CS435 Algorithms Mid-Term exam

April 2013

Name

[40 points] Fill in the blank(s) for questions 1-23. Please write clearly; your answers to these 23 **questions must be on the exam, not on a separate paper**.

1. The sum of the powers of two $(2^0+2^1+2^2+\ldots+2^n)$ is equal to _____.
2. The sum of increasing integers $(1+2+3+\ldots+n)$ is equal to _____.
3. $\log_b b^x$ is equal to _____.
4. $\log_b 1$ is equal to _____.
5. $b^{\log_b a}$ is equal to _____.
6. A logarithmic algorithm has a time complexity of $O()$.
7. A quadratic algorithm has a time complexity of $O()$.
8. A linear algorithm has a time complexity of $O()$.
9. A binary tree that has 10,000 internal nodes will have a height between ___ and ___ (Hint: $2^{13}=8{,}192$ and $2^{14}=16{,}384$).
10. A red-black tree that has 10,000 internal nodes will have a height between ___ and ___.
11. An AVL tree that has 10,000 internal nodes will have a height between ___ and ___.
12. Post-order traversal of a binary tree means the parent, left child, and right child nodes are "visited" in the following order ___, ___ and ___.
13. In-order traversal of a binary tree means the parent, left child, and right child nodes are "visited" in the following order ___, ___ and ___.
14. The maximum depth of any external node of tree T is called the ___ of T .
15. In a circular, growable array implementation of the Queue ADT, the enqueue and dequeue operations run in ___ and ___ amortized time if the array size is increased by a **constant** C each time it has to be enlarged.
16. In a growable array-based implementation of a Stack ADT, the push operation runs in O(1) amortized worst-case time when the array size increases by ___ each time resizing is necessary.
17. In a has table implementation of the Dictionary ADT, the insertItem, findElement, and removeElement operations run in ___, ___, and ___ expected time respectively.
18. In an unsorted, growable array implementation of the Dictionary ADT, the insertItem, findElement, and removeElement operations run in ___, ___, and ___ time respectively.
19. In a sorted, growable array implementation of the Dictionary ADT, the insertItem, findElement, and removeElement operations run in ___, ___, and ___ time respectively.
20. In a red-black tree implementation of the Dictionary ADT, the insertItem, findElement, and removeElement operations run in ___, ___, and ___ time respectively.

21. To sort a large sequence of keys that cannot fit in memory, ___ is the recommended choice.
22. In a real-time scenario where a sort must be completed within a fixed amount of time and the input can fit in memory, then ___ is the recommended choice.
23. To sort a sequence of less than fifty keys, ___ is the recommended choice.

For questions 24-33, circle True or False (1 point each). Giving a justification is **optional**.

24. The standard Bucket-sort distributes the keys into buckets, sorts each bucket, then concatenates the buckets. This algorithm runs in linear time no matter how the keys are distributed over the buckets.
    a. True
    b. False
25. A sorting algorithm is considered in-place if it uses some memory in addition to the memory used by the input sequence, but only a constant amount, i.e., not an amount that increases as n increases.
    a. True
    b. False
26. In a self-balancing binary search tree, to remove a key-element pair, the *removeElement(k)* method calls *expandExternal(v)*
    a. True
    b. False
27. In a binary search tree, every internal node has either one or two children.
    a. True
    b. False
28. The lower bound on sorting by key comparisons is $O(n)$ since we can always do a bucket or radix sort.
    a. True
    b. False
29. All implementations of an unordered dictionary are necessarily inefficient for finding items since the entire dictionary might have to be scanned to find the key.
    a. True
    b. False
30. In Radix-sort, the key is divided into components and Bucket-sort is run on each component, starting with the most-significant (high order) component down to the least significant component (low order).
    a. True
    b. False

31. In a Red-Black tree, the restructuring and recoloring operations are sometimes necessary when searching the tree.
    a. True
    b. False
32. Quick-sort is an example of the divide-and-conquer approach with worst-case time complexity that is no better than Selection-sort or Insertion-sort.
    a. True
    b. False
33. In a heap, external nodes cannot appear on more than one level.
    a. True
    b. False
34. [5 points] You are given an algorithm A with running time $\Theta(n^{13})$ in every case (best case, worst case, and average case) and algorithm B with running time $\Theta(n)$ in every case. Very briefly describe why A might be faster than B on some inputs. Your answer **cannot** have anything to do with memory limitations, disk accesses, or paging (i.e., memory is unbounded, disk accesses take 1 unit of time, etc.).

Algorithm Design:

35. [15 points] (a) Given a sequence of all books in a library (containing title, author, call number, and publisher) and another Sequence of 30 publishers, design an efficient algorithm to determine how many of the books were published by each publisher.
    (b) [5 points] What is the time complexity of your algorithm? Justify your answer.
36. (a) [20 points] Given a Sequence B of thousands of credit card bills and another Sequence *P* of thousands of payments, design an efficient algorithm to create a Sequence of credit card bills that were not paid in full. The elements of Sequence *B* contain the credit card number, amount due, name, and address. The elements of Sequence *P* contain the credit number, amount paid, and name. The output of the algorithm should be a newly created Sequence containing the unpaid bills, i.e., elements with the credit card number, name, address, amount due, and amount paid for those customers for which the amount paid is less than the amount due. Note that you must handle the case where there is a bill, but there is no payment.
    (b) [5 points] What is the time complexity of your algorithm? Justify your answer.
37. [20] Design an efficient algorithm to calculate the height and the balance factor of each internal node of a binary tree. The balance factor of an internal node is the height of the left subtree minus the height of the right subtree. Use the methods of the Tree and Binary Tree ADTs to traverse the tree. To set the height and balance factor of each internal node, use methods ***setHeight***(v, h) and ***setBalFactor***(v, bf) where **v** is a node of the tree, **h** is the height calculated for node **v**, and **bf** is the balance factor of **v**. For

example, in the Red-Black tree of Figure 1 below, the height of the node containing 13 is 1 and the balance factor is 0. Similarly, the height of the node containing 10 is 3 and the balance factor is 0 whereas, the height of the node containing 15 is 4 and the balance factor is +1 (since the height of its left subtree is one more than the height of its right subtree, i.e., 3-2=1).

(b) [5 points] What is the time complexity of your algorithm if the tree is a red-black tree? What if the tree is neither a red-black tree nor an AVL tree? Justify your answers.

Sequence ADT:
first(), last(), before(p), after(p), replaceElement(p, o),
swapElements(p, q), insertBefore(p, o), insertAfter(p, o),
insertFirst(o), insertLast(o), remove(p), size(), isEmpty(),
elemAtRank(r), replaceAtRank(r, o), insertAtRank(r, o), removeAtRank(r),
atRank(r), rankOf(p)
Dictionary ADT
findElement(k), insertItem(k, e), removeElement(k), items(), keys(), elements()
OrderedDictionary
findElement(k), insertItem(k, e), removeElement(k), items(),
closestKeyBefore(k), closestKeyAfter(k),
closestElemBefore(k), closestElemAfter(k)
PriorityQueue
removeMin(), minKey(), minElement(k), insertItem(k, e)
BinaryTree ADT:
root(), parent(v), children(v), leftChild(v), rightChild(v), sibling(v),
isInternal(v), isExternal(v), isRoot(v), size(), elements(), positions(),
swapElements(v, w), replaceElement(v, e)

2-4 and Red-Black Trees:

38. [5 points] Draw the corresponding 2-4 tree for the red-black tree in Figure 1.

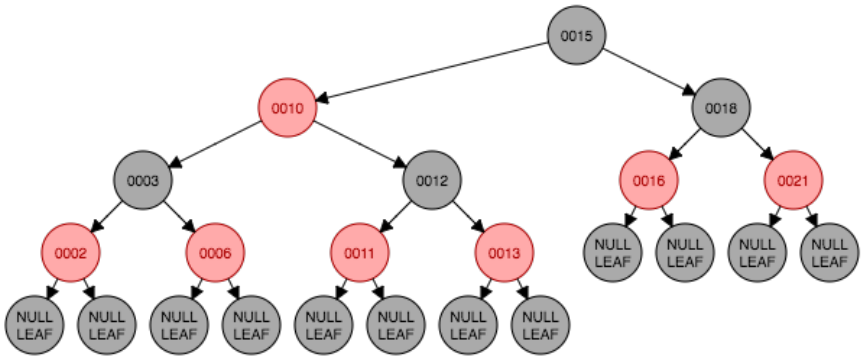

Figure 1

39. [10 points] For the red-black tree in Figure 1, insert the keys 8, 7, and 5 (in this order) and redraw the tree after any necessary re-coloring and rebalancing. Clearly label the red nodes with R. Show the tree after key 8 is inserted, another tree after 7 is inserted, and a final tree after 5 is inserted (if you show other intermediate steps, then clearly label these three trees).

Recurrence Relations:

40. For the following two questions, use the Master Method to find the closed-form solution of the following recurrence equations: (**you MUST show your work**).

$$T(n)=\begin{cases} c, \wedge if\ n<d \\ aT\left(\frac{n}{b}\right)+f(n), if\ n\geq d \end{cases}$$

Case 1. If $f(n)$ is $O(n^{\log_b a-c})$, then $T(n)$ is $\Theta(n^{\log_b a})$

Case 2. If $f(n)$ is $\Theta(n^{\log_b a}\log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a}\log^{k+1} n)$

Case 3. If $f(n)$ is $\Omega(n^{\log_b a+c})$, then $T(n)$ is $\Theta(f(n))$, provided $af\left(\frac{n}{b}\right)\leq \delta f(n)$ for some $\delta<1$.

(a) [5 points]

$$T(n)=3T\left(\frac{n}{3}\right)+bn^2$$

(b) [5 points]

$$T(n)=4T\left(\frac{n}{2}\right)+bn^2$$

41. [5 points] For recurrence equation $T(n)=8T\left(\frac{n}{2}\right)+n^2$, prove that $T(n)$ is $O(n^3)$ using the guess-and-test method (**Hint**: use as your guess (n^3-n^2) , which is $O(n^3)$).
42. [5 points] Let A be an array of n integers. What is the time complexity of the following algorithm? Give your answer as a summation that specifies the precise number of times that the statement $A[i] \leftarrow A[i]+A[j]+A[k]$ is executed.

```
for i ← 0 to n – 1 do
        for j ← 0 to i do
                for k ← 0 to i do
                        A[i] ← A[i]+A[j]+A[k]
```