Data Structures and Programming
Spring 2017, Final Exam. Solutions

June 20, 2017

1. (16 pts) True or False? (Score = max $\{0, \text{Right} - \frac{1}{2}\text{Wrong}\}$). No explanations are needed.

   (1) Shell-sort is stable.
   F

   (2) Insertion sort makes no more than $O(n \log n)$ pairwise comparisons.
   F

   (3) Linear probing is equivalent to double hashing with a secondary hash function of $h_2(k) = 1$.
   T

   (4) In the union/find data structure of n items, if we use union-by-size without path compression, then any combination of m union and/or find operations takes at most $O(m \log n)$ time.
   T

   (5) Walking a red-black tree with n nodes in in-order takes $\Theta(n \log n)$ time.
   F

   (6) From a sorted array with n elements, a red-black tree can be built in $O(n)$ time.
   T

   (7) A binary search tree can be constructed in $O(n)$ time from a set of n integers in the range 1 to n^3.
   T

   (8) Consider the efficient disjoint set data structure that uses the combined union-by-rank and path-compression heuristics on n elements. The operation FIND-SET(x) is performed to find the set containing x. Every future FIND-SET operations on the same element x will take constant time (i.e., $O(1)$ time).
   F

   (9) Any red-black tree that is not a full binary tree must contain at least one red node.
   T

   (10) Inserting an element into a skew heap of n items could take $\Omega(n)$ time in the worst case.
   T

   (11) AA-trees can be used to implement an optimal comparison-based sorting algorithm.
   T

   (12) Any n-node unbalanced tree can be turned into a balanced red-black tree using $O(\log n)$ rotations.
   F

   (13) An AVL tree is always a leftist tree.
   F

   (14) The decrease-key operation for skew heaps can be done in $O(logn)$ in the worst case.
   F

   (15) The depths of any two leaves in a max binary heap differ by at most 1.
   T

   (16) Start with an empty splay tree, a sequence of n operations (each of which is a *search, insertion*, or *deletion*) takes $O(n \log n)$ time in the worst case.
   T

2. (10 pts) For each of the following situations, name the best sorting algorithm from among those we studied. There may be more than one answer deserving full credit, but you only need to give one answer for each.

   (1) The input array is in random order, and average case time is most important.
   Ans: Quicksort

   (2) The array is in perfect sorted order.
   Ans: Insertion sort or bubble sort

(3) You have a large data set, but you know all the values are between 0 and 999.
Ans: Radix sort or bucket sort

(4) Copying your data is very fast, but comparisons are relatively slow.
Ans: Merge sort

(5) Suppose that exchanges of the items in the array are very, very expensive. In other words, which method does the fewest "swaps" or "moves" of the elements of the array?
Ans: Selection

(6) The input array consists of bits (0s and 1s) and the sort must be stable.
Ans: Straight radix

Please provide an example of each of the following (or say NONE if no example exists) from among the sorting algorithms we studied.

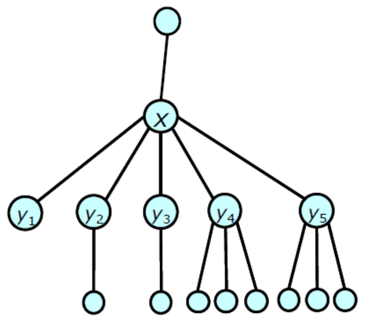
(7) A stable comparison sort with a worst-case $O(n^2)$ running time.
Ans: Insertion, bubble, selection

(8) A stable comparison sort with a worst-case $O(n)$ running time.
Ans: NONE

(9) An efficient (i.e. $O(n \log n)$ worst case) stable comparison sort.
Ans: Merge sort

(10) A stable sort with a worst-case running time linear in n.
Ans: Radix sort or bucket sort

3. (10 pts) Consider the node x and its children $y_1, \cdots, y_5$ in the portion of the Fibonacci heap shown below. Assume that x has never lost any children and that y_1 was the first to become a child of x, followed by y_2, y_3 and so forth. Which children of x must have a mark bit that is set (5 pts)? Explain your answer (5 pts). (Hint: Recall that in a Fibonacci heap, when a node u first becomes a child of a node v, u and v must have the same rank.)



Ans: y_3 and y_5 must both have a mark bit that is set. When each y_i became a child of x, it had to have had $i-1$ children. Nodes y_2 and y_4 still have all of their children, but y_3 and y_5 both have one fewer than the number they must have had when they became children of x. Therefore, their mark bits must be set.

4. (10 pts) Consider the following splay tree. Perform a search for element h.

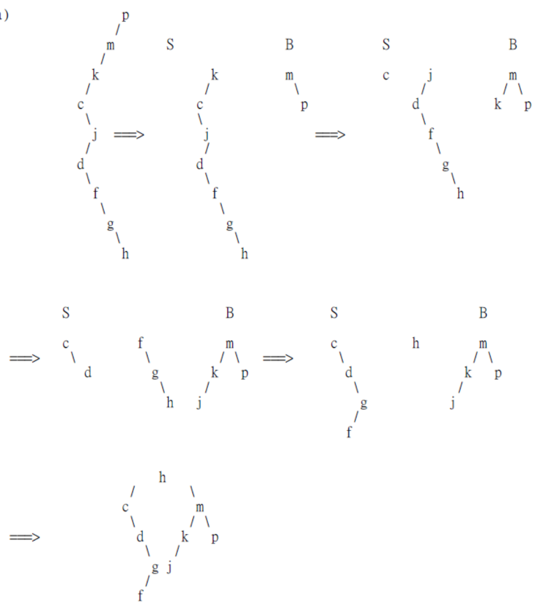
(1) (5 pts) assuming a Top-down splay tree;

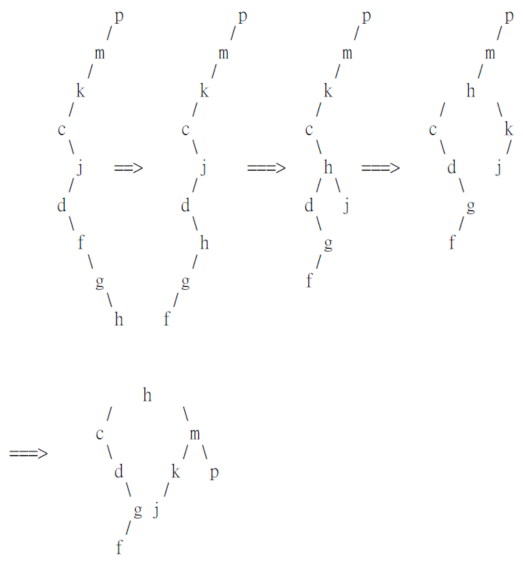
(2) (5 pts) assuming a Bottom-up splay tree.

For each of the above, show the final tree. No need to show the intermediate trees.

```
          p
         /
        m
       /
      k
     /
    c
     \
      j
     /
    d
     \
      f
       \
        g
         \
          h
```

(a)

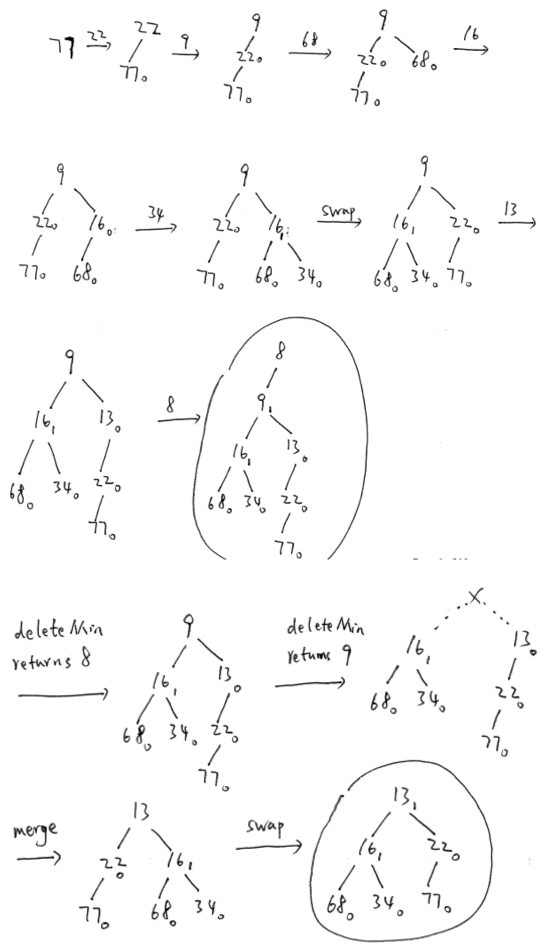




5. (10 pts) Answer the following two questions:

   (1) (7 pts) Draw the leftist min-heap after inserting **77, 22, 9, 68, 16, 34, 13, 8** in that order into an initially empty leftist heap. To receive full credit, you must show the resulting heap after each insertion is completed.

   (2) (3 pts) For the leftist tree obtained in step (1) above, perform TWO delete-min operations. Show the final heap.



6. (8 pts) Consider an initially empty hash table of size M and hash function $h(x) = x \ mod \ M$. In the worst case, what is the time complexity (in Big-oh notation) to insert n keys into the table for each of the following cases:

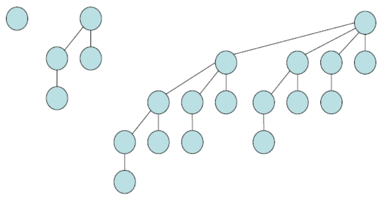
   (1) Separate chaining with each entry (bucket) of the table stores an unordered linked list, and the new element is inserted at the beginning of the list.
   Ans: n

   (2) Separate chaining with each entry (bucket) of the table stores a sorted linked list.
   Ans: n^2

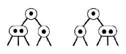
   (3) Separate chaining with each entry (bucket) of the table implemented as a splay tree.
   Ans: $n \log n$

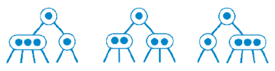
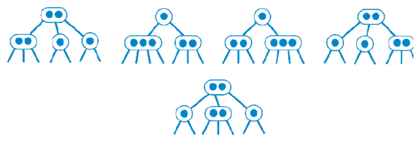
   (4) Linear probing with $n \leq M/2$.
   Ans: n^2

7. (5 pts) Draw the structure of a binomial heap with 21 nodes.



8. (10 pts) The table below lists all possible 2-3-4 tree shapes that could result from inserting N distinct keys into an initially empty tree using top-down insertion, for N between 1 and 6. The left column is the number of keys, the next column is the number of possible trees with that many keys, all if which are drawn on the right (with dots indicating the key values). Complete the two bottom rows of the table by (a) drawing the tree with 5 keys, (b) entering the count and (c) drawing the trees with 6 keys. **Solution**

1	1	
2	1	
3	1	
4	2	
5	3	(a)
6	(b)	(c)

5	3	
6	5	

9. (12 pts) Answer the following questions on red-black trees.

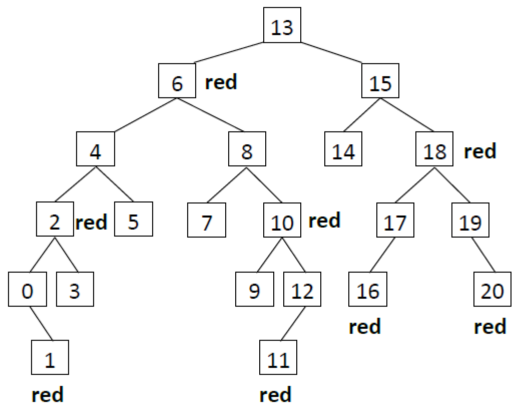
(1) (6 pts) The following tree is a red-black tree, but the colors are not shown. Also, the empty leaf nodes (i.e., external nodes) are not shown. In this question, you are asked to give all the red nodes in the tree.
Ans: 1, 2, 6, 10, 11, 16, 18, 20

(2) (6 pts) Recall that the height of a tree is the distance (i.e., number of nodes) from the root to the deepest external node (i.e., *nil*). Also, the black-height of a red-black tree is the number of black nodes along each path from the root to nil. For instance, the height of the above tree is 6.

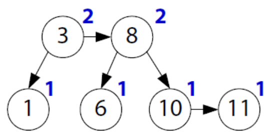
(1) What is the smallest possible number of keys in a red-black tree with black-height h?
Ans: $2^h - 1$

(2) What is the largest possible number of keys in a red-black tree with black-height h?
Ans: $4^h - 1$

(3) What is the largest possible number of red nodes in a red-black tree with black-height h?
Ans: $2(4^h - 1)/3$



10. (9 pts) Consider the following AA-tree in which each node is annotated by its level. (1)(3 pts) Draw an equivalent 2-3 tree. (2) (6 pts) Insert 9 into the AA-tree. Show intermediate steps in sufficient detail.

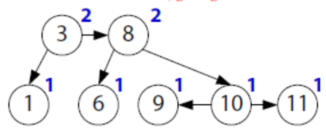


Ans:

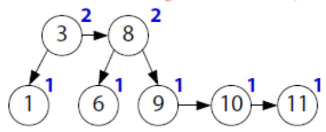
(1) The root (a 3-node) is (3, 8), with 1st child (1), 2nd child (6) and 3rd child (10, 11).

(2)

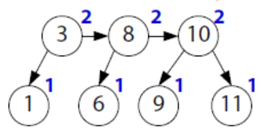
First we do BST insertion, giving the new node a level of 1:



Now 10 has the same height as its left child, so we do a skew (rotation):



Now 10 has become a "4-node", so we do a split, which lifts 10 up:



Now 3 has become a "4-node", so we do another split, which lifts 8 up: