University of Waterloo
Department of Electrical and Computer Engineering
ECE250
Algorithms and Data Structures
Fall 2017

Midterm Examination

Instructor: Dr. Ladan Tahvildari, PEng, SMIEEE
Date: Wednesday, October 25, 2017, 8:45 p.m.
Location: RCH-301 and RCH-302
Duration: 75 minutes
Type: Closed Book

Instructions:

- There are 4 questions. Answer all 4 questions.
- Standard calculator allowed but no additional materials allowed.
- The number in brackets denotes the relative weight of the question (out of 100).
- If information appears to be missing from a question, make a reasonable assumption, state it and proceed.
- Write your answers directly on the sheets.
- If the space to answer a question is not sufficient, use overflow page.
- When writing an algorithm, you may use any mixture of pseudocode/C++ constructs as long as the meaning is clear.

Name	Student ID

Question	Mark	Max	Marker
1	A: B:	30	A: B:
2		15	
3	A: B: C:	35	A: B: C:
4		20	
Total		100	

Question 1: Algorithm Analysis [30]

Part A [15].

Remember the *MERGE* procedure discussed in the lecture. Briefly speaking a call to $MERGE(A,p,q,r)$ merges two sorted sub-arrays $A[p..q]$ and $A[q+1..r]$ into a sorted sub-array $A[p..r]$, using linear time. We use this procedure in the following sorting algorithm:

$SORT(A:array, p:\text{int}, r:\text{int})$

if $p \geq r$	→ *O(1)*
then return	
$m \leftarrow \left\lfloor \frac{r-p-1}{3} \right\rfloor$	→ *O(1)*
$SORT(A, p, p+m)$	→ *The first 1/3 of the array*
$SORT(A, p+m+1, r-m-1)$	→ *The second 1/3 of the array*
$SORT(A, r-m, r)$	→ *The last 1/3 of the array*
$MERGE(A, p, p+m, r-m-1)$	→ *Linear time*
$MERGE(A, p, r-m-1, r)$	→ *Linear time*

Write a recurrence for the worst-case running time of the *SORT* algorithm and find a tight asymptotic bound on the worst-case running time.

$$T(n) = 3T(n/3) + \mathrm{O}(n)$$

$$a = 3, \qquad b = 3, \qquad n^{\log_b a} = n$$

$$f(n) = n = \Theta(n)$$

$$T(n) = \Theta(n \lg n) \qquad MasterMethod; Case 2$$

Part B. [15]

Give asymptotic upper and lower bounds for the following recurrence. Assume that $T(n)$ is constant for sufficiently small n. Make your bounds as tight as possible, and justify your answer.

$$T(n) = 3T(\frac{n}{4}) + n^2 \lg n$$

$$a = 3, b = 4, f(n) = n^2 lgn$$

$$n^{log_b^a} = n^{log_4^3} = O(n^{0.793})$$

$$f(n) = \Omega\left(n^{log_4^3 + \epsilon}\right) \qquad where\ \ 0 < \epsilon \leq 1.207$$

Master Method; Case 3

Regularity condition should hold for $f(n)$

$$af\left(\frac{n}{b}\right) \leq cf(n) \qquad \text{for some constant } c < 1$$

$$3\left(\frac{n}{4}\right)^2 \times \lg\left(\frac{n}{4}\right) \leq cn^2 lgn$$

$$\frac{3}{16}\lg\left(\frac{n}{4}\right) \leq clgn \qquad \frac{3}{16} \leq c < 1$$

$$T(n) = \Theta(n^2 lgn)$$

Question 2: Elementary Data Structures [15]

Propose a stack data structure that supports the following three operations:

a) **Push (Stack s, x):** This operation adds an item x to stack s
b) **Pop (Stack s):** This operation removes the top item from the stack s
c) **Concatenate (Stack s1, Stack s2):** This operation combines two stacks; the content of stack s2 goes on the top of stack s1

Time complexity of all above operations should be $O(1)$.

*We can use "**a linked list"** with two pointers:*

- *One pointer to the first node; also used as "Top" when elements are added to or removed from the stack.*
- *The other pointer is needed for the last node so that we can quickly merge the linked list of s2 on top of s1.*

Here are all the three operations:

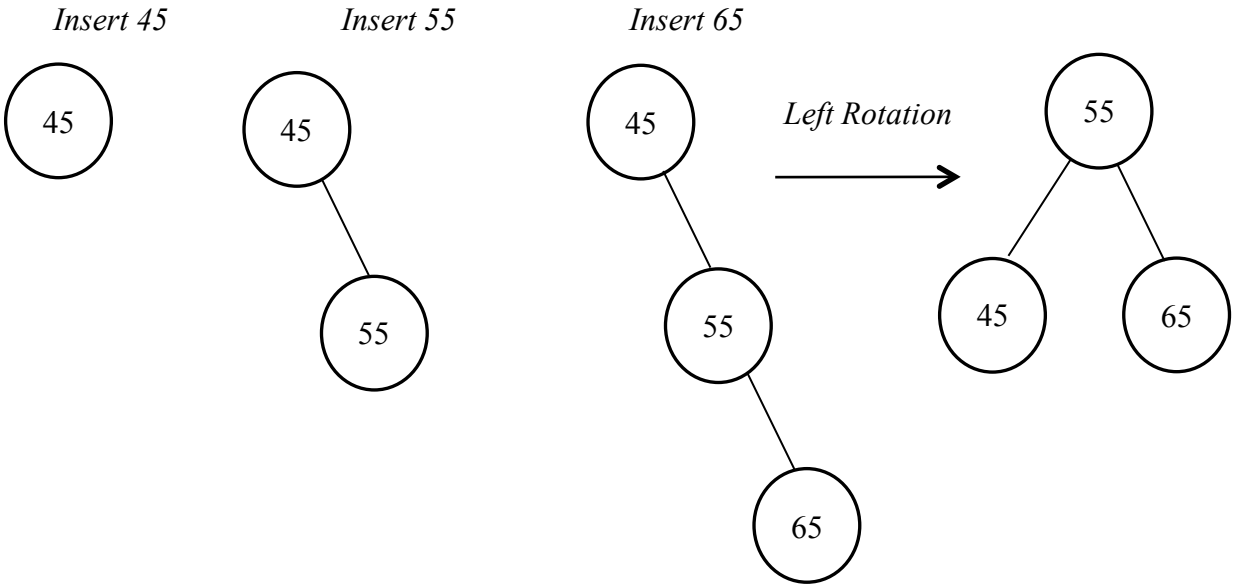
a) ***Push:** Adds the new item at the beginning of the linked list using the first pointer which takes O(1).*
b) ***Pop:** Removes an item from the beginning using the first pointer which takes O(1).*
c) ***Concatenate:** Links the last pointer of the second stack (s2) as the next of the first pointer of the first stack (s1). This takes O(1).*

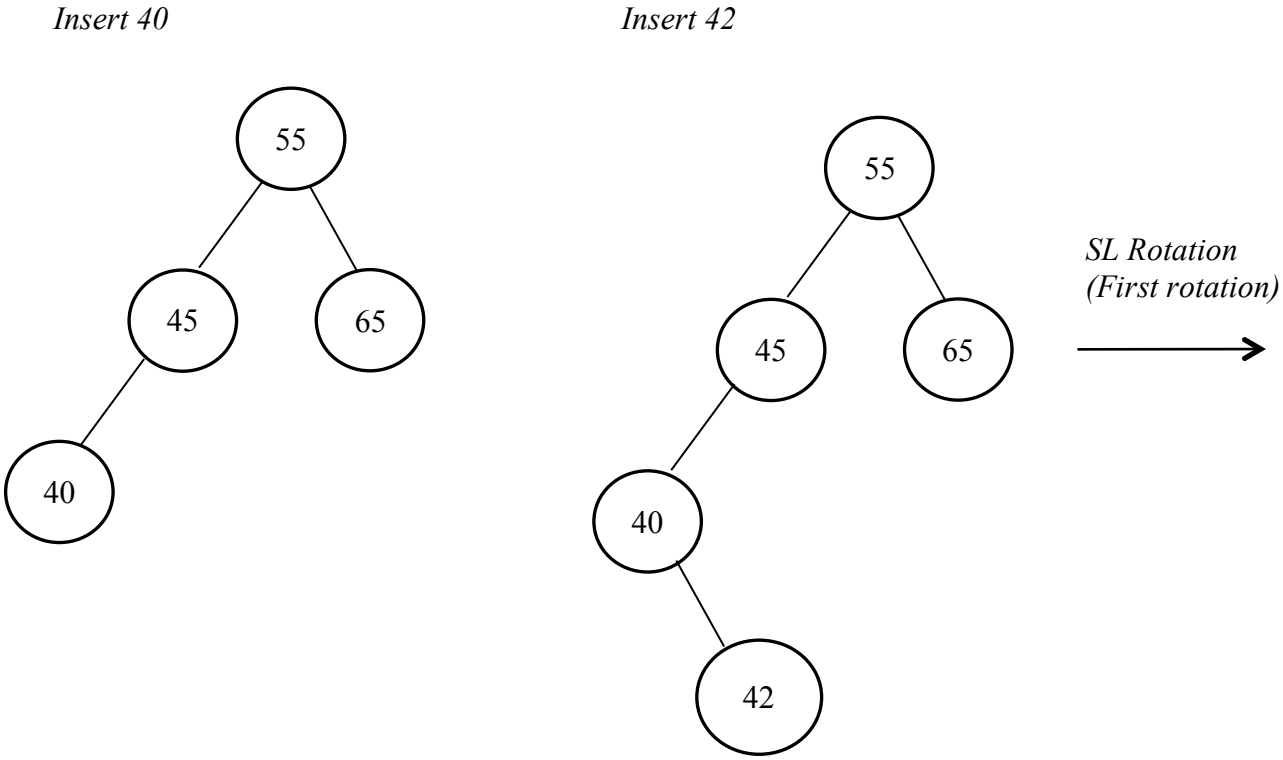
***NOTE:** If we use **"array"** implementation of stack, then **Concatenate** is not possible to do in O(1) time as we have to create a new array for s1 with the size equal to the size of the old array for s1 plus the size of s2, and then copy the old contents of s1 and s2 to the new array for s1. These operations take O(n) time.*

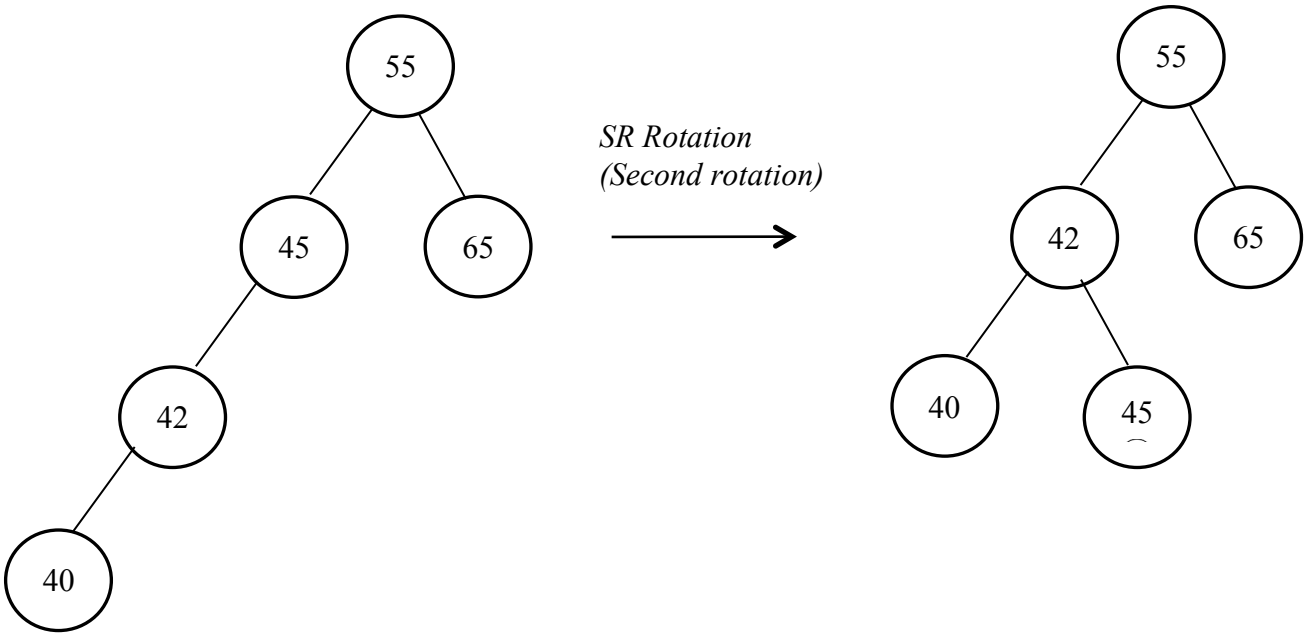
Question 3: Trees and Tree Traversals [35]

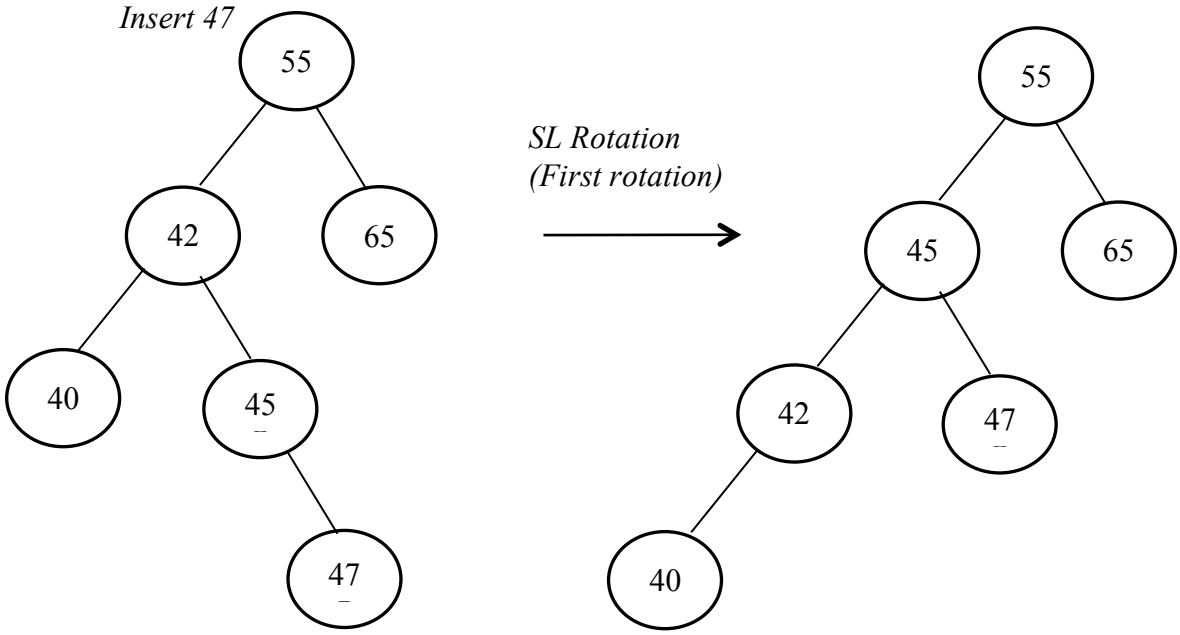
Part A. [10]

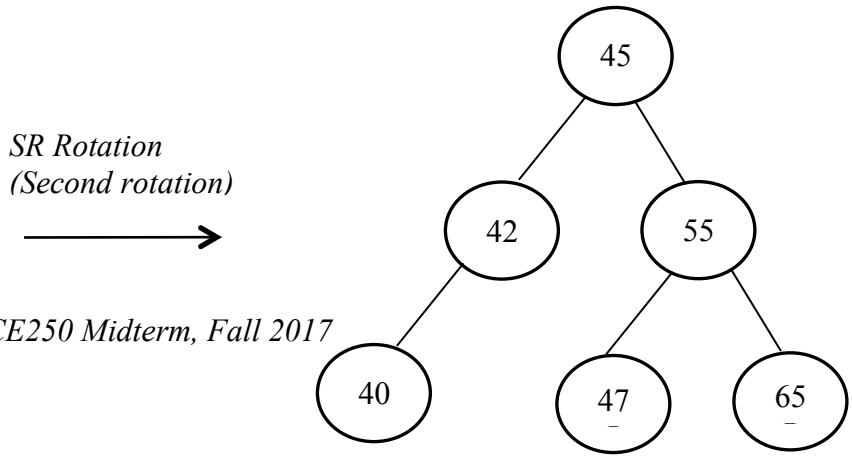
Start with an empty AVL tree and insert the following keys in the given order: 45, 55, 65, 40, 42, and 47. Draw the trees following each insertion, and also after each rotation. **Specify the rotation types.**





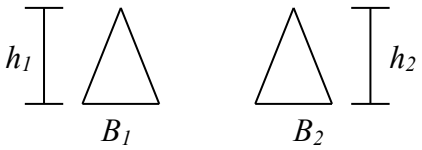

55
45
65
42
40
SR Rotation
(Second rotation)
55
42
65
40
45


Insert 47
55
42
65
40
45
47
SL Rotation
(First rotation)
55
45
65
42
47
40


SR Rotation
(Second rotation)
45
42
55
40
47
65

Part B. [12]

Let B_1 and B_2 be two binary search trees (BST) that together store keys $k_1, k_2, ..., k_n$. Suppose we know that every key in B_1 is smaller than every key in B_2 (according to some comparison operator). Write an algorithm that merges B_1 and B_2 into a single BST in $O(\min\{h_1, h_2\})$ time, where h_1 and h_2 are the heights of B_1 and B_2, respectively. Pointers b_1 and b_2 to the roots of each BST are given.



Find largest in B_1 or smallest in B_2 (call it node x):

while not at bottom of a tree
go right in B_1
go left in B_2
end while

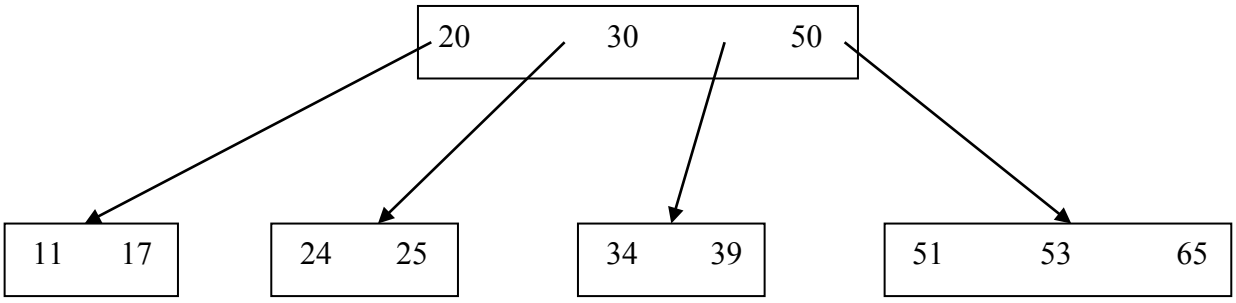
O(min{h_1, h_2})

delete node x — *O(1)*

move x to root of new tree with B_1 on left and B_2 on right — *O(1)*

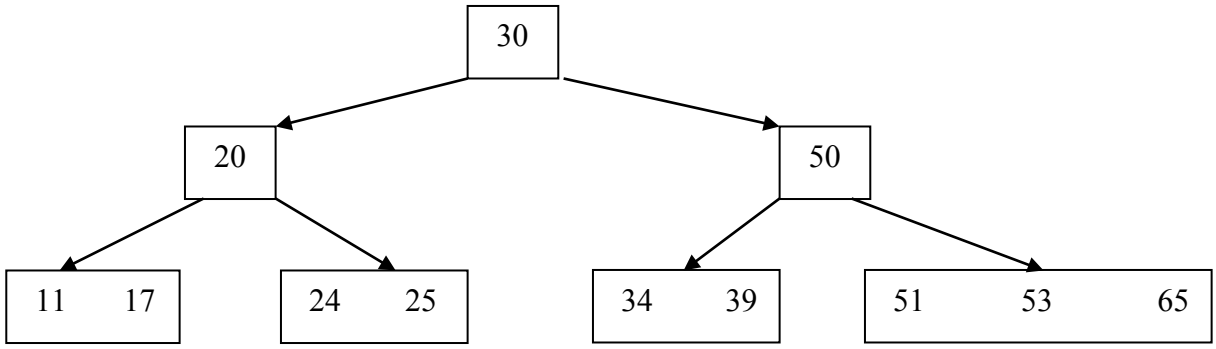
Part C. [13]

Consider the following B-Tree of order t=2.

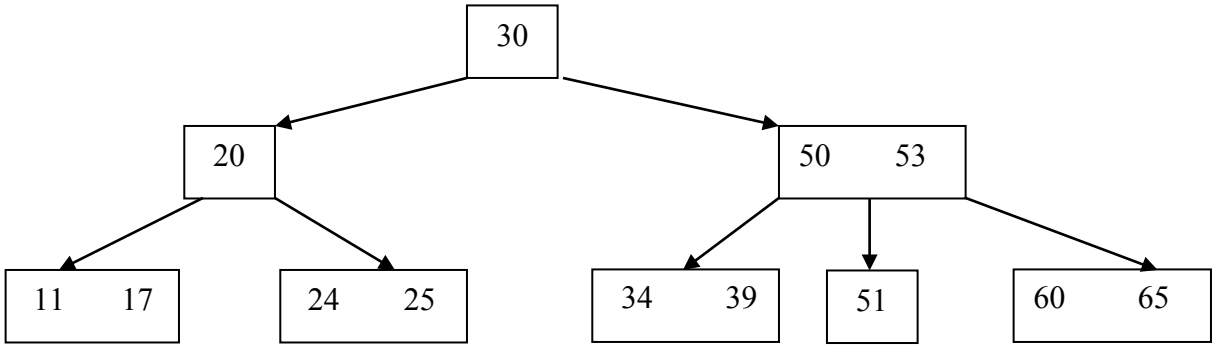


Draw the tree after inserting 60. **Show all your work.**

To insert 60, first we need to split the root as it is full.



Key 60 should be inserted to the node (51 53 65) but it is already full. Need to split the node first and then insert 60.



Question 4: Hashing [20]

Consider a hash table of size 11. Suppose the hash function uses division method. Insert, in the given order, keys: 10, 22, 31, 4, 15, 28, 17, 88 and 59 into the hash table. Double hashing technique with the secondary hash function $h_2(k) = 1 + \left(k \mod (m-1)\right)$ is used to resolve collisions. **Show all your work.**

- *h(10) = 10*

- *h(22) = 0*

- *h(31) = 9*

- *h(4) = 4*

- *h(15) = 15 mod 11 = 4 → collision*
  h_2 (15) = 1 + (15 mod 10) = 6
  h'(15,1) = (4 + 6) mod 11 = 10 → collision
  *h'(15,2) = (4 + 2*6) mod 11 = 5*

- *h(28) = 28 mod 11 = 6*

- *h(17) = 17 mod 11 = 6 → collision*
  h_2(17) = 1+(17 mod 10) = 8
  h'(17,1) = (6+8) mod 11 = 3

- *h(88) = 88 mod 11 = 0 → collision*
  h_2(88) = 1+(88 mod 10) = 9
  h'(88,1) = (0+9) mod 11 = 9 → collision
  *h'(88,2) = (0+2*9) mod 11 = 7*

- *h(59) = 59 mod 11 = 4 → collision*
  h_2(59) = 1+(59 mod 10) = 10
  h'(59,1) = (4+10) mod 11 = 3 → collision
  *h'(59,2) = (4+2*10) mod 11 = 2*

22	0
	1
59	2
17	3
4	4
15	5
28	6
88	7
	8
31	9
10	10