資料結構 Data Structures (Fall 2004) Midterm Exam

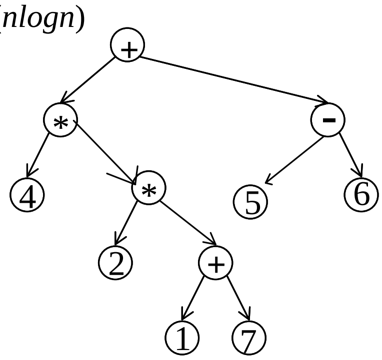
作答在答案卷上； 題目不用繳回。 答案卷上務必寫上姓名、系級、 學號

1. (10 pts) True/False questions (**Score=max{0, Right – Wrong}**, i.e., 答錯倒扣一分)

1) To have a place to store the tail and head values, it is common for a circular queue implementation to waste one table element.
2) When evaluating a prefix expression, the stack contains both operands and operators
3) Inserting an element into an n-node splay tree takes at most ($\log_2 n + 1$) comparisons.
4) The worst case complexity of searching a key in an n-node binary heap is O(n).
5) Amortized time complexity of inserting one element into a splay tree of size n is O(log n)
6) For ordinary binary search trees of size n, deleting the minimum value takes O(n) time in average (i.e., the average-case complexity is O(n) time)
7) $8n^{1.5} + 10\, n\, (\log n)^2 = O(n\, (\log n)^2)$
8) If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 2$, then $g(n) = \Theta(f(n))$
9) The *minimum* number of elements in a 2-3 tree of height 3 is 8, assuming that a tree with one node has height 1.
10) For the recurrence equation $T(n) = 2T(n/2) + n$, $T(n)$ is $\Theta(n\log n)$

2. (12 pts) Consider the expression tree on the right.

(i) (4 pts) Give the prefix expression corresponding to the tree.
(ii) (4 pts) Give the postfix expression corresponding to the tree.
(iii) (4 pts) What is the value of the tree?

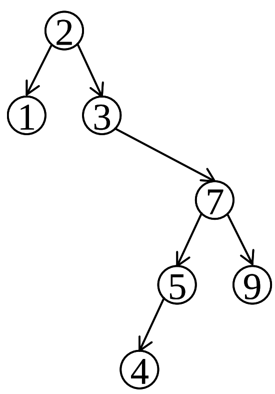


3. (10 pts) Insert the list of numbers **15, 18, 25, 20, 21, 19, 27** into an initially empty *AVL tree*. Show the tree after each insertion. Specify what kind of rotations where performed after each insertion.

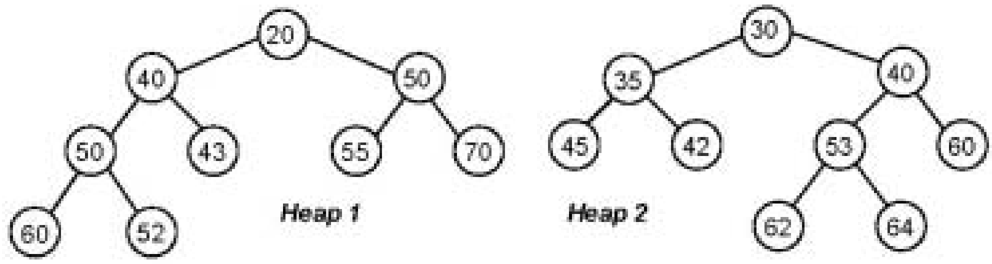
4. (10 pts) Insert the list of numbers **20, 15, 25, 10, 5, 12, 17, 18** into an initially empty *top-down Red-Black tree*. Show the tree after each insertion and specify which rotations and color changes where performed. You can draw red nodes as squares and black nodes as circles.

5. (10 pts) Insert the list of numbers **30, 10, 20, 7, 9, 22, 25** into an initially empty *AA tree*. Show the tree after each insertion and each skew or split, specifying which one you performed.

6. (10 pts) Insert 10 in a *top-down* fashion to the splay tree on the right. Show your work in detail.



7. (10 pts) Draw the resulting *leftist heap* after you merge the following two *leftist heaps*. Show your work in detail.



8. (16 pts) Consider the following 12 data structures (heaps are assumed to be min-heaps.). Answer the following questions:

1. Sorted (in increasing order) array	2. Unsorted array	3. Unsorted doubly linked list	4. Sorted (in increasing order) singly linked list
5. Simple binary search tree	6. 2-3-4 tree	7. Top-down splay tree	8. AVL tree
9. Top down Red-black tree	10. AA tree	11. Binary Heap	12. Leftist Heap

(a) List those for which a *delete-minimum* can be done in *O(log n)* time in the worst-case. (*** Notice that *O(log n)* includes *O(1)*.***)

(b) List those for which a *union* (i.e., merge) can be done in *O(log n)* time in the worst-case.

(c) List those for which *finding an arbitrary key* may require *Ω(n)* time in the worst case.

(d) List those for which performing a sequence of *insertions* can be done in *O(n log n)* time in the worst case.

9. (12 pts) In a binary search tree T, let *size[y]* be the number of keys stored in the subtree rooted at y (including y itself). Node y is said to be ***weight-balanced*** if the number of nodes in each subtree of y is at most *2/3*size[y]* (i.e., 2/3 the number of nodes in the tree rooted at y). T is said to be weight-balanced if every node in T is weight-balanced. Consider the tree below such that **adding one additional key to the subtree rooted at z will make node y no longer weight-balanced.** Suppose we now insert a key into the subtree rooted at z. Explain how to use *rotations* (single/double) to rebalance the tree so that y becomes weight-balanced again. Show in detail ***how*** and ***why*** (give a justification) your rebalancing method works. *(Hint:* consider different structures of the subtree rooted at z, and perform rotations accordingly. For convenience, you may assume that size[x], size[y], and size[z] are sufficiently large.)