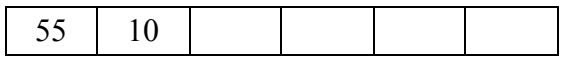
CS 112 Practice Midterm Solutions

Question 1)

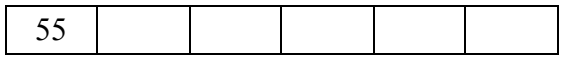
```
class TreeNode<T>
{
        void set_left (TreeNode<T> new_left); // Update lchild to point to new_left
        void set_data (T new_data); // Set the data field to new_data
        T data;
        TreeNode<T> lchild;
        TreeNode<T> rchild;

        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,



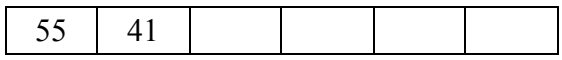
s.size = 2

s.pop(); // = 10



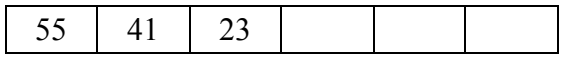
s.size = 1

s.push(41);



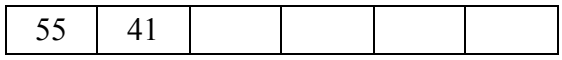
s.size = 2

s.push(23);



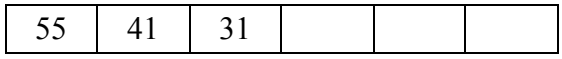
s.size = 3

s.pop(); // = 41



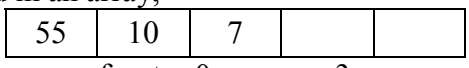
s.size = 2

s.push(31);



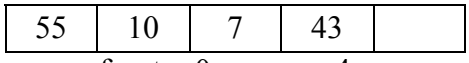
s.size = 3

Question 2b) (consistent with class version)
The queue stored in an array,



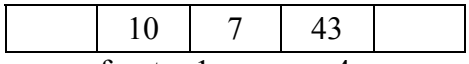
q.front = 0, q.rear = 3

q.enqueue(43)



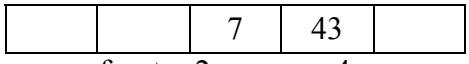
q.front = 0, q.rear = 4

q.dequeue() // = 55



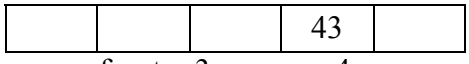
q.front = 1, q.rear = 4

q.dequeue() // = 10



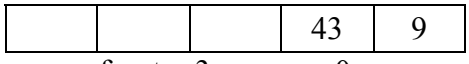
q.front = 2, q.rear = 4

q.dequeue() = 7



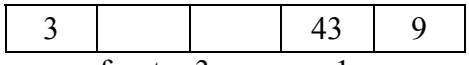
q.front = 3, q.rear = 4

q.enqueue(9)



q.front = 3, q.rear = 0

q.enqueue(3)



q.front = 3, q.rear = 1

Question 2c)

q.size = (q.front <= q.rear) ? (q..rear – q.front) : (q.rear + (N – q.front));

OR using modulo arithmetic,

q.size = Math.mod(q.rear - q.front , N);

Examples of modulo operation: mod(2, 5) = 2
mod(-2, 5) = 5 – 2 = 3
mod(-3, 5) = 5 – 3 = 2

Question 2b) **(inconsistent with class version only for reference)**
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 <= q.rear) ? (q..rear – q.front + 1) : ((q.rear + 1) + (N – q.front));

OR using modulo arithmetic,

q.size = 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, incorrect, fix below
(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)   // bounds checking missing
(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]);
        }
        //  swap (a[ i ], a[ r ]); missing here
(15)    return i;
}
```

Solution:

Note from the pair of while loops: all a[i] <= pivot v should be in the left partition and a[i] > pivot v should be in the right partition.

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

(9) while (i < r && a[i] <= v)   // need bounds checking for i

```
(14 - 15) swap (a[ i ], a[ r ]);  // move the pivot value to the end of left
                                  // partition, so that we satisfy the condition
                                  // a[ l to (i – 1) ] <= a[ i ] < a[ (i + 1) to r ]
                                  // we can now call quicksort on
                                  // the two partitions.
```

Complete test code:

```
package test;
import java.util.*;

public class TestSort {

	public static void main(String[] args) {

		int [] arr = new int[1000];
		Random rand = new Random(1234);
		for(int i = 0; i < arr.length; i++)
			arr[i] = rand.nextInt(arr.length/10);
		quicksort(arr, 0, arr.length - 1);

		System.out.println(Arrays.toString(arr));
	}

	static void quicksort(int [] a, int l, int r) {

		if (r <= l) return; // base case
		int i = partition (a, l, r);
		// quicksort the right side, then the left side
		quicksort(a, i+1, r);
		quicksort(a, l, i-1);
	}

	static int partition(int [] a, int l, int r) {
		int i = l, j = r-1, v = a[r];
		for (;;)
		{
			// find the next i such that a[i] is larger than v
			while ( i < r && a[ i ] <= v )
				i++;

		// find the previous j such that a[j] is smaller than v
			// stop if j goes outside the array bounds.
			while (v < a[j]) {
				j--;
				if (j == l-1) break;
			}
			if (i >= j) break; // pointers crossed
			else{
				//swap (a[i], a[j]);
				int temp = a[i]; a[i] = a[j]; a[j] = temp;
			}
		}
		// swap(a[i], a[r]);
		int temp = a[i]; a[i] = a[r]; a[r] = temp;
		return i;
	}
}
```

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;
        }
        // this pair of loops is linear: in each while loop iteration we either
        // (a) delete a node or (b) move to the next node.
}
```

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 q to next node (non-duplicate)
                }

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

Analysis: The best case for unsorted is when all elements are duplicates, in this case inner loop for q does one pass removing duplicates and we are done => O(N) operations.

The worst case is with a linked list of all distinct elements. In this case, the inner loop for q scans the following number of nodes,
N nodes in outer iteration 1,
(N – 1) nodes in outer iteration 2,
(N – 2) nodes in outer iteration 3,
…..
1 node in outer iteration N,

The total computation is 1 + 2 + 3 + …. + N is O(N^2)

The unsorted algorithm is hence O(N^2)