CS1332 Campus Spring 2023 Exam 2 Version A1

Aaryan Vinay Potdar

TOTAL POINTS

80.5 / 100

QUESTION 1

Efficiency - Multiple Choice 14 pts

1.1 Part A 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 incorrect

1.2 Part B 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 blank

1.3 Part C 2 / 2

+ 0 pts $$O(1)$$

+ 0 pts $$O(\log\,n)$$

+ 0 pts $$O(n)$$

✓ + 2 pts *$$O(n \log n)$$*

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

+ 0 pts incorrect

1.4 Part D 2 / 2

✓ + 2 pts *$$O(1)$$*

+ 0 pts $$O(\log\,n)$$

+ 0 pts $$O(n)$$

+ 0 pts $$O(n \log n)$$

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

+ 0 pts blank

1.5 Part E 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 blank

1.6 Part F 2 / 2

+ 0 pts $$O(1)$$

+ 1 pts $$O(\log\,n)$$

✓ + 2 pts *$$O(n)$$*

+ 0 pts $$O(n \log n)$$

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

+ 0 pts incorrect

1.7 Part G 0 / 2

+ 2 pts $$O(1)$$

✓ + 0 pts *$$O(\log\,n)$$*

+ 0 pts $$O(n)$$

+ 0 pts $$O(n \log n)$$

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

+ **0 pts** incorrect

QUESTION 2

Tree Identification - Multiple Select 10 pts

2.1 Part 1 **2.5 / 2.5**

✓ **+ 2.5 pts** *Fully Correct*

Partial Credit

+ **1 pts** Binary Tree

+ **1 pts** MaxHeap

+ **0 pts** Incorrect

2.2 Part 2 **2.5 / 2.5**

✓ **+ 2.5 pts** *Fully Correct*

Partial Credit

+ **0.5 pts** Binary Tree

+ **1 pts** BST

+ **1 pts** AVL

+ **0 pts** Incorrect

2.3 Part 3 **2.5 / 2.5**

✓ **+ 2.5 pts** *Fully Correct*

Partial Credit

+ **1 pts** Binary Tree

+ **1 pts** BST

+ **0 pts** Incorrect

2.4 Part 4 **2 / 2.5**

+ **2.5 pts** Fully Correct - Binary Tree Only

Partial Credit

✓ **+ 1 pts** *Binary Tree*

✓ **+ 1 pts** *MinHeap*

+ **0 pts** Incorrect

QUESTION 3

Code ID - Multiple Choice 8 pts

3.1 Part 1 **4 / 4**

+ **0 pts** `peek()`

✓ **+ 4 pts** *`enqueue(T data)`*

+ **0 pts** `pop()`

+ **0 pts** `push(T data)`

3.2 Part 2 **4 / 4**

✓ **+ 4 pts** *`peek()`*

+ **0 pts** `enqueue(T data)`

+ **0 pts** `pop()`

+ **0 pts** `push(T data)`

QUESTION 4

4 HashMap - Diagramming **9 / 12**

+ **12 pts** Completely Correct:

Partial

+ **0.5 pts** Key `17` appears once, and is updated with value `a`

+ **0.5 pts** Key `17` appears once, and is at index 9

✓ **+ 1 pts** *`<16, x>` is added*

✓ **+ 2 pts** *Key `16` is at index 6*

✓ **+ 1 pts** *`<6, s>` is added*

✓ **+ 2 pts** *Key `6` is at index 8*

✓ **+ 1 pts** *Key `7` is present in the hashmap*

✓ **+ 1 pts** *Key `7` has value `d`*

✓ **+ 1 pts** *key `7` can be probed to from index 7*

+ 1 pts Key `7` is at index 1

+ 1 pts No other data has been added, removed, or moved, and there are no duplicate keys

Partial - Resized Backing Array

+ 1 pts Backing array is resized to correct length: 21 `(2 * length) + 1`

+ 0.5 pts Key `10` is rehashed at correct index based on new length (index 10 for length 21)

+ 0.5 pts Key `3` is rehashed at correct index based on new length & rehashed entries (index 3 for length 21)

+ 0.5 pts Key `16` is rehashed at correct index based on new length & rehashed entries (index 16 for length 21)

+ 0.5 pts Key `27` is rehashed at correct index based on new length & rehashed entries (index 6 for length 21)

+ 0.5 pts Key `6` is rehashed at correct index based on new length & rehashed entries (index 7 for length 21)

+ 0.5 pts Key `17` is rehashed at correct index based on new length & rehashed entries (index 17 for length 21)

+ 0.5 pts Key `7` is rehashed at correct index based on new length & rehashed entries (index 8 for length 21)

+ 1 pts `DEL` entry is not copied to new backing array

+ 2 pts All values are correctly copied and updated (i.e. `10` has value `o`, `17` has value `a`, `7` has value `d`, etc.),

AND there are no duplicate keys

+ 0 pts Incorrect/ No Answer

QUESTION 5

5 Skiplist - Diagramming 10 / 10

✓ **+ 10 pts** *Completely Correct: **HHT HT HHHT HT HHHHT***

Partial Credit - First Substring

+ 2 pts Completely correct HHT

+ 1 pts Has 1 extra H: HHHT (counting the last level as promotion)

+ 0 pts No answer / Incorrect

Partial Credit - Second Substring

+ 2 pts Completely correct HT

+ 1 pts Has 1 extra H: HHT (counting the last level as promotion)

+ 0 pts No answer / Incorrect

Partial Credit - Third Substring

+ 2 pts Completely correct: HHHT

+ 1 pts Has 1 extra H: HHHHT (counting the last level as promotion)

+ 0 pts No answer / Incorrect

Partial Credit - Fourth Substring

+ 2 pts Completely correct: HT

+ 1 pts Has 1 extra H: HHT (counting the last level as promotion)

+ 0 pts No answer / Incorrect

Partial Credit - Fifth Substring

+ 2 pts Completely correct: HHHHT

+ 1 pts Has 1 extra H: HHHHHT (counting the last level as promotion)

+ 0 pts No answer / Incorrect

+ 4 pts Used T instead of H for promotion. Only get credit if completely correct when replacing H with T

+ 4 pts String is trying to add in data from left to right. Has to be completely correct in this order to earn this credit.

- 1 pts Minor error

+ 0 pts No answer / Incorrect

QUESTION 6

6 AVL - Diagramming 12 / 12

add

✓ **+ 6 pts** *completely correct*

![Screenshot_2023-03-08_at_11.00.19_PM.png](/files/af7b3eac-5da5-49a0-a9a5-ad4315edee2a)

+ 1.5 pts PARTIAL: `43` is added as a leaf node

+ 1.5 pts PARTIAL: tree is balanced

+ 1.5 pts PARTIAL: tree preserves order property

+ 1.5 pts PARTIAL: other than `43`, no other data is added or removed

+ 0 pts INCORRECT/BLANK

remove

✓ **+ 6 pts** *completely correct*

![Screenshot_2023-03-08_at_11.00.25_PM.png](/files/a483e1d5-2bbb-49c0-a15d-498f67161a09)

+ 1.5 pts PARTIAL: `25` is not present in tree

+ 1.5 pts PARTIAL: tree is balanced

+ 1.5 pts PARTIAL: tree preserves order property

+ 1.5 pts PARTIAL: other than `25`, no other data is added or removed

+ 0 pts INCORRECT/BLANK

QUESTION 7

7 2-4 Tree - Diagramming 10 / 10

Add

✓ **+ 4 pts**

![add8.png](/files/708a6175-6375-4bf9-9b40-dcafaf117a30)Completely Correct

+ 3 pts

![wrAdd8.jpeg](/files/fc353c99-ae37-4391-9c81-3ba8bbf0ef94)Promoted the wrong element

+ 1 pts Valid 2-4 Tree

+ 1 pts 8 has been added

+ 1 pts Resulting tree has 3 levels

- 0.5 pts Minor Error (an element is in the wrong node)

Remove

✓ **+ 6 pts**

![remove22.png](/files/45be209b-e7e9-4733-aa35-5b20f6b83b8d)Completely correct

+ 4.5 pts

![wrRem22.jpeg](/files/0c67a58e-44af-48e8-aedb-582fb8a69f67)Fuses with the wrong parent data/Uses wrong successor/predecessor

+ 1.5 pts Valid 2-4 tree

+ 1.5 pts 22 has been removed

+ 1.5 pts Tree has 2 levels

- 0.5 pts Minor Error (an element is in the wrong

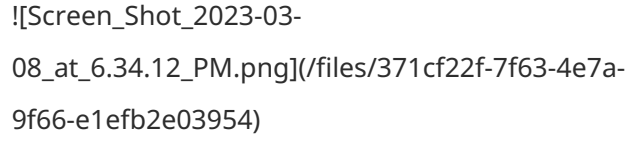
node)

+ 0 pts Incorrect

QUESTION 8

8 BST - Coding 7.5 / 15

+ 0 pts



✓ **+ 1.5 pts** *Attempts to initialize a list*

+ 1.5 pts Correctly initializes an arraylist or linkedlist in the public method given in the question

✓ **+ 1.5 pts** *Returns something*

+ 1 pts Returns the list

✓ **+ 1.5 pts** *Has a helper method*

+ 1 pts Throws an exception somewhere

+ 0.25 pts Correctly throws NoSuchElementException if data is not in the tree

+ 0.25 pts Returns without adding to the list if found data

+ 1.5 pts Attempts to check if current node's data is odd

+ 1 pts Correctly checks if current node's data is odd

+ 1 pts Adds to the list if current node's data is odd

✓ **+ 1.5 pts** *There is a recursive call somewhere*

✓ **+ 1 pts** *Attempts to compare curr.data and data*

✓ **+ 0.25 pts** *If curr.data is less than data recurse to the right*

✓ **+ 0.25 pts** *Else if curr.data is greater than data recurse to the left*

- 0.25 pts Efficiency

- 0.25 pts Minor Error

- 0.25 pts Syntax Error

+ 0 pts Incorrect/No answer

QUESTION 9

9 MinHeap - Coding 1.5 / 9

✓ **+ 1.5 pts** *Throws IndexOutOfBoundsException when A or B is <= 0*

✓ **+ 1 pts** *Returns a boolean*

+ 1 pts Correctly returns isParent/isChild (single if-statement or part of loop)

✓ **+ 1.5 pts** *Has a while loop or uses recursion*

+ 1 pts Loop/recursion is not infinite

+ 1 pts Loop/recursion breaks as soon as isAncestor/isDescendant is determined

✓ **+ 1 pts** *Correctly handles case A == B (can be implicit)*

+ 1 pts Correctly returns isAncestor/isDescendant in all other cases

- 0.5 pts Efficiency: i.e. Checks `index*2` and `index*2 +1` recursively, unnecessary operatons

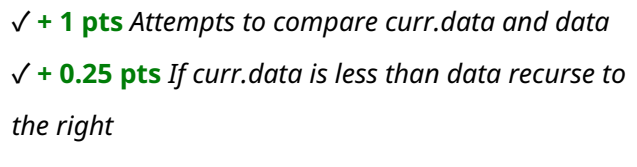
- 0.5 pts Minor error: (e.g. attempts to use size or backing array)

✓ **- 0.5 pts** *Syntax*

✓ **- 3 pts** *LATE*

+ 0 pts Incorrect/No Answer

+ 0 pts



1 must return a boolean

QUESTION 10

10 Bonus **1 / 0**

✓ **+ 1 pts** *Correct*

+ 0 pts Incorrect

+ 0 pts FLAG

CS 1332 Exam 2 - Version A1
Spring Semester 2023 - March 8, 2022

Name (print clearly including your first and last name): AARYAN POTDAR

Section (8:25 am - A, 9:30 am - B, 2:00 pm D, 3:30 pm - C, Online - O): B

Signature: AP

GT account username (msmith3, etc): apotdar31

GT account number (903000000, etc): 903795148

- You must have your BuzzCard or other form of identification on the table in front of you during the 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.
- If you plan on using ear plugs (foam or silicone, NOT AirPods) during the exam, you must show them to the instructor for approval.
- Pens/pencils and erasers are allowed. Do not share.
- If you brought a duck with you to the exam, you may silently consult with it at any time.
- All work entered on this exam, whether code, diagrams or multiple choice, must be implemented as was presented in lecture and recitation.
- All code must be in Java.
- Efficiency matters. For example, if you code something that uses O(n) time when there is a way to do it in O(1) time, your solution may lose credit. If your code traverses data 5 times when once would be sufficient, this also is considered poor efficiency even though both are O(n).
- Comments are not required unless a question explicitly asks for them.

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

This page is intentionally left almost blank.

(If you use this page for your work, please reference this page number on the question you are answering so we can find your response)

Table of Contents

Question	Points
1) Efficiency - Multiple Choice	14
2) Tree ID - Multiple Select	10
3) Code ID - Multiple Choice	8
4) Hashmap - Diagramming	12
5) Skiplist- Diagramming	10
6) AVL- Diagramming	12
7) 2-4 Tree - Diagramming	10
8) BST - Coding	15
9) MinHeap - Coding	9
10) Bonus	1

1) Efficiency - Multiple Choice [14 points]

For each of the operations listed below, determine the time complexity of the operation as it pertains to the data structure. Select the bubble corresponding to your choice in the space provided, and completely fill in the bubble. 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 *unless* otherwise specified.**

A) Calling get() on Hashmap that uses quadratic probing and is guaranteed to have no DEL markers.

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

B) Finding the maximum element in a MinHeap assuming we have access to the heap's backing array.

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

C) Constructing an AVL by adding n elements to it.

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

D) Calling remove() on a Hashmap that uses external chaining and has no chains with multiple elements.

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

E) Removing an element in a 2-4 tree given that the element is in a leaf node.

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

F) Finding the maximum data in a Skiplist capped at log n levels. The Skiplist is implemented as taught in class.

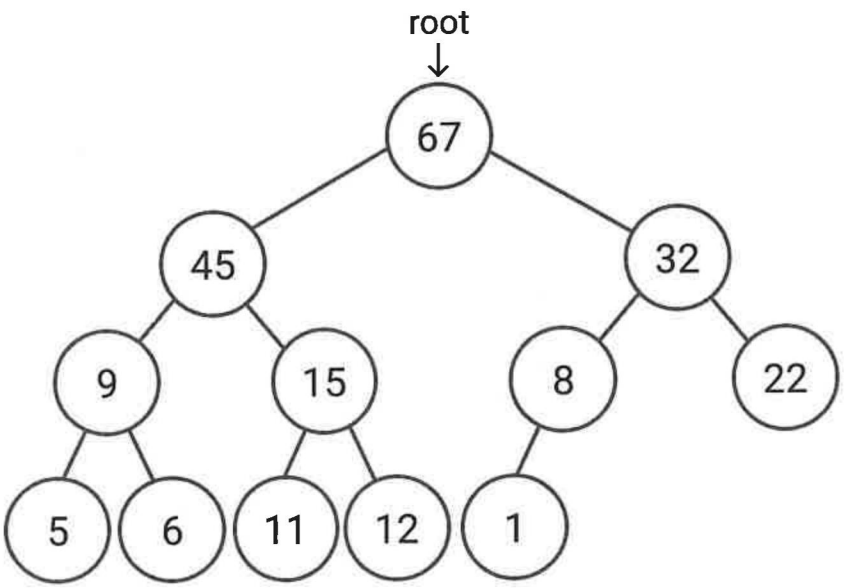
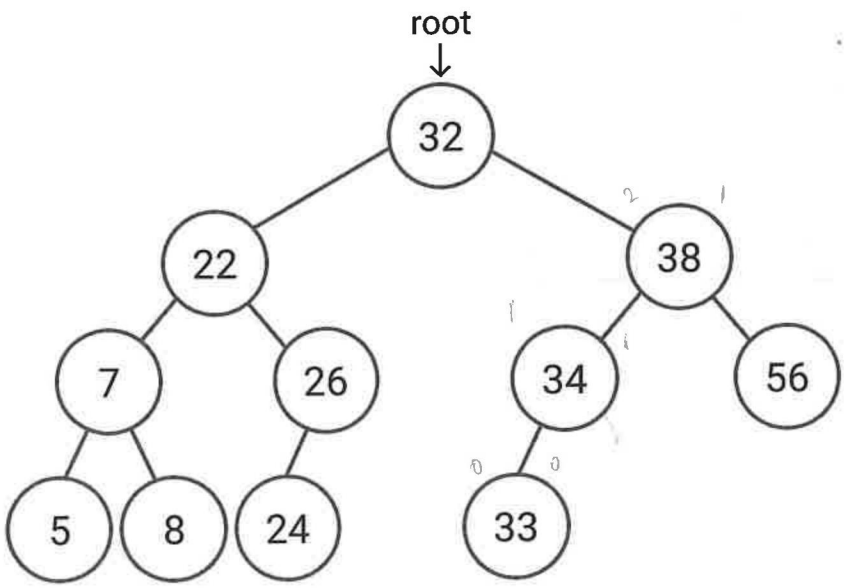
○ O(1) ○ O(log n) ● O(n) ○ O(n log n) ○ $O(n^2)$

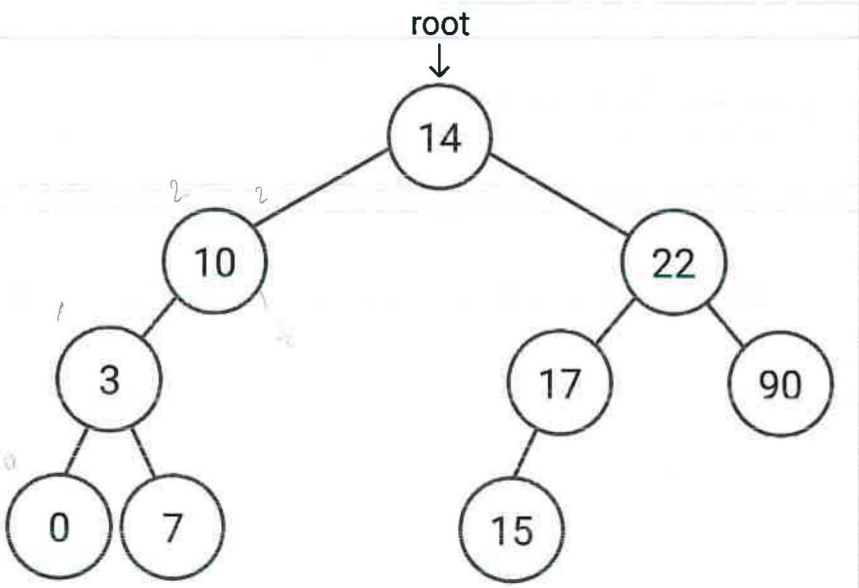
G) Finding the length of the path between the root of an AVL and the deepest leaf.

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

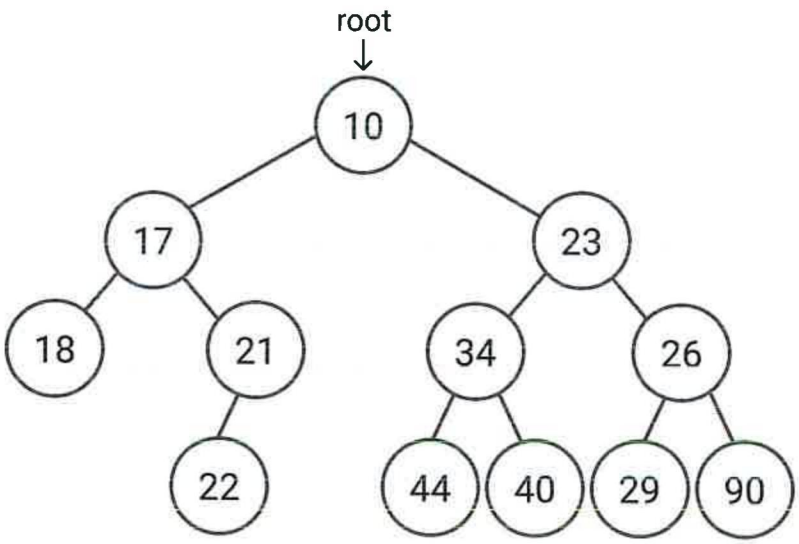
2) Tree Identification - Multiple Select [10 points]

For each of the given trees, **select all** tree type(s) that apply to the given tree. There are 4 tree types to choose from. For selected answers please completely fill the squares.

Given Tree	Tree Type (Select all that apply)
	■ Binary Tree ☐ Binary Search Tree ■ MaxHeap ☐ AVL
	■ Binary Tree ■ Binary Search Tree ☐ MinHeap ■ AVL



- [x] Binary Tree
- [x] Binary Search Tree
- [ ] MaxHeap
- [ ] AVL



- [x] Binary Tree
- [ ] Binary Search Tree
- [x] MinHeap
- [ ] AVL

3) Code ID - Multiple Choice [8 points]

Choose the correct name for the method which is described by a provided code snippet. The methods given below apply to the following data structures based on implementations taught in class - **Stack, Queue, Heap, BST**.

`if (data == null) {`
`  throw new IllegalArgumentException("Data cannot be null");`
`}`
`if (size == backingArray.length) {`
`  resize();`
`}`
`backingArray[(front + size) % backingArray.length] = data;`
`size++;`
	○ peek()
	● enqueue(T data)
	○ pop()
	○ push(T data)

`if (size == 0) {`
`  throw new java.util.NoSuchElementException("Empty");`
`}`
`return backingArray[size - 1];`
	● peek()
	○ enqueue(T data)
	○ pop()
	○ push(T data)

4) HashMap – Diagramming [12 points]

Goal: The HashMap below is backed by an array of capacity 10. **Perform all of the below operations in order** on the HashMap and draw the **final backing array** at the bottom of the page. The maximum load factor for this HashMap is 0.85. Circle the final answer if you draw multiple arrays.

Requirements: If you need a collision resolution strategy, use **Linear Probing** as taught in lecture. The hashcode of a particular number is the absolute value of the number itself. The compression function is to mod by the table length. If an element is deleted, represent it by writing "DEL" for the deleted element's index.

Initial Backing Array:

0	1	2	3	4	5	6	7	8	9
<10, o>	<17, a>	<7, d>	DEL	<3, k>		<16, n>	<27, v>	<7, m>	<17, q>

DEL
<6, s>

Operations:

- **put(17, "a")**
- **put(16, "x")**
- **remove(7)**
- **put(6, "s")**
- **put(7, "d")**

Final Backing Array:

0	1	2	3	4	5	6	7	8	9
<10, o>	<17, a>	<7, d>	DEL	<3, k>		<16, n>	<27, v>	<6, s>	<17, q>

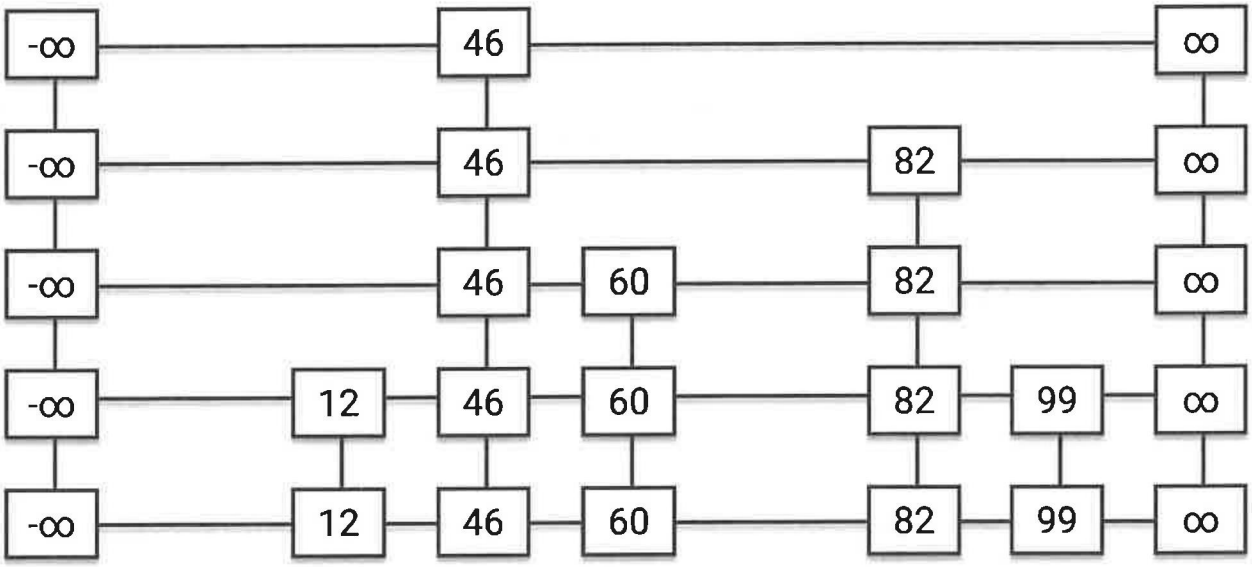
Do not write below this line. It may not be scanned, therefore it will not be graded.

5) Skiplist Diagramming [10 points]

Description: Starting with an empty skiplist, after the series of **add()** operations below, the resulting Skiplist is as illustrated.

Goal: Write the sequence of Heads (H) and Tails (T) coin tosses that was used in the add operations below to construct the Skiplist that was drawn below. Your answer, for example, should be in the format "HHTHTT", with H representing Head and T representing Tail in the sequence of coin tosses. **Use Heads as promotion in your answer.**

The series of add() operations were performed **in the following order:**
add(60)
add(12)
add(82)
add(99)
add(46)

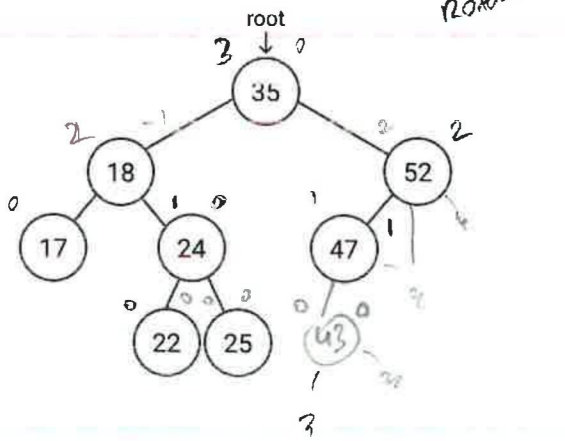


Write the coin toss sequence used in constructing the Skiplist above in the provided table below. You may not have to use all cells:

H	H	T	H	T	H	H	H	T	H	T	H	H	H	H	T				

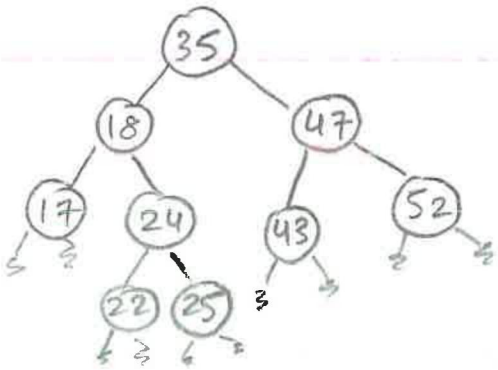
6) AVL - Diagramming [12 points]

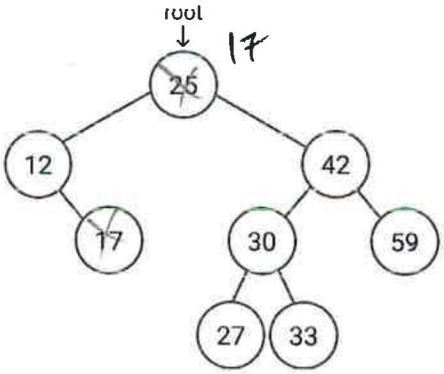
Given the following initial AVL, perform the stated operation and **draw your final answer in the box provided**. If necessary for any operation, use the **predecessor node**.



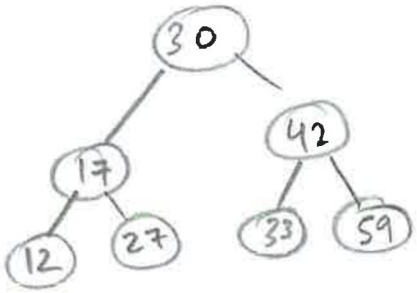
Right rotation

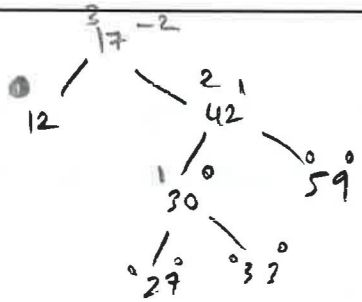
add(43)



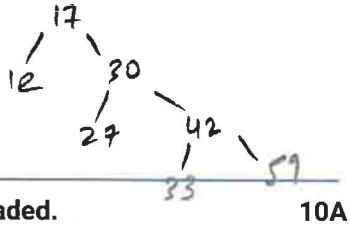


remove(25)





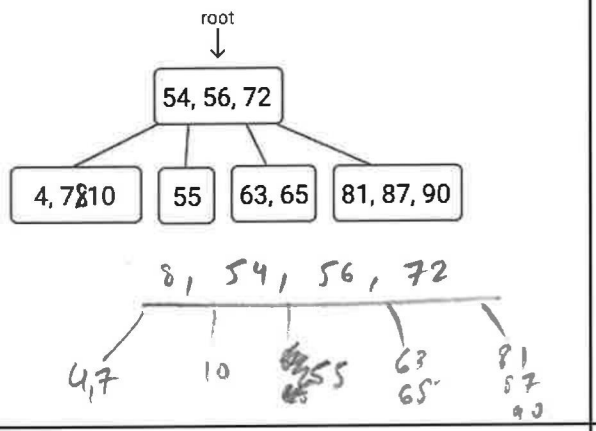
Right-left rotation



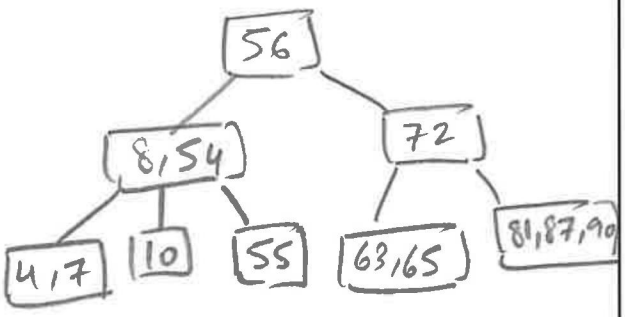
7) 2-4 Tree - Diagramming [10 points]

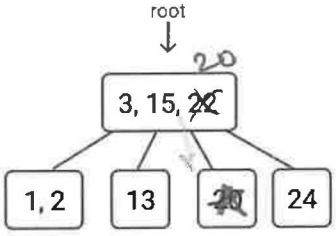
Given the following initial 2-4 trees in the left column below, perform the stated operation. Draw the resulting 2-4 tree in the right column. If you want, you can draw multiple steps (**circle the final step if you do so**). Follow the implementation taught in the **1332 module videos and live lectures**. **Please read the instructions below carefully.**

Implementation Details:
If you need to promote an element from a node, use the **third** element. When needed, use the **predecessor**. When checking if a transfer is possible, check the **left** sibling before the right sibling. If a fusion is necessary and the node has more than one parent data, choose the **left** parent data.

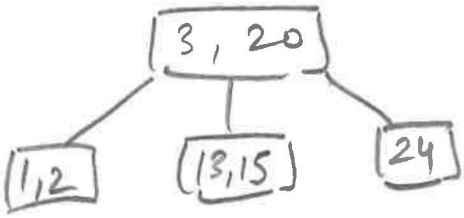


add(8)





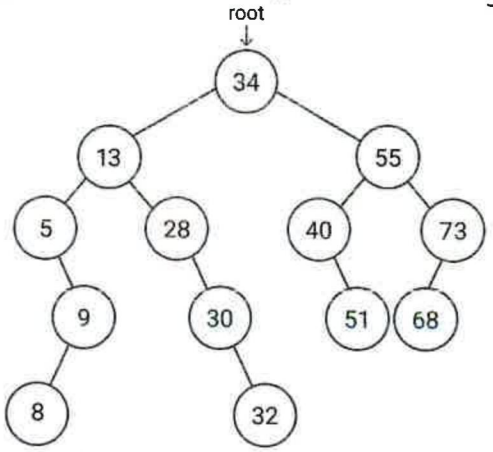
remove(22)



8) BST - Coding [15 points]

Goal: Given a binary search tree with integers, write a method which returns a list of all of the odd integers that are ancestors of the passed in data. **Odd integers refer to data in the node being odd, not the level.** An ancestor node is **either the parent of the node or the parent of some ancestor of the node**. Data contained in the list should be in the order of **increasing depth**. **The data being searched for should not be included, even if it is odd.**

Requirements: Your implementation should be **recursive**. Do not modify the tree or the method header provided. Feel free to write any helper methods you consider to be necessary. Since **Node** is an inner class, access its fields directly (e.g. node.left). **Assume the List, ArrayList, LinkedList, and NoSuchElementException classes are imported.** Your code should be as efficient as possible.



Examples: (on the tree above):

- listOddAncestors(8) returns [13, 5, 9]
- listOddAncestors(68) returns [55, 73]

```
public class BST {

  private static class Node {
    int data;
    Node right, left;
  }
  private Node root;

  /**
   * Records the odd ancestors of input data in order of increasing depth.
   * This should be implemented recursively.
   * @param data - data to be searched for
   * @return a list containing all odd ancestors
   * @throws NoSuchElementException if data is not present in BST
   */
  public List<Integer> listOddAncestors(int data) {
```

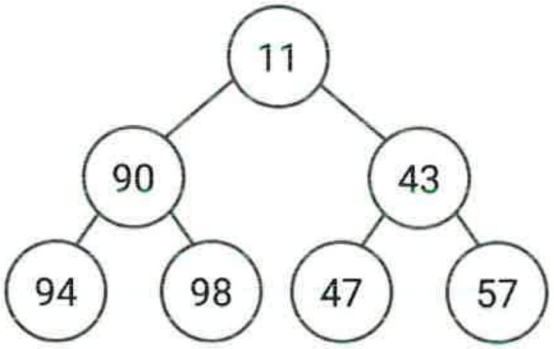
```
List<Integer> listNum = new List<>();
return recurse(listNum, data, root);
}
private void recurse(List<Integer> list, int data, Node curr){
if(curr == null){
return list;
}
if(curr.data > data){
return recurse(list, data, curr.left);
if(curr.data < data){
return recurse(list, data, curr.right);
}
list.add(curr.data)
return list;
}
```

```
    } // END OF METHOD
} // END OF CLASS
```

9) MinHeap - Coding [9 points]

Goal: Given two integers, **indexA and indexB**, that represent the **indices** of data in a MinHeap, write a method that determines **the node at index*A is a descendant of the node at indexB***. An element in a heap is considered a ***descendant*** of another element if it exists in the subtree of the parent element (see example below). An element is also considered a descendant of itself.

Requirements: The method should throw an **IndexOutOfBoundsException** if indexB or indexA is less than the start index of the heap. The method should return **true** if A is a descendant of B, and false otherwise. You do not have access to the backing array. Remember, **indexA and indexB are indices in the array!**



Examples (With tree and array representations of a heap for reference):

- isDescendant(5, 1) returns *true*
  - Index 5 (98) is a descendant of index 1 (11)
- isDescendant(4, 3) returns *false*
  - Index 4 (94) is NOT a descendant of index 3 (43)
- isDescendant(1, 1) returns *true*

0	1	2	3	4	5	6	7	8	9
x	11	90	43	94	98	47	57	x	x

```
public class Solution {

    /**
    * Returns a boolean value representing whether or not
    * index a is a descendant of index b.
    * @param indexA The index of the descendant element.
    * @param indexB The index of the ancestor element.
    * @return true if indexA is descendant of indexB, false otherwise.
    * @throws java.lang.IndexOutOfBoundsException if indexB <= 0 or
    * indexA <= 0
    **/
    public boolean isDescendant(int indexA, int indexB) {
        //WRITE YOUR METHOD HERE
            if(indexA <= 0 || indexB <= 0) {
                throw new IndexOutOfBoundsException("index not valid");
            }
```

```
if (indenA >= indenB) {
    boolean val = recurseCheck(indenA, indenB);
}
return val;
}
private boolean recurseCheck(int i, int j)
    if (i == j) {
        return true
    }
    if (j = j*2 || j = i*2+1) {
        return
    } else {
        return recurseCheck((i+1), j);
    }
    if (j > i) {
        return false;
    }
```

late

```
} // END OF METHOD
} //END OF CLASS
```

10) Bonus - Fill in the Blank [1 point]

For 1 extra point, please describe ONE city in the world 🌎 you would like to visit 😃 and why (inappropriate content will not receive credit):

Paris → It is touristic and has many monuments