CMPSCI 311: Introduction to Algorithms
Second Midterm Exam

April 11, 2018.

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.
- **Hint:** To make sure you get as many points as possible, remember that it is often possible to answer later parts of a question without being able to answer earlier parts of a question.

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*): *The recurrence* $T(n) \leq 3T(n/2) + cn$ *with* $T(2) \leq c$ *solves to* $T(n) = \Theta(n^{\log_2 3})$.
Solution: True, just apply the formula we saw in class.

1.2 (*2 points*): *Let* f *be some flow in a flow network and let* S *be some cut. Then it may be the case that* $v(f) > capacity(S)$*, i.e. the value of the flow* f *is larger than the capacity of the cut* S.
Solution: False, by Max Flow-Min Cut Theorem

1.3 (*2 points*): *Given a collection of* k*-bit integers* $w_1, \ldots, w_n$ *and a* k*-bit integer budget* W*, the dynamic programming algorithm can find the maximum subset sum in running time that is polynomial in* n *and* k.
Solution: False, the running time is $O(n2^k)$.

1.4 (*2 points*): *Let* G *be a weighted directed graph possibly with negative edge weights, but with no negative cycles. Then there exists a polynomial time algorithm to find the shortest* $s - t$ *path in* G.
Solution: True, we can use Ford-Fulkerson algorithm.

1.5 (*2 points*): *If there exists an* $s - t$ *path in the residual graph* G_f *for some flow* f*, then* f *is not the maximum flow.*
Solution: True, we can increase the flow by pushing along the found s-t path.

Question 2. (*10 points*) In this problem we consider the sequence alignment problem. Consider two strings

A L G O
T E S T

2.1 (*3 points*): *Suppose the gap penalty $\delta = 1$ and the alignment costs are $C(x, y) = \mathbf{1}\{x \neq y\}$.*

1. *(2 points) What is the optimal alignment?* **Solution:** (A-T), (L-E), (G-S), (O-T)
2. *(1 point) What is the cost of the optimal alignment?* **Solution:** 4

2.2 (*3 points*): *Suppose the gap penalty $\delta = 0.5$ and the alignment costs $C(x, y)$ is 0 if $x = y$, 1 if x and y are both vowels, 1 if x and y are both consonants, and 2 otherwise.*

1. *(2 points) What is the optimal alignment?* **Solution:** There are many, e.g., (A-E), (L-S), (G-T)
2. *(1 point) What is the cost of the optimal alignment?* **Solution:** 4.

Now suppose we are given two strings x, y and would like to find a substring y' of y that minimizes the alignment cost with the entire string x. Formally, we would like to find a substring y' that minimizes $\text{SeqAlign}(x, y', C, \delta)$ among all substrings y' of y, where SeqAlign is the standard sequence alignment algorithm, C is a cost matrix, and δ is a gap penalty. Recall that a substring y' is a contiguous sequence of characters $y_i, y_{i+1}, \ldots, y_j$ of y where $1 \leq i \leq j \leq n$. Suppose that x is length n and y is length m, also assume $\delta \geq 0$ and $C[a, b] \geq 0$ for all characters, a, b.

2.3 (*2 points*): *Let $F[i, j]$ denote the cost of the optimal alignment between all of $x_1, \ldots, x_i$ and the best* suffix *of $y_1, \ldots, y_j$. How should we set $F[0, j]$ to handle the base case?*
Solution: We should set $F[0, j] = 0$ for all j, since when x is the empty string, the alignment cost with the empty suffix of y is zero. In other words, we don't have to pay for putting gaps at the beginning of x.

2.4 (*2 points*): *After changing the initialization, we run the standard forward program for sequence alignment to compute $F[i, j]$ for all i, j. From this matrix, what is the cost of the optimal substring alignment? (Note you do not have to recover the alignment)*
Solution: The cost is $\max_{0 \leq j \leq m} F[n, j]$, which is the optimal cost for aligning all of x, with the best suffix of $y_1, \ldots, y_j$, for the best prefix of y. This is precisely the cost of aligning all of x with the best substring of y.

Question 3. (*10 points*) In this problem we will study an unfamiliar recurrence, and we will derive an asymptotic upper bound from first principles. Consider an algorithm whose running time on an input of size n satisfies

$$T(n) \leq 4T(n/2) + cn^2, \qquad T(2) \leq c.$$

For the following questions, it may be helpful to draw the recursion tree.

3.1 (*2 points*): *Derive an upper bound on $T(n)$ in terms of $T(n/4)$ (and n, c, and any numerical constants, but* not *in terms of $T(m)$ for $m \neq n/4$). In other words, unroll the recursion twice.*
Solution:

$$T(n) \leq 4T(n/2) + cn^2 \leq 16T(n/4) + 4c(n/2)^2 + cn^2 = 16T(n/4) + 2cn^2.$$

3.2 (*2 points*): *For any $i \geq 0$, how many subproblems of size $n/2^i$ must we solve (express your answer as a function of n and i)?*
Solution: 4^i.

3.3 (*2 points*): *For any $i \geq 0$, how much additional work (not including the recursive calls), must we do for* all *problems of size $n/2^i$ (express your answer as a function of n and i)?*
Solution: $O(n^2)$.

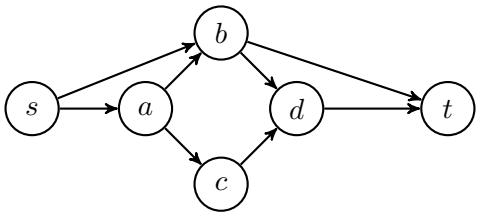
3.4 (*2 points*): *We may write the total running time as*

$$T(n) \leq \sum_{i=0}^{???} \textit{total work for problems of size } n/2^i$$

What should we put in for the upper limit of the summation (where ??? is)? In other words, what is the largest value of i we must consider?
Solution: Acceptable answers are $\log_2(n), \lceil \log_2(n) \rceil, \lfloor \log_2(n) \rfloor, \log_2(n) + 1, \log_2(n) - 1$.

3.5 (*2 points*): *What is the Big-Oh running time of the algorithm, as a function of n?*
Solution: $O(n^2 \log_2(n))$.

Question 4. (*10 points*) Let $G = (V, E)$ be a directed acyclic, unweighted, graph with a source node s and a target node t. There are no incoming edges to s and no outgoing edges from t. Here, we will design an algorithm to count the number of distinct $s - t$ paths in the graph. Two paths P, P' are distinct if there is some edge e that is in P but not in P', or vice versa (i.e. they are not the exact same sequence of edges).



4.1 (*2 points*): *How many distinct $s - t$ paths are in the graph above?* **Solution:** 5

4.2 (*2 points*): *Consider a "greedy" algorithm that finds any path P, modifies the graph by removing all edges in P, and repeats, returning the number paths found. Draw a graph with at most* 5 *nodes (including s, t) where this algorithm returns an incorrect answer.*
Solution: Remove c in the above graph.

4.3 (*2 points*): *Let* ForwardPaths(v) *denote the number of distinct paths* from s *to some vertex $v \in V$, with* ForwardPaths(s) $= 1$. *Write down a recursive formula for* ForwardPaths(v).
Solution:

$$\textsc{ForwardPaths}(v) = \sum_{(u,v)\in E} \textsc{ForwardPaths}(u)$$

4.4 (*2 points*): *We can also write down a backwards recursion. Let* BackwardPaths(u) *denote the number of paths* from *some vertex $u \in V$ to t, with* BackwardPaths(t) $= 1$. *Write down a recursive formula for* BackwardPaths(u).
Solution:

$$\textsc{BackwardPaths}(u) = \sum_{(u,v)\in E} \textsc{BackwardPaths}(v)$$

4.5 (*2 points*): *Fill in the following pseudocode to complete the algorithm.*

Algorithm 1 CountPaths(G)

order = ***Solution:*** *Topological order of G* {Fill me in}
for v in order **do**

ForwardPaths(v) = ***Solution:*** $\sum_{(u,v)\in E}$ ForwardPaths(u) {Fill me in}
end for
return ForwardPaths(t)

Question 5. (*10 points*) Suppose we are given an array of objects $A[1 \dots n]$. Any object that occurs in strictly more than $n/2$ positions is called a *majority element*, and in this problem, we will design an algorithm for finding a majority element, or deciding that no such element exists. Assume that you can test if two objects are equal, but they may not be sorted (i.e. there is no notion of "less than" for these objects).

For example, the list $[o_1, o_2, o_1, o_1, o_3]$ has o_1 as the majority, but $[o_1, o_2, o_1, o_1, o_3, o_4, o_3]$ has no majority.

5.1 (*3 points*): *In a divide and conquer approach, suppose we recurse on the left and right halves of the array, but both recursive calls report that no majority element exists. What are the remaining possibilities for the majority element of the original array? Briefly explain why.*
Solution: There are no remaining possibilities for the majority element. This is true because every element appears less than $n/4$ times in the left and also less than $n/$ times on the right, so it appears less than $n/2$ times overall.

5.2 (*3 points*): *Now, suppose that the left recursive call reports that o_L is a majority element of the left half, while the right reports that o_R is the majority element of the right half. What are the remaining possibilities for the majority element of the original array? Briefly explain why.*
Solution: The only two possibilities are o_L and o_R. For any other element, it appears less than $n/4$ times on the left (since o_L appears more than $n/4$ times on the left), and similarly on the right. So no other element can be a possibility. o_L and o_R remain possibilities since o_L might have appears $n/2$ times on the left, and 1 time on the right.

5.3 (*2 points*): *Describe how to test if object o is a majority element in a list of length n in $O(n)$ time.*
Solution: Simply traverse the array and check equality between o and every element, and return true if we find more than $n/2$ elements that are equal.

5.4 (*2 points*): *The above arguments inspire a divide and conquer algorithm. What is the running time of that algorithm?*
Solution: $O(n \log_2 n)$. The algorithm has the same structure as MergeSort.