Midterm 1

6:00-8:00pm, 2 March

Notes: There are **four** *questions on this midterm. Answer each question part in the space below it, using the back of the sheet to continue your answer if necessary. If you need more space, use the blank sheet at the end.* **None of the questions requires a very long answer, so avoid writing too much! Unclear or long-winded solutions will be penalized.** *The questions vary quite a bit in difficulty, so if you are having problems with part of a question, leave it and try the next one. The approximate credit for each question part is shown in the margin (total 65 points). Points are not necessarily an indication of difficulty!*

Your Name: **Your Section No:**

1. True or False?

For each of the following statements, say whether each statement is True or False. If it is True, give a **brief** explanation (~ one sentence); if it is False, give a **simple** counterexample.

(i) For any two non-negative functions $f(n)$ and $g(n)$, both of which tend to infinity, we must have either $f(n) = O(g(n))$ or $g(n) = O(f(n))$. *3pts*

(ii) Suppose the pre- and post-visit labels of two vertices in a depth-first search on an *undirected* graph $G = (V, E)$ satisfy $\text{pre}(u) < \text{post}(u) < \text{pre}(v) < \text{post}(v)$. Then there can be no edge between u and v. *3pts*

(iii) Suppose the pre- and post-visit labels of two vertices in a depth-first search on a *directed* graph $G = (V, E)$ satisfy $\text{pre}(u) < \text{post}(u) < \text{pre}(v) < \text{post}(v)$. Then there can be no edge (in either direction) between u and v. *3pts*

[continued on next page]

[Q1 continued]

(iv) In a DFS of a directed graph G, the set of vertices reachable from the vertex with *lowest* post-visit label is a strongly-connected component of G. *3pts*

(v) In a DFS of a directed graph G, the set of vertices reachable from the vertex with *highest* post-visit label is a strongly-connected component of G. *3pts*

(vi) If an optimal Huffman code has codewords of length 1 and 3, then it must also have at least one codeword of length 2. *3pts*

[continued on next page]

2. Divide-and-Conquer

Each of the following scenarios outlines a divide-and-conquer algorithm. In each case, write down the appropriate recurrence relation for the running time as a function of the input size n and give its solution. You need not give a full derivation of your solution, but you should indicate how you arrived at it (e.g., by appealing to the Master Theorem). You may assume that n is of some special form (e.g., a power or multiple of some number), and that the recurrence has a convenient base case with cost $\Theta(1)$.

(a) An input of size n is broken down into five subproblems, each of size $n/2$. The time taken to construct the subproblems, and to combine their solutions, is $\Theta(n^2)$. *4pts*

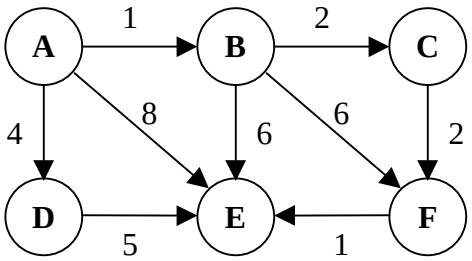
(b) An input of size n is broken down into three subproblems, each of size $n/2$. The time taken to construct the subproblems, and to combine their solutions, is $\Theta(n^2)$. *4pts*

(c) An input of size n is broken down into $\sqrt{n}$ subproblems, each of size $\sqrt{n}$. The time taken to construct the subproblems, and to combine their solutions, is $\Theta(n)$. [HINT: Write $\sqrt{n}$ as $n^{1/2}$. Note that $n^{1/\log n}$ is constant.] *4pts*

[continued on next page]

3. Dijkstra's algorithm

(a) Run Dijkstra's algorithm on the following directed graph, to compute distances from node *A* to all the other nodes. Fill in the entries of the table, showing the "dist" label of each node after each iteration. *6pts*



	Iteration					
Node	0	1	2	3	4	5
A	0					
B	∞					
C	∞					
D	∞					
E	∞					
F	∞					

(b) Consider the following problem. You are given a directed graph representing a network of highways, together with a black box that gives you the *predicted travel time* for each edge in the graph. Because traffic conditions vary, this travel time depends on the time of day at which you start to traverse the edge. Specifically, the travel time on edge $e = (u, v)$ is a positive function $\ell_e(t)$, such that if you leave node u at time t then you will arrive at v at time $t + \ell_e(t)$. The black box allows you to query the predicted travel time $\ell_e(t)$ for any edge e and any time t in advance. Travel times satisfy the *physicality* constraint: $t + \ell_e(t) < t' + \ell_e(t')$ if $t < t'$, i.e., you cannot arrive at v earlier by leaving u later. Your goal is to figure out a route, starting at time t_0 from node s, to a destination node z, so that your predicted arrival time at z is as early as possible. Explain carefully how to modify Dijkstra's algorithm so that it solves this problem. (You do not need to give the full pseudocode; just indicate how the algorithm changes.) *6pts*

(c) Explain *briefly* why your algorithm breaks down if the physicality constraint is removed. *2pts*

[continued on next page]

4. Yet another MST algorithm

We have seen several algorithms for computing a Minimum Spanning Tree (MST) in a connected, undirected graph. This question discusses yet another algorithm for this problem. For simplicity, we assume that all the edge weights are distinct.

(a) For each vertex v, let e_v be the edge with minimum weight incident on v. Prove that there is a MST of G that contains e_v. *4pts*

(b) The new algorithm is based on the observation in part (a). It uses the operation of "contracting" an edge: when an edge $e = \{u, v\}$ is contracted, its endpoints u, v are merged into a single vertex. If either (or both) of u, v have an edge to another vertex w, then the merged vertex also has an edge to w; the weight of this new edge is the minimum of the weights of the edges $\{u, w\}$ and $\{v, w\}$. Here is the algorithm: *5pts*

```
function MST(G = (V, E))
     X=∅
     while |V| > 1:
          for each v in V:
               e_v = minimum edge weight incident on v
          for each edge e_v:
               add e_v to X
               contract e_v
     return X
```

Using part (a), explain carefully why this algorithm outputs a MST of G.

[continued on next page]

[Q1 continued]

(c) Show that the number of iterations of the while-loop is $O(\log |V|)$. [HINT: The number of vertices remaining after the first iteration of the loop is at most $|V|/2$ – why?] *4pts*

(d) Show that the entire algorithm can be implemented to run in time $O(|E| \log |V| \log^* |V|)$. *5pts*

(e) Can you reduce the running time to $O(|E| \log |V|)$? *3pts*

[The end]