Data Structures and Programming

Spring 2018, Midterm Exam. Solutions

May 1, 2018

1. (10 pts) Consider the following two functions:

$$f(n) = \frac{1}{10}n^2 \qquad g(x) = \begin{cases} n^5, n \leq 5 \\ 99n + \log n, \text{ otherwise} \end{cases}$$

Show that $f(n) = \Omega(g(n))$ is true by finding c and n_0 that satisfy the definition of big-Omega. Please show your work in sufficient detail. (Recall that $f(n) = \Omega(g(n))$ if $\exists c \in R^+, n_0 \in N$ such that $\forall n \geq n_0, f(n) \geq cg(n)$, where R^+ is the set of positive real numbers and N is the set of natural numbers.)

Solution: First, note that $g(n) = 99n + log(n)$ when $n > 5$. So, to simplify our analysis, we will show that $f(n) \in \Omega(g(n))$ specifically for $n \geq 6$.

Next, to show that the definition of big-Ω holds, we must find some c and n_0 such that $\frac{1}{10}n^2 \geq c(99n + log(n))$ is true for all $n \geq n_0$. In order to do so, we first observe that the following chain of inequalities are true:
$c(99n + log(n)) \leq c(99n + n)$ for $n \geq 1$
$c(99n + n) \leq c100n$
$c100n \leq c100n^2$ for $n \geq 1$
Therefore, we conclude that $c100n^2 \geq c(99n + log(n))$ holds for all $n \geq 1$.
Next, observe that $\frac{1}{10}n^2 \geq c100n^2$ is true when $c = \frac{1}{1000}$ and for all values of n. Therefore, if we apply the inequality we discovered previously, it must be the case that $\frac{1}{10}n^2 \geq c(99n + log(n))$ is true for the same value of c. We previously declared $g(n) = 99n + log(n)$ only when $n \geq 6$. Therefore, we conclude $f(n) \geq cg(n)$ for $c = \frac{1}{1000}$ and $n_0 = 6$.

2. (20 pts) For each of the following program fragments (a) - (e), give a Θ bound of the worst-case runtime with respect to n. You do not need to justify your answer.
Sol: (a) $\Theta(n^4)$, **(b)** $\Theta(n^2)$, **(c)** $\Theta(n^2)$, **(d)** $\Theta(2^n)$, **(e)** $\Theta(n)$

(a)
```
void f1(int n) {
  for(int i=0; i < n; i++) {
    for(int j=0; j < n; j++) {
      for(int k=0; k < n; k++) {
        for(int m=0; m < n; m++) {
          System.out.println("!");
  } } } }
}
```

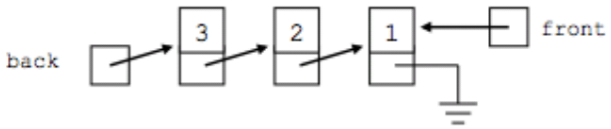
(b)
```
void f2(int n) {
  for(int i=0; i < n; i++) {
    for(int j=0; j < 10; j++) {
      for(int k=0; k < n; k++) {
        for(int m=0; m < 10; m++) {
          System.out.println("!");
  } } } }
}
```

(c)
```
int f3(int n) {
  int sum = 73;
  for(int i=0; i < n; i++) {
    for(int j=i; j >= 5; j--) {
       sum--;
    }
  }
  return sum;
}
```

(d)
```
int f4(int n) {
  if (n < 10) {
    System.out.println("!");
    return n+3;
  } else {
    return f4(n-1) + f4(n-1);
  }
}
```

(e)
```
int f5(int n) {
  if (n < 10) {
    System.out.println("!");
    return n+3;
  } else {
    return f5(n-1) + 1;
  }
}
```

3. (6 pts) Suppose that we had chosen to assign the front and back references of a queue using a singly linked list in the following way. What is the time complexity (in big-Oh) of executing the following operations to a queue of size n? Why?



(a) Insert (at back)

(b) delete(from front)

Solution (a) $O(1)$; (b) $O(n)$, as we need to know the address of the predecessor of the node to be deleted.

4. (30 pts) True or False? (Score = max {0, Right - $\frac{1}{2}$Wrong }). No explanations are needed.

(1) Binary search trees are abstract data types supporting insertion, search and deletion operations.
Sol: False

(2) $n^{1.5} = O(n \log n)$
Sol: False

(3) If $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$, then $f_1(n) \times f_2(n) = \Omega(g_1(n) \times g_2(n))$.
Sol: True

(4) A stack follows a FIFO (first-in-first-out) rule.
Sol: False

(5) In a circular doubly linked list with 10 nodes, we will need to change 4 links if we want to delete a node other than the head node.
Sol: False; We can change the link fields of the nodes that precede and follow the node we want to delete. So we just need to change 2 links if we want to delete a node other than the head node.

(6) $(a \oplus b) \oplus (c \oplus b) \oplus (c \oplus a) = 1$, where $a, b, c \in \{0, 1\}$ and $\oplus$ is the exclusive-or operator.
Sol: False

(7) It is possible to insert any batch of N keys into an initially empty binary search tree using at most $2N$ compares.
Sol: False

(8) The height of any AVL tree with N keys is always between $\log_2 N$ and $2 \log_2 N$.
Sol: True

(9) The subtree rooted at any node of an AVL tree is itself an AVL tree.
Sol: True

(10) The element at the root node of any AVL tree is the median element in the tree.
Sol: False

(11) The minimum element in any binary search tree is on a leaf node.
Sol: False

(12) The node that stores the second smallest element in any AVL tree has a left child.
Sol: False

(13) The depths of any two leaf nodes in any AVL tree differs by no more than two. (Note: The depth of a node is the number of edges from the node to the tree's root node.)
Sol: False

(14) If we traverse an AVL tree in-order, then the we visit the elements in their increasing order.
Sol: True

(15) Binary search on a sorted length-n array takes $O(n)$ time.
Sol: True

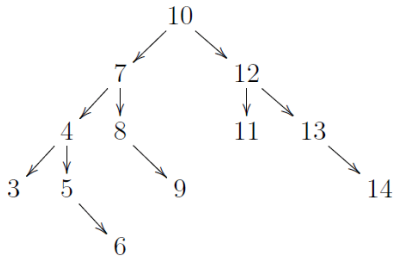
5. (10 pts) Answer the following questions.

(a) A binary search tree on n distinct elements can be uniquely reconstructed given only its *pre-order traversal*. True or False? Give a convincing argument.
Sol: True. The first number of the sequence is the root of the tree, which partitions the remaining sequences into two (sub)sequences, those smaller than the root (called LEFT) and those larger than the root (called RIGHT). Apply the same reasoning to LEFT and RIGHT recursively yields a unique binary tree.

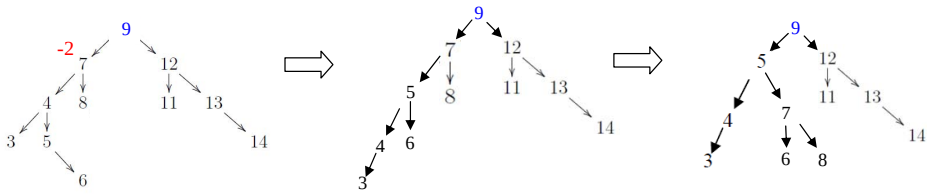
(b) A binary search tree on n distinct elements can be uniquely reconstructed given only its *in-order traversal*. True or False? Give a convincing argument.
Sol: False. Consider T_1: root 2, left child 1; T_2: root 1, right child 2. Both have $1, 2$ as their in-order sequence.

6. (8 pts) What are the postfix and prefix expressions of "$(a + b) * c - d + (e - f)/g$"? No need to give the detail of the conversion.
SOlution
Postfix: $ab + c * d - ef - g/+$
Prefix: $+ - * + abcd/ - efg$

7. (10 pts) Given the following AVL tree, delete node 10 and re-balance the tree if necessary. When deleting a node, replace it with its immediate predecessor (i.e., the largest key smaller than the key to be deleted). Show your steps in sufficient detail.



Solution



8. (6 pts) Insertion of a new element into an AVL tree may result in a violation of the balance condition of AVL trees. In such a case a (single or double) rotation restores the balance condition. According to the lectures, a rotation (single or double) is performed on the first node on the path from the inserted element to the root that is in violation of the AVL balance condition. Give an example to show that if we would instead choose the first node on the path from the root to the inserted element that is in violation of the AVL balance condition to carry out a rotation (single or double), the entire AVL tree may not be balanced after the rotation.
Solution