NATIONAL UNIVERSITY OF SINGAPORE

ANSWER

SCHOOL OF COMPUTING
SEMESTER II AY2019/2020

FINAL ASSESSMENT
IT5003: DATA STRUCTURES AND ALGORITHMS

December 2019
Time Allowed: 2 Hours

STUDENT NUMBER:

A	**0**							

INSTRUCTIONS TO CANDIDATES:

1. Write your student number in the space provided above using a **PEN**.
2. This examination paper consists **SIX (6) questions** and comprises **FOURTEEN (14)** printed pages including this front page and an empty page 14.
3. Answer all questions directly in the space given after each question. You can **use PENCIL or PEN.**
4. Marks allocated to each question are indicated. Total marks for the paper is **100**.
5. This is an OPEN BOOK assessment. Non-programmable Calculators are allowed. Other electronic devices, e.g. laptop, smart phone are **prohibited**.

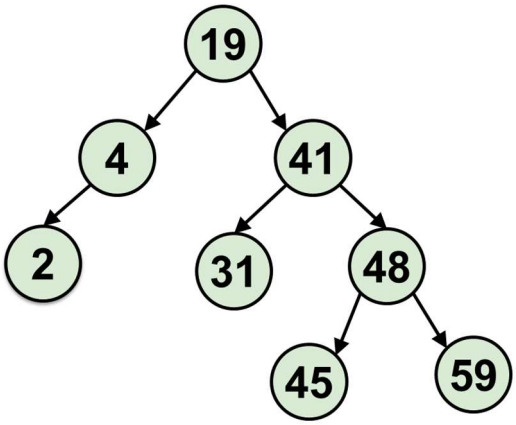
Questions	Possible	Marks
Q1	16	
Q2	18	
Q3	18	
Q4	14	
Q5	18	
Q6	16	
Total	**100**	

Question 1 (16 marks)

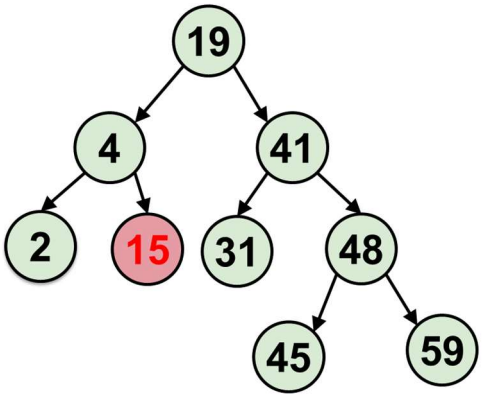
This question is on **AVL Tree**.

For part (a) to (d) Starting from the AVL Tree below, perform the operations specified in each subparts. If the operation causes rotation(s), give the intermediate AVL tree after each rotation operation.

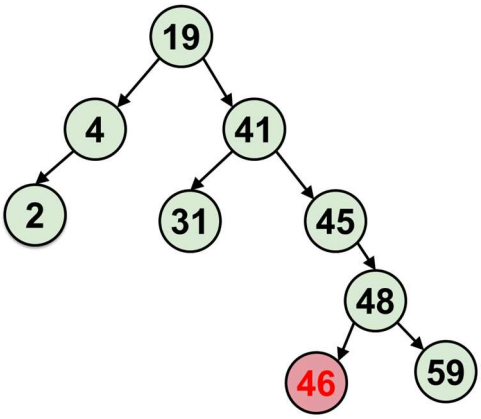
Please note that each part is independent, i.e. they all starts from the tree below and not a continuation from subpart to subpart.

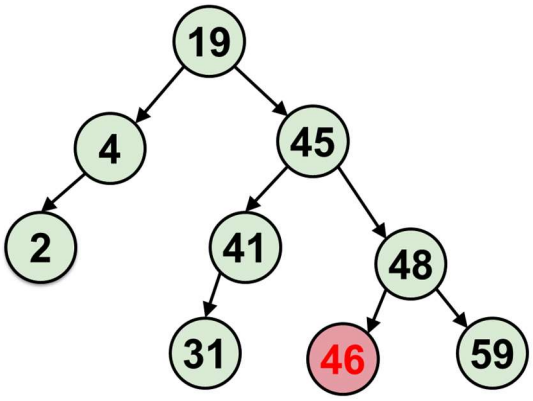


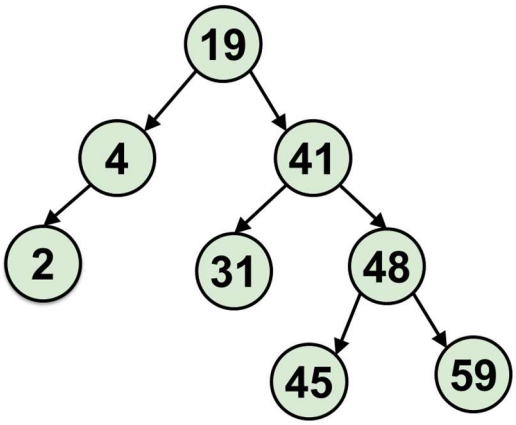
a. Insert 15 [2 marks]



b. Insert 46 [3 marks]

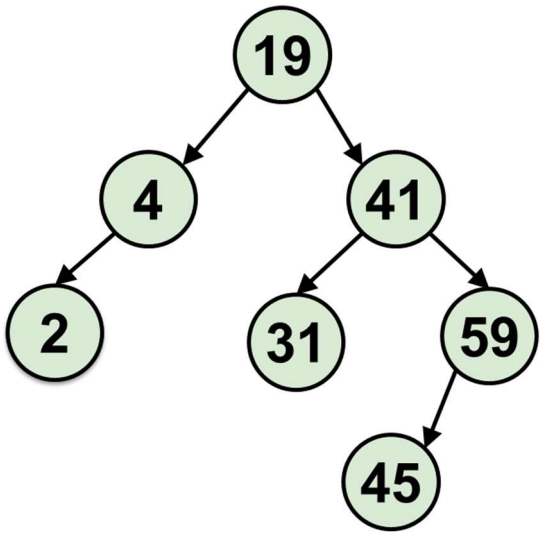






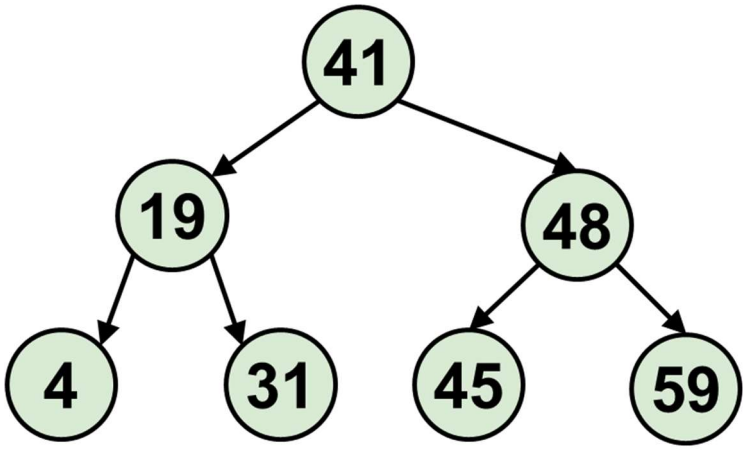
(duplicated for your reference)

c. Delete 48 [2 marks]



Acceptable alternative: use 45 to replace 48 instead

d. Delete 2 [3 marks]



e. Suppose we want to support a new operation for AVL tree:

```
updateKey( oldKey, newKey)
```

This method change the **oldkey** to **newKey** and perform the necessary steps to maintain the AVL tree operation. Give **pseudo code** for this operation. You can reuse all existing operation in the AVL Tree (e.g. rotate left/right, findMin, insert, delete, search). **Give the complexity of your approach.** [6 marks]

```
delete(oldKey)

insert(newKey)
```

Complexity is `O(lg N)`

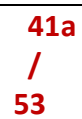
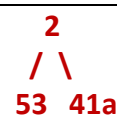
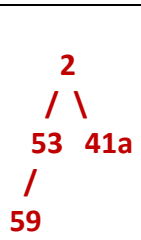
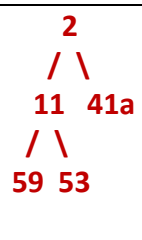
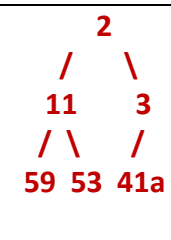
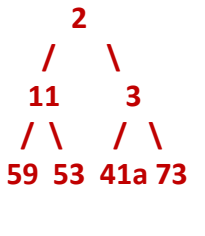
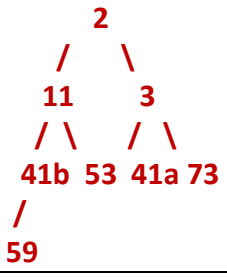
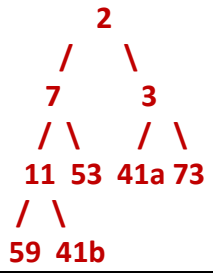
Question 2 (18 Marks)

This question looks at **heap**. To test your understanding, we choose to use the variant **min heap** instead of the **max heap**. If you are unable to do the question with **min heap**, you can do all parts with **max heap**, but with a **penalty of 25%.** If you choose to do so, please write a clear "**max heap**" at the side of your attempts.

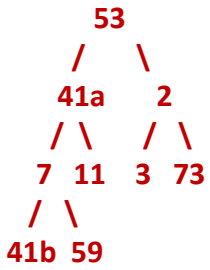
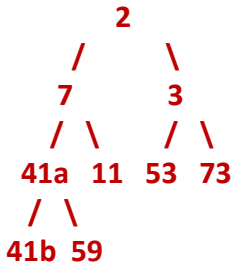
a. Give a **definition** for **min heap**. [2 marks]

A complete binary tree where root is smaller **or equal to all items** in left and right subtrees **recursively.** [Or each node follow the same property]

b. Use heap insertion to insert the following number 53, 41_a, 2, 59, 11, 3, 73, 41_b, 7 into a **min heap**. Make sure the two copies of 41 are notated with subscript for checking purpose. Use the space given to show the evolution of the heap, i.e. show the heap after each insertion as indicated. [4 marks]

53	41a	2
		
59	**11**	**3**
		
73	**41b**	**7**
		

c. Use heapify to construct a **min heap** from the same set of values, i.e. **53, 41$_a$, 2, 59, 11, 3, 73, 41$_b$, 7**. Show the heap when the heapify algorithm reaches the number **59** (i.e. after heapify the semi-heap where 59 is the root) and 53 (i.e. whole heap). [6 marks]

After heapifying 59	After heapifying 53
 2 marks	 4 marks

d. Suppose we perform the following operation with a min heap **h** with **N unique items** and an initially empty BST **bt**:

```
while h is not empty:
    item = h.delete()
    bt.insert(item)
```

What is the height of the bt **in the best case?** Express using Big-O notation and briefly explain. [6 marks]

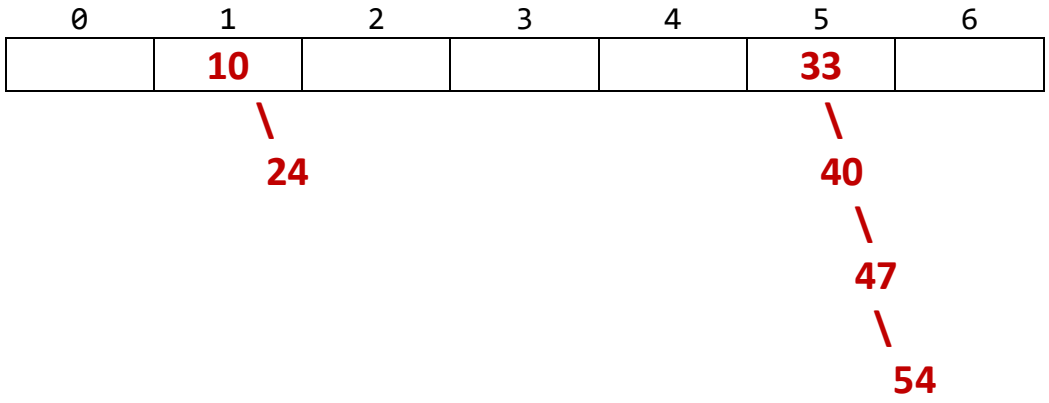
Height is O(**N**)

Since values are unique, min heap deletion gives values from smallest to largest in strictly increasing order ➔ the BT is essentially a linked list with all values on the left subtree only.

Question 3 (18 Marks)

a. Mr.Smaralec decided to implement a variant of **separate chaining** for his hash table. Instead of linked list at each location, he uses a **BST** instead. Given that the hash table is size 7 and the hash function is h(key) = key % 7, give the **insertion sequence** for the keys { 47, 33, 54, 24, 10, 40 } that causes the worst possible performance for this variant. Insertion sequence refers to the order to insert a set of keys. Give the content of the hash table at the end of the insertions. [4 marks]

Insertion sequence is **{33, 40, 47, 54} {10, 24} (simplest: two set of values that clashes, just ensure they are in order (ascending /decending)).**



Give the **worst case complexity** for Mr.Smaralec's approach in terms of **N** (number of items) and **M** (size of hash table). [2 marks]

Complexity is O(N) (all hashed to the same slot and in order)

b. Ms.Evansmar uses **double hashing** for her hash table:

- Size = 7
- H(Key) = key % 7
- H2(Key) = ((2*key) % 6) + 1

For each of the following part, insert the given number into the partially filled hash table, give the number of **probe(s).** Note that the probe number starts from 0 (home address), then 1, 2, ……

Insert 11 [4 marks]

0	1	2	3	4	5	6
11		**2**		**25**	**54**	

Number of probe = _____2 (will accept 3)____

Insert 22 [4 marks]

0	1	2	3	4	5	6
7	**71**	**2**	**31**	**25**	**22**	**356**

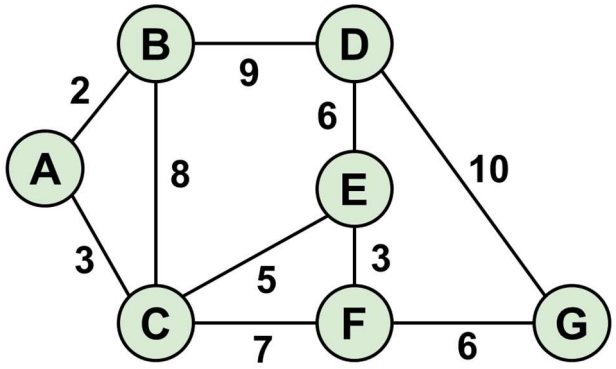
Number of probe = _____6(will accept 7)___

After the insertions above, Ms.Evansmar came to an interesting conclusion: "*Any number can essentially end up in **any slot in the hash table** under this scheme.*". Is she right? Briefly explain. [4 marks]

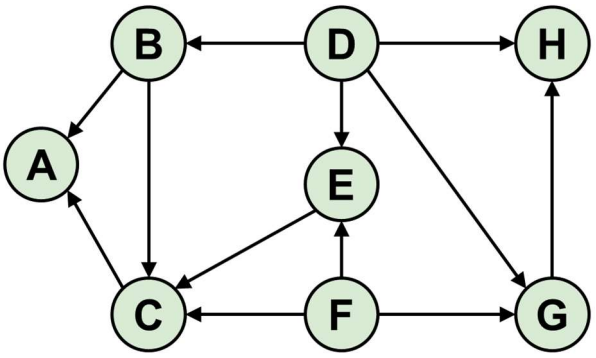
Yes. H_2(key) in this case gives co-prime of table size ➔ double hashing will visit all slots ➔ any number can be inserted at any slot

Question 4 (14 Marks)

a. The **Single Source Shortest Path** algorithm works well on an **undirected graph** as well. Perform the algorithm on the graph below, using the source node **"A"**. [8 marks]



			Shortest Distance from Source						
Step	**v**	**S**	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Init	-	-	**0**	**∞**	**∞**	**∞**	**∞**	**∞**	**∞**
1	**-**	**[a]**	**0**	**2**	**3**	**∞**	**∞**	**∞**	**∞**
2	**b**	**[a, b]**	**0**	**2**	**3**	**11**	**∞**	**∞**	**∞**
3	**c**	**[a, b, c]**	**0**	**2**	**3**	**11**	**8**	**10**	**∞**
4	**e**	**[a, b, c, e]**	**0**	**2**	**3**	**11**	**8**	**10**	**∞**
5	**f**	**[a, b, c, e, f]**	**0**	**2**	**3**	**11**	**8**	**10**	**16**
6	**d**	**[a, b, c, e, f, d]**	**0**	**2**	**3**	**11**	**8**	**10**	**16**
7	**g**	**[a, b, c, e, f, d, g]**	**0**	**2**	**3**	**11**	**8**	**10**	**16**



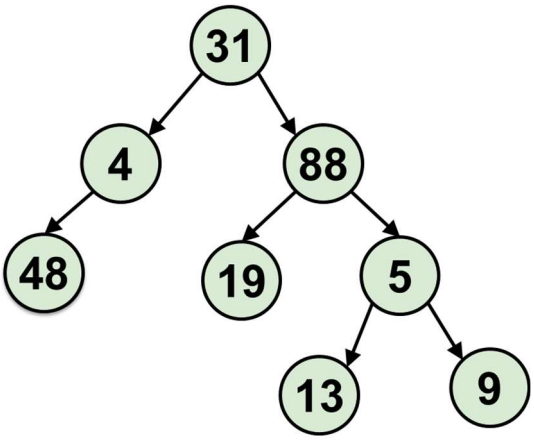
b. Give a **Topological Sort** sequence for the directed graph above. **If there are multiple possible nodes to choose from, you should always choose the smallest alphabetically**. E.g. if you can choose "A" or "B", you should choose "A". If the **topological sort** sequence cannot be generated, point out the issue. [6 marks]

full mark:
D F B E G C H A (using algorithm in lecture)
D B F E C A G H (just pick 0-degree every round)
partial mark: D B E F C G A H (Valid, but not alpha order)

Question 5 (18 Marks)

a. Draw the Binary Tree from the following pre-order and in-order traversals:

- Pre-order Traversal: **31, 4, 48, 88, 19, 5, 13, 9**
- In-order Traversal: **48, 4, 31, 19, 88, 13, 5, 9** **[4 marks]**



b. Suppose we use a binary tree to model the **sales performance** of a company. For simplicity, we assume each sales employee has **at most two subordinates**, represented as his/her two child nodes in the BT. Each of the BT node, in addition to the standard left/right references, contains `{name, month_sales}` information for an employee. Give the **pseudo code** for the following operation:

```
def meetQuota( T, quota ):
""" Returns a list of employee names, whose team meet the sales quota."""
```

This operations reports **all employees** whose combined monthly sales of his/her team (i.e. themselves and their subordinates, sub-subordinates, …. etc) exceeds the **quota** specified. [6 marks]

```
def sumSale( T ):
    if T == None:
      return 0
    return T.month_sales + sumSale(T.leftT) + sumSale(T.rightT)

def meetQuota( T, quota):
   if T == None:
       return []

   result = meetQuota( T.leftT, quota) + meetQuota(T.rightT, quota)
   if sumSale(T) > quota:
        result.append( T.name )
   return result
```

c. Give an alternative version for part (b) where the only the **lowest employee** in the BT that meets the quota is reported. In other words, if B is a subordinate of A, and B's team meet the monthly quota, then only B is reported but not A. [8 marks]

```
#reuse sumSale() from (b)

def meetQuota( T, quota ):
    if T == None:
        return []

    result = meetQuota( T.leftT, quota) + meetQuota(T.rightT, quota)
    if result == [] and sumSale(T) > quota:
        result.append( T.name )
    return result
```

Question 6 (16 Marks)

Write a Python function to duplicate a specific node in a singly linked list a number of times. For example, if the given linked list **L** is 7→2→3→1→5→2 and we want to "duplicate node "2" **three** times, we get: 7→2→**2→2→2**→3→1→5→2→**2→2→2** (duplicated nodes are **bolded and underlined**). The function header is given below:

```
def duplicate( L, target, nCopy ):
"""" duplicate all [target] nodes in linked list [L], [nCopy] times. Return the modified linked list."""
```

For example, we can call ***duplicate*(L, 2, 3)** to achieve the end result as shown above.

a. Write an **iterative version of the function.** [8 marks]

```
def duplicate( L, target, nCopy ):
    ptr = L

    while ptr != None:
        nxt = ptr.next
        if ptr.item == target:
            chain = nxt
            for i in range(nCopy):
                chain = SinglyNode(target, chain)
            ptr.next = chain
        ptr = nxt
    return L
```

b. Write an **recrusive version of the function.** [8 marks]

```
def duplicateR( L, target, nCopy ):
    if L == None:
        return L

    nxt = duplicateR(L.next, target, nCopy)

    if L.item == target:
        chain = nxt
        for i in range(nCopy):
            chain = SinglyNode(target, chain)
        nxt = chain
    L.next = nxt

    return L
```

<Blank Page>

~~~ End of Paper ~~~