Name: ______________________________________ NetID: ___________________________

- **WRITE your name and NetID on EVERY page.**
- **DO NOT REMOVE** THE STAPLE IN YOUR EXAM.
- **DO NOT BEGIN** UNTIL INSTRUCTED TO DO SO.
- WRITE NEATLY AND CLEARLY. If we cannot read your handwriting, you will not receive credit. Please plan your space usage. No additional paper will be given.
- This exam is worth 150 points.

Problem 1 – Special Linked Structures (20 points)

Implement the following method to delete the last occurrence (starting from the front) of an item from a circular linked list, given a pointer rear to its last node. For example:

Input (rear points to 4):

5→3→7→2→3→4

Resulting list after deleting the last occurrence of 3 (rear points to 4):

5→3→7→2→4

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

public class CLL {
   public CLLNode rear; // point to last node in a CLL
   ...

   // Deletes LAST occurrence (from front) of given item from a CLL
   // Returns pointer to rear node of the updated CLL
   // Throws NoSuchElementException if item is not in the CLL

   public void deleteLastOccurrence ( int item )
   throws NoSuchElementException {
        // COMPLETE THIS METHOD
   }
}
```

Name: ______________________________________ NetID: ___________________________

```
public CLLNode deleteLastOccurrence(int item)
    throws NoSuchElementException {
      if (rear == null) {
        throw new NoSuchElementException();
      }
      CLLNode ptr = rear.next, prev=rear, match=null, matchprev = null;

      // find last occurrence
      do {
        if (ptr.data == item) {
          match = ptr;
          matchprev = prev;
        }
        prev = ptr;
        ptr = ptr.next;
      } while (ptr != rear.next);

      if (match == null) {
        throw new NoSuchElementException();
      }

      if (match == rear) {
        if (rear == rear.next) {
          return null;
        } else {
          matchprev.next = rear.next;
          return matchprev;
        }
      }

      matchprev.next = match.next;
      return rear;
    }
```

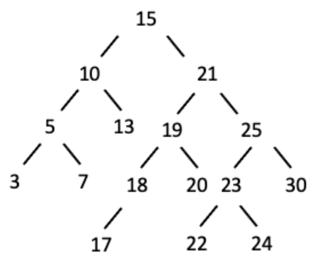
Partial Credit

Above is one way of doing, students may have other approaches.
- 1 pts: Throwing exception if item is not found
- 3 pts: Proper declaration and initialization of all variables
- 8 pts: Logic for match happening at last node
- 8 pts: Logic for match happening elsewhere

Name: ____ NetID: ____________________________

Problem 2 – Binary Search Tree (30 points)

Implement the following method to return the inorder successor of a node.



On the BST tree to the left:

- the inorder successor of node 21 is node 22.
- the inorder successor of node 10 is 13.

```
// Node class
private class Node <K extends Comparable<K>, V> {
        K key;
        V value;
        Node left;
        Node right;
 }

// Returns the inorder successor of node h
public Node successor (Node h) {
    // COMPLETE THIS METHOD
}
```

```
// Returns the inorder successor of node h
public Node successor (Node h) {

        if ( h == null || h.right == null ) return null;
        Node ptr = h.right;  // 5 points if student returns any node on the right
subtree

        while ( ptr.left != null ) {
            ptr = ptr.left;
        }
        // 20 points for returning the successor
    return ptr;  // 5 points for returning a node
    // MAX of 10 points if student return inorder predecessor
}
```

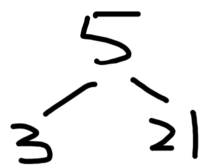
Name: ______________________________________ NetID: __________________________

Problem 3 – 2-3 trees and left-leaning red-black trees (40 points)

Construct a 2-3 tree and the corresponding left-leaning red-black tree whose keys are inserted in the following sequence. Label the links as R (red) or B (black), or use color in your answer.

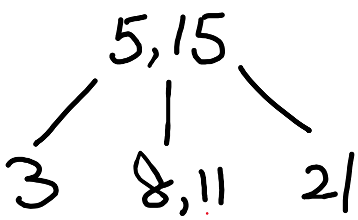
3 5 21 8 15 11 26 9 2

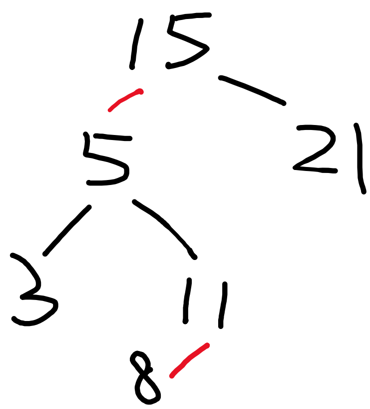
(a) **(10 points)** Draw the 2-3 tree after the insertions of 3 5 21, and the corresponding left-leaning red-black tree.



// 5 points for each correct tree

(b) **(12 points)** Draw the 2-3 tree after the insertions of 3 5 21 8 15 11, and the corresponding left-leaning red-black tree.

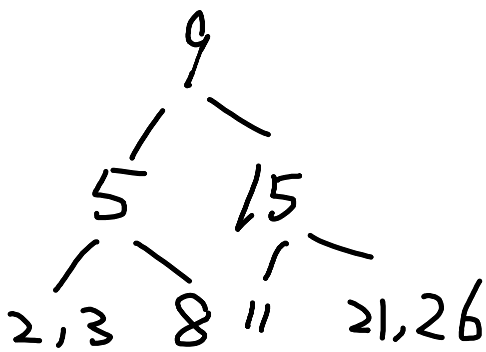


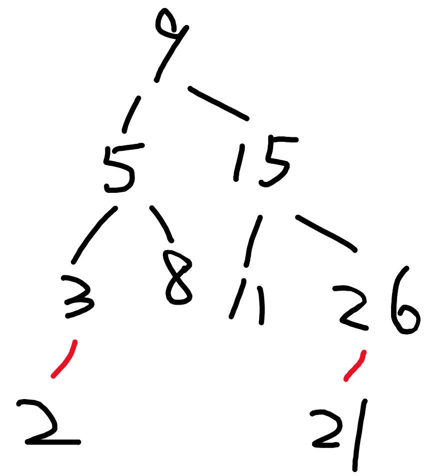


// 5 points for 2-3 tree
// 5 points for correct shape/keys of r-b tree
// 2 points for correct link color of r-b tree

Name: ______________________________________ NetID: ___________________________

(c) **(14 points)** Draw the 2-3 tree after the insertion sequence completed, and the corresponding left-leaning red-black tree.





// 5 points for 2-3 tree
// 5 points for correct shape/keys of r-b tree
// 4 points for correct link color of r-b tree

(d) **(4 points)** What is the minimum height of a 2–3 tree with n keys?

A. ~ $\log_3 n$
B. ~ $\log_2 n$
C. ~ $2 \log_2 n$
D. ~ n

Sol: A

Name: ______________________________________ NetID: ___________________________

Problem 4 - Priority Queue (30 points)

(a) **(5 points)** Assume the array below will be used to hold the keys of a MAX priority queue. Based on the array contents shown below. Is this a valid binary heap? Justify your answer according to the properties of a valid binary heap.

index	0	1	2	3	4	5	6	7	8	9	10	11
Key[		100	19	36	2	3	25	1	17	7		

No; it's a complete binary tree, however, not heap-ordered; parent's key must be greater than or equal to (no smaller than) the keys in child nodes

1 point if the answer is NO without an explanation

(b) Below is an array representing a valid binary heap for a MAX priority queue.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key[		60	53	3	49	46	1	2	48	16	25	40				

(b.1) **(10 points)** Show the array contents after 2 insertions: first insert key 55 and then insert key 44.

// 2 points for each correct 55 (index 3), 44 (index 6), 1 (index 12), 3 (index 13)
// 2 points for all other keys

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key[		60	53	55	49	46	44	2	48	16	25	40	1	3		

(b.2) **(10 points)** After the above insert operations, assume 2 (two) delMax() operations were performed, show the contents of the array.

// 1 point for each correct 53 (index 1), 49 (index 2), 44 (index 3), 48 (index 4), 3 (index 6), 1 (index 8)
// 1 points for each indices 12 and 13 empty
// 2 points for all other keys

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key[		53	49	44	48	46	3	2	1	16	25	40				

Name: ______________________________________ NetID: ___________________________

(c) **(5 points)** Assume there are n keys in a binary heap for a MAX priority queue. If a delMax() operation is performed, how many compares in the worst case, in terms of n, to restore the heap-order property? Justify your answer.

$2 \log_2 n$;
in a delMax() operation, sink() operation is required to restore the heap-order property starting at the root node. Each iteration of sink involves 2 compares, 1 compare to determine the larger key in children and 1 compare between the parent and the larger child to determine if an exchange operation is necessary to restore the property.

// 2 points for 2 log n
// 3 points for description

Name: ______________________________________ NetID: ___________________________

Problem 5 – Hash Table (30 points)

Assume the following keys 4371, 1323, 6173, 4199, 4344, 9679, 1989 are inserted in sequence to a hash table of size 10 where the hash function is hash(key) = key % 10. For simplicity, we omit the "values" associated with the keys and assume that no rehashing happens.

(a) **(10 points)** Show the hash table if separate chaining is used (insert at front of a chain).

[0] --->
[1] ---> 4371
[2] --->
[3] ---> 6173 ---> 1323
[4] ---> 4344
[5] --->
[6] --->
[7] --->
[8] --->
[9] ---> 1989 --> 9679 --> 4199

// 7 points: 1 point for each key in the correct array slot
// 3 points: the linked lists are in the correct order

(b) **(10 points)** Show the hash table if linear probing is used.

index

0	1	2	3	4	5	6	7	8	9
9679	4371	1989	1323	6173	4344				4199

// 7 points: 1 point for each key in its correct array slot
// 3 points: the sequence of keys is identical to solution

(c) **(10 points)** Assume the following code segment implements the delete() method as part of the Linear Probing API for (b) , show the hash table after deleting the key 4199 from the hash table in (b). The method `contains(key)` returns true if key is present in the hash table.

Name: ______________________________________ NetID: ___________________________

```
public void delete(Key key) {
   if (key == null) return;
   if (!contains(key)) return;
   int i = hash(key);
   while (!key.equals(keys[i])) {
      i = (i + 1) % m;
   }

   keys[i] = null;
   vals[i] = null;

   i = (i + 1) % m;
   while (keys[i] != null) {
       Key keyToRehash = keys[i];
       Value valToRehash = vals[i];
       keys[i] = null;
       vals[i] = null;
       n--;
       put(keyToRehash, valToRehash);
       i = (i + 1) % m;
   }
   n--;
}
```

// 1 point for 4371 (index 1), 1323 (index 3), 9679 (index 9)
// 2 points each for 6173 (index 4), 4344 (index 5)
// 3 points for 1989 (index 0)

index

0	1	2	3	4	5	6	7	8	9
1989	4371		1323	6173	4344				9679