

CS 1332 Exam 2B

TOTAL POINTS

91 / 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 28 to index 3
- ✓ + 2 pts Put 17 at index 6
- ✓ + 3 pts Put 39 at index 7
- ✓ + 2 pts Put 28 at index 10
- + 0 pts Incorrect / No Answer

QUESTION 2

2 Priority Queue - Coding 3 / 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 NSEEException when backingHeap.isEmpty()
 - ✓ + 1 pts Dequeue: calls AND returns backingHeap.remove()
 - + 0 pts Incorrect / No Answer
- 主义思想() should be called on the backingHeap

QUESTION 3

AVL - Diagram 15 pts

3.1 Add - double rotation 6 / 6

- ✓ + 1 pts 12 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 10
 - + 0 pts Incorrect / No Answer
 - 1 pts Changing unnecessary nodes

3.2 Remove - double rotation 9 / 9

- ✓ + 1 pts 72 is no longer in the tree
- ✓ + 1 pts The tree is a valid BST
- ✓ + 2 pts The tree is a valid AVL
- ✓ + 2 pts 59 is the new root
- ✓ + 3 pts A correct double rotation was done about 54
 - + 0 pts Incorrect / No Answer
 - 1 pts Losing extra nodes

QUESTION 4

4 Skip List - Diagram 10 / 10

- ✓ + 1 pts Has column of negative infinity
- ✓ + 2 pts Has 19, 28, 44, 76 in the bottom row
- ✓ + 2 pts Has 19, 28, 44, 76 in order throughout the structure
- ✓ + 1 pts 19 is only in the 2 bottom layers
- ✓ + 1 pts 28 is only in the 3 bottom layers
- ✓ + 1 pts 44 is only in the 4 bottom layers
- ✓ + 1 pts 76 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

QUESTION 5

Tree Identification - Multiple Choice 15 pts

5.1 AVL 5 / 5

- + 0 pts BST
- ✓ + 5 pts AVL
- + 0 pts Heap
- + 0 pts None of the Above / No Answer / Multiple Answers

5.2 BST 0 / 5

- + 5 pts BST
- + 0 pts AVL

+ 0 pts Heap

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

- It cannot be an AVL based on the order the items are added.

5.3 Heap 5 / 5

+ 0 pts BST

+ 0 pts AVL

✓ + 5 pts Heap

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

QUESTION 6

6 Build Heap 10 / 10

✓ + 1 pts Swap 28 and 71

✓ + 1 pts Swap 46 and 87

✓ + 1 pts Swap 44 and 81

✓ + 2 pts Swap 44 and 78 (may be on same line as item above)

✓ + 5 pts Final row is a valid max heap and correctly did build heap algorithm

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

+ 0 pts Incorrect / No Answer

- 2 pts Did not show any intermediate steps

QUESTION 7

Efficiency - Matching 10 pts

7.1 BuildHeap 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

7.2 Removing bad hash 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

7.3 Resizing hashtable 2 / 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

7.4 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

7.5 Postorder 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

QUESTION 8

8 BST Ancestors - Coding 20 / 20

✓ + 2 pts Throws a `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

- 3 pts Adds data to new list

CS 1332 Exam 2B

Spring Semester 2018 - March 7, 2018

Name (print clearly including your first and last name) _____

Signature: _____

GT account username (msmith3, etc): _____

GT account number (903000000, etc): _____

- ☞ You must have your BuzzCard or other form of identification during the exam. When you turn in your exam, you will have to show your ID to the TA. If 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 28 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 $(2 \times \text{initial capacity}) + 1$. 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.

resize

table[0] =	table[0] =
table[1] =	table[1] =
table[2] = 17	table[2] =
table[3] = 28	table[3] =
table[4] = 39	table[4] =
$z(s) = 10 + 1$	\downarrow
	[0] =
	[1] =
	[2] =
	[3] =
	[4] =
	[5] =
	[6] = 17
	[7] = 39
	[8] =
	[9] =
	[10] = 28

N	$N \% 5$	$(h(u) + i^x) \% 5$	K	L-F
17			2	$1/5 = 0.2$
39			4	$2/5 = 0.4$
28	$28 \% 5 = 28 - 25 = 3$		3	$3/5 = 0.6$
17	$17 \% 11 =$ $17 - 11 =$ $\frac{11}{6} = 6$		6	$4/11 \approx \text{not } .5$
39	$39 \% 11$ $\frac{33}{6} = 6$	$(6 + 0^2) \% 11 =$ $(6 + 1^2) \% 11 = 7 \% 11$	7	
28	$28 \% 11 =$ $\frac{22}{6} = 6$	$6 + 0^2$ $6 + 1^2$ $6 + 4 = 10 \% 11 = 10$		

2) 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<? super T>> {

    /**
     * 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() {
        backingHeap = (T[]) new Comparable[ ];

    }

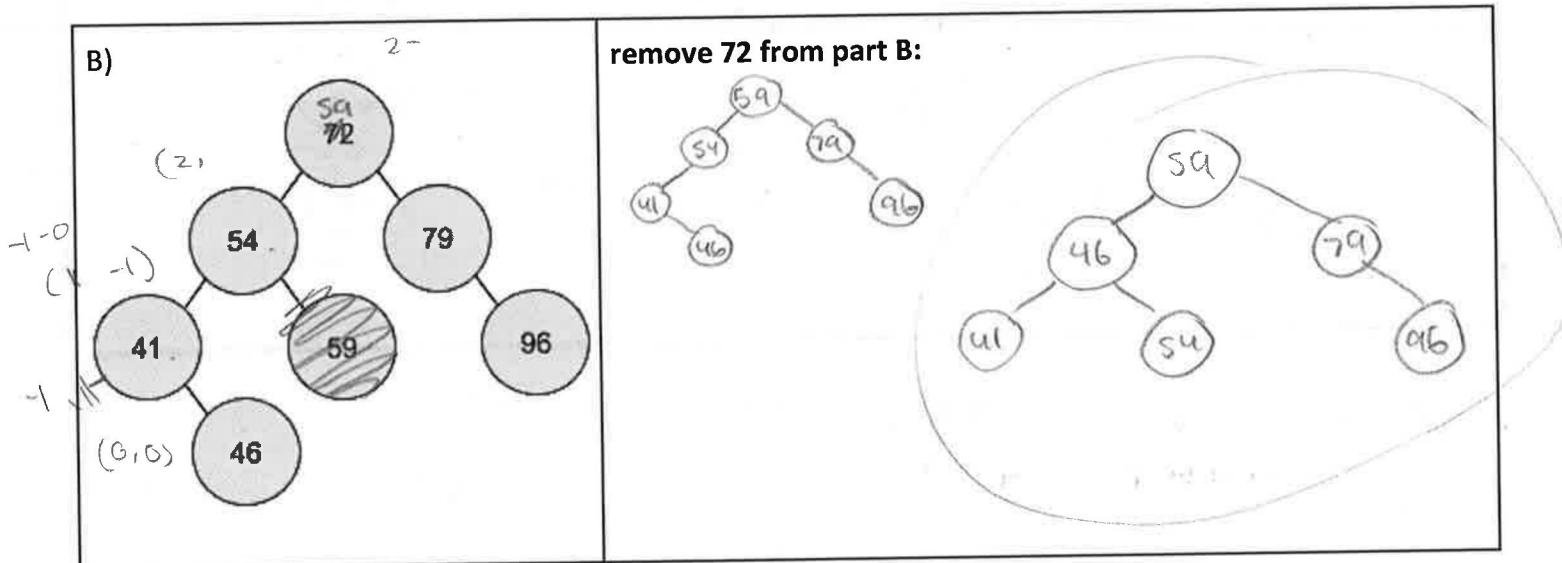
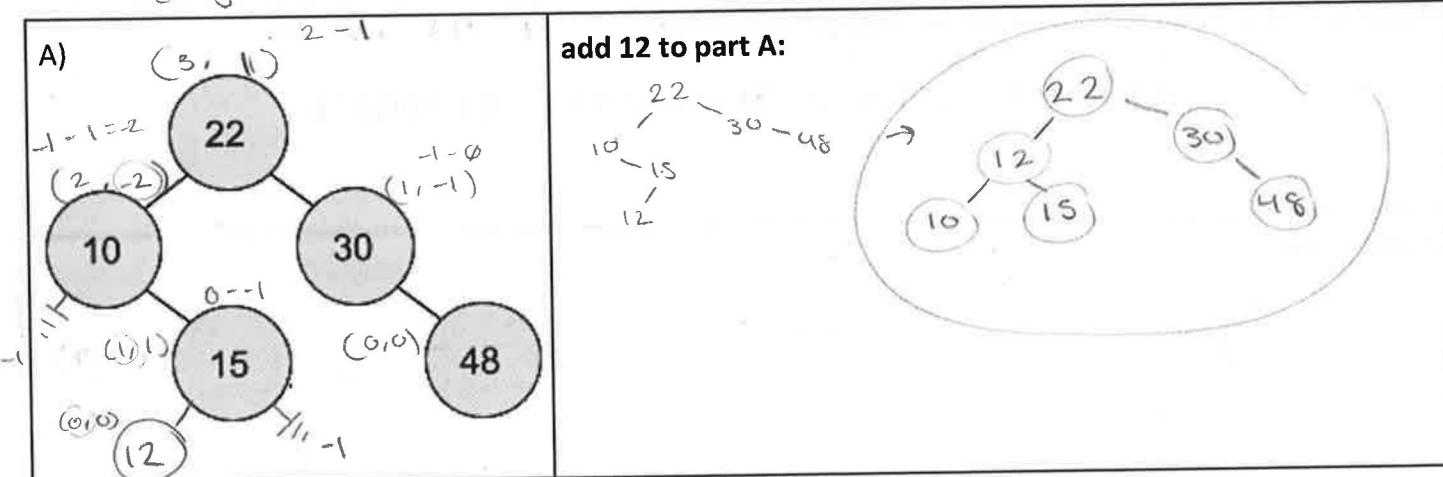
    /**
     * Adds (@code item) to the PQ such that the minimum element is still the root
     * @throws java.lang.IllegalArgumentException if the parameter is null
     * @param item generic object to be added
     */
    public void enqueue(T item) {
        if(item == null) throw new java.lang.IllegalArgumentException("Error, input cannot be null");
        backingHeap.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(isEmpty()) throw new NoSuchElementException("Bor, queue empty");
        return backingHeap.remove();
    }
}
```

3) 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.

(height 1 B+F)



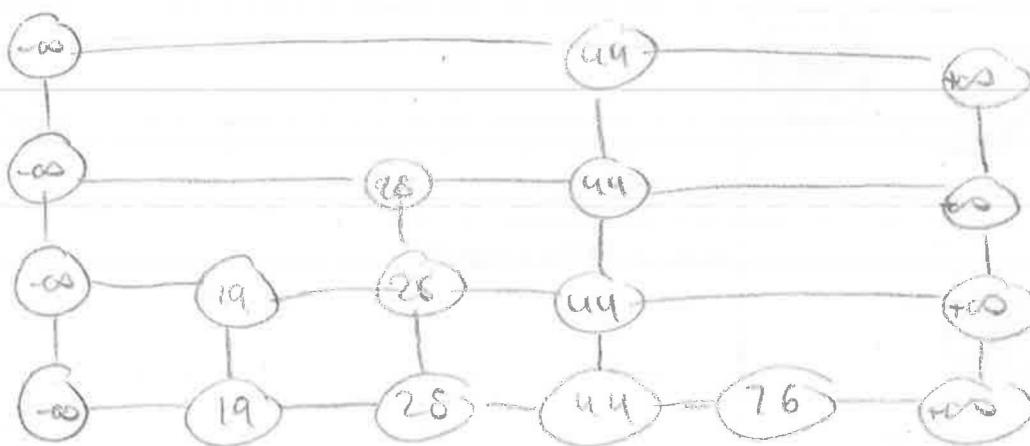
-2
-2

4) 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: **19, 44, 28, 76**. You will not receive any credit for intermediate steps; **only the final Skip List will be graded**. Include $-\infty$ 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.

HT HHHHT HHTHHHTHTTHHTTHTTHHTH

Final SkipList:

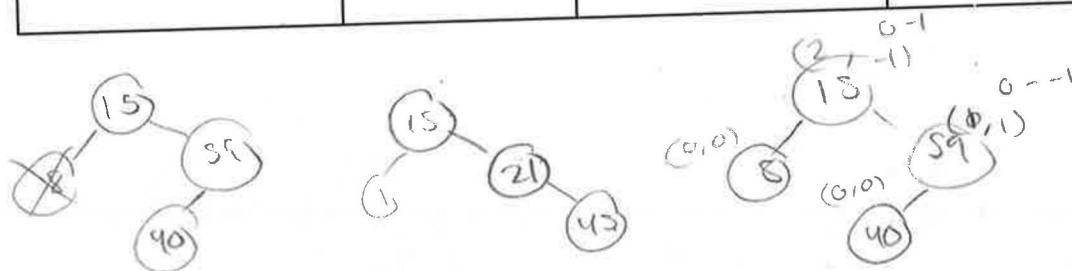


19 44 28 76
HT HHHHT HHT T

5) 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 - 59 Add - 8 Add - 40 Remove - 8		<input checked="" type="radio"/> Binary Search Tree <input checked="" type="checkbox"/> <input checked="" type="radio"/> AVL Tree <input checked="" type="checkbox"/> <input checked="" type="radio"/> Heap <input checked="" type="checkbox"/> <input checked="" type="radio"/> None of the Above <input checked="" type="checkbox"/>
	Add - 21 Add - 45 Add - 1		<input checked="" type="radio"/> Binary Search Tree <input checked="" type="checkbox"/> <input checked="" type="radio"/> AVL Tree <input checked="" type="checkbox"/> <input checked="" type="radio"/> Heap <input checked="" type="checkbox"/> <input checked="" type="radio"/> None of the Above <input checked="" type="checkbox"/>
	Add - 59 Add - 8 Add - 40		<input checked="" type="radio"/> Binary Search Tree <input checked="" type="checkbox"/> <input checked="" type="radio"/> AVL Tree <input checked="" type="checkbox"/> <input checked="" type="radio"/> Heap <input checked="" type="checkbox"/> <input checked="" type="radio"/> None of the Above <input checked="" type="checkbox"/>



6) Max Heap - Diagram [10 points]

The data below was put into an initial Max Heap via the constructor as a Collection. Perform the **build-heap** 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	92	44	46	28	81	87	63	59	28	78
	92	44	46	71	81	87	63	59	28	78
	92	44	87	71	81	46	63	59	28	78
	92	81	87	71	44	46	63	59	28	78
	92	81	87	71	78	46	63	59	28	44
	92	81	87	71	78	46	63	59	28	44

i = size/2 ; i > 0 ; i--

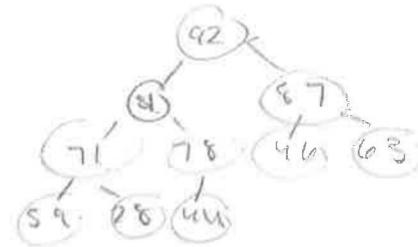
i = 10/2 = 5

downHeap(5) LC = 5*2 = 10 (78)
 81 RC = 5*2 + 1 = 11

downHeap(1) LC = 2
 92 RC = 3

downHeap(4) LC = 4 * 2 = 8 = 59
 28 RC = 4 * 2 + 1 = 9 71 ✓

downHeap(3) LC = 6 → 87 ✓
 46 RC = 7 → 63



downHeap(2) LC = 4 → 28 → LC =
 81 RC = 5 → 28 ✓ → LC = 10

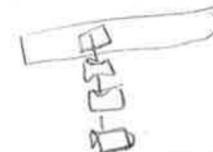
7) Efficiency - Matching [10 points]

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.) The BuildHeap Algorithm for a min heap.

- O(1) O(log n) O(n) O(n log n) O(n^2)

B.) Average case of removing from a hashtable using external chaining where the hash function always returns 7. *avg for a LL*



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

- 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.) Performing a postorder traversal in an AVL Tree.

- O(1) O(log n) O(n) O(n log n) O(n^2)

*L
R
Par*

8) 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 {

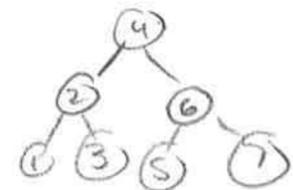
    private class BSTNode {
        int data;
        BSTNode left;
        BSTNode right;
    }

    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 (current == null) {  
        throw new NoSuchElementException("data not in tree!");  
    }  
  
    if (data < current.data) {  
        // no indent  
        ancestorHelper(current.left, list, data));  
        list.add(current.data);  
        return;  
    }  
  
    else if (data > current.data) {  
        // no indent  
        ancestorHelper(current.right, list, data));  
        list.add(current.data);  
        return;  
    }  
  
    else {  
        list.add(current.data);  
        return;  
    }  
  
}  
}
```



(5, 6,

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.