CMPSCI 311: Introduction to Algorithms
First Midterm Exam

October 13, 2016.

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 $\sum_{i=1}^{n} i \geq \sum_{i=n/2}^{n} i \geq \sum_{i=n/2}^{n} n/2 = n^2/4$.

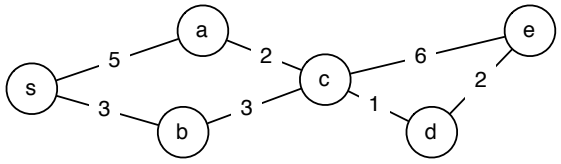
1.2 (*2 points*): *If T is a BFS tree for a graph G and (u, v) is an edge in G that is not in T, then u and v are the same distance from the root of the T.*
Solution. False. The level of u and v can disagree by at most 1. We proved this in class, but a square graph on 4 nodes has this property. Consider 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*): *The recurrence $T(n) = 3T(n/2) + n$ solves to $T(n) = \Theta(n^{\log_2 3})$.*
Solution. True. We know that $T(n) = \sum_{i=0}^{\log_2 n} \frac{3^i n}{2^i}$ by unraveling the recurrence. Using the formula for partial geometric series, this expression is $\Theta(n^{\log_2 3})$.

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 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.,* $\max_{1 \le t \le n} A[t]$.

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 days that have the same value, i.e., does there exists* $i \neq j$ *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., $\max_{1 \le t \le n} A[t]$.

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*) Let $T = (V, E)$ be a balanced binary tree with $n = 2^k - 1$ nodes including a root named r. Suppose every node $v \in V$ has an associated weight $w(v)$ that can be either positive or negative. We say $T' = (V', E')$ is a *rooted subtree* of T if

1. $V' \subseteq V$ and $E' \subseteq E$.
2. $r \in V'$ and whenever u is a node in T' then the parent of u is also in V'.
3. Whenever $u, v \in V'$ and (u, v) in an edge of T then $\{u, v\}$ is an edge of T'.

We say the weight of T' is $\sum_{v \in V'} w(v)$.

4.1 (*2 points*): *If $n = 7$, how many rooted subtrees does T have?*

Solution. Let $T(n)$ be the number of rooted subtrees on n nodes. If $n = 3$ there are 4 rooted subtrees, so $T(3) = 4$ (root, root+left, root+right, all three). Now when $n = 7$, there are $1 + T(3) + T(3) + T(3)^2$, which is root, root+something from left, root+something from right, and root+something from both. Thus there are 25 rooted subtrees.

4.2 (*2 points*): *Solve the recurrence $f(n) \leq 2f(n/2)+c$ and $f(1) = c$ where c is a positive constant.*

Solution. The recurrence solves to $f(n) = O(n)$. Unraveling the recurrence gives,

$$\sum_{i=0}^{\log_2} 2^i c \leq 2c \times 2^{\log_2 n} = 2cn$$

Or you can count how much work is done per edge in the tree, which is $O(1)$ per edge, but there are n edges.

4.3 (*6 points*): *Design and analyze an efficient algorithm for finding the rooted subtree of maximum weight. Remember to explain why your algorithm is correct and give the running time.*

Solution. The algorithm is by divide and conquer. Find the rooted subtree of maximum weight on the left and right and then we can combine in $O(1)$ time as follows. Take max of $w(r)$ (the weight of the root), $w(r) + \text{best}_L$ (weight of the root plus best rooted subtree on the left), $w(r) + \text{best}_R$ (weight of the root plus best rooted subtree on the left), and $w(r) + \text{best}_L + \text{best}_R$ (weight of the root plus best rooted subtree on both sides). You should also return a representation of the subtree, but building this subtree requires $O(1)$ operations given the subtrees that achieve best_L and best_R.

The running time is given by the recurrence in the previous part, which solves to $O(n)$.

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**	**No**
$d_1 = 9, d_2 = 4$, *and* $d_3 = 13$*?*	**Yes**	**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 ith 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 any i such that $m(i) - t_{i-1} \leq 0$. If you find one, report that ordering without the $i-1$st job (so you don't complete the $i-1$st assignment). If you plan to remove some job $i-1$, then the optimal ordering for minimizing lateness is to order by deadline, which is exactly what we have here. Moreover, subtracting t_{i-1} from $m(i)$ effectively shifts every job back by t_{i-1} for the purpose of checking for lateness. So if there exist some i such that $m(i) - t_{i-1} \leq 0$ then removing job $i-1$ will give a valid ordering. On the other hand if this never happens, then after removing $i-1$ there is always some job that is still delayed, since $m(i) > 0$.

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