Mid-term. February 3, 2017

CS180: Algorithms and Complexity
Winter 2017

Guidelines:

- The exam is closed book and closed notes. Do not open the exam until instructed to do so.
- Write your solutions clearly and when asked to do so, provide complete proofs. You may use results we proved in class without proofs as long as you state what you are using.
- I recommend taking a quick look at all the questions first and then deciding what order to tackle to them in. Even if you don't solve the problems fully, attempts that show some understanding of the questions and relevant topics will get reasonable partial credit.
- You can use extra sheets for scratch work, but try to use the white space (it should be more than enough) on the exam sheets for your final solutions.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly and any cheating reported.

Problem	Points	Maximum
1	10	10
2	4	4
3	4	4
4	4	4
5	4	4
Total	26	26

Name	Solutions Midterm 1
UID	None
Section	None

1 Problem

The answers to the following should fit in the white space below the question.

1. For each pair (f, g) below indicate the relation between them in terms of O, Ω, Θ. For each missing entry, write-down Y (for YES) or N (for NO) to indicate whether the relation holds (no need to justify your answers here). For example, if $f = O(g)$ but not $\Omega(g)$, then you should enter Y in the first box and N in the other two boxes. Similarly, if $f = \Theta(g)$, then you should enter Y in all the boxes. [1 point]

f	g	O	Ω	Θ
$\log_3 n$	$\log_9 n$	**N**	**Y**	**N**
3^n	6^n	**Y**	**N**	**N**

2. Is the following True or False: Consider a divide and conquer algorithm which solves a problem on an instance of length n by making five recursive calls to instances of length ($\lfloor n/2 \rfloor$) each, and combines the answers in $O(n^2)$ time. Then, the time-complexity of the algorithm is $O(n^2)$. [1 point]
False, O($n^{log_3 5}$) according to Master's Theorem Case 1

3. State the principles behind the divide and conquer technique for designing algorithms. [1 point]
Dividing the problem into smaller subproblems, solve the subproblems, and combine the subproblems into the final solution

4. What is the solution to the recurrence $T(1) = 1$, $T(n) = 2T(n/2) + 100n$? [1 point]
O($nlogn$) by Master's Theorem Case 2

5. Write down the definition of the discrete Fourier Transform for signals of length n (i.e., write down the expression we used for $DFT_n(a_0, a_1, \ldots, a_{n-1})$). [1 point]
For i =[0,1,...n-1]

$$Si = \sum_{k=0}^{n-1} w^{ik} * a_k = 1$$

$DFT_n(a_0, a_1, \ldots, a_{n-1}) = (S_0 S_1 ... S_{n-1})$

6. Write down some pros and cons of the adjacency-list and adjacency-matrix representations of graphs. [1 point]
Pros of adjacency list: O(E) storage space
Cons of adjacency list: O(V) to check if edge exists
Pros of adjacency matrix: O(1) to check if edge exits
Cons of adjacency matrix: O(V^2) storage space

7. Let G be a weighted directed graph with positive weights. Suppose we ran Dijkstra's algorithm starting from a vertex s to compute the shortest distances from s to all vertices in G and let T be the tree formed by the PARENT links that are computed during the run of the algorithm.

   Now suppose we change the weights of the graph as follows: for every edge e that is **not** part of T, its weight is doubled, i.e., its weight w_e is replaced with $2 \cdot w_e$. The weights of edges in T are not changed. This creates a new instance G' of the problem with the same underlying graph but different costs on all the edges which are **not part of** T.

   True or false: "The shortest distances from s to other vertices in the new instance are the same as they were in the original weighted graph." If true, provide a brief explanation why and if false, provide an example of a graph G where the statement fails. [2 points]
   True, scaling the edges not in T by a factor of 2 would not modify the list of edges in T because they would still be less than edges not in T. (Following Dijkstra's algorithm.

8. Let $G = (V, E)$ be a weighted undirected graph with positive weights and let s be a vertex in G. Consider a variant of Dijkstra's algorithm where we grow the set of vertices S by picking the vertex $v \notin S$ that has the shortest edge to any vertex in S. That is, consider the following algorithm:

   (a) Set $S = \{s\}$. Set $c(s) = 0$ and $c(v) = \infty$ for all $v \neq s$.

   (b) While $S \neq V$:

      i. For each vertex $v \notin S$, let $c'(v) = \min\{\ell_{(u,v)} : u \in S, (u, v) \in E\}$.

      ii. Find the vertex $v \notin S$ with least $c'(v)$ and let $u \in S$ be the corresponding vertex that achieves the minimum in the definition of $c'(v)$.

      iii. Set $c(v) = c(u) + \ell_{(u,v)}$.

   Does the above algorithm compute the lengths of the shortest paths from s to all other vertices? That is, are the numbers $c(v)$ computed by the algorithm the distances to v from s? If yes, provide a brief explanation why this may be true. If not, provide an example of a graph G where the algorithm fails to compute the lengths of the shortest paths. [2 points]
   No, the algorithm would not find the shortest path. (Example will be shown in discussion)

2 Problem

Let n be an even integer and let Q_n denote the $n \times n$ matrix with rows and columns indexed by $0 \leq j, k \leq n-1$ and $Q_n[j,k] = e^{-2\pi i (j \cdot k)/n}$.

1. Can you identify any repeating pattern in the matrix Q_n (like the one we saw in our derivation of FFT for computing the discrete Fourier Transform)? [2 points]

2. Can you connect the matrices in the pattern to the matrix $Q_{n/2}$? [2 points]
   See lecture transcript for January 18, pages 23-26.

3 Problem

An array $A[0, 1, \ldots, n-1]$ is said to have a *majority element* if more than half of its elements are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as mp3 files, say; so in particular, you cannot sort the elements.) However you can answer questions of the form: "is $A[i] = A[j]$" in constant time.

Give an algorithm to solve the problem. For full-credit, your algorithm should run in time $O(n \log n)$. (You don't have to prove correctness or analyze the time-complexity of the algorithm.) [4 points]

(Hint: Split the array A into two arrays A_L and A_R of half the size each. Does knowing the majority elements of A_L and A_R help you figure out the majority element of A? If so, you can use a divide-and-conquer approach.)

We split the original array into A_L and A_r . The only way for the original array A to have a majority is if one of the subarrays (A_L or A_R) have a majority. This is true because if both subarrays did not have a majority element, then that means that all elements occured less than n/4 times. Therefore it would be impossible for any one of those elements to have occured over n/2 times in the array.

Using divide and conquer we can divide the problem until we hit the base case (1 element sized array) and then when merging, we can check to see if the the majority elements from each respective subproblems is still the majority in the merged array. This would cost O(n) at each level. The recursion for this algorithm would look like: T(N) = 2T(N/2) + O(n) and therefore have the complexity of O(nlog(n). *Note that solutions in linear time were also accepted.

4 Problem

Let $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{\{1, 2\}, \{1, 6\}, \{2, 5\}, \{2, 6\}, \{3, 4\}, \{3, 5\}, \{3, 6\}, \{4, 6\}, \{5, 6\}\}$. Suppose that G was given to you in adjacency list representation where the elements in the adjacency list are ordered in increasing order. For example, the adjacency list of vertex 2 would be $[1, 5, 6]$.

1. Draw the BFS tree that you would get when doing BFS starting from 1. [2 points]

2. Draw the DFS tree that you would get when doing DFS starting from 1. [2 points]

(You don't have to show all the stages of the algorithms just the final trees. Also, keep in mind that you process elements of the adjacency list in increasing order. For example, when doing DFS, you push vertices from an adjacency list onto the stack in increasing order.)

Solution will be discussed in discussions.

5 Problem

Let $G = (V, E)$ be an undirected (unweighted) graph and for any two vertices $u, v \in G$, let $distance_G(u, v)$ be the length of the shortest path between u, v if one exists and ∞ if they are not connected. Define the diameter of a graph G to be the maximum distance between any two vertices of the graph G; that is, $diameter(G) = \max\{distance_G(u, v) : u, v \in G\}$. Give an algorithm that given a graph $G = (V, E)$ (in adjacency list representation) as input, computes the diameter of G, $diameter(G)$. For full-credit, your algorithm should run in time $O(|V|^2 + |V| \cdot |E|))$. [4 points]

(You don't have to prove correctness or analyze the time-complexity of your algorithm.)

The basic idea is to run BFS on each vertex, s and return the maximum level/farthest vertex distance. This is a brute force approach because you are traversing through each possible vertext and finding the max distance. The only catch is that you would need to check to see if any of the vertices in G are disconnected, meaning the max distance would be infinity. We can check this at the end of the BFS algorithm for each vertex and verify that all vertices are in fact discovered/reached. The complexity comes out to be O(V* (V+E)) = O($V^2 + V * E$**)**