Data Structures and Programming

Spring 2014, Final Exam.

Date: June 16, 2014

(A) (30 pts) True or False? (Mark O for true and $\times$ for false.) Score = max$\{0$, Right - $\frac{1}{2}$ Wrong$\}$. No explanations are needed.

(1) If every node in a binary tree of n nodes has either 0 or 2 children, then the height of the tree is $O(\log n)$.
$\times$ **False**

(2) The depths of any two leaves in a binary heap differ by at most 1.
◯ **True.**

(3) A binary heap A (of n keys) has each of its keys randomly increased or decreased by 1. The random choices are independent. We can restore the heap property on A in linear time (i.e., $O(n)$ time).
◯ **True**

(4) The following array is a max binary heap: $[10, 3, 5, 1, 4, 2]$.
$\times$ **False.**

(5) Every directed acyclic graph has exactly one topological ordering.
$\times$ **False.**

(6) Given a graph $G = (V, E)$ with positive edge weights, the Bellman-Ford algorithm and Dijkstra's algorithm can produce different shortest-path trees despite always producing the same shortest-path weights.
◯ **True.**

(7) Dijkstra's algorithm may not terminate if the graph contains negative-weight edges.
$\times$ **False.**

(8) If a depth-first search on a directed graph $G = (V, E)$ produces exactly one back edge, then it is possible to choose an edge $e \in E$ such that the graph $G_0 = (V, E - \{e\})$ is acyclic.
◯ **True.**

(9) Given an undirected graph $G = (V, E)$, it can be tested to determine whether or not it is a tree in $O(|V| + |E|)$ time. A tree is a connected graph without any cycles.
◯ **True.**

(10) If a directed graph G is cyclic but can be made acyclic by removing one edge, then a depth-first search in G will encounter exactly one back edge.
$\times$ **False.** For example, in graph $G = (V, E) = (\{a, b, c\}, \{(a, b), (b, c), (b, a), (c, a)\})$, there are two cycles (a, b, a) and (a, b, c, a) and a DFS from a in G returns two back edges (b, a) and (c, a), but a single removal of edge (a, b) can disrupt both cycles, making the resulting graph acyclic

(11) The Bellman-Ford algorithm applies to instances of the single-source shortest path problem which do not have a negative-weight directed cycle, but it does not detect the existence of a negative-weight directed cycle if there is one.
$\times$ **False**

(12) The topological sort of an arbitrary directed acyclic graph $G = (V, E)$ can be computed in linear time (i.e.,$O(|V| + |E|)$ time) .
◯ **True**

(13) We know of an algorithm for the single source shortest path problem on an arbitrary graph with no negative-weights that works in $O(|V| + |E|)$ time.
$\times$ **False**

(14) If the load factor of a hash table is less than 1, then there are no collisions.
$\times$ **False**

(15) If an operation takes $O(n)$ worst case time, then it takes $O(n)$ amortized time.
◯ **True**

(B) (10 pts) Give an $O(k \log k)$ time algorithm to return the k^{th} smallest element in a min-heap H of size n, where $1 \leq k \leq n$. Explain why your algorithm takes $O(k \log k)$ time .
(Hint: Create a new min-heap I which is initially empty. Then insert $(H(0), 0)$ into I, where the first component $H(0)$ is the root of H and the second component 0 is the index of $H(0)$. At any point in time, a node in I is of the form $(H(p), p)$ where p is an index. Use the first component of $(H(p), p)$ (i.e., $H(p)$) as the key in

creating I.)
Solution:
Create an initially empty min-heap I with each of its elements of the form $(H(p), p)$. The first step is to insert $(H(0), 0)$ into I. Then do the following:

For $i = 1, .., k$
  $(v, p) = I.extractMIN$
  if $i == k$ then *return* v else
    Insert both of p's children into I

The reason to insert both of p's children into I is that the smallest element in the remainder of H is the smallest of the current elements in I plus the two children of p. Since the number of elements of I is bounded by $2k$, the $O(k \log k)$ bound of the algorithm follows.

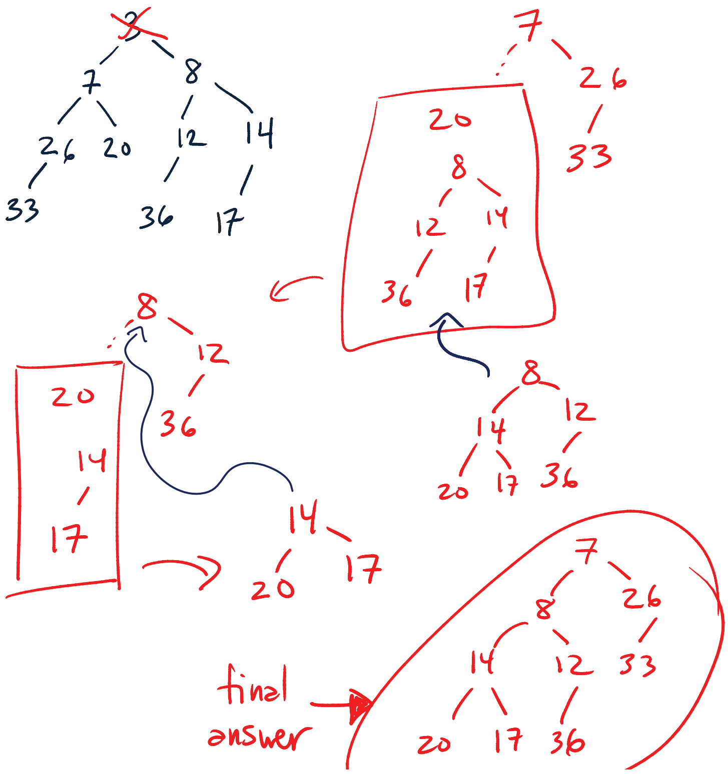
(C) (5 pts) Suppose we have a priority queue data structure that supports EXTRACT-MIN and DECREASE-KEY on integers in $\{0, 1, ..., u-1\}$ in $O(\log \log u)$ time per operation. What is the resulting running time of Dijkstra's algorithm on a weighted directed graph $G = (V, E)$ with edge weights in $\{0, 1, ..., W-1\}$? Why?
Solution Dijkstra's algorithm will call EXTRACT-MIN $O(|V|)$ times and DECREASE-KEY $O(|E|)$ times. In total, the runtime of Dijkstra's using this new priority queue is $O((|V| + |E|)lglg(|V||W))$. Note that the maximum key in the priority queue is bounded by $|V| \times W$.

(D) (5 pts) Give an efficient algorithm to compute the union $A \cup B$ of two sets A and B of total size $|A| + |B| = n$. Assume that sets are represented by arrays that store distinct elements in an arbitrary order. In computing the union, the algorithm must remove any duplicate elements that appear in both A and B. For full credit, your algorithm should run in $O(n)$ time.
Solution

You will still receive full credit if you give an $O(n \log n)$-time algorithm.

For an $O(n)$ algorithm, use perfect hashing with $O(1)$ search time in the worst case for static data. Let H be an initially empty hash table, and R be an initially empty growable array. For each element e in A and B, do the following. If e is in H, skip over e. Otherwise, append e to R and insert e into H.

(E) (10 pts) Draw the skew heap that results from doing a delete-min on the skew heap shown below. Show the details.
Solution



(F) (15 pts) Running Time Analysis: Give the tightest possible upper bound for the worst case running time for each of the following in terms of N. You MUST choose your answer from the following (not given in any particular order), each of which could be re-used. $O(N^2)$,$O(N^{\frac{1}{2}})$, $O(N \log N)$, $O(N), O(N^2 \log N)$, $O(N^5), O(2^N)$, $O(N^3)$, $O(\log N), O(1)$, $O(N^4)$, $O(N^N)$, $O(N^6)$, $O(N(\log N)^2)$, $O(N^2(\log N)^2)$

(1) The decrease-key operation of a min-Fibonacci heap of N nodes.
O(N)

(2) Finding the minimum spanning tree of a weighted graph of N nodes using Kruskal's algorithm. (You may assume that the graph is dense, i.e., $|E| = O(N^2)$)
$\mathbf{N^2 \log N}$

(3) Finding an element in a hash table of size N using open addressing with linear probing.
O(N)

(4) Finding (but not removing) the minimum value in a Fibonacci heap containing N elements.
O(1)

(5) Finding (but not removing) the minimum value in a binomial heap containing N elements.
$\mathbf{O}(\log \mathbf{N})$

(6) Finding an element in a hash table containing N elements where separate chaining is used and each bucket points to an AVL tree. The table size = N.
$\mathbf{O}(\log \mathbf{n})$

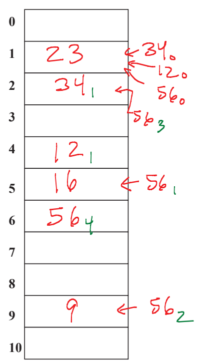
(7) Finding the median value in a leftist heap containing N elements. (You dont know what the median value is ahead of time.) You may assume N is odd.
$\mathbf{O}(\mathbf{N} \log \mathbf{N})$; DO N/2 deletions

(8) Breadth-first search of a graph with N nodes and $N \log^2 N$ edges.
$\mathbf{O}(\mathbf{N} \log^2 \mathbf{N})$

(9) In union-find, what is the worst case running time of a single Find operation (without path compression), assuming that Union-by-size has been used, where N = total number of elements in all sets.
$\mathbf{O}(\log \mathbf{N})$

(10) The height of a leftist heap of N nodes.
O(N)

(G) (5 pts) Consider a hash table of size 11. Open addressing with double hashing is used to resolve collisions. The hash function used is $H(k) = k \ mod \ 11$ The second hash function is $H_2(k) = 5 - (k \ mod \ 5)$. What values will be in the hash table ($A[0..10]$) after the following sequence of insertions? 16, 23 9, 34, 12, 56.
Solution



(H) (20 pts) For each of the following operations, decide whether it is supported by the following three types of min heaps: **A = (classical) Binary Heaps. B = Binomial Heaps. C = Fibonacci Heaps**. This is a Multiple-choice question, i.e., your answer should look like: (1) A C (2) NONE (3) B, ..., for instance. No explanations are needed.

(1) heapify($a_1, ..., a_n$): create a heap on a given set of n elements in $O(n)$ steps.
ABC

(2) makeheap(a): create a heap on 1 element in $O(1)$ steps.
ABC

(3) insert(a, H): insert element a to heap H in $O(1)$ amortized steps.
BC

(4) insert(a, H): insert element a to heap H in $O(\log n)$ steps.
ABC

(5) deletemin(H): delete the minimum element of heap H in $O(1)$ steps.
NONE

(6) deletemin(H): delete the minimum element from heap H in $O(1)$ amortized steps.
NONE

(7) deletemin(H): delete the minimum element from heap H in $O(\log n)$ amortized steps.
ABC

(8) meld(H_1, H_2): combine heaps H_1 and H_2 into one heap in $O(\log n)$ amortized steps.
BC

(9) delete(a, H): remove element a from heap H in $O(\log n)$ amortized steps.
ABC

(10) decrement(a, t, H): decrease the value of the element a in H by amount t in amortized $O(1)$ steps.
C