Midterm 2

Graded

Student

AKSHAJ KAMMARI

Total Points

63 / 150 pts

Question 1

Problem 1 - Special Linked Structures

Resolved **0** / 20 pts

+ 1 pt Throws an exception if element not found

```
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;
}
```

+ 3 pts Proper declaration and initialization of all variables

```
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;
    }
```

+ 8 pts Logic for match happening at last node

```
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;
    }
```

+ 8 pts Logic for match happening elsewhere

```
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;
    }
```

✔ **+ 0 pts** Incorrect

Regrade Request **Submitted on: Apr 28**

I was wondering if I could get maybe a couple points or a point or two back because of my matches because though they may not have aligned correctly with the grading criteria it still

may have the potential of solving the same problem with a circular linked list.

I wasn't able to give points because the first while loop will never work, since the node itself will never be equal to the data contained inside of the node. Additionally, ptr is not defined until after it is used which causes an issue. For this question, it was necessary to remove the last Occurrence of an item. It seems that you return a certain node, which was not the objective as the method has a void return type.

Reviewed on: Apr 28

Question 2

Problem 2 - Binary Search Tree

Resolved **0** / 30 pts

+ 5 pts Returns any node in right subtree

```
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
}
```

+ 5 pts Returns a node

```
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
}
```

+ 20 pts Returns successor

```
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
}
```

✔ **+ 0 pts** Incorrect

Regrade Request **Submitted on: Apr 28**

Is there any there I can get maybe a point or two for trying to do the opposite and returning 0?

sorry I can't see in any way how you can get partial credit

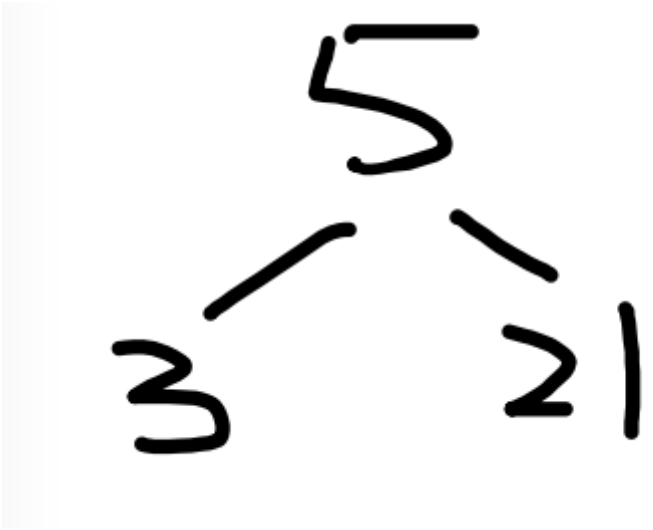
Reviewed on: Apr 29

Question 3

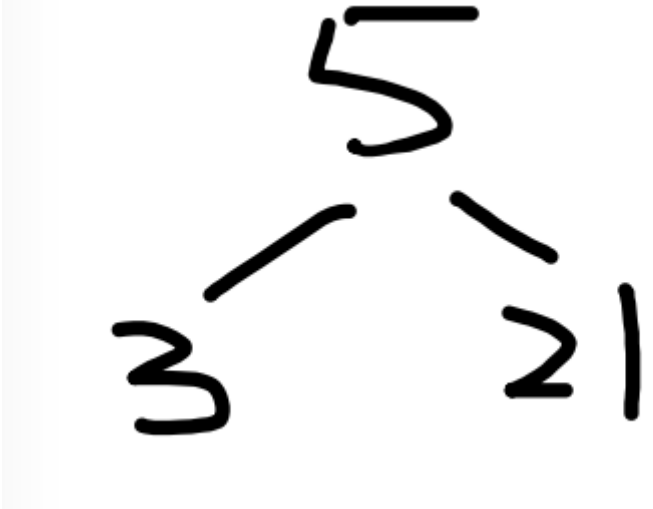
Problem 3 - 2-3 trees and left-leaning red-black trees **20** / 40 pts

3.1 (a) **10** / 10 pts

✔ **+ 5 pts** Correct 2-3 tree



✔ **+ 5 pts** Correct left-leaning red-black tree.

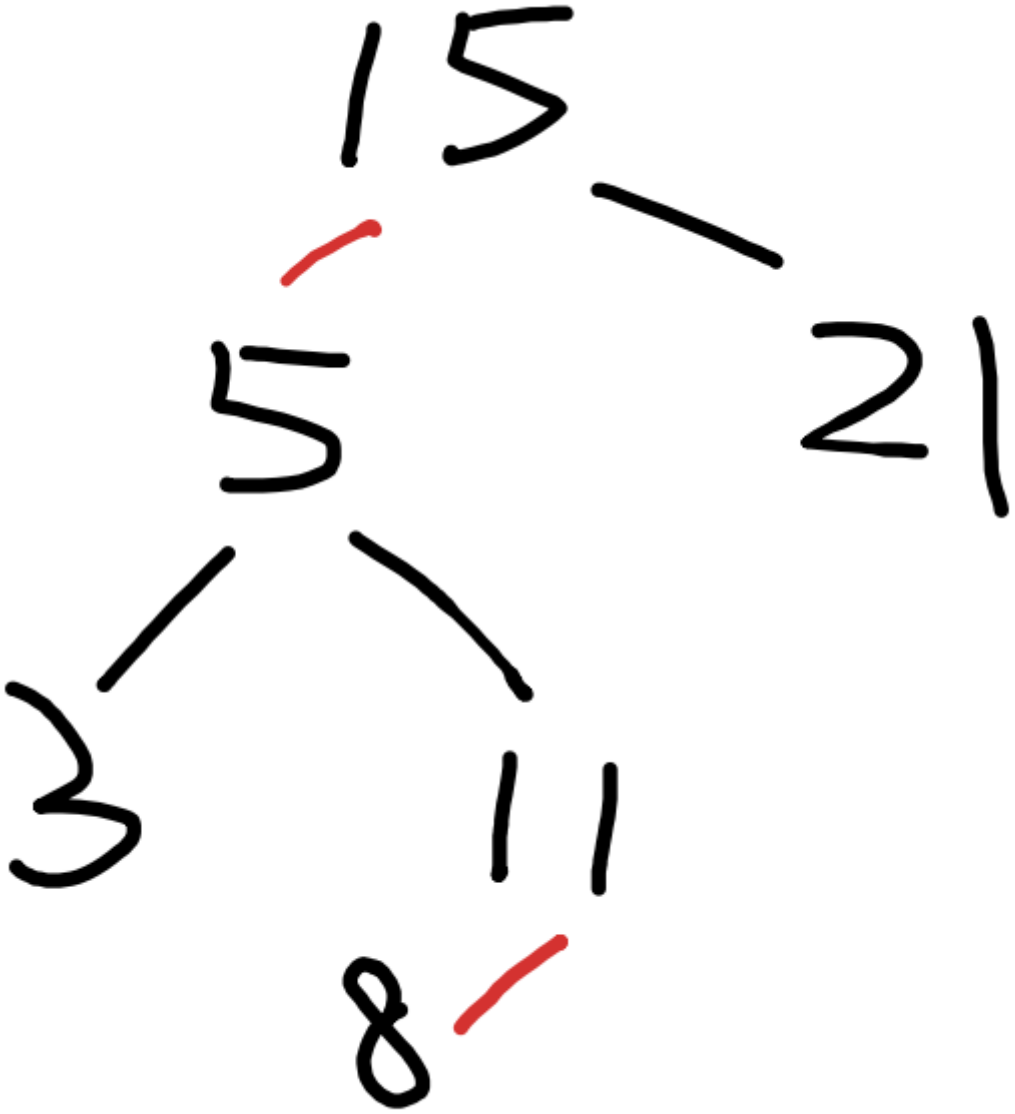


+ 0 pts Incorrect

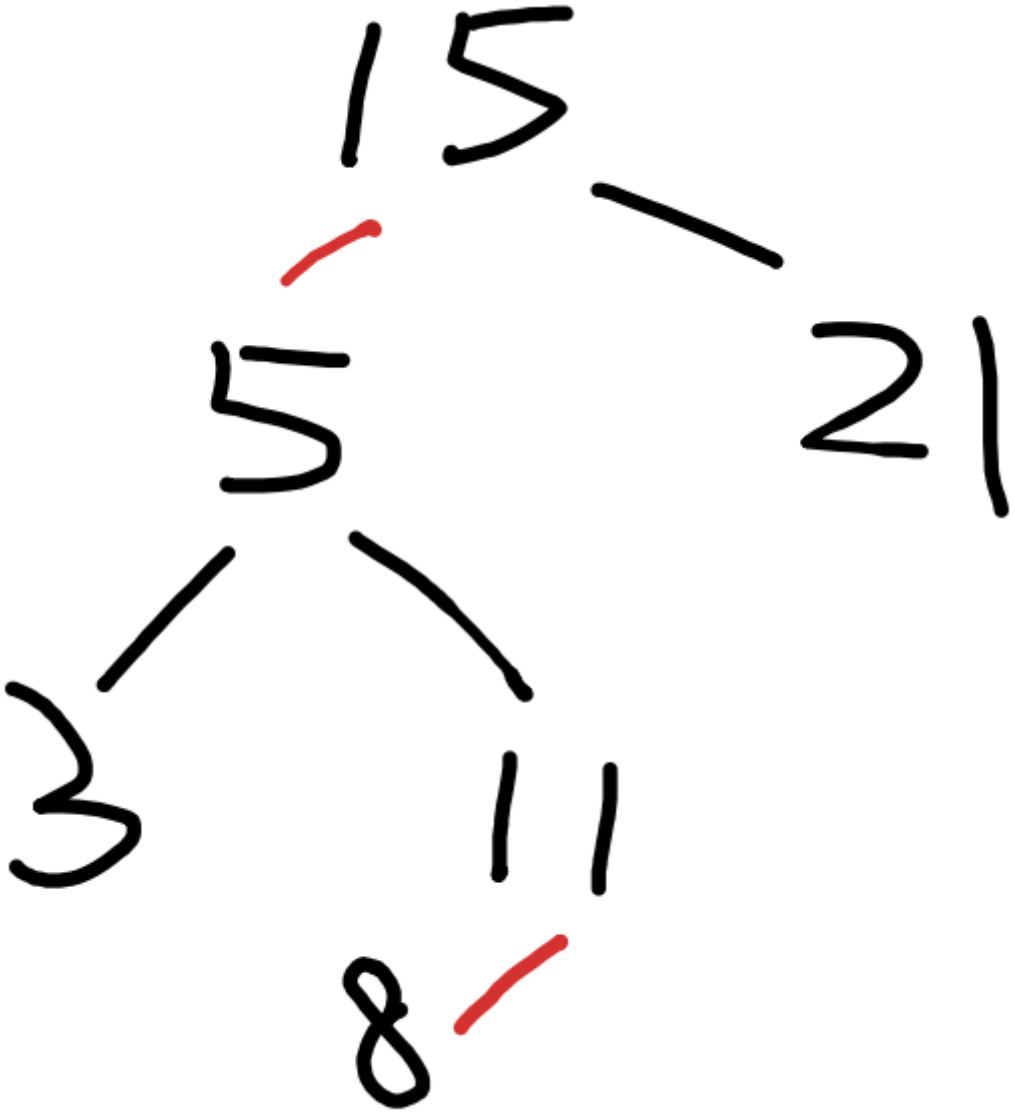
3.2 **(b)** **5** / 12 pts

✔ **+ 5 pts** Correct 2-3 tree

+ 5 pts Correct shape/keys of red-black tree



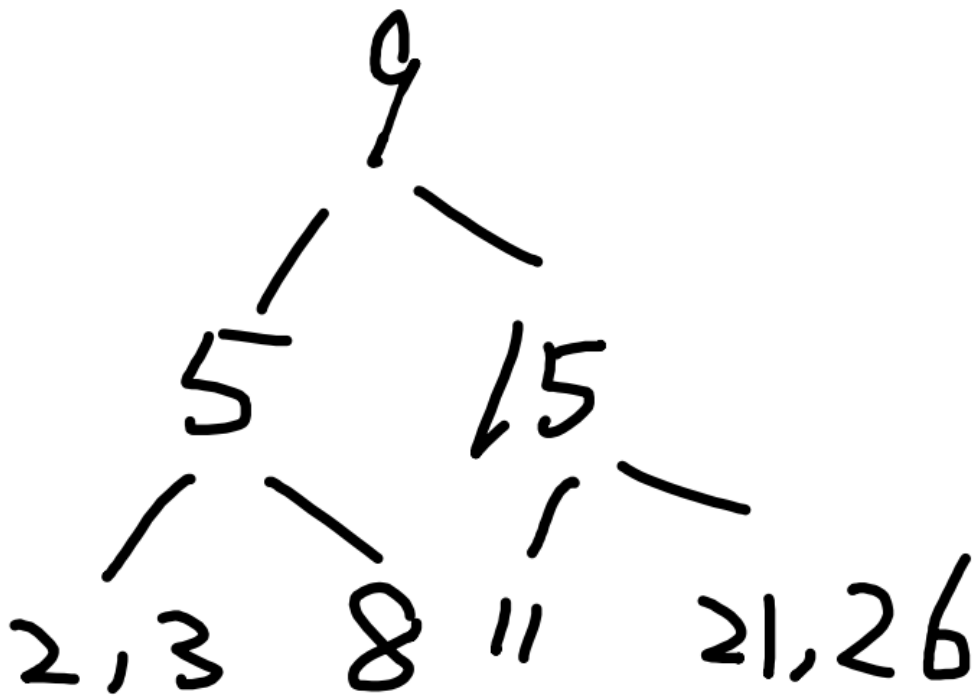
+ 2 pts Correct link colors of red-black tree



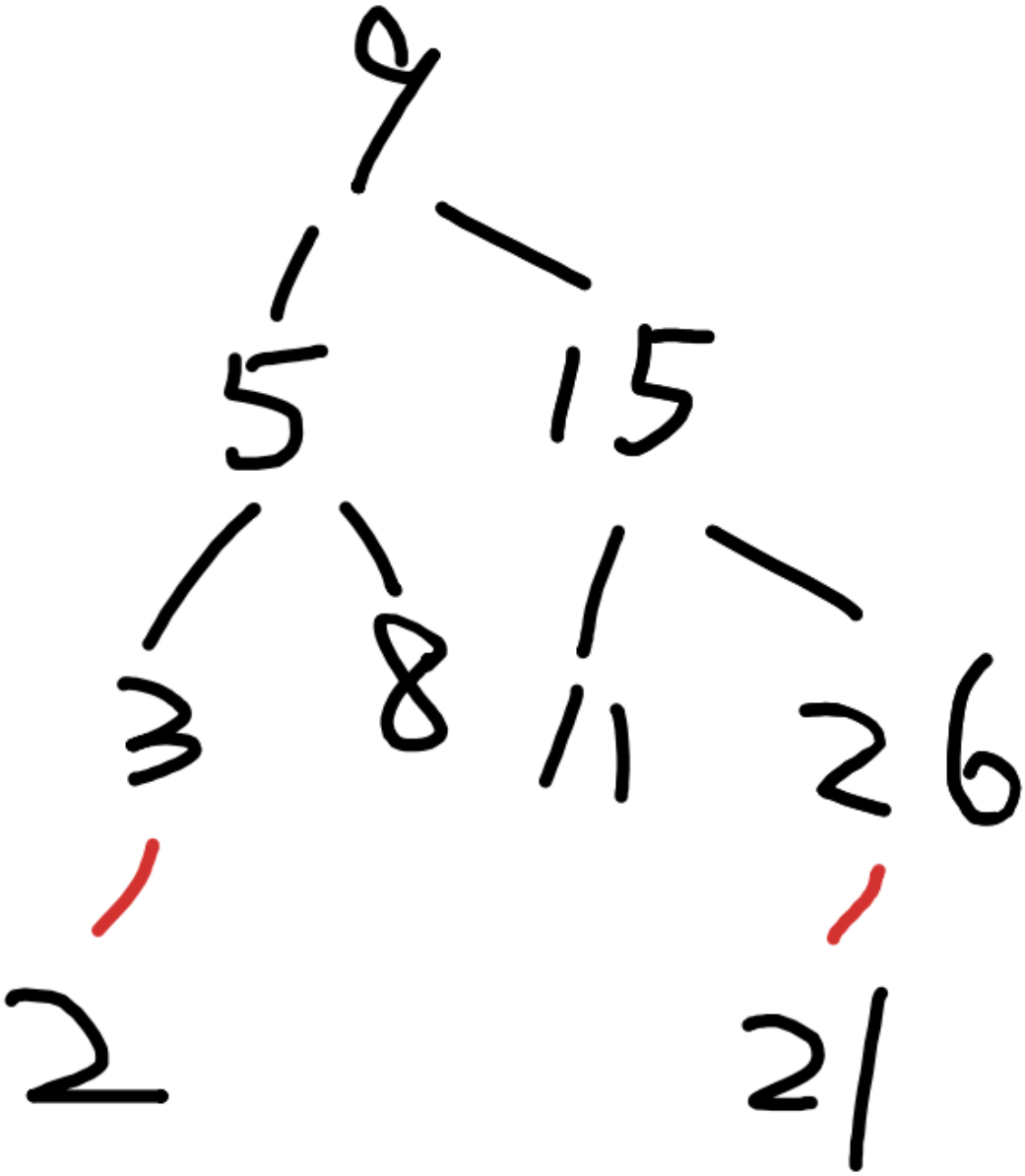
+ 0 pts Incorrect

3.3 **(c)** **5** / 14 pts

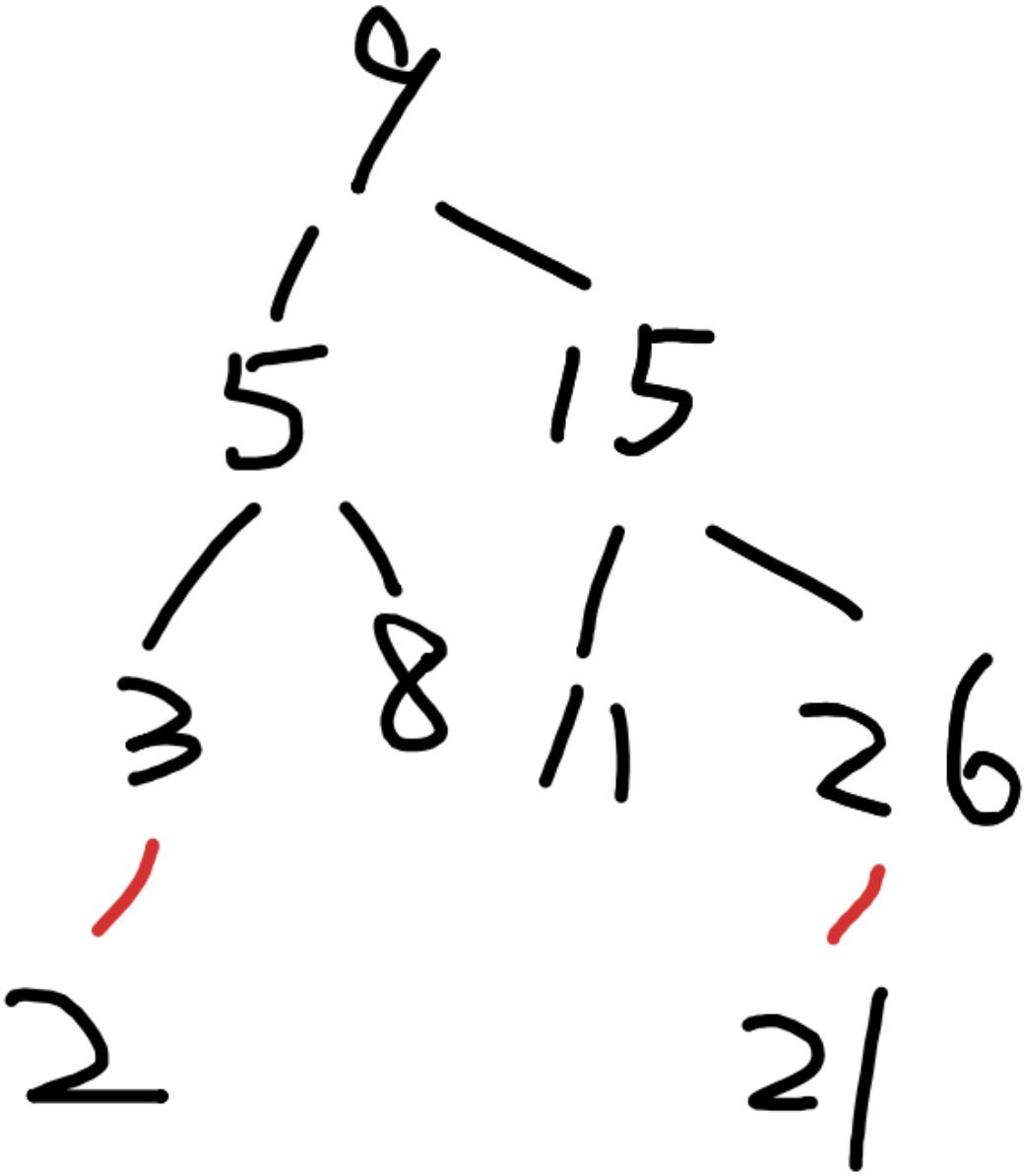
✔ **+ 5 pts** Correct 2-3 tree



+ 5 pts Correct shape/keys of red-black tree



+ 4 pts Correct link colors of red-black tree



+ 0 pts Incorrect

3.4 **(d)** **0** / 4 pts

+ 4 pts A. ~$\log_3(n)$

✔ **+ 0 pts** Incorrect

Question 4

Problem 4 - Priority Queue **13** / 30 pts

4.1 (a) **5** / 5 pts

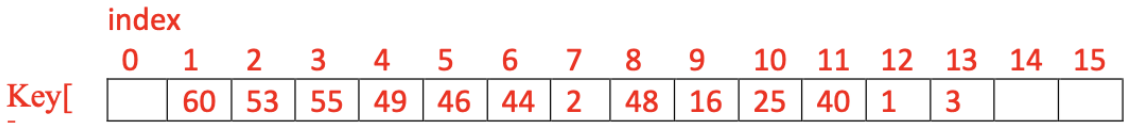
✔ **+ 1 pt** NO

✔ **+ 4 pts** 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

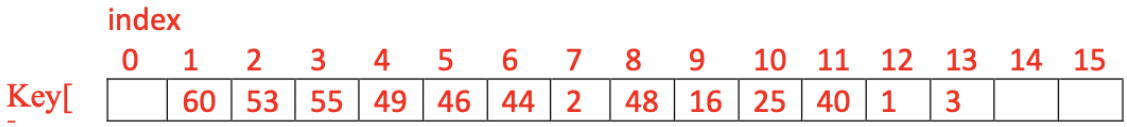
+ 0 pts Incorrect

4.2 (b.1) **0** / 10 pts

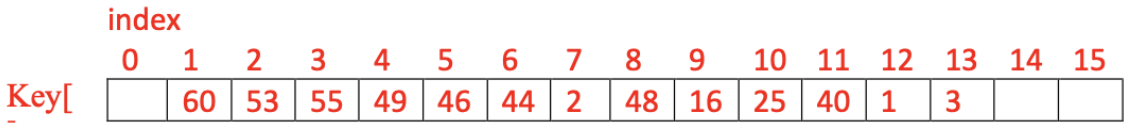
+ 2 pts Index 3 has value 55



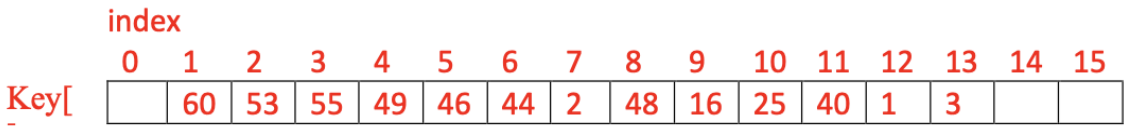
+ 2 pts Index 6 has value 44



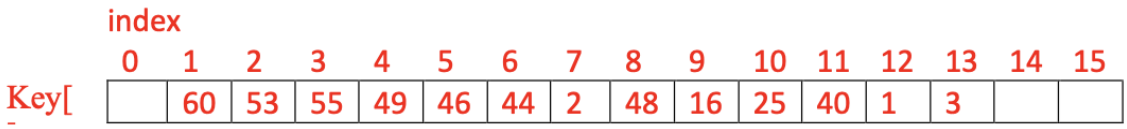
+ 2 pts Index 12 has value 1



+ 2 pts Index 13 has value 3



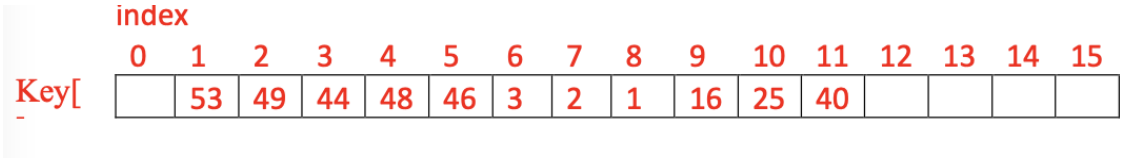
+ 2 pts All other keys match



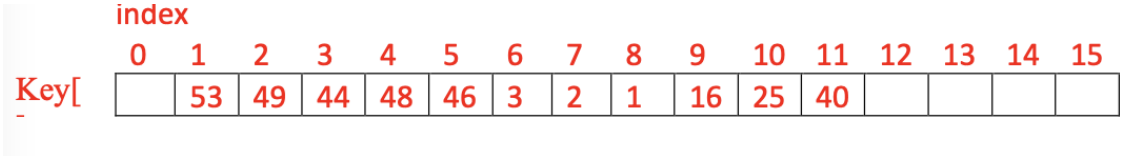
✔ **+ 0 pts** Incorrect

4.3 (b.2) Resolved 4 / 10 pts

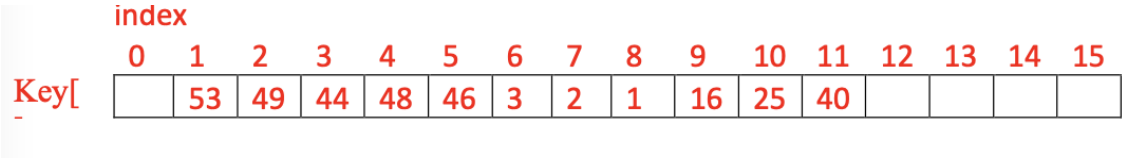
✔ + 1 pt Index 1 has value 53



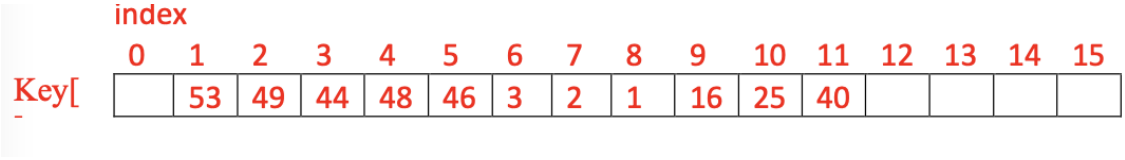
✔ + 1 pt Index 2 has value 49



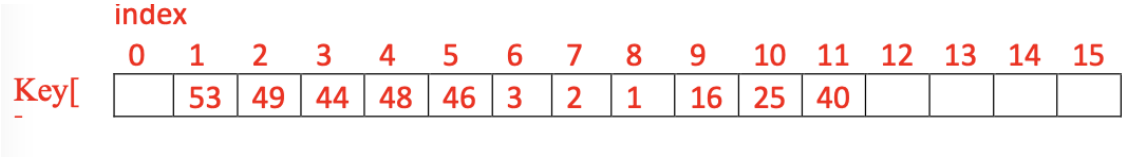
+ 1 pt Index 3 has value 44



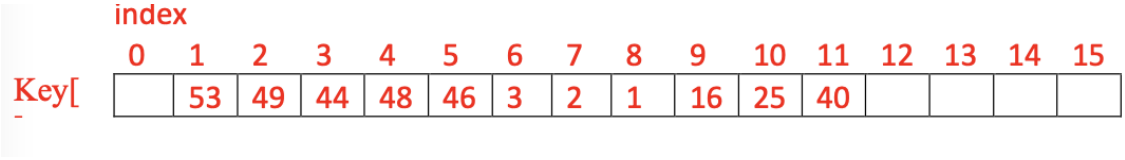
+ 1 pt Index4 has value 48



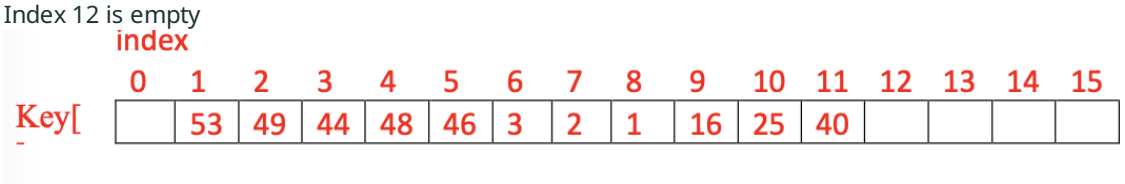
+ 1 pt Index 6 has value 3



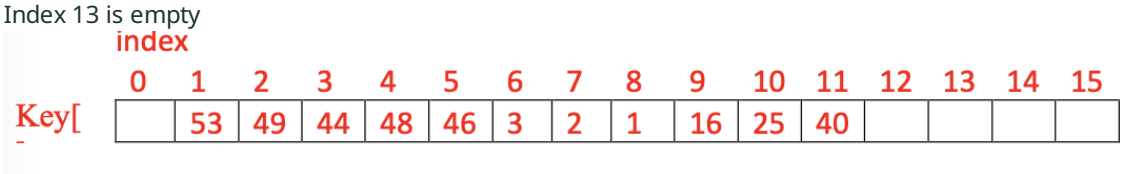
+ 1 pt Index 8 has value 1



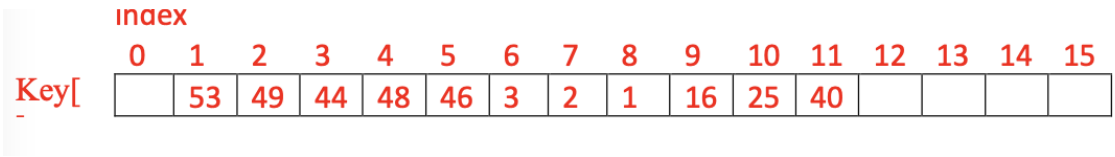
✔ + 1 pt Index 12 is empty



✔ + 1 pt Index 13 is empty



+ 2 pts All other keys match



+ 0 pts Incorrect

Regrade Request **Submitted on: Apr 28**

Is there a chance I can get a point because the last 3 keys match?

Sadly I can't, I was looking to give you points for the "all other keys match" part but since your index 5 is wrong I can't give points there.

Reviewed on: Apr 29

4.4 **(c)** **4** / 5 pts

✔ **+ 2 pts** $2 \log_2(n)$

+ 3 pts 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.

+ 0 pts Incorrect

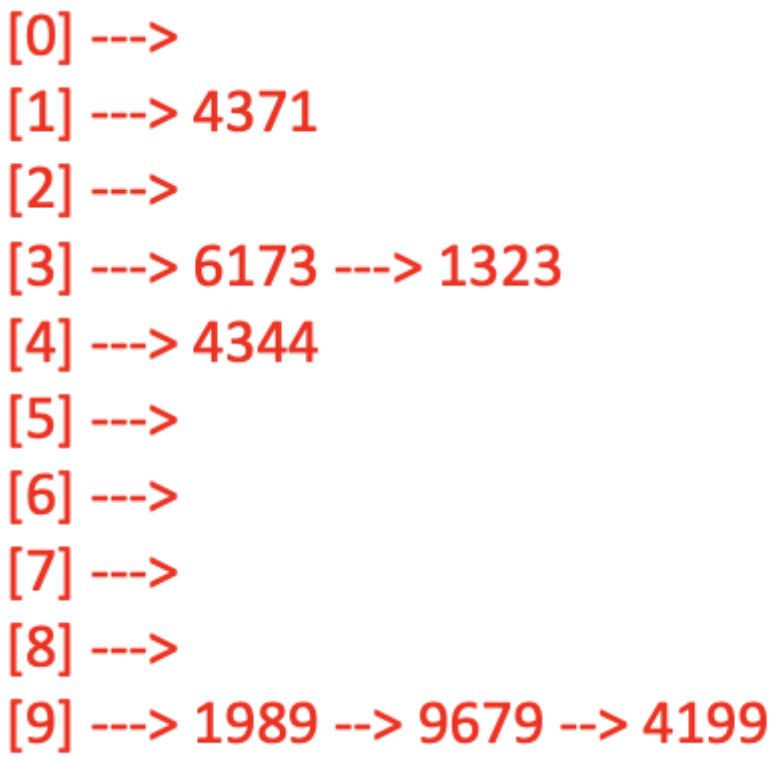
+ 2 pts The explanation is almost correct, just did not specifically mention the comparisons for the parent and child nodes.

Question 5

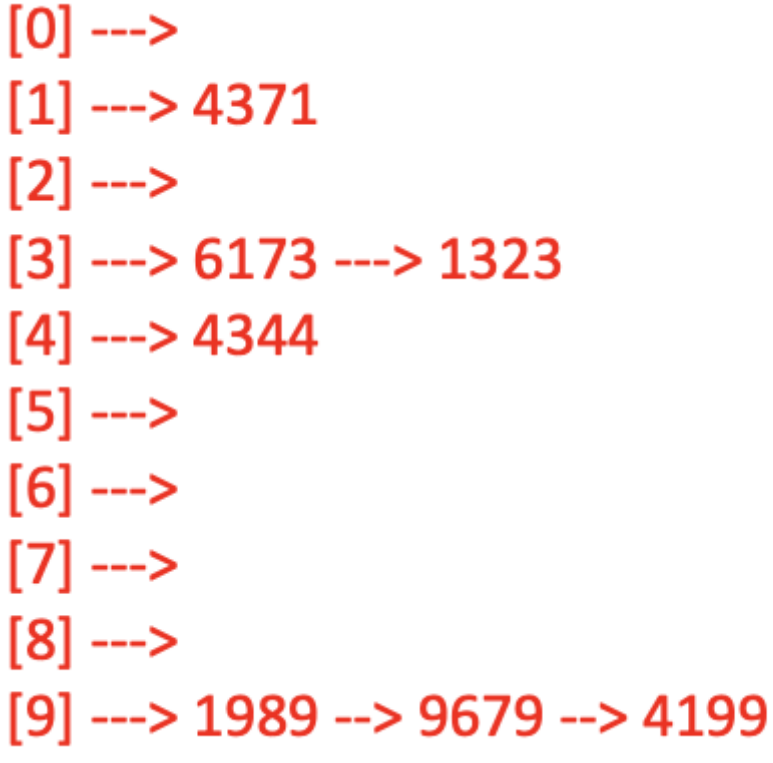
Problem 5 - Hash Tables **30** / 30 pts

5.1 (a) Resolved **10** / 10 pts

✔ **+ 1 pt** Index 1 has 4371



✔ **+ 1 pt** Index 3 has 6173



✔ **+ 1 pt** Index 3 has 1323

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

✔ **+ 1 pt** Index 4 has 4344

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

✔ **+ 1 pt** Index 9 has 1989

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

✔ **+ 1 pt** Index 9 has 9679

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

✔ **+ 1 pt** Index 9 has 4199

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

✔ **+ 3 pts** Linked lists for each array slot is in the correct order

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

+ 0 pts Incorrect

Regrade Request **Submitted on: Apr 28**

Is there any possibility that I can get 2 points for the other array slots being correct other

than 3?

points awarded

Reviewed on: Apr 30

5.2 **(b)** **10** / 10 pts

✔ **+ 1 pt** Index 0 has value 9679

index

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

✔ **+ 1 pt** Index 1 has value 4371

index

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

✔ **+ 1 pt** Index 2 has value 1989

index

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

✔ **+ 1 pt** Index 3 has value 1323

index

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

✔ **+ 1 pt** Index 4 has value 6173

index

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

✔ **+ 1 pt** Index 5 has value 4344

index

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

✔ **+ 1 pt** Index 9 has value 4199

index

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

✔ **+ 3 pts** The sequences of keys is identical to the solution

index

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

+ 0 pts Incorrect

5.3 **(c)** **10** / 10 pts

✔ **+ 3 pts** Index 0 has value 1989

index

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

✔ **+ 1 pt** Index 1 has value 4371

index

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

✔ **+ 1 pt** Index 3 has value 1323

index

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

✔ **+ 2 pts** Index 4 has value 6173

index

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

✔ **+ 2 pts** Index 5 has value 4344

index

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

✔ **+ 1 pt** Index 9 has value 9679

index

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

+ 0 pts Incorrect

CS112 Data Structures - Midterm 2 - Spring 2022

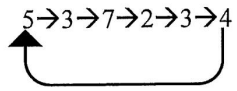
Name: Akshaj kammari NetID: gk 1990

- **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.

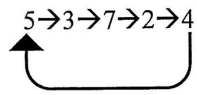
Problem 1 – Special Linked Structures (20 points)

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):



Resulting list after deleting the last occurrence of 3 (`rear` points to 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: Akshaj kammari NetID: ak1990

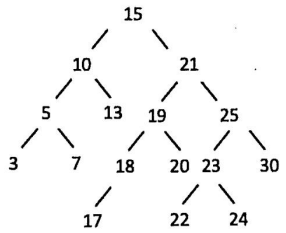
```
public void deleteLast Occurance (int item)

  While (ptr != head.data) {
ptr = head.next;
ptr2 = ptr.next;
while ( ptr2 != ptr) {
    ptr.next; }
if (ptr2 = ptr) {
return ptr2;
}
return rear;
```

Name: Akshaj kammari NetID: ak1990

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
}
```

```
Key k = h.key;
if((k*2).data == null){
  return 0;
  }
else{return (k*2).data;}
if((k*2+1).data == null){
return 0;
}
else{return (k*2+1).data;}

return 0;
```

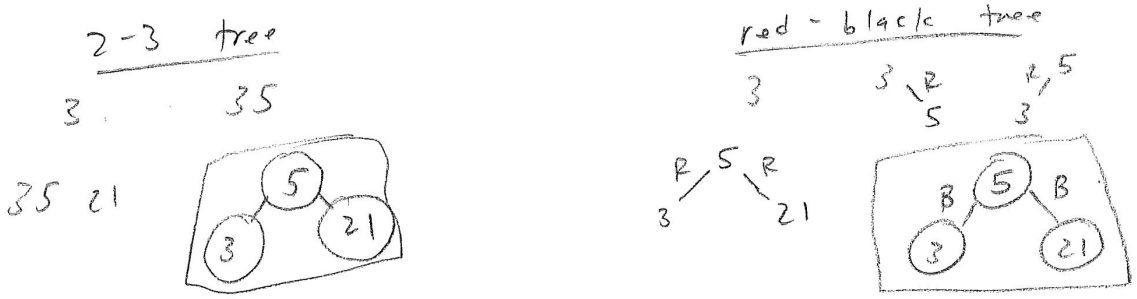
Name: Akshaj Kammari NetID: ak1990

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

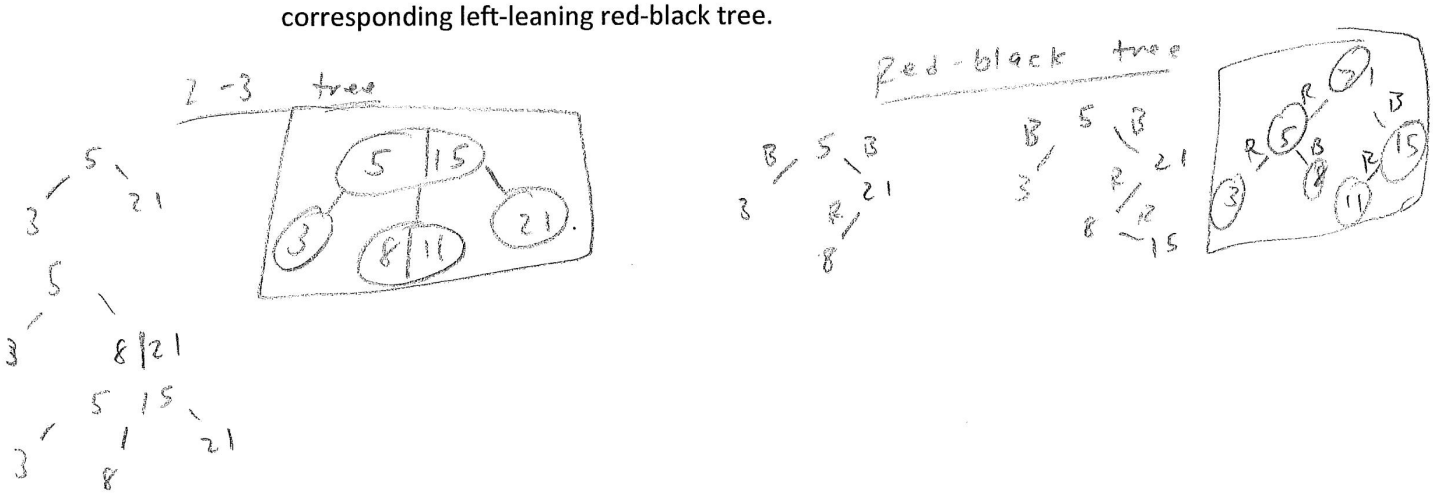
Construct a 2-3 tree and the corresponding left-leaning red-black tree whose keys are inserted in the following sequence. Label the links as R (red) or B (black), or use color in your answer.

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.



(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.

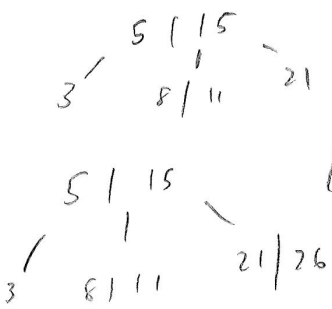


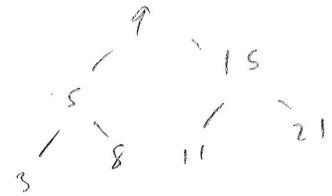
Name: Akshaj Kamman NetID: ak1990

3 5 21 8 15 11 26 9 2

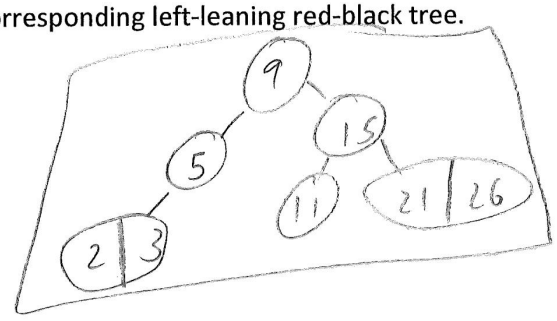
(c) **(14 points)** Draw the 2-3 tree after the insertion sequence completed, and the corresponding left-leaning red-black tree.

2-3 tree





Red - Black 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

Name: Akshaj Kammari NetID: ak1990

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, because the parent of index 8 & 9 is smaller than its value

(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.

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

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

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

Name: Akshaj Kammari NetID: ak1990

(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.

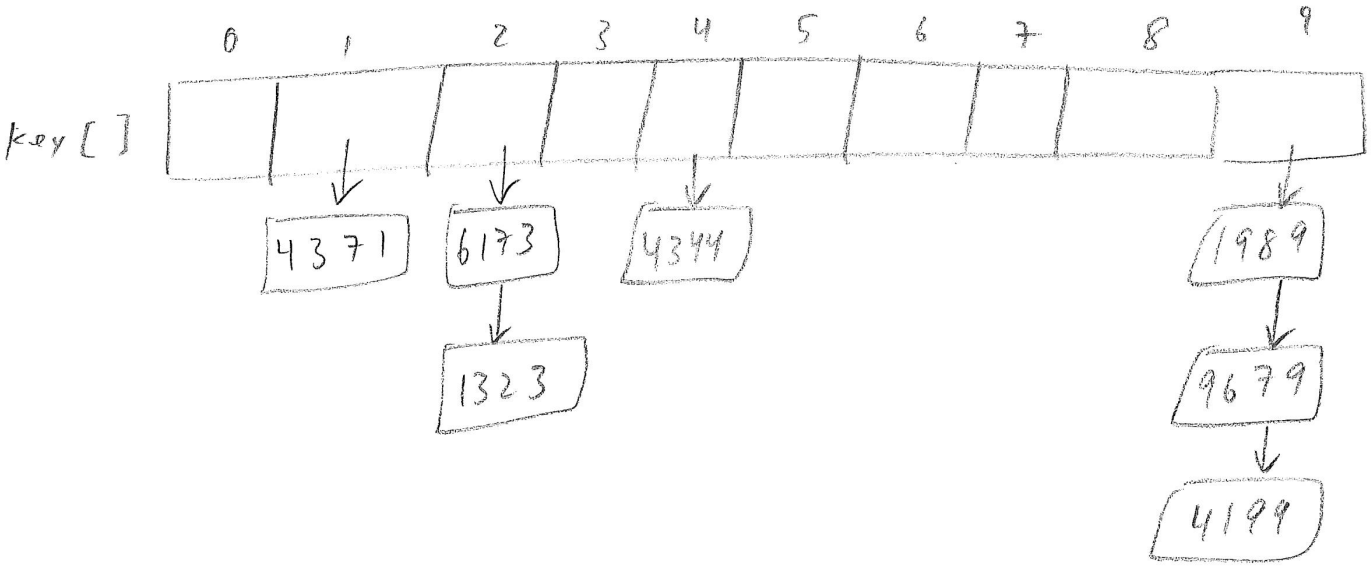
2 log n compares because the the value at the top of the key would need to be sunk down & the new max which is may have been at the bottom would need to swim up

Name: Akshaj Kammari NetID: ak1990

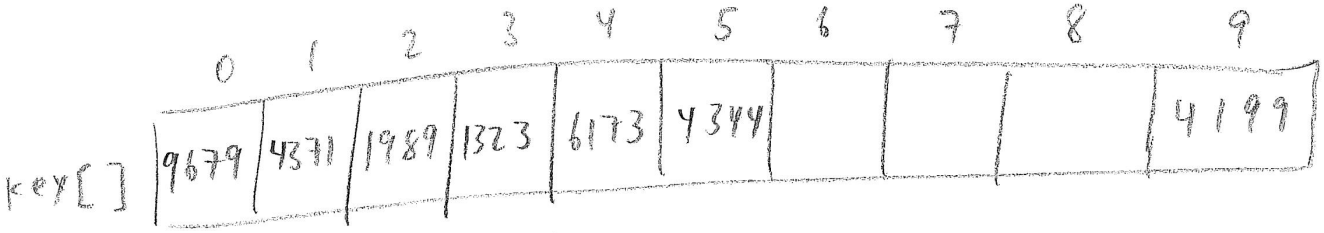
Problem 5 – Hash Table (30 points)

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).



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



Name: Akshaj kammari NetID: ak1990

(c) **(10 points)** Assuming 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.

```
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--;
}
```

key[ ]

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