the lateness of jobs 1 through $i-1$ is unchanged in the new schedule; this means the lateness of those jobs is alreay zero in the original schedule. Also note that the finish times of jobs $i+1, \ldots, n$ each decrease by exactly t_i; since they now finish on time, we know that $m(i+1) - t_i \leq 0$.

So, we are guaranteed the existence of an index i such that: (a) the first i jobs currently finish on time and (b) $m(i+1) - t_i \leq 0$. By finding the first index i such that (b) is true we are guaranteed to find the correct job.

This can be done in linear time with one pass through the list.