CS 1332 Exam 2A

Jordan Shartar

TOTAL POINTS

71 / 100

QUESTION 1

1 Hash Table - Diagram 15 / 15

✓ **+ 3 pts** **Resize the array**

✓ **+ 5 pts** **Resize the array to a new capacity of 11, OR just add 23 to index 3**

✓ **+ 2 pts** **Put 12 at index 1**

✓ **+ 3 pts** **Put 34 at index 2**

✓ **+ 2 pts** **Put 23 at index 5**

+ 0 pts Incorrect / No Answer

QUESTION 2

AVL - Diagram 15 pts

2.1 Add - double rotation 6 / 6

✓ **+ 1 pts** **36 is in the new tree**

✓ **+ 1 pts** **The new tree is a valid BST**

✓ **+ 2 pts** **The new tree is a valid AVL**

✓ **+ 2 pts** **A double rotation was done about 30**

+ 0 pts Incorrect / No Answer

- 1 pts Losing extra nodes

2.2 Remove - double rotation 9 / 9

✓ **+ 1 pts** **94 is no longer in the tree**

✓ **+ 1 pts** **The tree is a valid BST**

✓ **+ 2 pts** **The tree is a valid AVL**

✓ **+ 2 pts** **81 is the new root**

✓ **+ 3 pts** **A double rotation was done about 76**

+ 0 pts Incorrect / No Answer

- 1 pts Extraneous modifications to the tree (unrelated node lost)

QUESTION 3

Tree Identification - Multiple Choice 15 pts

3.1 Heap 5 / 5

+ 0 pts BST

+ 0 pts AVL

✓ **+ 5 pts** **Heap**

+ 0 pts None of the Above / No Answer / Multiple Answers

3.2 AVL 5 / 5

+ 0 pts BST

✓ **+ 5 pts** **AVL**

+ 0 pts Heap

+ 0 pts None of the Above / No Answer / Multiple Answers

3.3 BST 5 / 5

✓ **+ 5 pts** **BST**

+ 0 pts AVL

+ 0 pts Heap

+ 0 pts None of the Above / No Answer / Multiple Answers

QUESTION 4

4 Build Heap 10 / 10

✓ **+ 1 pts** **Swaps 92 and 63**

✓ **+ 1 pts** **Swaps 81 and 44**

✓ **+ 1 pts** **Swaps 87 and 46**

✓ **+ 2 pts** **Swaps 87 and 59 (may be on same line as item above)**

✓ **+ 5 pts** **Final row is a valid min heap and correctly did build heap**

+ 4 pts Final row is a valid max heap and correctly did build heap algorithm

+ 0 pts Incorrect / No Answer

QUESTION 5

Efficiency - Matching 10 pts

5.1 Preorder 2 / 2

+ 0 pts O(1)

+ 0 pts O(log n)

✓ **+ 2 pts** **O(n)**

+ 0 pts O(n log n)

+ **0 pts** O(n^2)

+ **0 pts** No Answer

5.2 Resize HashTable 0 / 2

+ **0 pts** O(1)

+ **0 pts** O(log n)

✓ **+ 0 pts** **O(n)**

+ **0 pts** O(n log n)

+ **2 pts** O(n^2)

+ **0 pts** No Answer

5.3 BuildHeap 2 / 2

+ **0 pts** O(1)

+ **0 pts** O(log n)

✓ **+ 2 pts** **O(n)**

+ **0 pts** O(n log n)

+ **0 pts** O(n^2)

+ **0 pts** No Answer

5.4 Skip List 2 / 2

+ **0 pts** O(1)

✓ **+ 2 pts** **O(log n)**

+ **0 pts** O(n)

+ **0 pts** O(n log n)

+ **0 pts** O(n^2)

+ **0 pts** No Answer

5.5 Removing bad hash 0 / 2

+ **0 pts** O(1)

✓ **+ 0 pts** **O(log n)**

+ **2 pts** O(n)

+ **0 pts** O(n log n)

+ **0 pts** O(n^2)

+ **0 pts** No Answer

QUESTION 6

6 BST Ancestors - Coding 4 / 20

+ **2 pts** Throws an NoSuchElementException if and only if data isn't in the tree

✓ **+ 2 pts** **Is recursive**

✓ **+ 4 pts** **Recurses left AND right**

+ **5 pts** Only adds data from the node's ancestors to the list and adds all of these ancestors.

✓ **+ 2 pts** **Adds data of the desired node (can add parameter data)**

+ **5 pts** Adds all data in the correct order (from data to root)

+ **0 pts** Incorrect / No Answer

✓ **- 1 pts** **Syntax Error**

- 2 pts NullPointerException

✓ **- 3 pts** **Inefficient**

- 3 pts Adds node instead of data to list

1. After each recursion, current will still be current.left or current.right, so it will loop again and add unnecessary elements.
2. You use list.addToFron(), which in case of ArrayList will have O(n), which is inefficient.

QUESTION 7

7 Priority Queue - Coding 2 / 5

+ **1 pts** Constructor: assigns a new MinHeap to backingHeap with generics

✓ **+ 1 pts** **Enqueue: throws an IAException when item == null**

✓ **+ 1 pts** **Enqueue: calls backingHeap.add(item)**

+ **1 pts** Dequeue: throws a NSEException when backingHeap.isEmpty()

+ **1 pts** Dequeue: calls AND returns backingHeap.remove()

+ **0 pts** Incorrect / No Answer

QUESTION 8

8 Skip List - Diagram 4 / 10

✓ **+ 1 pts** **Has column of negative infinity**

+ **2 pts** Has 12, 21, 37, 69 in the bottom row

+ **2 pts** Has 12, 21, 37, 69 in order throughout the structure

✓ **+ 1 pts** **12 is only in the 2 bottom layers**

✓ **+ 1 pts** **21 is only in the 3 bottom layers**

✓ **+ 1 pts** **37 is only in the 4 bottom layers**

+ **1 pts** 69 is only in the bottom layer

+ **1 pts** All nodes are fully connected with straight lines or double-ended arrows (can miss up to 2 edges)

+ **0 pts** Incorrect / No Answer

CS 1332 Exam 2A
Spring Semester 2018 - March 7, 2018

Name (print clearly including your first and last name): Jordan Shartar

Signature: _____

GT account username (msmith3, etc): JShartar6

GT account number (903000000, etc): 903131050

- You must have your BuzzCard or other form of identification on the table in front of you during the exam. When you turn in your exam, you will have to show your ID to the TAs before we will accept your exam. It is your responsibility to have your ID prior to beginning the exam.
- You are not allowed to leave the exam room and return. If you leave the room for any reason, then you must turn in your exam as complete.
- Signing and/or taking this exam signifies you are aware of and in accordance with the Academic Honor Code of Georgia Tech and the Georgia Tech Code of Conduct.
- Notes, books, calculators, phones, laptops, smart watches, headphones, etc. are not allowed.
- Extra paper is not allowed. If you have exhausted all space on this test, talk with your instructor. There are extra blank pages in the exam for extra space.
- Pens/pencils and erasers are allowed. Do not share.
- All code must be in Java.
- Efficiency matters. For example, if you code something that uses O(n) time or worse when there is an obvious way to do it in O(1) time, your solution may lose credit. If your code traverses the data 5 times when once would be sufficient, then this also is considered poor efficiency even though both are O(n).
- Style standards such as (but not limited to) use of good variable names and proper indentation is always required. (Don't fret too much if your paper gets messy, use arrows or whatever it takes to make your answer clear when necessary.)
- Comments are not required unless a question explicitly asks for them.

This page is purposely left blank. You may use it for extra space, just mention on the page of the question that you want work here to be graded.

1) Hash Table - Diagram [15 points]

The hash table below is backed by an array of capacity 5, and has a current size of 2. Add **23** to the hash table. The maximum load factor for this hash table is **0.5**. If you need a collision resolution strategy, use **quadratic probing**. If you need to resize the table, resize it to a capacity of **(2 × initial capacity) + 1.** When resizing, **completely redraw** the backing array like the backing array given to you. The hashcode of a particular number is the number itself. The compression function is mod the table length.

size = 2
length = 5
load = 0.5

```
table[0] =
table[1] =
table[2] = 12
table[3] =
table[4] = 34
```

11 | 34
34
0

1 term
0 + 1² ✓

11 | 23
2
2 2
1

1 term
1 + 1 term
0 + 2² = 5 ✓

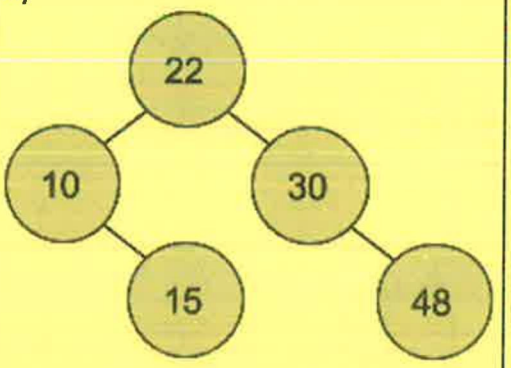
New table

```
table[0] =
table[1] = 12
table[2] = 34
table[3] =
table[4] =

table[5] = 23
table[6] =
table[7] =
table[8] =
table[9] =
table[10] =
```

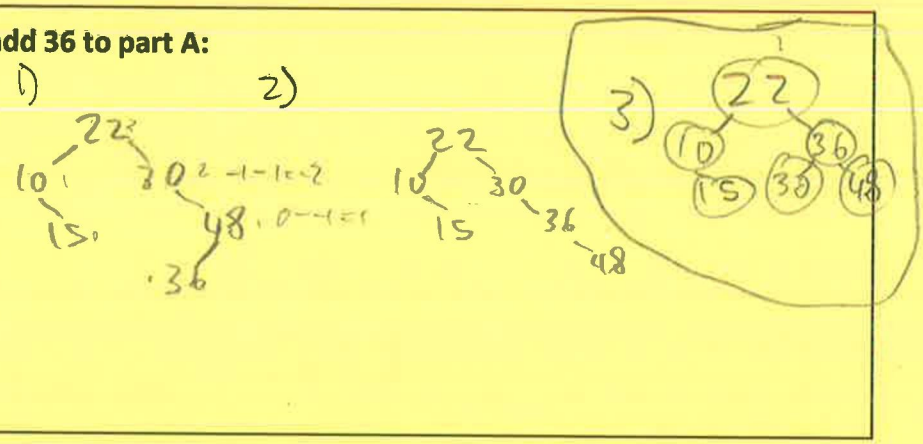
2) AVL - Diagram [15 points]

Given both of the following AVL trees below. Perform the specified add or remove operation and draw the new tree in the box to the right of the original tree. If you want to show multiple steps in a box, circle the final tree. If needed in an operation, use the **predecessor**.

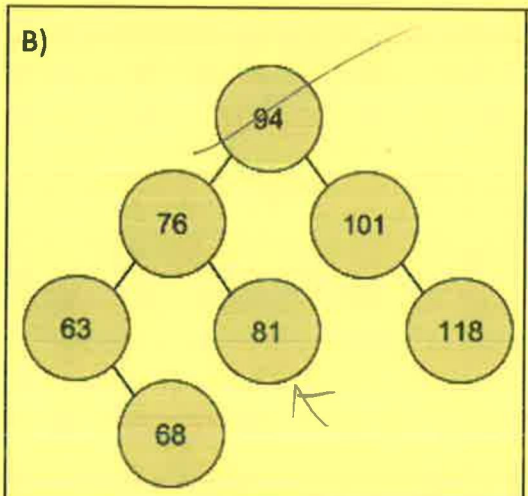
A)



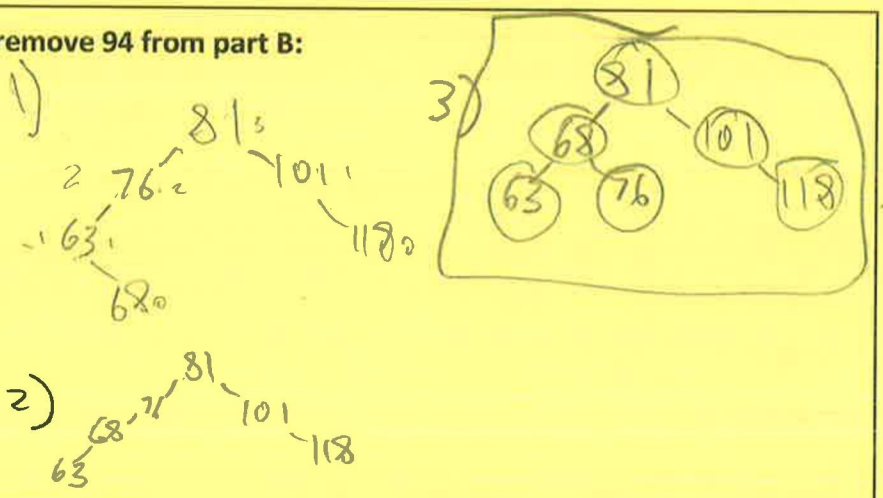
add 36 to part A:



B)



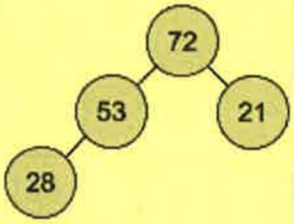
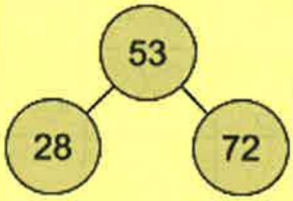
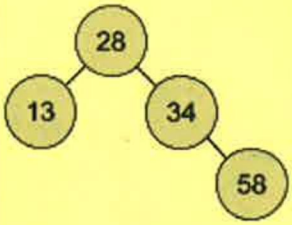
remove 94 from part B:

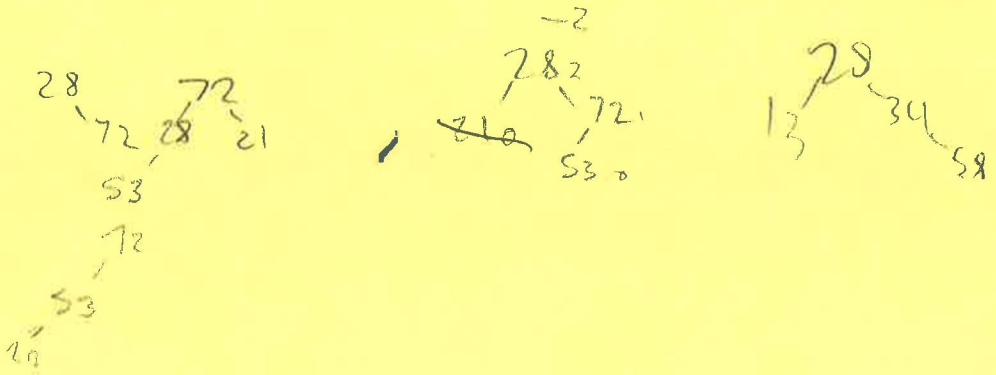


Do not write below this line. It may not be scanned, therefore it will not be graded.

3) Tree Identification - Multiple Choice [15 points]

Given the initial trees, the given operations that are performed on them, and the final trees that result after the operations: select the best option of which tree type each tree is.

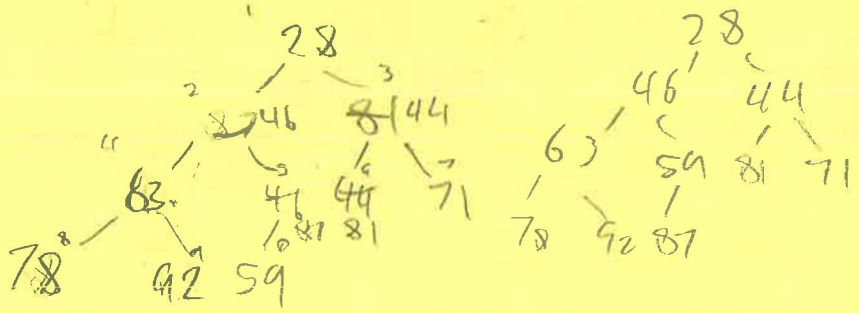
Initial Tree	Operations	Final Tree	Tree Type? (Choose one)
	Add - 72 Add - 21 Add - 53		○ Binary Search Tree ○ AVL Tree ● Heap ○ None of the Above
	Add - 72 Add - 21 Add - 53 Remove - 21		○ Binary Search Tree ● AVL Tree ○ Heap ○ None of the Above
	Add - 34 Add - 58 Add - 13		● Binary Search Tree ○ AVL Tree ○ Heap ○ None of the Above



4) Min Heap - Diagram [10 points]

The data below was put into an initial Min Heap via the constructor as a Collection. Perform the ***build-heap*** algorithm on the given array below to create the valid min-heap from the data. Show all intermediate steps/swaps on a new line in the grid. If the array changes, you should show the changed array. If the array doesn't change during a step, you do not need to rewrite the array.

Index 0	1	2	3	4	5	6	7	8	9	10
Initial Array	28	87	81	92	46	44	71	78	63	59
	28	87	81	63	46	44	71	~~78~~	92	59
	28	87	44	63	46	81	71	78	92	59
	28	46	44	63	87	81	71	78	92	59
	28	46	44	63	59	81	71	78	92	87



5) Efficiency - Matching [10 points]

For each of the operations listed below, determine the time complexity of the operation. Select the bubble corresponding to your choice in the space provided. Unless otherwise stated, assume the ***worst-case*** time complexity. However, make sure you choose the tightest Big-O upper bound possible for the operation. Do **not** use an amortized analysis for these operations.

A.) Performing a preorder traversal in an AVL Tree.

○ O(1) ○ O(log n) ● O(n) ○ O(n log n) ○ $O(n^2)$

B.) Resizing a hashtable using a linear probing collision strategy (Hint: Consider collisions upon resizing).

○ O(1) ○ O(log n) ● O(n) ○ O(n log n) ○ $O(n^2)$

C.) The BuildHeap Algorithm for a max heap.

○ O(1) ○ O(log n) ● O(n) ○ O(n log n) ○ $O(n^2)$

D.) Average case of searching in a skip list where data has been added with a 1/3 chance of being promoted each level.

○ O(1) ● O(log n) ○ O(n) ○ O(n log n) ○ $O(n^2)$

E.) Average case of removing from a hashtable using external chaining where the hash function always returns 0.

○ O(1) ● O(log n) ○ O(n) ○ O(n log n) ○ $O(n^2)$

6) BST - Coding [20 points]

Given the starter code below, you are to implement the helper method logic for the **ancestors()** method. Details on what the method should do are given below. Your **code** must be recursive. Do **not** add any import statements. Do **not** create any helper methods; all of your logic must be contained in the private method we've provided for you. You **must** be as efficient as possible. Do **not** assume any other methods are available to you that are not shown. There are no getters/setters; just **access the values directly** since Node is an inner class.

```
import java.util.*;
public class BST {

   private class BSTNode {
       int data;
       BSTNode left;
       BSTNode right;
   }

   private BSTNode root;
   private int size;

   /**
   * Retrieves the ancestors of a node sorted in order of ancestor distance with the
   * closest ancestors first, including the data itself. In other words, a list of a
   * node, that node's parent, that node's parent's parent, and so on in that order.
   *
   * DO NOT MODIFY THIS METHOD!
   *
   * Example:           4
   *                   /   \
   *                  2     6
   *                 / \   / \
   *                1   3 5   7
   *
   * Here, ancestors(5) should return (5, 6, 4)!
   *
   * @throws java.util.NoSuchElementException if the data is not in the BST
   * @param data the data to retrieve the ancestors of
   * @return the list of ancestors sorted by increasing distance from the data
   */
   public List<Integer> ancestors(int data) {
        List<Integer> list = new ArrayList<>();
        ancestorsHelper(root, list, data);
        return list;
   }


  // See next page for the method you need to implement!
```

```
/**
* Recursive helper method to retrieve the ancestors of the data.
*
* YOU MUST IMPLEMENT THIS METHOD RECURSIVELY! Do NOT change the method signature of
* this method.
*
* @throws java.util.NoSuchElementException if the data is not in the BST
* @param current the current node in traversing the BST
* @param list the list of ancestors
* @param data the data of which to find the ancestors
*/
private void ancestorsHelper(BSTNode current, List<Integer> list, int data) {
    If (size == 0) {
        throw new NoSuchElementException("data not in BST");
    }
    while (current != null) {
        if (data < current.data) {
            list.addFront(current.data);
            current = current.left;
            ancestorHelper(current, list, data);
        } else if (data > current.data) {
            list.addFront(current.data);
            current = current.right;
            ancestorHelper(current, list, data);
        } else if (data == current.data) {
            list.addFront(current.data);
            return
        }
        list = new ArrayList<>();
        throw new NoSuchElementException("data not in BST");
}
```

7) Priority Queue - Coding [5 points]

Given the following *MinPriorityQueue* class, fill in the constructor and the methods *enqueue* and *dequeue*. This Priority Queue (PQ) is backed by a *MinHeap<T>*. You may **NOT** import any classes or create any helper methods or instance variables (local variables are fine). You must be as efficient as possible.

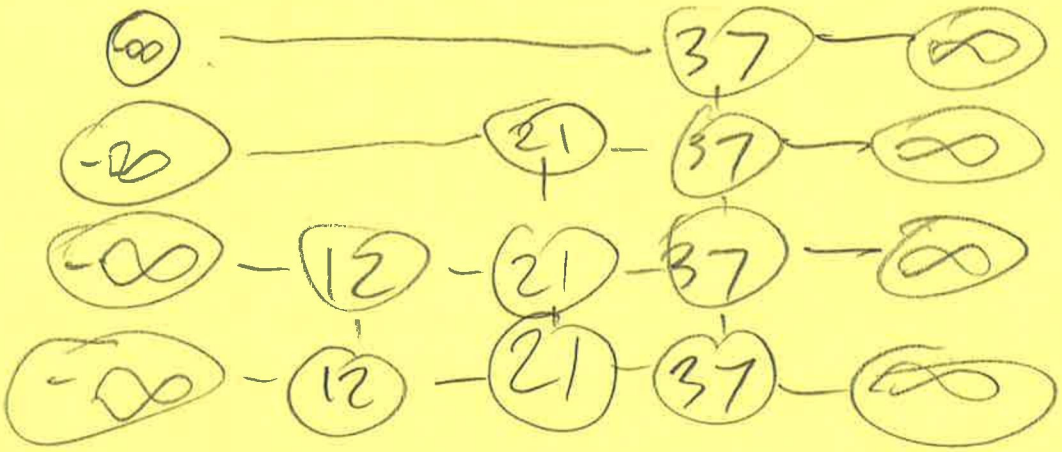
```
import java.util.*;
public class MinPriorityQueue<T extends Comparable<? superT>> {

    /**
    * A MinHeap of generic typing backed by an array.
    *
    * Has the following methods:
    * add(T data), remove(), isEmpty()
    */
     private MinHeap<T> backingHeap;

     /**
     * Constructor for the PQ. Fill in as necessary.
     */
     public MinPriorityQueue() {
          this.backingHeap =(T[]) new Comparable<>();


     }

     /**
     * Adds (@code item) to the PQ such that the minimum element is still the root
     * @throws java.lang.IllegalArgumentException if the parameter is null
     * @param item generic object to be added
     */
     public void enqueue(T item) {
          If (item ==null) {
               throw new IllegalArgumentException ("item is null cannot enqueue");
          }
     }    add (item);

     /**
     * Removes and returns the minimum element from the PQ such that the next smallest
     * element is the new root
     * @throws java.util.NoSuchElementException if the PQ is empty
     * @returns the minimum element from the PQ
     */
     public T dequeue() {
          IF (backingHeap[0] == null) {
               throw new NoSuchElementException("no item in root");
          }
     }    T out = backingHeap[0];
          remove();
}                                        return out;
```

8) Skip List - Diagram [10 points]

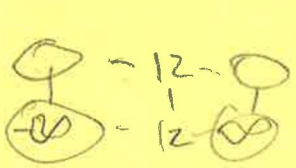
Given the following sequence of Heads and Tails, draw a diagram of the **final** Skip List produced by adding this data in this order: **12, 37, 21, 69**. You will not receive any credit for intermediate steps; **only the final Skip List will be graded**. Include -∞ to denote the beginning of each level. You may use a single line without arrow heads to denote pointers going in both directions. If you draw arrow heads, we will assume that it is a pointer in that direction. You do **not** need an empty level at the top of the Skip List. Interpret the coin flips as was taught in lecture.

HTHHHTHHTTHHHTHTTHHHTTTHTTHHTH

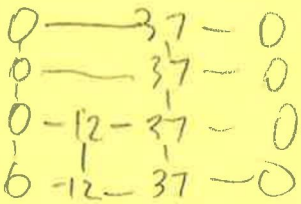
Final SkipList:



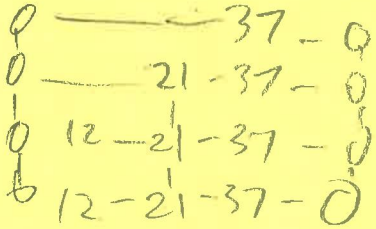
1)



2)



3)



Do not write below this line. It may not be scanned, therefore it will not be graded.

This page is purposely left blank. You may use it for extra space, just mention on the page of the question that you want work here to be graded.