Data structures and algorithms

Consider a binary tree data structure with N nodes, where each node of the tree has the following records:

key – the key (an integer) stored in the node
parent – a pointer to the parent of the node in the tree
left, *right* – pointers to the left and right children of the node in the tree

a. [30 %] Use pseudo-code to describe a *recursive* algorithm *PrintKeys(root)* with time complexity *O(N)*, which, given the root of the tree, prints out the key of each node in tree.

b. [40 %] Use pseudo-code to describe a *non-recursive* algorithm *PrintKeys(root)* with time complexity *O(N)*, which, given the root of the tree, prints out the key of each node in tree. Use a stack as an auxiliary data structure.

c. [30 %] Suppose you need to develop an algorithm *SameKeys(T1,T2)* that, given as input the roots of two distinct trees *T1* and *T2*, determines whether the two trees store the exact same set of N keys. Is it possible to design *SameKeys(T1,T2)* with time complexity better than $O(N^2)$? Justify your answer and state your assumptions

(you can refer to well-known algorithms and do not need to show their pseudo-code implementations, as long as you explain their behavior and complexity)

Operating Systems

Consider a single-processor computer with 32-bit virtual and physical address spaces. The operating system (O/S) supports multi-tasking with processes and paged virtual memory. **Assume the page table for any process has only 4 entries**, and page sizes are fixed (64Kbytes). Page table entries (PTEs) hold, in addition to the Virtual Address (VA) and physical Page Frame (PF), the following bits:

R	(R=0: not readable, R=1: readable)
W	(W=0: not writable, W=1: writable)
X	(X=0: not executable, X=1: executable)
V	(V=0: translation invalid; V=1: valid translation)

Assume there are only *two user processes* (P1 and P2) in the system, each with the following page tables (addresses are in base-16 hexadecimal format)

P1's VA	**PF**	**R**	**W**	**X**	**V**
0x0000	0x0003	1	0	1	1
0x0001	0x0002	1	0	0	1
0x0002	0x0001	1	1	0	1
0x0003	0x0000	0	1	0	1

P2's VA	**PF**	**R**	**W**	**X**	**V**
0x0000	0x0004	1	0	1	1
0x0001	0x0002	1	1	0	1
0x0011	0x0005	1	1	0	0
0x0012	0x0006	1	1	0	1

a) [40 %] Using the notation "Px(PC): LD R, addr" for a memory load instruction, issued by process Px (P1 or P2) from address PC, which retrieves a memory word from addr and stores in register R, and "Px(PC): ST R, addr" to denote a store instruction. Determine, for each of the following, whether the instruction **always, never, or possibly** requires handling by the O/S kernel. In each case, provide a one-sentence justification for your answer.

i) P1(PC=0x00000000) LD R, addr=0x00000000

ii) P2(PC=0x00010000) LD R, addr=0x00110800

iii) P1(PC=0x00000100) ST R, addr=0x00020000

b) [30 %] Assuming the above page tables, can P1 and P2 share a busy-wait mutual exclusion lock in memory, without invoking system calls? Explain.

c) [30 %] In this scenario, under what conditions (if any) would a Translation Lookaside Buffer (TLB) with 8 entries deliver better performance than a TLB with 4 entries**?** Explain.