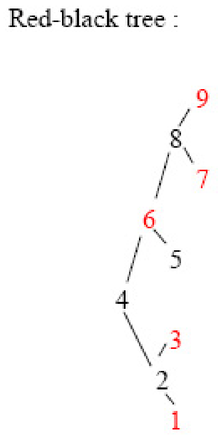
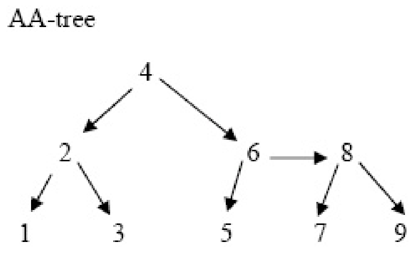
Data Structures and Programming

Final Exam (June 21, 2010)

1. (16 pts) Insert the following sequence **5 4 3 2 1 6 7 8 9** into an initially empty (a) **top-down Red-Black Tree**, and (b) **AA-tree**. Show all the intermediate steps in sufficient detail. For red-black trees, you can draw red nodes as squares and black nodes as circles.
   Solution





2. (15 pts)

   (a) (5 pts) In a Fibonacci heap, a node is marked if it looses one of its children. Consider the node x and its children $y_1, ..., y_5$ in the portion of the Fibonacci heap shown in Figure 1-(1). 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? Explain your answer.
   Solution: 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 $iV1$ 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.

   (b) In the Fibonacci heap shown in Figure 1-(2), the numbers are the key values, and $\surd$ indicates a marked node.
   (1) (4 pts) What is the potential value (based on the potential function discussed in class) of the heap? Why?
   Solution 6+2*4=14
   (2) (6 pts) Show the heap that results from performing a *delete-min* on the heap. Draw the resulting heap, and show all the marked node.
   Solution

3. (10 pts) Consider the priority queue represented by the leftist tree shown in Figure 2. Show the modified tree under each of the following operations: (a) **Insertion of the key 7.** (b) **Deletion of the minimum key.** Show your derivation in sufficient detail. (Note: The two operations are independent. Each of them starts from the original tree.)

4. (10 pts) Let the height of a tree be the number of nodes in the longest path from the root to the leaves. For each of the following types of trees, determine the minimum number of nodes a tree of height 4 can have. **(1) Binary heaps (2) Leftist trees (3) (Standard) binary search trees (4) AVL**

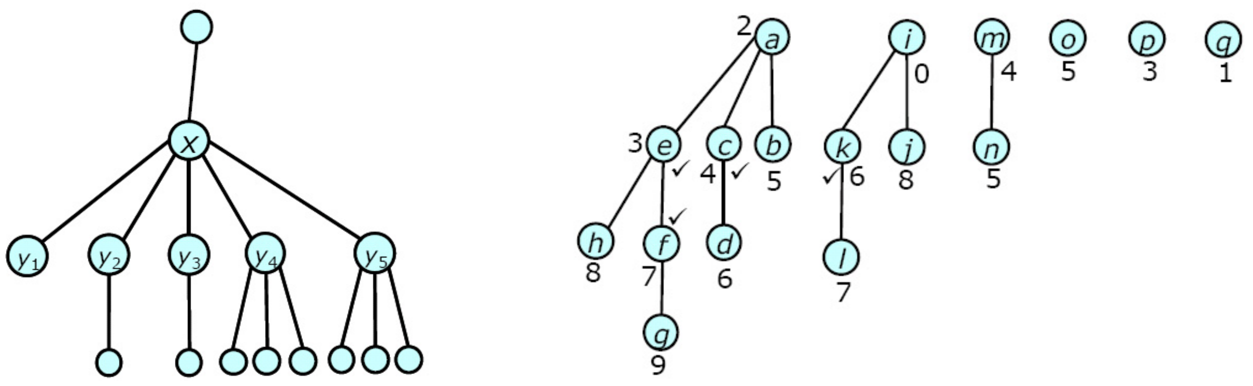
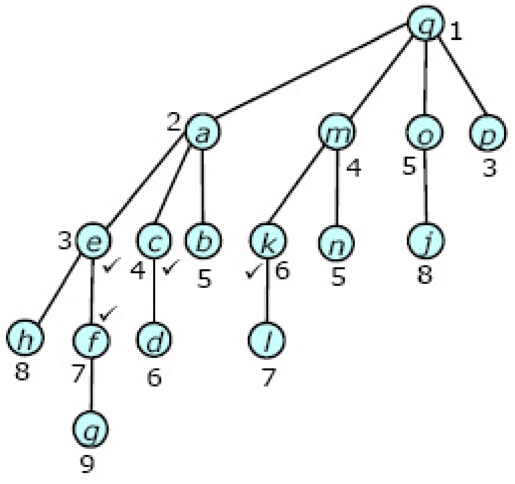


Figure 1: (1) Portion of a Fibonacci heap; (2) A Fibonacci heap



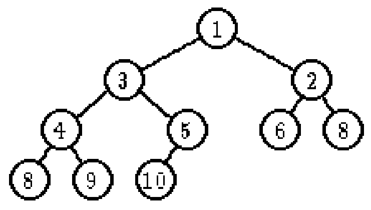
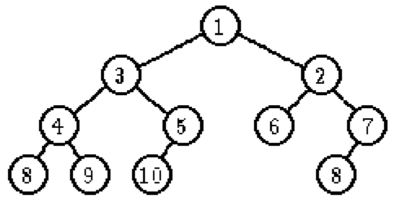
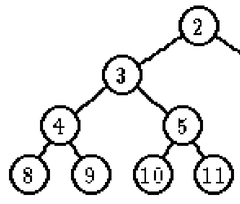


Figure 2: A leftist heap



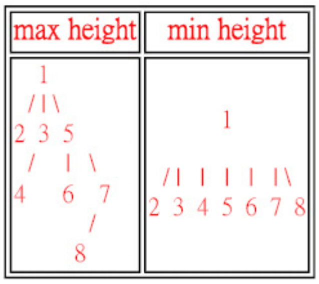


trees (5) Red-black trees
Solution: (1) 8. (2) 4. (3) 4. (4) 7. (5) 6.

5. (15 pts) Consider the union/find problem in which the union-by-size strategy is applied. Suppose we initialize our sets so that each integer between 1 and 8 (inclusive) is contained within its own set.

   (a) (5 pts) Give a sequence of seven unions that produces a tree whose height is as large as possible. Your answer should be a sequence of procedure calls of the form *union*(a, b) where a and b are integers between 1 and 8. Draw the resulting tree.
   Solution: There are lots of possible sequences that would work here. One of them is: U(1,2),

U(3,4), U(5,6), U(7,8), U(1,3), U(5,7), U(1,5).



(b) (5 pts) Give a sequence of seven unions, on the original eight sets, that produces a tree of minimum height. Draw the resulting tree.
Solution: Again, there are many correct answers. Here's one: U(1,2), U(1,3), U(1,4), U(1,5), U(1,6), U(1,7), U(1,8).

(c) (5 pts) We know that a sequence of m union/find operations on n singleton sets using union-by-size and path compression takes time $O(m \cdot \alpha(m, n))$ where $\alpha(m, n)$ grows extremely slowly. Suppose all the union operations are done first. What is the complexity of a sequence of n unions followed by m finds? Give a convincing argument.

Solution: It takes time $O(n + m)$. First recall that each Union operation takes time O(1); thus, all we have to show is that not too much time is spent on Finds. Observe that each edge of a tree has to be created by a Union operation; thus, the total number of tree edges is O(n). Suppose we want to count only edges that are deep in a tree, i.e., just those edges that are not edges from the root. There are O(n) such deep edges.

A Find operation on item i is slow only when i is deep in a tree; then it takes time proportional to the depth of i. The Find operation also converts a bunch of deep edges into root edges. As a matter of fact, the Find operation takes time proportional to a constant plus the number of deep edges converted to root edges. Since there aren't very many deep edges (just O(n) of them), the total time spent on Find operations is at most O((number of Find operations)+(number of edges converted from deep edges to root edges)) or O(m+n).

6. (8 pts) A *ternary counter* is a string of k *"trits"* $t_{k-1}...t_0$ each of which can be 0, 1, or 2. As with a binary counter, we can perform the operation INCREMENT on a ternary counter. If we start with every trit equal to 0, then after n INCREMENTs, the counter holds the number n written in base 3. For example, $k = 4$ and $n = 6$ is shown in Figure 3. The cost of each INCREMENT is the number of trits that change. Suppose we start from all 0's and perform n INCREMENTs, what is the worst-case total cost of the n operations? What is the amortized complexity of an INCREMENT? Explain why.
Solutio: Let $WCSC(n)$ be the worst case complexity of performing n INCREMENTS.

$$WCSC(n) = \sum_{i=0}^{\ell} n/3^i,$$

where ℓ is the index of the largest trit that ever becomes non-zero. Therefore,

$$WCSC(n) \leq n \sum_{i=0}^{\infty} 1/3^i = 3n/2.$$

t_3	t_2	t_1	t_0
0	0	0	0
0	0	0	1
0	0	0	2
0	0	1	0
0	0	1	1
0	0	1	2
0	0	2	0

Figure 3: A ternary counter;

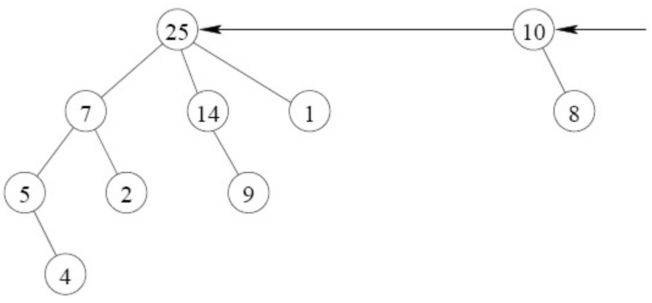
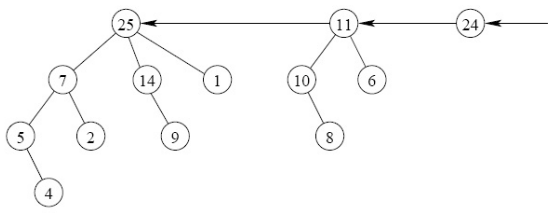


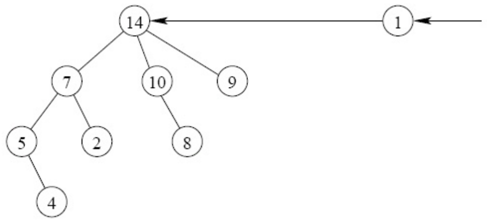
Figure 4: A max-binomial heap.

7. (14 pts) Consider the *max*-binomial heap B shown in Figure 4.

   (a) (7 pts) Insert keys 6, 11, 24 into B in the given order, and show the resulting binomial heap.

   (b) (7 pts) Perform the *delete-max* operation from the original binomial heap B and show the resulting binomial heap.





8. (12 pts) For each of the following questions, pick an answer from one of the following $\Theta(1)$, $\Theta(\log^* n)$, $\Theta(\log_2 n)$, $\Theta(n)$, $\Theta(n \log n)$, $\Theta(n^2)$, $\Theta(n \log^* n)$:

   (a) Worst-case complexity of the *decrease-key* operation of a *min*-Fibonacci heap of n nodes.
   Solution: $\Theta(n)$

   (b) The maximum height of a leftist heap of n nodes.
   Solution: $\Theta(n)$

   (c) Worst-case complexity of n union-find operations if union-by-size is applied but path compression is not used.
   Solution: $\Theta(n \log n)$

   (d) Worst-case complexity of the *union* operation of a Fibonacci heap of n nodes.
   Solution: $\Theta(1)$

   (e) The maximum number of *rotations* needed to rebalance a bottom-up red-black tree of n nodes after an insertion is carried out.
   Solution: $\Theta(1)$

   (f) The worst-case complexity of the *delete-min* operation of a *min*-leftist heap of n nodes.
   Solution: $\Theta(\log_2 n)$