Data Structures

Fall 2019, Midterm Exam. (Solutions)

Nov. 11, 2019

1. (12 pts) **Asymptotic notations**
Let A be an algorithm, $T(n)$ be the worst-case time complexity of A on inputs of length n, and $g(n)$ a function, $g : N \to N$. Fill in the missing parts of each of the following statements, i.e., blanks (1)-(6), with "$\exists$" (i.e., *there exists*), "$\forall$" (i.e., *for all*), "*at least*" or "*at most*" so that they hold:

   - $T(n)$ is $O(g(n))$ if and only if $\exists c > 0$, $\exists n_0 > 0$, ... (1) ... $n \geq n_0$, ... (2) ... x of size n, A takes ... (3) ... $c \cdot g(n)$ steps when it is executed on input x.
   - $T(n)$ is $\Omega(g(n))$ if and only if ... (4) ... $c > 0$, $\exists n_0 > 0$, $\forall n \geq n_0$, ... (5) ... x of size n, A takes ... (6) ... $c \cdot g(n)$ steps when it is executed on input x.

   Solution: (1) $\forall$ (2) $\forall$ (3) at most (4) $\exists$ (5) $\exists$ (6) at least

2. (12 pts) **Time analysis of programs**
Consider the following four pieces of code, what are the running times of $f1, f2, f3, f4$ (in $\Theta()$)? No need to give explanations.

```
void f1 (int n) {
    int x = 2;
    int y = 1;
    while (y ≤ n) {
        y = y + x;
        x = x + 1;
}   }

void f2 (int n) {
    int x = 2;
    int y = 1;
    while (y ≤ n) {
        y = y + x;
        x = 2*x;
}   }

void f3 (int n) {
    int x = 2:
    int y = 1;
    while (y ≤ n) {
        y = y*x;
        x = 2*x;
}   }

void f4 (int n) {
    int x = 2:
    int y = 1;
    while (y ≤ n) {
        y = y*x;
        x = x*x;
}   }
```

head--->

Index	Key	Next
[0]	10	[6]
[1]	N/A	N/A
[2]	87	[8]
[3]	9	NULL(-1)
[4]	31	[3]
[5]	10	[2]
[6]	8	[5]
[7]	N/A	N/A
[8]	90	[4]

Linked list

Solution: $f1 : O(\sqrt{n})$ $f2 : O(\log n)$ $f3 : O(\sqrt{\log n})$ $f4 : O(\log \log n)$

Note:

- For $f3$, $y = 2^{g(k)}$, where $g(k) = g(k-1) + (k-1), k > 1$; $g(1) = 0$. $g(k) = 2^{k^2}$. Let $g(k) = 2^{k^2} = n$, $k = \sqrt{\log n}$.
- For $f4$, $y = 2^{h(k)}$, where $h(k) = h(k-1) + 2^{k-1}, k > 1$; $h(1) = 0$. $h(k) = 2^{2^k}$. Let $h(k) = 2^{2^k} = n$, $k = \log \log n$.

3. (10 pts) **Array, List**
A singly linked list is stored in an array as shown in the right figure above. Each array element has 2 fields, which are the *Key* value and the *Next* node index (N/A means no value). The Next value indicates the index of the next node (in the list) stored in this array. The head of the list is shown above at index [0] which is the 1st node. Answer the following questions.

   (a) (2 pts) What is the index of the tail node of this singly linked list?
   Solution: [3]

   (b) (2 pts) What is the index of the 3rd node in this list?
   Solution: [5]

   (c) (6 pts) A new node is inserted between the 3rd and the 4th node of this list and the content is stored in the [7] index of the array (the key value is 999). List ALL changed parts of this array. (For example, if you think [6]Key

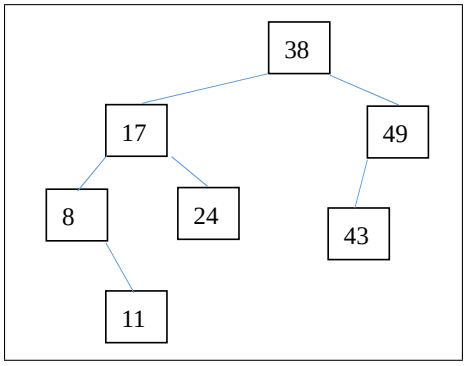
is changed to "10", write [6]Key=10. You only have to write down the changes.)

head--->

Index	Key	Next
[0]	10	[6]
[1]	N/A	N/A
[2]	87	[8]
[3]	9	NULL(-1)
[4]	31	[3]
[5]	10	[7]
[6]	8	[5]
[7]	999	[2]
[8]	90	[4]

Solution:

4. (5 pts) **Tree Traversal**
Below is a post-order traversal of a binary search tree: 11 8 24 17 43 49 38. Can you uniquely determine the tree? Why?
Solution: Yes. In the given example, 38 is the root and the left (resp., right) subtree contains 11 8 24 17 (resp., 43, 49). Repeat the above for the left and right subtrees will yield the following:



5. (10 pts) **Balanced trees**

   (a) (6 pts) Consider a binary tree in which the number of nodes in each left subtree is within a factor of 2 of the number of nodes in the right subtree. More precisely, for each node in the tree, $\frac{1}{2}$*size(right subtree)* $\leq$ *size(left subtree)* $\leq 2 \times$ *size(right subtree)*. Let $N(h)$ be the minimum number of nodes contained in a tree with height h. Write a recurrence relation for $N(h)$, just like what we did in the AVL tree case. What is the worst-case height of such a tree of n nodes? Give your answer in $O()$, and explain why.
   Solution: $N(h) = 1 + N(h-1) + 1/2N(h-1) = 1 + 3/2N(h-1) \Rightarrow$ height $O(\log n)$.

   (b) (4 pts) Consider a binary tree in which the number of leaves (nodes with no children) in each left subtree is within one of the number of leaves in the corresponding right subtree. More precisely, for each node in the tree, |*number-of-leaves(left subtree) - number-of-leaves(right subtree)*| ≤ 1. What is the worst-case height of such a tree of n nodes? Give your answer in $O()$, and explain why.
   Solution: Consider a tree of n nodes, where node 1 is the root, and node $i > 1$ is the child of node $i - 1$. For each node, the left subtree has one leaf, whereas the right subtree has zero. This meets the balancing condition. The height of the tree is n.

6. (24 pts) **AVL trees**

   (a) What is the balance factor for the following nodes in the AVL tree shown in the left figure below: 14, 21, 35, and 41? Note that balance factor = height(right child)- height(left child).
   Solution Balance(14)=-1 Balance(21)= 1 Balance(35)=0 Balance(41)=0

   (b) Show the AVL tree after inserting 3 into the provided tree.

   (c) Show the AVL tree after inserting 36 into your answer from (b).

   (d) Show the AVL tree after inserting 70 into your answer from (c).

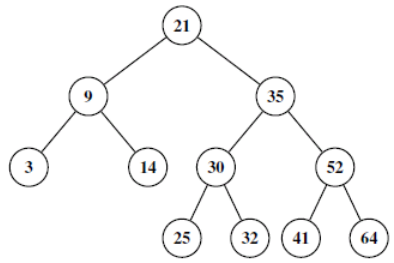
   (e) Show the AVL tree after deleting 52 from your answer from (d). Your deletion should use the immediate predecessor for replacement.

(f) Show the AVL tree after deleting 36 from your answer from (e). Your deletion should use the immediate predecessor for replacement.

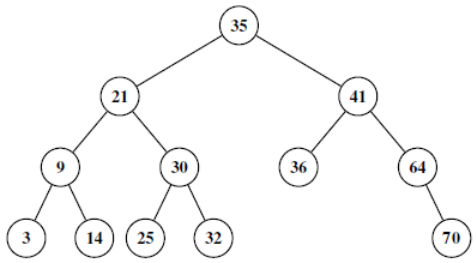
For each of (b)-(f) above, you only have to show the resulting tree; no need to show the derivation details.

Solution

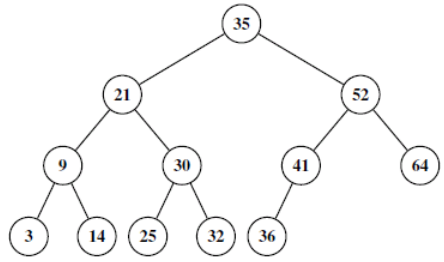
(b)



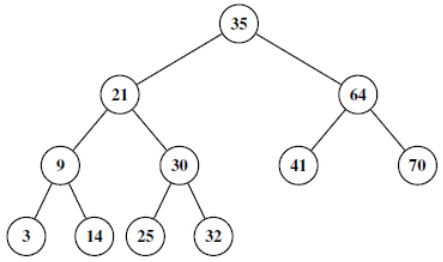
(e)



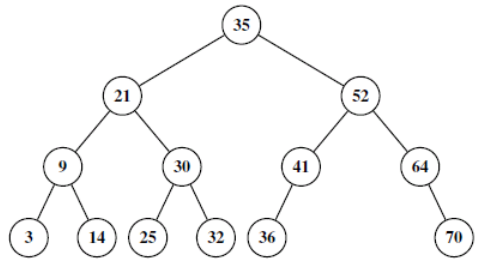
(c)

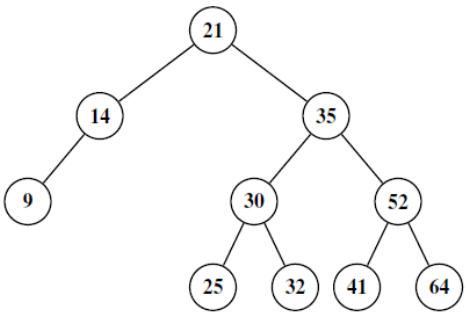


(f)

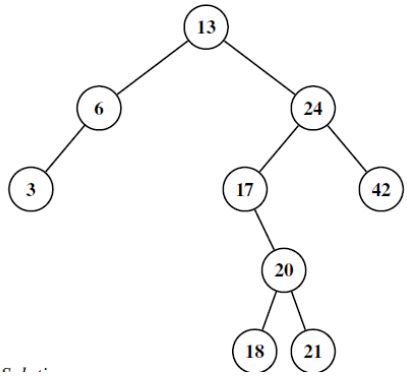


(d)





AVL tree



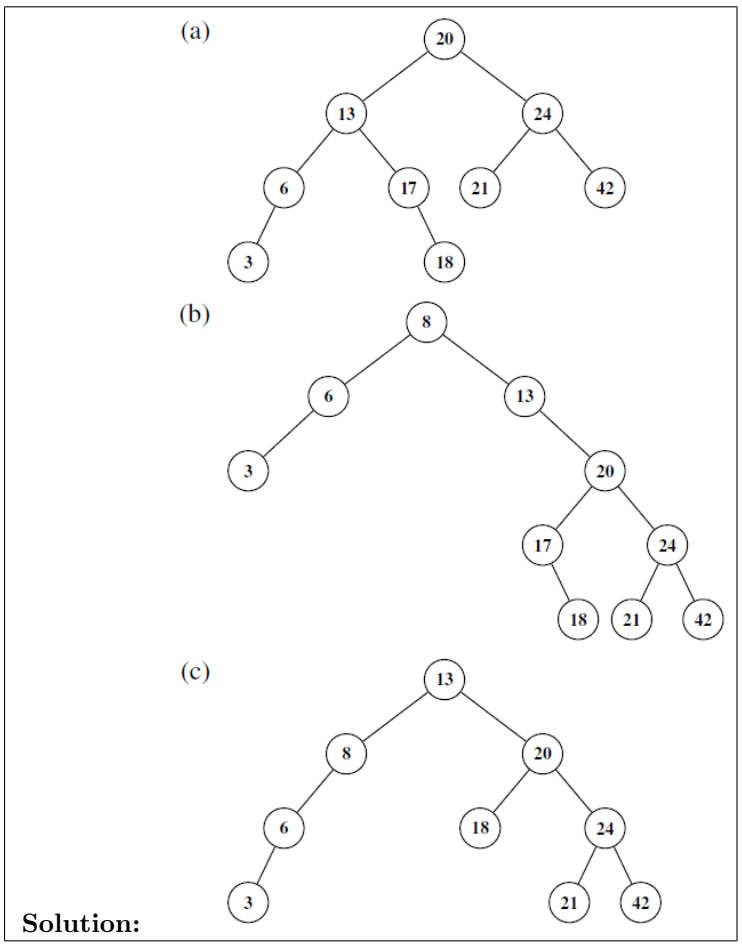
Splay tree

7. (15 pts) **Splay Trees**
Consider the (bottom-up) splay tree shown in the right figure above.

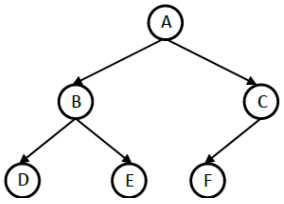
   (a) Show the splay tree after a call to find(20). Use 2-pass splaying (i.e., bottom-up splaying).

   (b) Using your answer from (a), show the splay tree after a call to insert(8). You should use the find/split approach discussed in class for inserting. That is, perform a splay at the parent of the location to be inserted first.

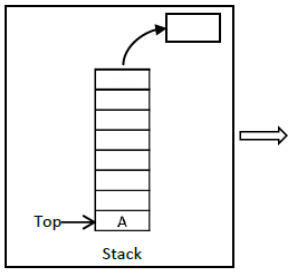
   (c) Using your answer (c), show the splay tree after a call to delete(17). Your should use the find/join approach discussed in class for deletion and use the max from the left subtree for replacement. That is, perform a splay at 17 first.



Solution:



Binary tree

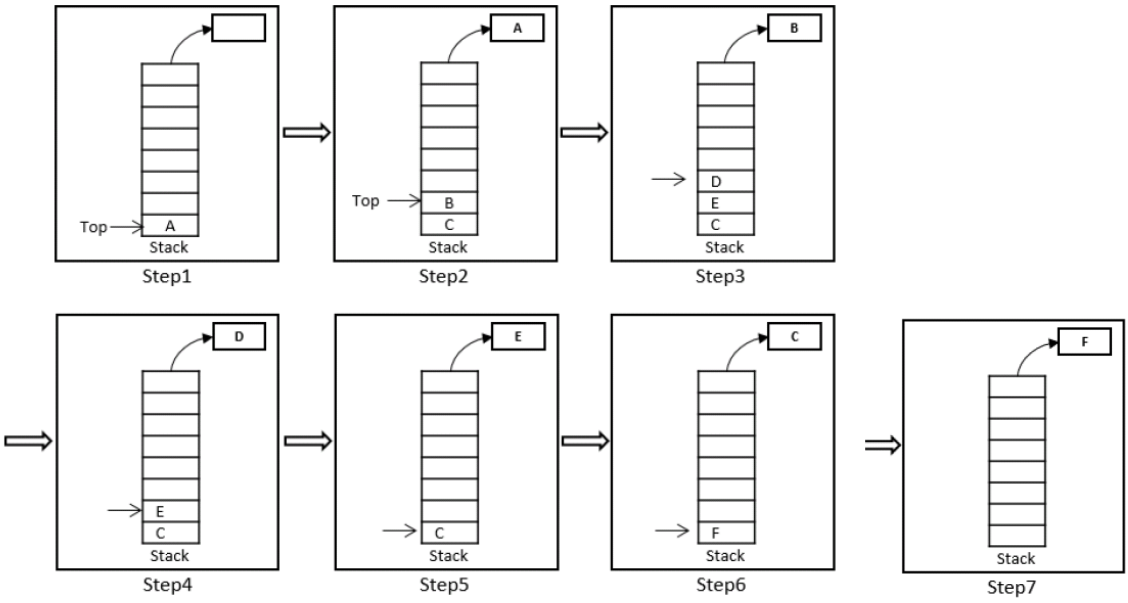


Step1

Stack in progress

8. (12 pts) Consider a *preorder traversal* of the given binary tree using a stack. Whenever an element (name of a node) is popped up from the stack, the element will be printed out and some elements will be pushed onto the stack (if not NULL). Show the stack contents and write the popped element in the top box in the subsequent Steps 2-7 in a way similar to Step 1. Note that in each of Steps 2-7, one and only one element will be popped up and printed; however, there might be more than one element pushed onto the stack in each step.

Solution


Top
A
Stack
Step1
A
Top
B
C
Stack
Step2
B
D
E
C
Stack
Step3
D
E
C
Stack
Step4
E
C
Stack
Step5
C
F
Stack
Step6
F
Stack
Step7