Data Structures and Programming

Spring 2019, Midterm Exam.

April 16, 2019

1. (15 pts) For each of the three methods below (foo(), boo(), and goo()),

   (a) (9 pts) what is the **exact runtime** of each method (in **steps**) in terms of **A, B, C**, and **N**? You should not count loop related operations (e.g., declaring, incrementing and comparing i or j). Be sure to give the exact time; do not use asymptotic notations such as O or Θ.
   Sol.
   - $foo() : A + BN + CN^2$
   - $boo() : A + BN + C(N(N+1)/2)$
   - $goo() : A + BN + 100CN$

   (b) (3 pts) For $N = 10$, which is the fastest algorithm? Please show work.
   Sol. $boo()$
   - $foo() : 100C + 10B + A$
   - $boo() : 55C + 10B + A$ (Fastest Algorithm)
   - $goo() : 100C + 10B + A$

   (c) (3 pts) As N grows infinitely large, which is the fastest algorithm? Why?
   Sol. $goo()$
   - $foo() : O(N^2)$
   - $boo() : O(N^2)$
   - $goo() : O(N)$ (Fastest Algorithm)

```
foo(int N)
    do stuff in A steps
    for i = 1..N
        do stuff in B steps
        for j = 1.. N
            do stuff in C steps
```

```
boo(int N)
    do stuff in A steps
    for i = 1..N
        do stuff in B step
        for j = 1.. i
            do stuff in C steps
```

```
goo(int N)
    do stuff in A steps
    for i = 1..N
        do stuff in B steps
        for j = 1.. 100
            do stuff in C step
```

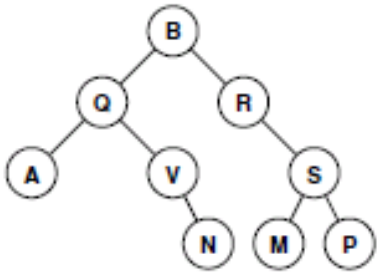
2. (11 pts) For each functions $f(n)$ along the left side of the table below and each function $g(n)$ across the top, write O, Ω, or Θ in the appropriate box depending on whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. If there is more than one relation between $f(n)$ and $g(n)$, write only the strongest one. The first entry is a demo solution.

$f(n) \mid g(n)$	$\log n$	$n \log^2 n$	$4^{\sqrt{n}}$
n^2	Ω	Ω	O
$n \log^4 n$	Ω	Ω	O
2^n	Ω	Ω	Ω
$(\frac{\log n}{n})^{10}$	O	O	O

3. (14 pts) Tree traversals.

   (a) (6 pts) Draw a binary tree (not necessarily a binary search tree) that would produce both of the following traversals. Note that the tree has 9 nodes. No need to show the detailed derivation.

   - In-order: AQVNBRMSP
   - Pre-order: BQAVNRSMP

   Sol.

   (b) (8 pts) If you know that a tree is a binary search tree, which of the following is or is not always sufficient to uniquely reconstruct it? For each one, write "yes" if it is enough to reconstruct the tree, or "no" if it is not. No need to justify your answers.

   (1) Pre-order traversal
   Sol. Yes

   (2) In-order traversal
   Sol. No

   (3) Post-order traversal
   Sol. Yes

   (4) Level-order traversal
   Sol. Yes

4. (10 pts) Consider stack and queue ADTs. By reading $0, 1, 2, 3, 4, 5, 6, 7$ in the given order (i.e., in ascending order), can the sequence $1, 2, 0, 6, 3, 4, 5, 7$ be printed using

   (a) a stack? Why?
   Sol. If the element popped from the stack can be printed directly without storing it in x first, then the sequence can be generated; otherwise (i.e., x has to be used to store an element popped from the stack), when 6 is printed, 3, 4, 5 must be in the stack with 5 at the top, followed by 4 then 3. As a result, when 3 is printed, 4, 5 must have been printed already.

   (b) a queue? Why?
   Sol. Yes. Insert 0, ..., 7 into the queue; circulate the data around the queue until the number to be printed is at the front.

   Only one temporary variable x can be used to hold the number popped from a stack (or dequeued from a queue). The value of x can be printed, if needed. Note that once an input is read, either the input is printed immediately, or it is stored in the stack (or queue).

5. (10 pts) Convert the following expression from Postfix notation: $H\ A\ B\ C\ +\ D\ *\ +\ F\ *\ G\ *\ E\ *\ *\ J\ +$

   (a) to Infix notation (full-bracketed expression):
   Sol. $((H * ((((A + ((B + C) * D)) * F) * G) * E)) + J)$

   (b) to Prefix notation:
   Sol. $+ * H * * * + A * + BCDFGEJ$

   Just show the results. No need to show the derivations.

6. (10 pts) Let $a = a_0, a_1, \ldots, a_{N-1}$ be an array of length N. An array b is a *circular shift* of array a if it consists of the subarray $a_k, a_{k+1}, \ldots, a_{N-1}$ followed by the subarray $a_0, a_1, \ldots, a_{k-1}$ for some integer k. In the example below, b is a circular shift of a (with $k = 7$ and $N = 10$).

   Suppose that you are given an array b that is a circular shift of some sorted array. Assume that the array b consists of N comparable keys (i.e., any two keys can be compared under the $<$ operation), no two of which are equal.

sorted array a[]

0	1	2	3	4	5	6	7	8	9
1	2	3	5	6	8	9	34	55	89

circular shift b[]

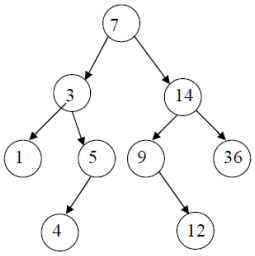
0	1	2	3	4	5	6	7	8	9
34	55	89	1	2	3	5	6	8	9

Design an efficient algorithm to find the k above. The order of growth of the running time of your algorithm should be $\log_2 N$ (or better) in the worst case, where N is the length of the array.

Sol. Use a version of binary search to find the index r of the smallest key. Notice that r is the unique index for which $b[r-1] > b[r]$. Initially, we set $lo = 0$ and $hi = N - 1$. If $b[lo] < b[hi]$, then $k = N$, indicating that there is no shift; otherwise, we maintain the invariant that $b[lo] > b[hi]$. Now, pick $mid = (lo + hi)/2$. There are three cases:

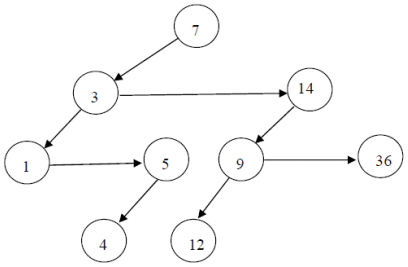
- if $hi = lo + 1$, return hi
- else if $b[mid] < b[hi]$, then set $hi = mid$
- else if $b[mid] > b[hi]$, then set $lo = mid$

7. (15 pts) Consider the following tree.



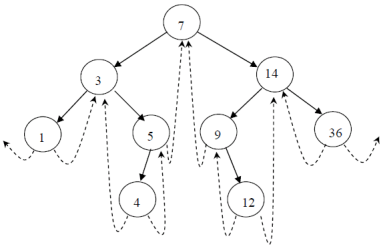
(a) Suppose the tree is regarded as an ordered tree, draw its binary tree representation (i.e., Left Child-Right Sibling Representation).

Sol.

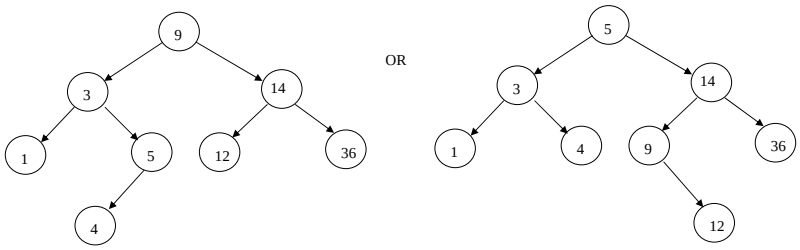


(b) Add threads to the tree to make it a threaded binary tree.

Sol.



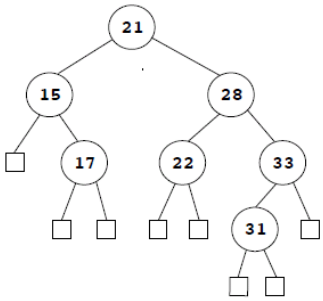
(c) Suppose the tree is a binary search tree. Draw what the tree would look like after deleting the value 7. Use one of the methods for deleting described in class or in the book. No need to rebalance the tree.

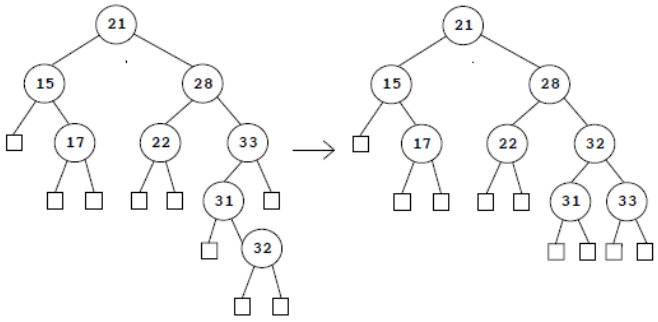


Sol.

8. (15 pts) AVL Tree

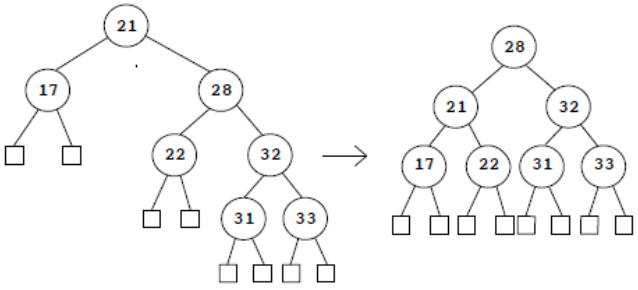
   (a) (7 pts) For the AVL tree shown below, draw the resulting tree after inserting 32 into the following tree. Please re-draw the tree each time the tree is restructured (i.e., show all the intermediate trees).





Sol.

   (b) (8 pts) Using the resulting AVL tree from (a) above, draw the resulting tree after removing 15. Please re-draw the tree each time the tree is restructured.



Sol.