

CS 112 Practice Midterm Solutions

Question 1)

```
class TreeNode<T>
{
    void set_left (TreeNode<T> new_left); // Update lchild to point to new_left
    void set_data (T new_data); // Set the data field to new_data
    T data;
    TreeNode<T> lchild;
    TreeNode<T> rchild;

    TreeNode( TreeNode<T> newLeft, TreeNode<T> newRight, T newData )
    {
        lchild = newLeft;
        rchild = newRight;
        data = newData;
    }
}

public class TreeNodeClient
{
    public static void main( String [] args )
    {
        // Create an instance of TreeNode with String data
        TreeNode<String> tree =
            new TreeNode<String>( null, null, "cs112" );
    }
}
```

Question 2a) The stack stored in an array,

55	10				
----	----	--	--	--	--

s.size = 2

s.pop(); // = 10

55					
----	--	--	--	--	--

s.size = 1

s.push(41);

55	41				
----	----	--	--	--	--

s.size = 2

s.push(23);

55	41	23			
----	----	----	--	--	--

s.size = 3

s.pop(); // = 41

55	41				
----	----	--	--	--	--

s.size = 2

s.push(31);

55	41	31			
----	----	----	--	--	--

s.size = 3

Question 2b)

The queue stored in an array,

55	10	7		
----	----	---	--	--

q.front = 0, q.rear = 2

q.enqueue(43)

55	10	7		43
----	----	---	--	----

q.front = 4, q.rear = 2

q.dequeue() = 7

55	10			43
----	----	--	--	----

q.front = 4, q.rear = 1

q.dequeue() = 10

55				43
----	--	--	--	----

q.front = 4, q.rear = 0

q.dequeue() = 55

				43
--	--	--	--	----

q.front = 4, q.rear = 4

q.enqueue(9)

			9	43
--	--	--	---	----

q.front = 3, q.rear = 4

q.enqueue(3)

		3	9	43
--	--	---	---	----

q.front = 2, q.rear = 4

Question 2c)

$q.size = (q.front \leq q.rear) ? (q.rear - q.front + 1) : ((q.rear + 1) + (N - q.front));$

OR using modulo arithmetic,

$q.size = \text{Math.mod}(q.rear - q.front, N) + 1;$

Question 3)

```
(1) void quicksort(Item [] a, int l, int r) {  
  
(2)     if (r >= l) return; // base case  
(3)     int i = partition (a, l, r);  
        // quicksort the right side, then the left side  
(4)     quicksort(a, i+1, r);  
(5)     quicksort(a, l, i-1);  
}  
  
(6) int partition(Item [] a, int l, int r) {  
(7)     int i = l, j = r-1, Item v = a[r];  
(8)     for (;;) {  
        // find the next i such that a[i] is larger than v  
(9)         while (a[i] <= v)  
(10)            i++;  
  
        // find the previous j such that a[j] is smaller than v  
        // stop if j goes outside the array bounds.  
(11)        while (v < a[j]) {  
(12)            j--;  
(13)            if (j == l-1) break;  
        }  
        if (i >= j) break; // pointers crossed  
(14)        else swap (a[i], a[j]);  
    }  
(15)    return i;  
}
```

Solution:

Note from the pair of while loops: all $a[i] \leq \text{pivot } v$ should be in the left partition and $a[i] > \text{pivot } v$ should be in the right partition.

(2) if (r <= l) return; // base case should check for length <= 1

(7) int i = l, j = r, Item v = a[r]; // $a[r]$ belongs to the left partition and we need to
// allow it to be swapped by initializing j at r

(9) while (i < r && a[i] <= v) // need bounds checking for i, otherwise:
// if we had all $a[i] \leq v$ then we would return
// $i = (r + 1)$ which leads to infinite recursion in
// (5) quicksort(a, l, i-1);

Question 5)

```
public class ListNode
{
    int data;
    ListNode next;
}
```

5a) remove duplicates in sorted linked list

```
public removeDuplicatesSorted( ListNode p )
{
    while( p != null )
    {
        while( p.next != null && p.data == p.next.data )
            p.next = p.next.next;

        p = p.next;
    }
}
```

5b) remove duplicates in unsorted linked list

```
public removeDuplicatesUnSorted( ListNode p )
{
    while( p != null )
    {
        // Remove all duplicates in list that match p
        // Need to check all nodes succeeding p since list is unsorted

        ListNode q = p; // start with p to allow next node delete
        while( q != null )
        {
            // Is next node to q a duplicate for p?
            if( q.next != null && q.next.data == p.data )
                q.next = q.next.next; // remove successive duplicates
            else
                q = q.next;           // advance to next node
        }

        p = p.next; // advance p
    }
}
```