Name: ANSWER KEY

State here the problem you are eliminating:

CS 112—Final Exam—Summer 1, 2017

There are 8 problems on the exam. The first and last are mandatory, and **you *must* eliminate any one of problems 2 – 7 by drawing an X through it.** Problem 1 is worth 10 points, and all other problems are worth 15. Please write in pen if possible. Circle answers when they occur in the midst of a bunch of calculations. If you need more room, use the back of the sheet and tell me this on the front sheet.

Problem One. (True/False – MANDATORY – 10 points) Write True or False to the left of each statement.

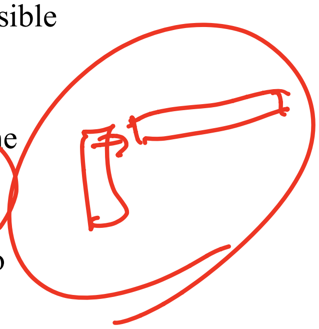
T 1. ~~The largest key in a 2-3 tree always occurs at a~~ leaf node. (WAY TOO HARD FOR THIS TIME)

REASON: ALL INSERTS AT ROOT, ONLY MEDIAN VALUES GET PUSHED UP.

F 2. A separate-chaining hash table can never have a load factor larger than 1.0.

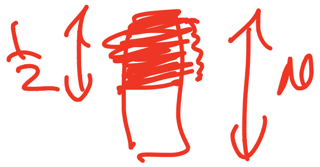
T 3. Selection sort performs the same number of comparisons of keys for all possible inputs.

F 4. A separate-chaining hash table containing N keys and M buckets (slots in the array) must have a worst-case lookup time of O(N / M). O(N)



T 5. The smallest key in a Max Heap always occurs at a leaf node, if there are no duplicates.

F 6. The smallest key in a binary search tree always occurs at a leaf node.



F 7. A perfect binary tree (perfect triangle) of height H will have 2^H -1 nodes.

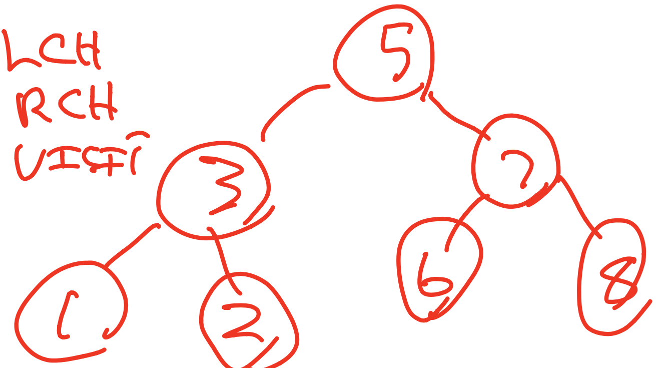
T 8. If a linear-probing hash table has a load factor of 0.5, then the worst-case lookup time for a key will be O(N), where N is the number of keys.

F 9. For any binary tree, if we list the nodes using a preorder traversal, and then list the nodes in postorder, these two lists will be the reverse of each other (i.e., if one produces a, b, c, d, then the other will be d, c, b, a)

F 10. A recursive definition can have at most one base case, but can have any number of recursive cases.

PRE
VISIT
LCH
RCH

POST
LCH
RCH
VISIT



5 3 1 2 7 6 8

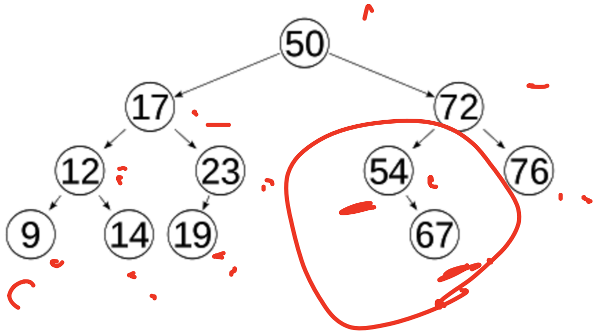
8 6 7 2 1 3 5

1 2 3 6 8 7 5

RCH

LCH

Problem Two (Binary Search Trees – 15 points). This problem has multiple parts, all referring to the diagram of the tree on the right; each part is independent and refers to the original diagram, i.e., start all over again with the original tree diagram for each part.



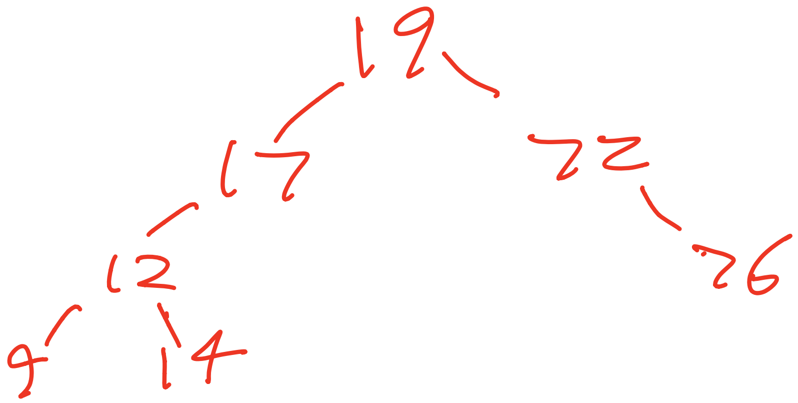
(A) Supposing that we can only insert integers into this tree, and no duplicates, what keys could be inserted to the immediate left of the node containing 54?

51, 52, 53

(B) Give a breadth-first (level order) traversal of this tree.

50 17 72 12 23 54 76
9 14 19 67

(C) As discussed in lecture, it is useful when deleting nodes from a BST that we alternately move up nodes from the right and left subtrees of the node to be deleted; draw the tree that results from deleting the root of the tree in the diagram four times, alternately moving up a node from the right, left, right, and left subtrees of the root.



(D) We studied six different recursive tree traversals this semester. For of the following listings of the keys in the tree given above, either give the name of the traversal that produced it, or state that “no possible traversal exists” if it could not be produced by one of the six.

(i) 76 67 54 72 19 23 14 9 12 17 50 REV. POSTORDER

(ii) 50 17 23 19 12 14 9 72 76 54 67 NONE X

(iii) 9 14 12 19 23 17 54 67 76 72 50

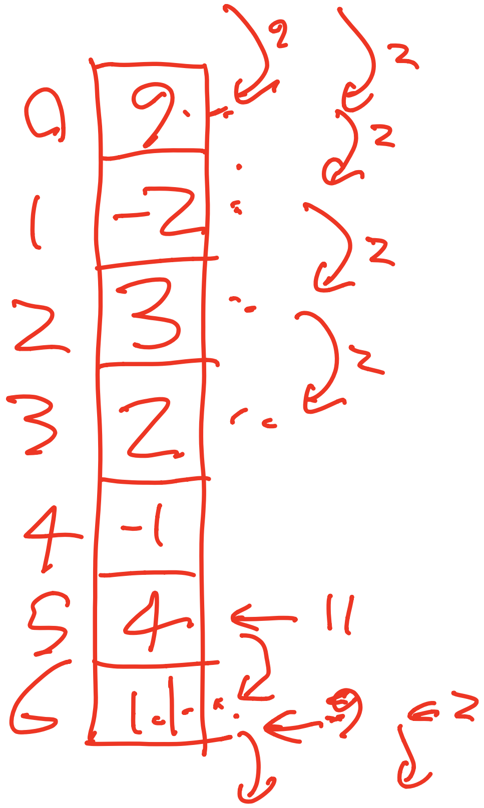
(iv) 50 72 76 54 67 17 23 19 12 14 9 REV. PREORDER

Problem Three (Hash Tables – 15 points). For a hash table implemented using the technique of **Linear Probing**, assume that you have an array size of 7 and a hash function

```
int hash(int key) {  return  ( 3 * key % 7 );  }
```

Suppose that you want to perform the following sequence of operations:

```
insert( 4 );
insert( 3 );
insert( 7 );
insert( 11 );
insert( 9 );
insert( 2 );
delete( 7 );
delete( 9 );
insert( 9 );
```



KEY	*3	%7
4	12	5
3	9	2
7	21	~~0~~
11	33	5
9	27	6
2	6	6
~~7~~		
~~9~~		
~~9~~		

(A) Show in the space above the result of performing these operations on an initially-empty table. Use ~~0~~ as a sentinel for "never used" and ~~1~~ for "used but currently empty."

-1 -2

3 7

(B) Recall that a hash function has the form (P * key % M) for suitable P and M. What would be the effect on a separate-chaining hash table's performance if P = 2 * M ? Be specific about the structure of the table and give the O(.....) estimated cost for looking up an item in such a table.

ALL ITEMS HASH TO BUCKET 0

O(N)

P = 14

M = 7

(C) Precisely how many slots in the array do you have to examine in each of the following tests for the hash table you got at the end of (A)?

9: 2

7: 5

2: 3

5: 4

2	3
3	2
4	-1
5	4
6	11

member(9)

member(2)

member(5) [unsuccessful test]

Problem Four (Max Heaps – 15 points). Show the Max Heap that would result after the following series of operations; you should work this out by using a tree structure to do the operations, and then (A) fill in the array representation for this final tree, (B) draw your final tree, and (C) answer the general questions below.

1. insert(8);
2. insert (3);
3. insert (9);
4. insert (5);
5. insert (6);
6. n = getMax(); // delete the root and assign its key to n
7. insert(12); ✓
8. insert(7);
9. m = getMax(); // delete the root and assign its key to m

(A)

0	1	2	3	4	5	6
8	6	7	3	5		

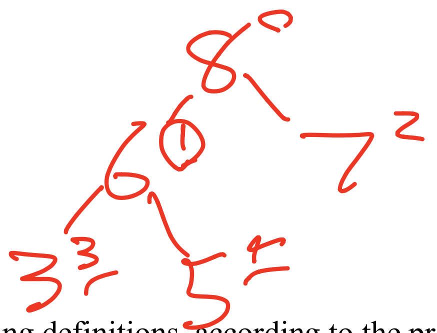
next: 5

Note: The array may not be entirely filled by the operations given.

n 9

m 12

(B) Draw your final tree:



(C) Complete the following definitions, according to the presentation in lecture on heaps:

// given index k in the array, return index of parent of this key

int parent(int k) { return (k-1)//2 }

// given index k in the array, return index of right child of this key

int rchild(int k) { return (2*k)+2 }

// given index k in the array, return index of left child of this key

int lchild(int k) { return (2*k)+1 }

Problem Five (Write an algorithm – 15 points). Let us say that two binary trees are symmetric if they are mirror images of each other (i.e., one is reversed left to right with respect to the other, as shown in the example at the bottom of the page). Write a recursive function

```
boolean isSymmetric(Node root1, Node root2) {

        ......

}
```

which takes two binary trees and returns true if they are symmetric with respect to each other. Note the following

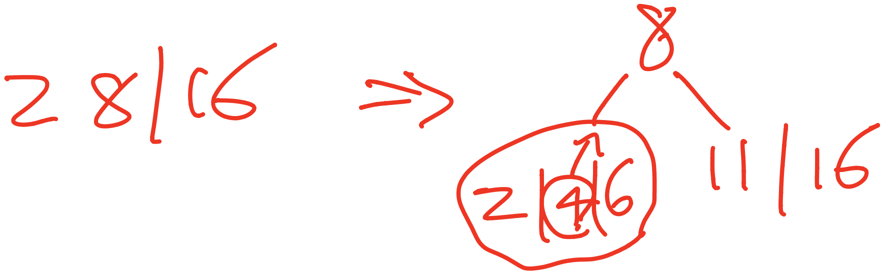
- You may not use a loop (the function or the helper function must be recursive).
- You may write a helper-function for this if you wish (meaning, the function above is a wrapper around another function).
- The assumed Node class is shown at right.
- You must account for the null pointer; two null trees are symmetric with respect to each other.
- You may NOT write a reverse function and then check for equality; you may write at most one recursive function to solve this problem.

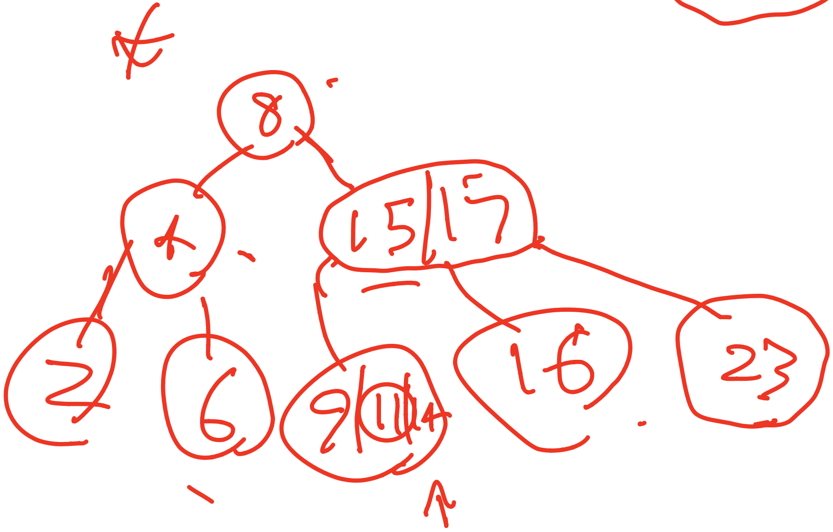
```
class Node {
    int key;
    Node left;
    Node right;
}
```

```
BOOLEAN ISSYM(NODE R1, NODE R2){
    IF (R1 == NULL && R2 == NULL)
        RETURN TRUE;
    ELSE IF (R1 == NULL || R2 == NULL)
        RETURN FALSE;
    ELSE
        RETURN ((R1.KEY == R2.KEY)
            && ISSYM(R1.LCHILD, R2.RCHILD)
            && ISSYM(R2.RCHILD,
                     R1.RCHILD));
}
```

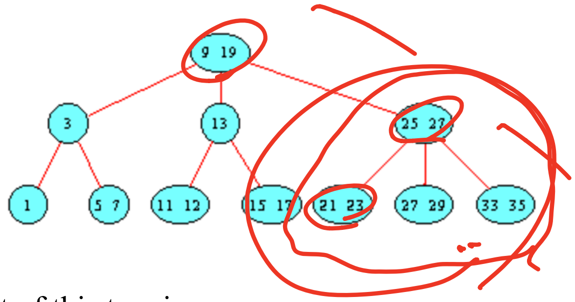
Problem Six (2-3 Trees -- 15 points). Assume you need to insert the integer keys 2, 8, 16, 11, 6, 4, 15, 9, 17, 23 in that order into an initially-empty 2-3 tree.

1. Show the data structure that would result after inserting these integers (use back of previous page or scrap paper for preliminary steps, and CIRCLE your final tree.





2. True or false? Suppose you have a key K < 21. Is it true that inserting K into the tree on the right could never cause its height to increase?



FALSE K=20 SPLITS ALL WAY UP.

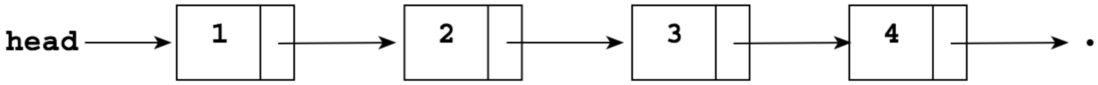
3. Give an example of a key which would make the height of this tree increase or write that "no such key exists."

31

4. In general what is the minimum number of keys we would need to insert into a 2-3 tree to guarantee that the tree is of height at least 2?

>8

Problem Seven (Algorithm Trace -- 15 points). Consider the following linked list:



Show what gets printed out by the following algorithm when applied to this list:

```
void mystery(Node p) {
      if (p == null) {
          System.out.println( "---");
      }
      else if (p.item % 2 == 1) {
          mystery(p.next);
          System.out.println(p.item);
      }
      else {
          System.out.println(p.item);
          mystery(p.next);
          mystery(p.next);
      }
}
```

---

odd

p.item even

((REST) ITEM) odd

(ITEM (REST) (REST)) EVEN

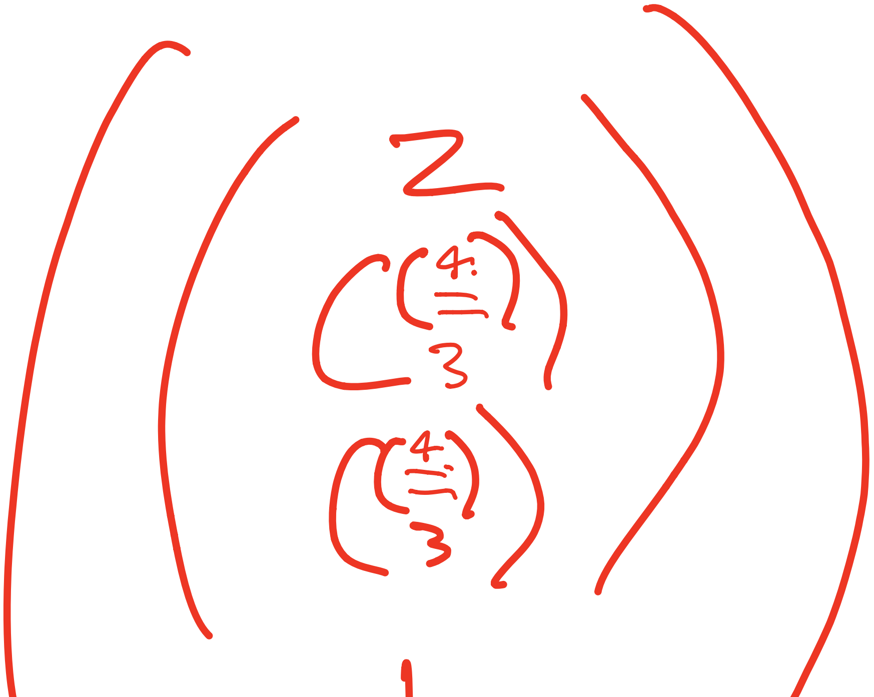
MYSTERY(HEAD)

1 (2 3 4)

2 (3 4)

3 (4)

4



2
4

3

Problem Eight. (MANDATORY – 15 points) For the following algorithms, for the first set, state the average- and worst-case time (in terms of Θ) for performing the indicating operation. The number of items stored in each data structure is N. Assume that none of the operations produces an error. Those entries in the table contain "X" do not have to be answered. For this question, your Θ(......) estimate is counting the number of times you must **touch** an element (e.g., to compare it or move it or print it out, etc.).

"Already-ordered" means that it is already solved, i.e., in the same order as it would be at the end of the sorting process.

Algorithm/Problem	**Average Case O(......)**	**Worst Case O(......)**
Find an item in an ordered array using Binary Search	X	O(LOG N)
Insert an item into an unordered list if duplicates are *not* permitted.	O(N) *	O(N)
Insert an item into an unordered list if duplicates *are* allowed.	O(1) *	O(1) *
Print out all the nodes in a Binary Search Tree using preorder search.	O(N)	O(N)
Sort an already-ordered list using Mergesort	O(N LOG N) *	O(N LOG N)
Sort an arbitrary list using Mergesort	"	"
Sort an already-sorted list using Selection Sort	$O(N^2)$	$O(N^2)$
Sort an already-sorted list using Insertion sort.	O(N)	O(N)
Delete an item from a max-queue implemented as a heap.	X	O(LOG N)
Dequeue an item from a queue.	O(1)	O(1)
Enqueue an item into a queue implemented as a ring buffer with resizing.	X	O(N) TRIGGERS RESIZE
Insert an item into a linear-~~probing hash table with a~~ load factor of 1.0.	X	O(N)
Delete an item from a Binary Search Tree.	O(LOG N)	O(N)
Insert an item into a 2-3 tree.	X	O(LOG N)
Delete an element from a separate-chaining hash table (with the assumptions from lecture)	O(1)	O(1)