Data Structures and Programming

Spring 2018, Final Exam.

June 26, 2018

1. (50 pts) True or False? (Score = max {0, Right - $\frac{1}{2}$Wrong }). Please write O for "True," and $\times$ for "False." No explanations are needed. Unless stated otherwise, the $\log n$ function is of base 2.

   (1) Given n keys, a red-black tree of the n keys can be built in $O(n^{1.5})$ time in the comparison model.
   O – $n \log n = O(n^{1.5})$

   (2) Performing a left rotation on a node and then a right rotation on the same node will not change the underlying tree structure.
   X

   (3) A binary max-heap can support both the INCREASE-KEY and DECREASE-KEY operations in $\Theta(\log n)$ time.
   O

   (4) In a Fibonacci heap, it is possible to have a tree of size k (i.e., the number of nodes equals k) for which the number of children of its root is greater than $\lceil \log_2 k \rceil$.
   O – $\lceil \log_\phi k \rceil > \lceil \log_2 k \rceil$, where ϕ is the golden ratio ≈ 1.4.

   (5) Given n integers in the range $0...k$, where k is a constant, it is possible to preprocess these integers into a data structure that can answer the following query in $O(1)$ time: given two integers a and b, how many integers fall within the range $a...b$ (i.e., the number of element x s.t. $a \leq x \leq b$)?
   O – same preprocessing as for bucket sort.

   (6) If a data structure supports an operation foo such that any sequence of n foo operations takes $\Theta(n \log n)$ time, the actual time of a single foo operation could be as high as $\Theta(n \log n)$ time.
   O

   (7) The total amortized cost of a sequence of n operations (i.e., the sum over all operations, of the amortized cost per operations) gives a lower bound on the total actual cost of the sequence.
   X – It gives an upper bound on the actual cost.

   (8) Given an unsorted array $A[1..n]$ of n integers, building a max-heap out of the elements of A can be done in $O(n)$ time regardless of whether the heap is a binary heap, a binomial heap or a Fibonacci heap.
   O

   (9) Given a binary min-heap, finding the 5th smallest key can be done in $O(1)$ time.
   O

   (10) Straight radix sort remains correct (i.e., producing the correct output) if we sort each individual digit using INSERTION SORT instead of BUCKET SORT.
   O – Insertion sort is a stable sort, so RADIX SORT remains correct

   (11) Given an array of n integers, each belonging to $\{-1, 0, 1\}$, we can sort the array in $O(n)$ time in the worst case.
   O – Use bucket sort, e.g., after adding 1 to all numbers

   (12) Consider a sequence of union and find operations while union-by-rank and path compression heuristics are used. For any node x, once x becomes a child of another node y, then the rank of x will never change in the remaining sequence.
   O

   (13) Let T be a complete binary tree with n nodes. Finding a path from the root of T to a given vertex $v \in T$ using recursive depth-first search requires $\Omega(n)$ time in the worst case.
   O – Breadth-first search requires $\Omega(n)$ time.

   (14) Dijsktra's shortest path algorithm runs in $O(|V|^3)$, where $|V|$ is the number of vertices of the graph.
   O – The question is big-Oh, not Θ.

   (15) Consider two positively weighted graphs $G = (V, E, w)$ and $G' = (V, E, w')$ with the same vertices V and edges E such that, for any edge $e \in E$, we have $w'(e) = w(e)^2$, where $w(e)$ and $w'(e)$ are the weights of e in G and G', respectively. For any two vertices $u, v \in V$, any shortest path between u and v in G' is also a shortest path in G.
   X – G has two paths from s to t: s-2-2-t; s-3-t

(16) Given an undirected graph represented as an adjacency list, it can be tested to determine whether or not the graph has exactly two connected components in $O(|V| + |E|)$ time.
O –Use BFS or DFS

(17) Let U be the universe and $|U| = m^2$. We consider hashing with chaining. For any hash function $h : U \rightarrow \{1, 2, ..., m-1\}$ (i.e., a hash table of size $m-1$), there exists a sequence of m insertions that leads to a chain of length at least m.
O – pigeonhole principle

(18) Consider a hash table H of a given size $n > 0$. Also assume a hash function $h(k) = (k \ mod \ n)$. Increasing the size of H to $2n$ (and modifying the hash function accordingly) necessarily mean that the number of collisions decreases by approximately one half.
X – For example, the keys 1, 5, 9, 13 and 17 will hash to position 1 both when the size of the hash table is 2 and 4.

(19) The union-find ADT is useful for effectively implementing Prim's minimum spanning tree algorithm.
X – Priority Queue

(20) $\log^*(2^n) = 2 \times \log^* n$
X – By definition

(21) The worst-case running time of a decrease-key operation of a Fibonacci min-heap is $O(\log n)$.
X – $O(n)$

(22) In a leftist heap of n nodes, the longest path from the root to a leaf node is of length bounded by $O(\log n)$
X – $O(n)$

(23) Given a sorted array in increasing order as the input, MergeSort can be done in $O(n)$ time.
X – $O(n \log n)$

(24) In a splay tree of n nodes, finding the largest element can be done in $O(\log n)$ amortized time.
O

(25) Depth-first search will take $\Theta(|V|^2)$ time on a graph $G = (V, E)$ represented as an adjacency matrix.
O

2. (10 pts) Each of the following arrays shows a comparison sort in progress on a random sequence, corresponding to an algorithm from the list: Selection-Sort, Insertion-Sort, Heap-Sort, Quick-Sort, Bubble-Sort, Shell-Sort, and Merge-Sort. Your task is to match each array to the algorithm that would most likely produce such an array during its execution.

(a) 02 04 01 07 09 08 12 19 13 27 25 33 44 35 51 85 98 77 64 56
QuickSort
Reason: The values show signs of having been partitioned (and by process of elimination)

(b) 12 25 51 64 77 08 35 09 01 07 04 33 44 19 02 85 98 13 27 56
InsertionSort
Reason: The first five elements are sorted but are not the minimum values

(c) 56 51 44 27 13 33 35 25 09 12 04 01 08 19 02 07 64 77 85 98
HeapSort
Reason: Maximum values are at the end and the rest of the array is a max-heap

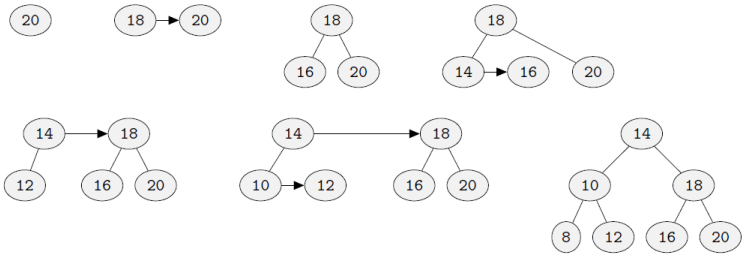
(d) 01 02 04 64 12 08 35 09 51 07 77 33 44 19 25 85 98 13 27 56
SelectionSort
Reason: The first few elements are the minimum values in the array

(e) 12 25 51 64 77 01 07 08 09 35 02 04 19 33 44 13 27 56 85 98
MergeSort
Reason: The array is divided into fourths of which some are in sorted order

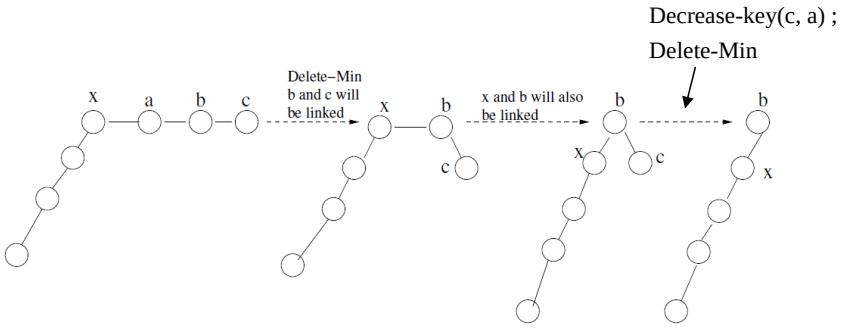
3. (10 pts) Insert 20, 18, 16, 14, 12, 10, 8 into an initially AA tree. Show intermediate trees $T_1 ..., T_7$, where

$$T_0 \overset{20}{\rightarrow} T_1 \overset{18}{\rightarrow} T_2 \overset{16}{\rightarrow} T_3 \overset{14}{\rightarrow} T_4 \overset{12}{\rightarrow} T_5 \overset{10}{\rightarrow} T_6 \overset{8}{\rightarrow} T_7,$$

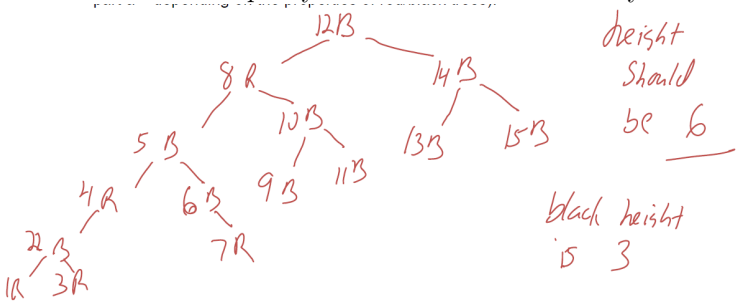
T_0 is an empty tree and $\overset{m}{\rightarrow}$ denotes insert(m). Ensure that horizontal and vertical links are drawn clearly and with arrows.



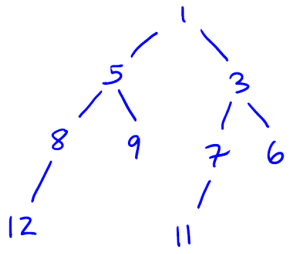
4. (10 pts) Consider the Fibonacci min-heap shown on the left-hand side of the following figure. Assume that $a < b < c < ... < x < ....$ Suppose we want to obtain the Fibonacci heap shown on the right-hand side through a sequence of operations α. Each operation in α is chosen from "insert(μ); delete-min; find-min; decrease-key(μ, ω)", where decrease-key(μ, ω) decreases the key from μ to ω ($\omega < \mu$) and $\mu, \omega \in \{a, b, c, ..., x, y, z\}$. What is α? Show the resulting tree after each of the operations in α is performed.

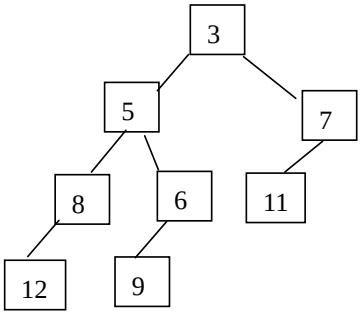


5. (10 pts) Draw the tallest red-black tree containing the values $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$ such that the root of the tree is 12. Specify black and red nodes clearly.

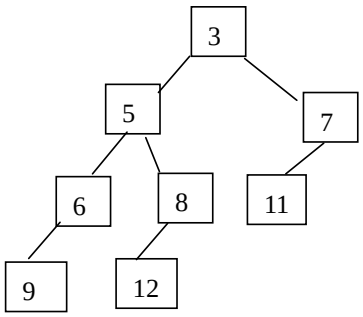


6. (10 pts) Draw the leftist heap and the skew heap that result from doing a deletemin on the min-heap shown below. You are only required to show the two final trees (one for the leftist heap and one for the skew heap).





Leftist heap



Skew heap