ATILIM UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

**Name: Name: Id:**

**Section:**

**Signature:**

**CMPE 226 Data Structures**

**Year** : 2018-2019 Fall

**Instructors** : E. Gökçay

**FINAL** Examination

**Date**: 14.01.2019 **Time**: 15:30-17:30

**Duration**: 120 minutes

**WARNINGS**

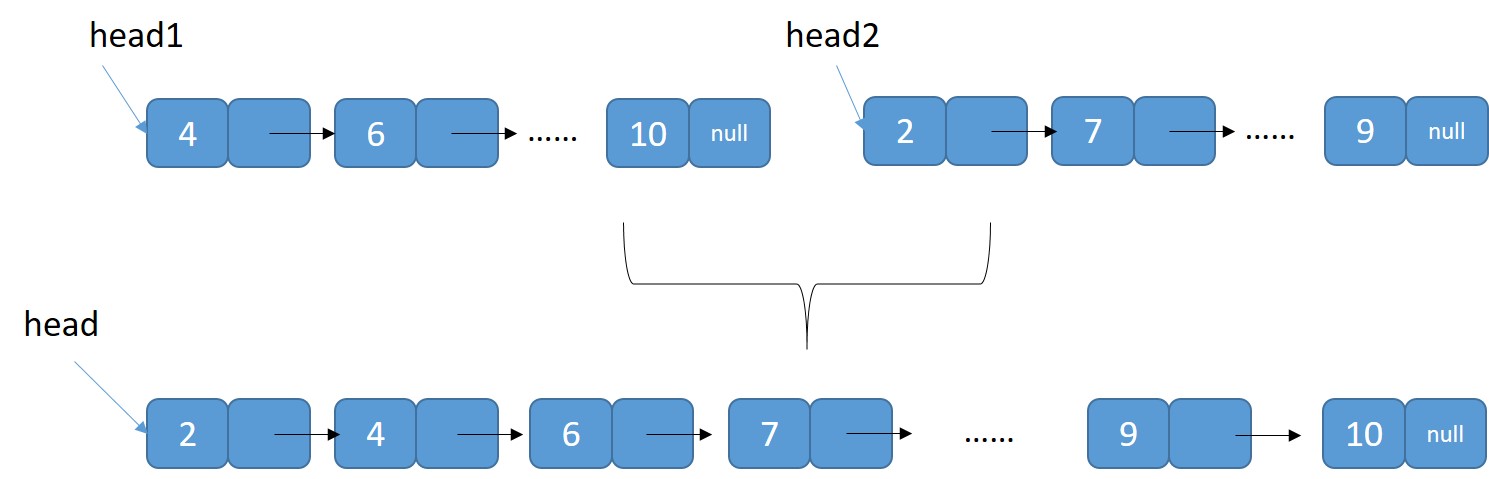
* It is forbidden to bring electronic data storage equipments (mobile phones, MP3 players, flash disks and so on.) to exams.
* Students who either cheat, attempt to cheat or provide a help to other(s) in cheating, get 0 (zero) grade from this examination. Also, based on the regulations, a disciplinary action will be taken.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q1 (10) | Q2 (15) | Q3 (25) | Q4 (10) | Q5 (15) | Total 100 |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Q1 (15) | Q2 (15) | Q3 (15) | Q4 (15) | Q5 (15) | Q6 (15) | Q7 (10) | Q8 (10) | Q9 (10) | Total 120 |
|  |  |  |  |  |  |  |  |  |  |

**Q1) (15pts)** (Part of HW2 Merge Sort) Assume that you have two head pointers of type

node<T> \*. Each head pointer points to a sorted LinkedList. Merge two sorted LinkedList into a single Sorted List. Don’t use the example in the code. Write a general program. There is no need to keep the original lists.



template <class T>

node<T> \* mergeLinkedLists(node<T> \*head1, node<T> \*head2) {

node<T> \*head, \*p;

if (head1 == NULL && head2 == NULL) return NULL;

if (head1 == NULL) return head2;

if (head2 == NULL) return head1;

// both pointers are not NULL

if (head1->data < head2->data) {

head = head1;

head1 = head1->next;

}

else {

head = head2;

head2 = head2->next;

}

p = head; // temp pointer to create the LinkedList

// Iterate both pointers until one them is NULL

while (head1 != NULL && head2 != NULL) {

if (head1->data < head2->data) {

p->next = head1;

head1 = head1->next;

p = p->next;

}

else {

p->next = head2;

head2 = head2->next;

p = p->next;

}

}

// copy the remaining elements after the loop

if (head2 == NULL)

p->next = head1;

else

p->next = head2;

return head;

}

**Q2) (15pts)** Assume that you have a Binary Search Tree. Write a function to find how many nodes are there with a value **less than** a given value. Check all extreme cases. You can use a recursive (easier) or a sequential program.

template <class T>

int int countNodesLessThan(T value, node<T> \*root) {

// Base case

if (!root) return 0;

// If current node is less than value, then include it in count and

// recur for left and right children of it

if (root->data < value )

return 1 + countNodesLessThan(value, root->left ) +

countNodesLessThan(value, root->right);

// If current node is larger than value, then recur for left

// child (root->data >= value)

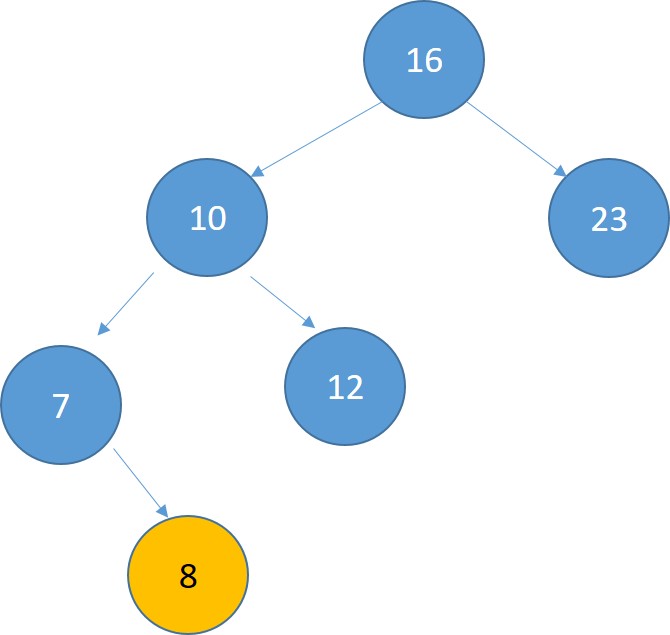
// No need to search the right child

else

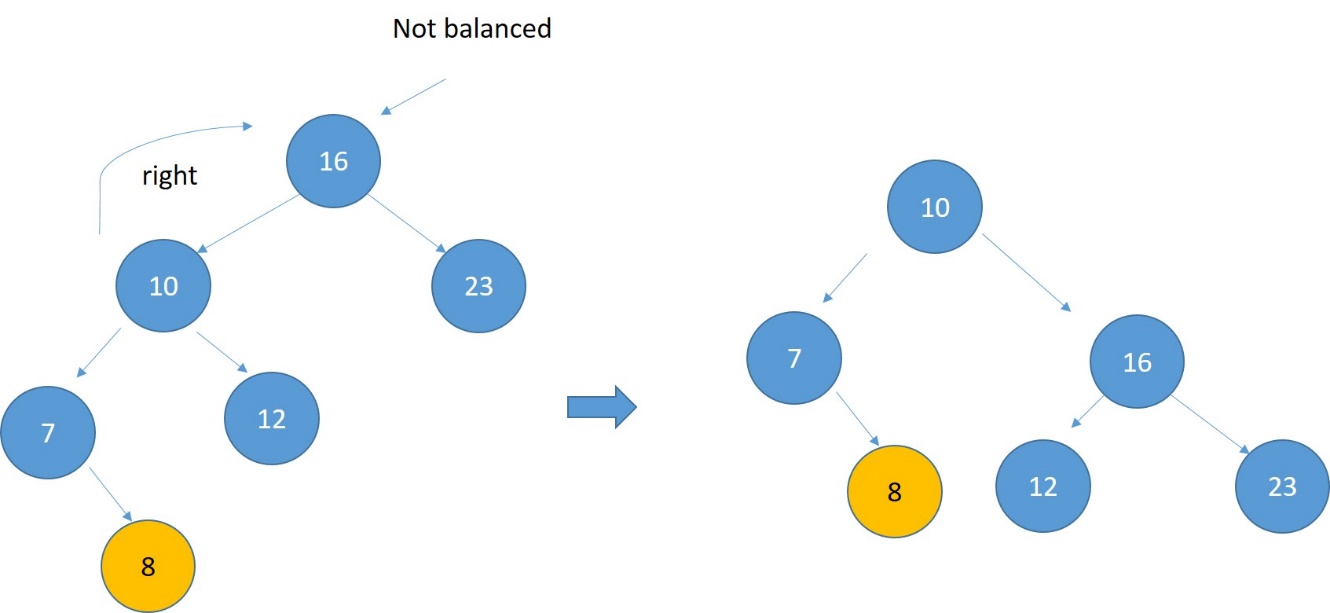
return countNodesLessThan(value, root->left);

}

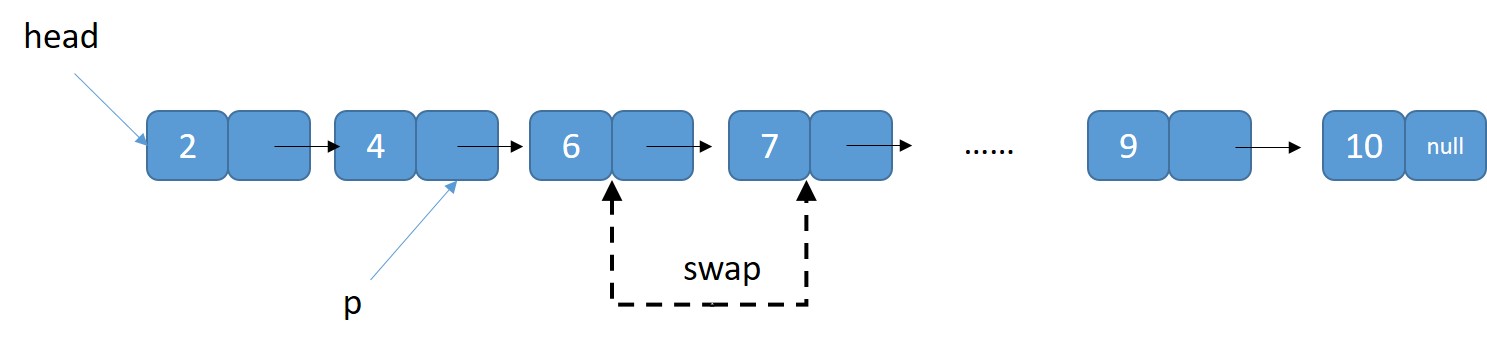
**Q3) (15pts)** A binary tree is given below. When the last node is added, the tree becomes unbalanced. Balance the tree using rotations. Show every step and indicate which node is going where.



**Node “16” is not balanced. The insertion takes places to the left of the left child of the unbalanced node. This is an outside case. It needs only a right rotation.**

****

**Q4) (15pts)** Given the following LinkedList and a pointer ***p***. Write a program to swap two nodes coming after the pointer ***p***. Don’t use the example. Write a general program for any ***p***.

****

template <class T>

void swapLinkedList(node<T> \* p) {

// check extreme conditions

if (p == NULL) return;

if (p->next == NULL)return;

if (p->next->next == NULL)return;

node <T> \*pn = p->next;

node <T> \*pnn = p->next->next;

p->next = pnn;

pn->next = pnn->next;

pnn->next = pn;

return;

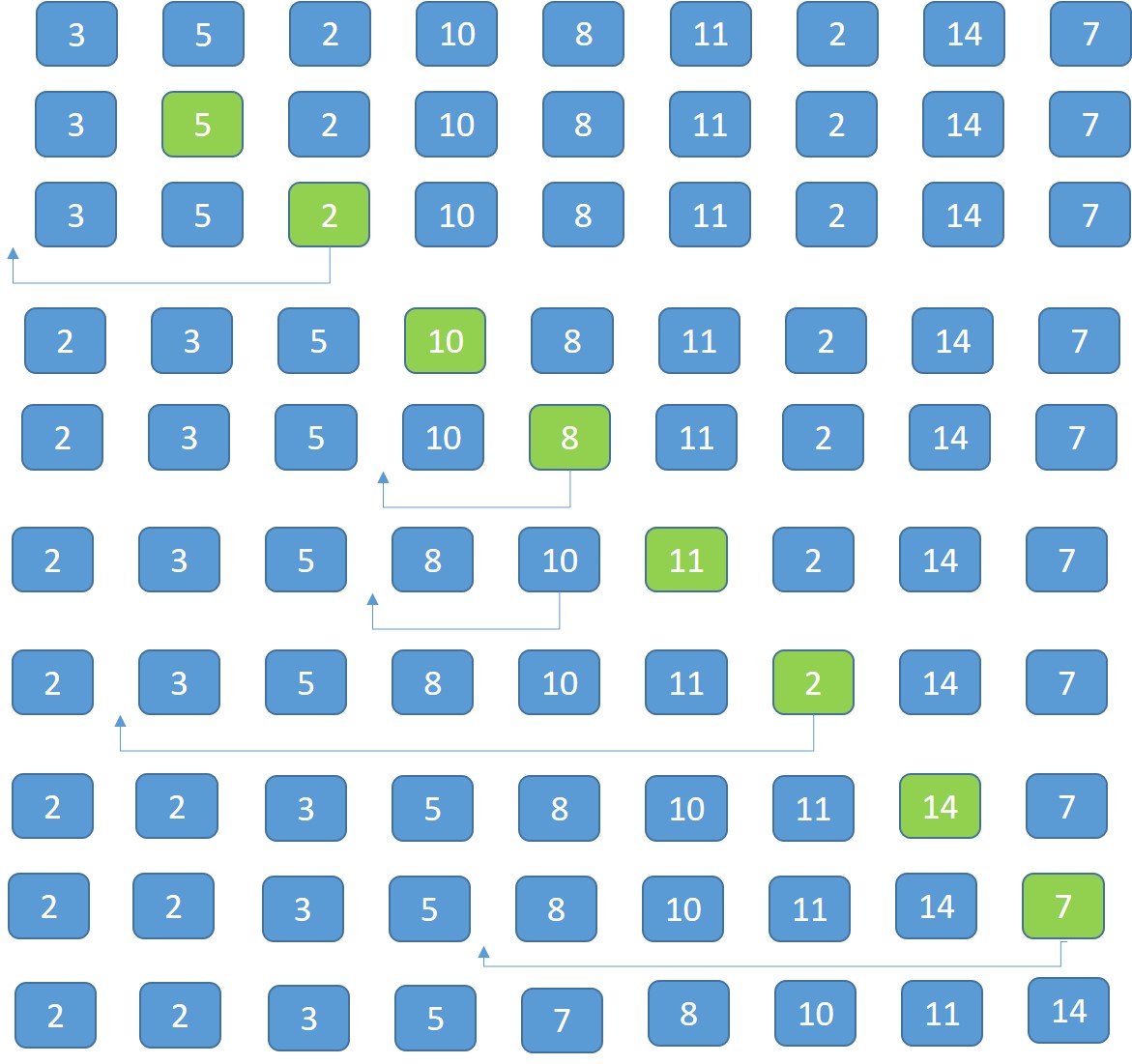
}

**Q5) (15pts)** Insertion Sort Algorithm is given as:

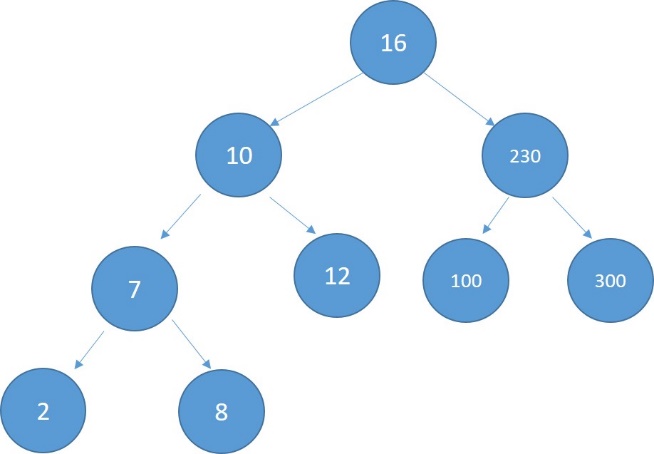
// Sort an arr[ ] of size n  
Loop from i = 1 to n-1.  
 Pick element arr[i] and insert it into sorted sequence arr[0…i-1]

Sort the following numbers using insertion sort. Show every step, move, shift operations.





**Q6) (15pts)** A complete binary tree can be represented as an array. Create the array version of the following binary tree and show how to access left and right child of a given parent.

****

**Array representation of a complete binary tree (copy to array level by level)**

**If k indicates the index of a parent node, the left child is at 2k+1 and the right child is at 2k+2 index in the array.**

**0 1 2 3 4 5 6 7 8**

16

10

230

7

12

100

300

2

8

**Q7) (10pts)** Calculate the following postfix expression

Postfix Value?

|  |  |
| --- | --- |
| 2 16 2 5 2 + \* - / | (2/(16-(2\*(5+2)))) = 1 |

**Q8) (10pts)** Explain why we are trying to keep a binary tree balanced (for example AVL trees). What happens if we don’t?

**If we don’t balance the tree, the height of the tree can be (worst case) proportional to N and not to LOG(N). This will increase the N\*LOG(N) search/delete/insert complexity to N\*N.**

**Q9) (10pts)** Explain **chaining** in a hash function implementation. Give an example as well.

**When there is a collision using the hash function, the collision is resolved by creating a LinkedList at the location.**

