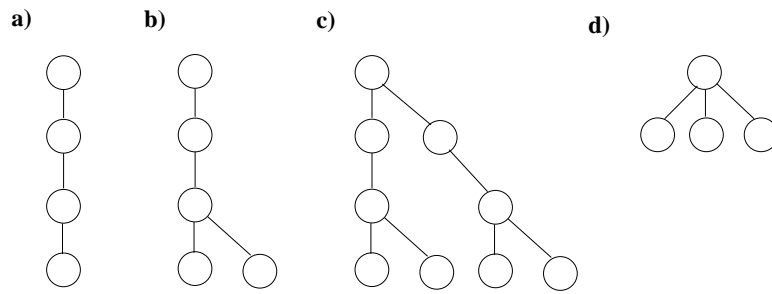


You can work in groups of up to 3. Please let us know, for each question, if you have seen the question before. You will be graded out of 100 points for this assignment. Unless otherwise specified, in all graph problem  $m$  is the number of edges and  $n$  is the number of vertices of a graph. In addition, Omega in scoring guides asks you to prove a lower bound for *all* deterministic algorithms, not just your algorithm. Please submit your solution in ALL of the following formats by email to [virgi+algo@cs.cmu.edu](mailto:virgi+algo@cs.cmu.edu) and [chengwen+gradalg@cs.cmu.edu](mailto:chengwen+gradalg@cs.cmu.edu): ps, pdf and tex. Please use 'Submission: names' in the subject line.

**Problem 1** For each of the following trees, is it a possible Fibonacci tree in a Fibonacci heap?



Scoring (for each):

5 pts for a correct answer and explanation.

**Problem 2** Consider a directed graph  $G$  with equal edge weights  $k > 0$  on all edges and a starting vertex  $s$ . Create an algorithm to compute the shortest path from  $s$  to all other nodes.

Scoring:

5 pts for  $O(m + n \log n)$

10 pts for  $O(m + n)$

And/Or 20 pts for proving this is as hard as single source shortest path (SSSP) problem.

And/Or 20 pts for  $\Omega(m + n \log n)$

**Problem 3** Consider a directed graph  $G$  with distinct weights on all edges, a starting vertex  $s$  and a terminal vertex  $t$ . Create an algorithm to compute the shortest path from  $s$  to  $t$ .

Scoring:

5 pts for  $O(m + n \log n)$

15 pts for  $O(m + n)$

And/Or 30 pts for proving this is as hard as SSSP problem.

And/Or 30 pts for  $\Omega(m + n \log n)$

**Problem 4** Add a Change-Key operation for binomial heap without changing its data structure. (e.g. You do not need to add additional pointers.) Specifically,  $\text{Change-Key}(x_i, k)$  changes the value of  $x_i$  to  $k$ . (You may assume a pointer to  $x_i$  is given.) If you are unsure of binomial heap's representation, please refer to Chapter 20 in CLR.

Scoring:

- 5 pts for  $O(\log^2 n)$
- 10 pts for  $O(\log n \log \log n)$
- 15 pts for  $O(\log n)$
- 20 pts for  $O(\log \log n)$
- 25 pts for  $O(1)$

**Problem 5** In this problem, you will be creating a data structure that supports 3 operations:

- $\text{MakeSet}(x_1, x_2, x_3 \dots, x_n)$  creates a set,  $S$ , of  $n$  elements with values  $v_1, v_2, \dots, v_n$ . (You may assume all  $v_i$  are distinct.)
- $\text{Increase-Key}(x_i, k)$  increases the  $v_i$  to  $k$ . (You may assume a pointer to  $x_i$  is given, and  $k > v_i$ )
- $\text{Find-Min}(S)$  returns the minimum of the set  $S$ .

1. Give an efficient data structure for each of the following scenarios. A user wants to make a single  $\text{MakeSet}$  of  $n$  elements, and interleave the following operations:

(a)  $n$   $\text{Increase-Key}$  operations, and  $\log n$   $\text{Find-Min}$  operations

Scoring: (Based on total running time for all operations, including  $\text{MakeSet}$ .)

- 5 pts for  $O(n \log n)$
- 15 pts for  $O(n \log \log n)$
- 20 pts for little  $o(n \log \log n)$
- 25 pts for  $O(n)$

(b)  $n \log n$   $\text{Increase-Key}$ , and  $n$   $\text{Find-Min}$  operations.

Scoring: (Based on total running time for all operations, including  $\text{MakeSet}$ .)

- 5 pts for  $O(n \log^2 n)$
- 10 pts for little  $o(n \log^2 n)$
- 15 pts for  $O(n \log n \log \log n)$
- 20 pts for little  $o(n \log n \log \log n)$
- 30 pts for  $O(n \log n)$

2. If we need  $\text{Increase-Key}$  operation to run in little  $o(\log n)$ , find an efficient data structure to support  $\text{Find-Min}$  operation.

Scoring: (Based on the running time for  $\text{Find-Min}$ )

- ( $n$  is the number of elements)
- 10 pts for little  $o(n)$
- 20 pts for  $O(\log^2 n)$

30 pts for  $O(\log n)$   
40 pts for anything better

and/or (i.e. Prove that for any deterministic data structure that supports Increase-Key in little  $o(\log n)$  and any running time for MakeSet, there exists a sequence of operations such that at least a single Find-Min must take longer than any of the followings.)

5 pts for any non-constant lower bound  
10 pts for  $\Omega(\log \log n)$   
15 pts for  $\Omega(\log n)$   
20 pts for anything better

3. We need Find-Min to run in  $O(n^{1-\epsilon})$  worst case for a small positive constant  $\epsilon$ . Find an efficient data structure to support both this Find-Min and an Increase-Key operation.

Scoring: (Based on the running time for Increase-Key)  
( $n$  is the number of elements)

5 pts for  $O(\log n)$   
10 pts for little  $o(\log n)$   
15 pts for  $O(\log \log n)$   
20 pts for little  $o(\log \log n)$

and/or

10 pts for any non-constant lower bound  
20 pts for  $\Omega(\log \log n)$   
30 pts for  $\Omega(\log n)$   
40 pts for anything better