CS1332 Campus Fall 2022 Exam 2 Section A Version B

Thomas Nam Minh Le

TOTAL POINTS

88 / 100

QUESTION 1

Efficiency - Multiple Choice 12 pts

1.1 1.A 0 / 2

+ **2 pts** Correct: $$O(log n)$$
+ **0 pts** Incorrect: $$O(1)$$
✓ + **0 pts** **Incorrect: $$O(n)$$**
+ **0 pts** Incorrect: $$O(n log n)$$
+ **0 pts** Incorrect: $$O(n^2)$$
+ **0 pts** No answer

1.2 1.B 2 / 2

✓ + **2 pts** **Correct: $$O(1)$$**
+ **0 pts** Incorrect: $$O(log n)$$
+ **0 pts** Incorrect: $$O(n)$$
+ **0 pts** Incorrect: $$O(n log n)$$
+ **0 pts** Incorrect: $$O(n^2)$$

1.3 1.C 0 / 2

+ **2 pts** Correct: $$O(n^2)$$
+ **0 pts** Incorrect: $$O(1)$$
+ **0 pts** Incorrect: $$O(log n)$$
✓ + **0 pts** **Incorrect: $$O(n)$$**
+ **0 pts** Incorrect: $$O(n log n)$$
+ **0 pts** No answer

1.4 1.D 2 / 2

✓ + **2 pts** **Correct: $$O(n)$$**
+ **0 pts** Incorrect: $$O(1)$$
+ **0 pts** Incorrect: $$O(log n)$$
+ **0 pts** Incorrect: $$O(n log n)$$
+ **0 pts** Incorrect: $$O(n^2)$$

1.5 1.E 2 / 2

✓ + **2 pts** **Correct: $$O(n)$$**
+ **0 pts** Incorrect: $$O(1)$$
+ **0 pts** Incorrect: $$O(log n)$$
+ **0 pts** Incorrect: $$O(n log n)$$
+ **0 pts** Incorrect: $$O(n^2)$$
+ **0 pts** No answer

1.6 1.F 2 / 2

✓ + **2 pts** **Correct: $$O(n log n)$$**
+ **0 pts** Incorrect: $$O(1)$$
+ **0 pts** Incorrect: $$O(log n)$$
+ **0 pts** Incorrect: $$O(n)$$
+ **0 pts** Incorrect: $$O(n^2)$$
+ **0 pts** No answer

QUESTION 2

2 Tree Ops - Diagramming 12 / 12

AVL

✓ + **3 pts** **H is correct**
+ **1.5 pts** E/C/B is some AVL, but not THIS AVL.
+ **0 pts** Incorrect

BST

✓ + **3 pts** **B is correct**
+ **1.5 pts** C is some BST, but not THIS BST.
+ **0 pts** Incorrect or no answer

Max Heap

✓ + **3 pts** **A is correct**
+ **1.5 pts** G is some max heap, but not THIS HEAP.
+ **0 pts** Incorrect or no answer

Min Heap

✓ + **3 pts** **F is correct**
+ **1.5 pts** D is some min heap, but not THIS MIN HEAP.
+ **0 pts** Incorrect or no answer

QUESTION 3

3 HashMap - Diagramming 9 / 10

correct resize

✓ + 2 pts **Final backing array length is **11**.**

✓ + 2 pts **There are exactly **4** entries in the final backing array.**

✓ + 1 pts **`<-2, D> ` is at index 2.**

+ **2 pts** `<24, G> ` is added to the **front** at index 2 (also award if there is only one entry).

✓ + 1 pts **`<24, G> ` is added to the **back** at index 2 (_exclusive_ with previous rubric item).**

✓ + 2 pts **`<14, D> ` is at index 3.**

✓ + 1 pts **`<9, C> ` is at index 9.**

partial credit - no resize

+ **1 pts** Final backing array length is **5**.

+ **1 pts** There are exactly **4** entries in the final backing array.

+ **0.5 pts** `<-2, D> ` is at index 2.

+ **0.5 pts** `<9, C> ` is at index 4.

+ **0.5 pts** `<14, D> ` is at index 4.

+ **1.5 pts** `<24, G> ` is added to the **front** at index 4

+ **0.5 pts** `<24, G> ` is added to the **back** at index 4 (_exclusive_ with previous rubric item).

+ **0 pts** Incorrect/No answer

QUESTION 4

4 Build Heap - Diagramming 4.5 / 10

✓ + 1 pts **Row 1: 10 and 5 are swapped**

+ **1 pts** Row 2: 9 and 4 are swapped

+ **1 pts** Row 3: 7 and 3 are swapped

+ **1 pts** Row 4: 10 and 2 are swapped

+ **1 pts** Row 5: 5 and 2 are swapped

+ **1 pts** Row 6: 10 and 1 are swapped

+ **1 pts** Row 7: 9 and 1 are swapped

+ **1 pts** Row 8: 8 and 1 are swapped

+ **2 pts** MaxHeap is Correct

Partial Credit

Check the student's swaps and award 0.5 points for every row that is valid. A **valid** row is one where:
- They must swap with a child that is less than the parent (it is okay if it is the larger child)
- The swap is be between a **valid** parent /child index pair
- for example: swapping index 5 and 8 is **not** valid even if the data at index 8 is less than the data at index 5

- TAs: IF the student was correct up to a point, grade with the **normal** rubric. Once the student does an incorrect swap, grade with the partial credit from that row forward.
- IF a student gets **any** partial credit, you may only award partial credit fianl minheap points **even if** the final minheap matches the actual answer

+ **0.5 pts** Row 1 valid, but incorrect swap

✓ + 0.5 pts **Row 2 valid, but incorrect swap**

✓ + 0.5 pts **Row 3 valid, but incorrect swap**

✓ + 0.5 pts **Row 4 valid, but incorrect swap**

✓ + 0.5 pts **Row 5 valid, but incorrect swap**

✓ + 0.5 pts **Row 6 valid, but incorrect swap**

✓ + 0.5 pts **Row 7 valid, but incorrect swap**

✓ + 0.5 pts **Row 8 valid, but incorrect swap**

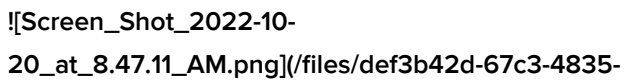
+ **1 pts** Partial Credit Maxheap (correct level order traversal from final row in table)

+ **0 pts** Incorrect/Blank

QUESTION 5

5 SkipList - Diagramming 8 / 8

✓ + 8 pts **Completely Correct**



9f03-9defc3039707)

+ **5 pts** Data on correct levels in **order of addition**
+ **3 pts** **All** data in the list in **sorted order**
- **1 pts** **One** data in the wrong sorted position
- **1 pts** **One** data on the wrong level
- **3 pts** **Two** data in the wrong sorted position
- **3 pts** **Two** data on the wrong level
- **2 pts** **One** data not in the list
- **3 pts** **Two** data not in the list
+ **0 pts** Incorrect/Blank

QUESTION 6

6 AVL - Diagramming 14 / 13

`add(15)`

✓ **+ 7 pts Completely Correct**

![Screenshot_2022-10-19_at_9.15.14_PM.png](/files/d4503e0d-67ea-4d66-8f1b-1d57162669f7)

+ **1 pts** **15** is added as a leaf node.
+ **2.5 pts** Tree is balanced.
+ **2 pts** Tree preserves order property.
+ **1 pts** Other than **15**, no other data is added or removed.
+ **0 pts** Incorrect/No answer

`remove(12)`

✓ **+ 7 pts Completely Correct**

![Screenshot_2022-10-19_at_9.16.13_PM.png](/files/babdcc6e-6b0d-4c52-ba37-7d7b1b65b8c6)

+ **1 pts** **12** is removed.
+ **2.5 pts** Tree is balanced.
+ **2 pts** Tree preserves order property.
+ **1 pts** Other than **12**, no other data is added or removed.
+ **0 pts** Incorrect/ No answer

QUESTION 7

7 2-4 Tree - Diagramming 10 / 10

add(2)

✓ **+ 4 pts Completely Correct**

![IMG_16FD29F494BB-1.jpeg](/files/53a21a0d-2515-4712-8c92-aee8341d11be)

+ **1 pts** The resulting is a valid 2-4 tree
+ **1 pts** The resulting tree has all the data including the added 2
+ **1 pts** The resulting tree has 3 levels

remove(15)

✓ **+ 6 pts Completely Correct**

![IMG_8C825EE7E8E6-1.jpeg](/files/c7111c42-2555-4324-8bb9-4d839a4444fe)

+ **1 pts** The resulting tree is a valid 2-4 tree
+ **1 pts** The resulting tree has 15 removed
+ **1 pts** The resulting tree has 2 levels
+ **0 pts** Blank

QUESTION 8

8 BSTs - Coding 11.5 / 15

+ **0 pts** Correct Answer

![Screen_Shot_2022-10-20_at_1.22.28_AM.png](/files/f30d009a-85b8-4066-b001-d29506359c53)

✓ **+ 2 pts Has helper method**

✓ **+ 2 pts Makes a call to the helper method in the public one**

✓ **+ 1 pts Helper method has a base case if node is null**

✓ **+ 1 pts Helper method returns 0 if node is null**

✓ **+ 2 pts Attempts to check if on odd level**

+ **1 pts** Correctly checks if on odd level

✓ **+ 1 pts If on odd level, returns data +**

helper(curr.left, level + 1) + helper(curr.right, level + 1)

✓ **+ 1 pts** **If on even level, returns helper(curr.left, level + 1) + helper(curr.right, level + 1)**

✓ **+ 2 pts** **Returns something**

+ 2 pts Returns the correct value in the public method

- 0.5 pts Minor error

- 0.5 pts Syntax

✓ **- 0.5 pts** **Efficiency**

+ 0 pts Blank/Incorrect

QUESTION 9

9 Deque - Coding 7 / 7

✓ **+ 1 pts** **Throws NSE if list is empty**

Size 1 Case

✓ **+ 0.5 pts** **Sets head to null**

✓ **+ 0.5 pts** **Sets tail to null**

General Case

✓ **+ 1.5 pts** **Sets tail to tail.prev**

✓ **+ 1.5 pts** **Sets tail's next to null**

✓ **+ 0.5 pts** **Decrements size**

✓ **+ 1.5 pts** **Returns removed data**

- 0.5 pts Efficiency

- 0.5 pts Syntax Error

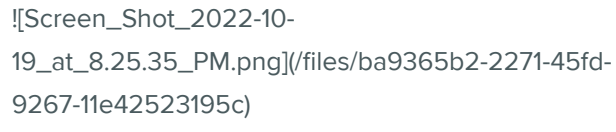
- 0.5 pts Minor Error

+ 0 pts Incorrect / No answer

QUESTION 10

10 Heap - Coding 3 / 3

+ 0 pts Correct Answer



✓ **+ 0.5 pts** **Has an if statement**

✓ **+ 0.5 pts** **Attempts to compare data at i and i/2**

✓ **+ 0.5 pts** **Correctly compares arr[i] and arr[i/2] using .compareTo()**

✓ **+ 0.5 pts** **Has a return statement (not the one we gave)**

✓ **+ 0.5 pts** **Returns false with the right condition**

✓ **+ 0.5 pts** **Decrements i**

- 0.5 pts Minor error

- 0.5 pts Syntax

- 0.5 pts Efficiency

+ 0 pts Blank/Incorrect

QUESTION 11

11 Bonus 1 / 0

✓ **+ 1 pts** **Correct**

+ 0 pts No answer

CS 1332 Exam 2 - Version B
Fall Semester 2022 - October 19, 2022

Name (including first and last name): Thomas Le

Signature: Thomas Le

GT account username (msmith3, etc): thomas30

GT account number (903000000, etc): 903 696 568

- 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 submit your exam as complete. (If you need to use the restroom for an emergency, then a TA will escort you.)
- 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, earbuds, etc. are not allowed. Extra paper is not allowed. If you have exhausted all space on this exam, 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 / module videos and recitation.
- 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).
- 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.

1) Efficiency - Multiple Choice [12 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) Removing data from a 2-4 Tree where all the leaves have 3 data

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

B) Finding the height of the tree represented by a Heap, if you are given its backing array and size

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

C) Adding an element to a HashMap that triggers a resize. The HashMap uses quadratic probing for collisions and has a bad hash function

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

D) Calling the get() method on a HashMap. The HashMap uses External Chaining for collisions, has a bad hash function

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

E) Removing from a SkipList that is capped at *log n* levels

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

F) Adding a set of *n* sorted elements one by one to an AVL

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

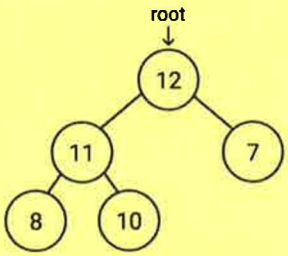
2) Tree Ops - Diagramming [12 points]

For each of the data structures listed below, write the letter of the diagram that corresponds to performing these operations in order: **add(10), add(8), add(7), add(12), add(11)**. Assume each data structure begins empty. You should write **exactly ONE letter per blank**, so some diagrams will not be used. Letters can be used more than once.

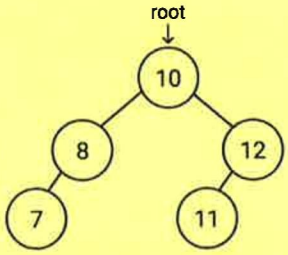
AVL: H MaxHeap: A

BST (not self-balancing): B MinHeap: F

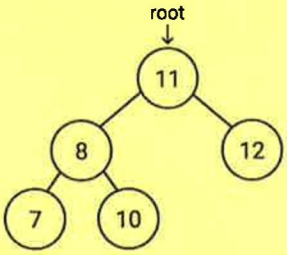
A.



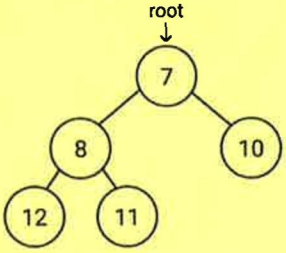
B.



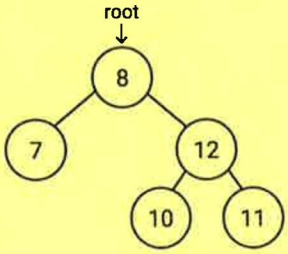
C.



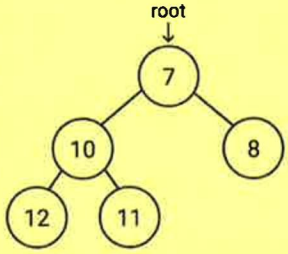
D.



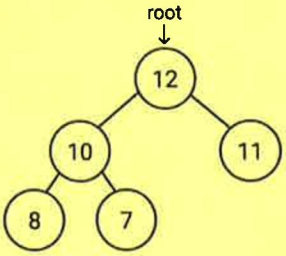
E.



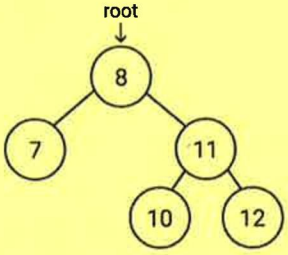
F.



G.



H.



This page is left blank for your use.

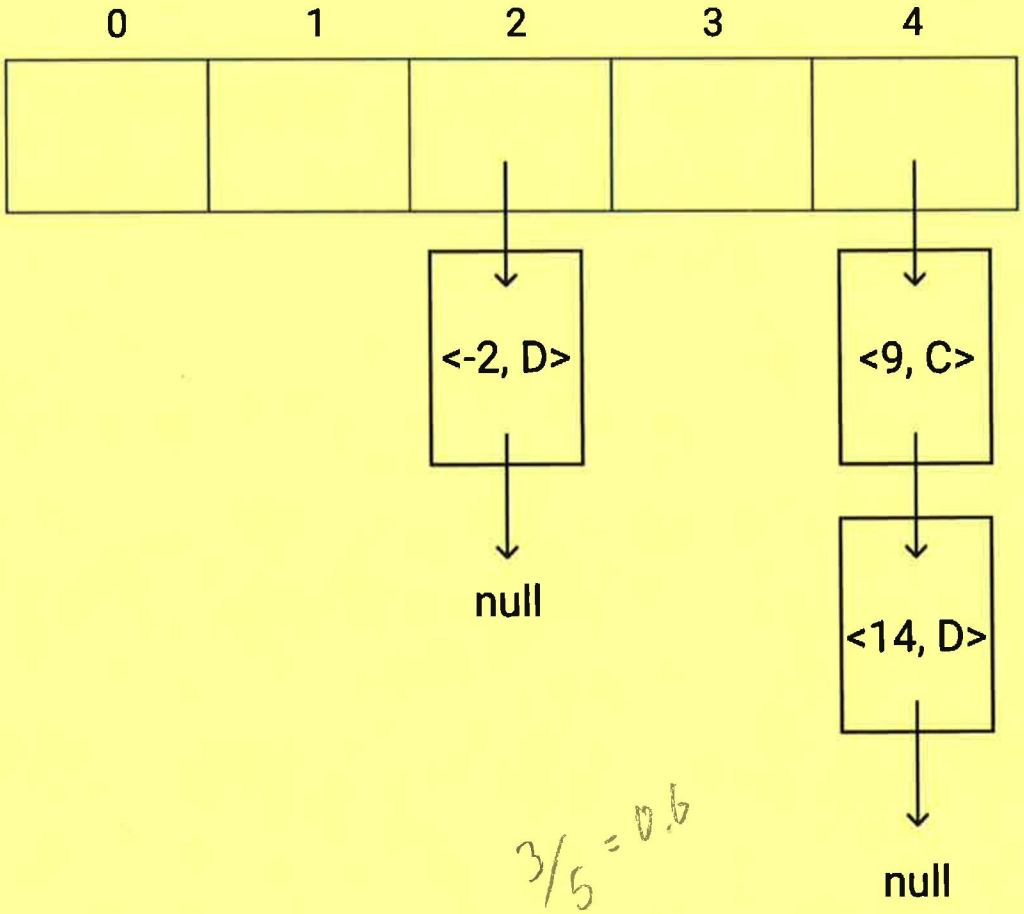
(If you use this page for your work, please reference this page number on the question you are answering so we can find your response)

3) HashMap – Diagramming [10 points]

Goal: The HashMap below is backed by an array of capacity 5. **Add (24, 'G')** to the HashMap and draw the **final backing array** on the next page. The maximum load factor for this HashMap is **0.65**.

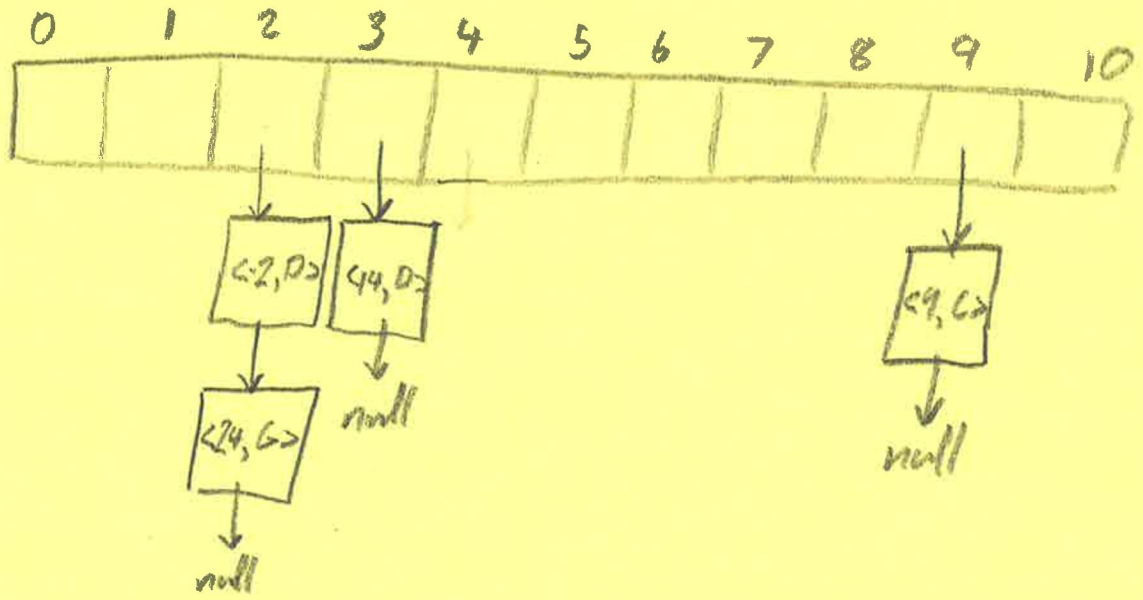
Requirements: If you need a collision resolution strategy, use **external chaining** where you add to the **front** of a **singly linked list**. If you need to resize the HashMap, resize it to a capacity of **(2 × current capacity) + 1**. The hashcode of a particular number is the **absolute value** of the number itself. The compression function is to mod by the table length.

Initial Backing Array:



3/5 = 0.6

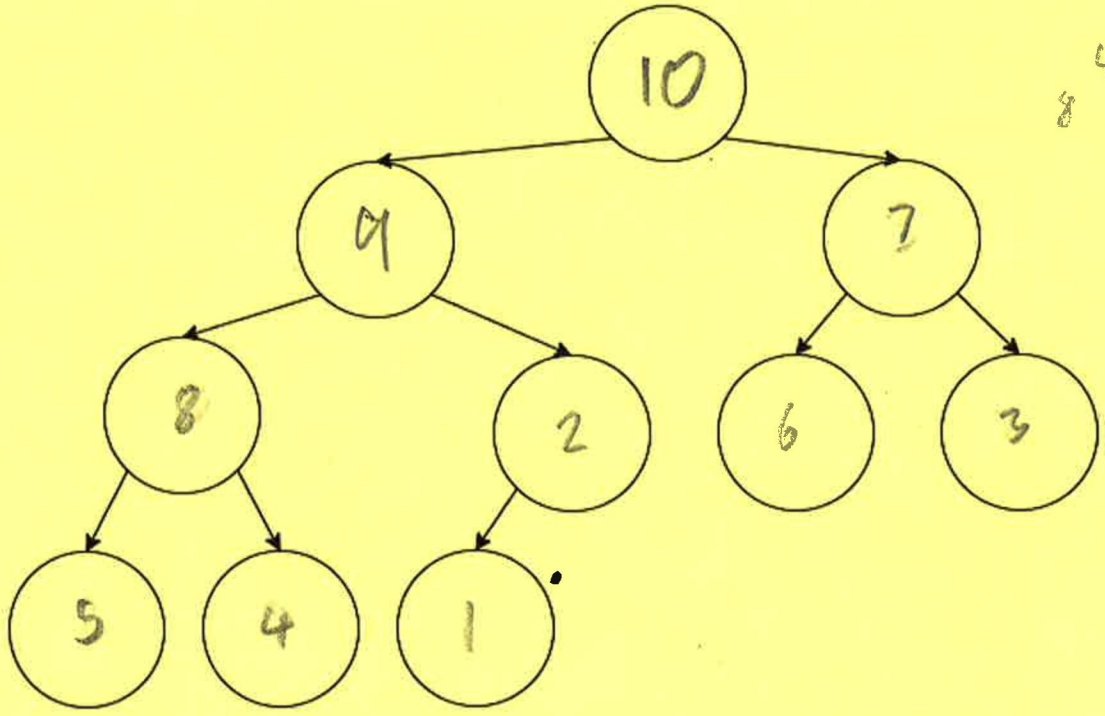
Final Backing Array:

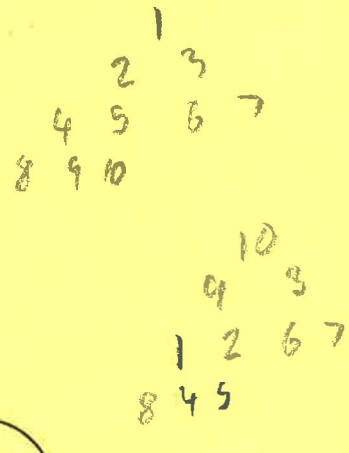


4) Build Heap - Diagramming [10 points]

Perform the buildheap algorithm on the given array below to produce a valid **Max Heap**. Show all intermediate steps by **circling swapped indices.** There should be **only 2** circled elements per row. Finally, fill in the final **Max Heap** at the bottom, in the given tree.

0	1	2	3	4	5	6	7	8	9	10
	1	2	3	4	5	6	7	8	9	10



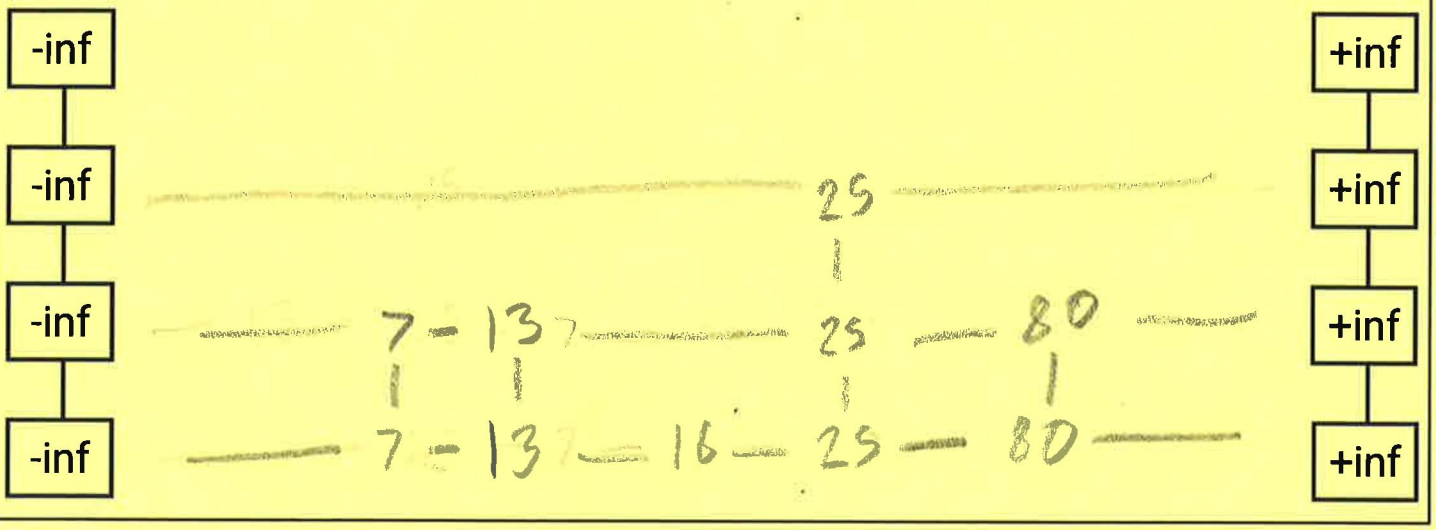


5) SkipList - Diagramming [8 points]

Given the following series of coin flips, draw the final skip list that would be produced if you were to add the data listed below in that order. Notice that you are provided more coin flips than needed, and the drawing may have more levels than needed. **Use heads as promotions.**

Coin Flips: ~~T T H T H T H T T H T~~ H H H T H T T T H H T
Data to Add: 13, 25, 7, 16, 80

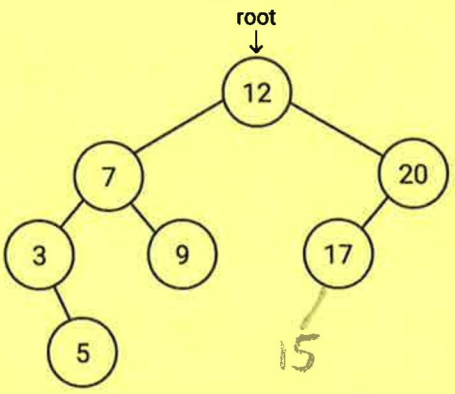
Final Skip List:



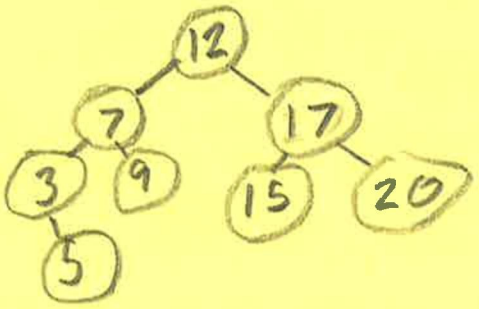
25
13 25 7 80
13 25 7 16 80

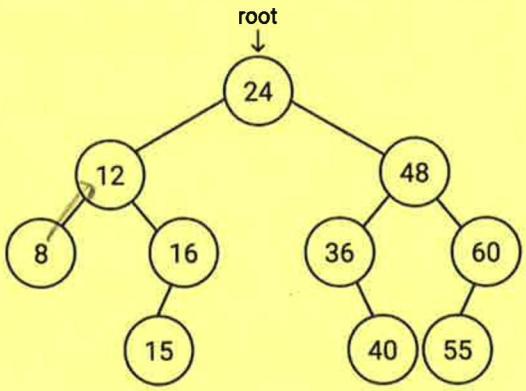
6) AVL - Diagramming [10 points]

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**.

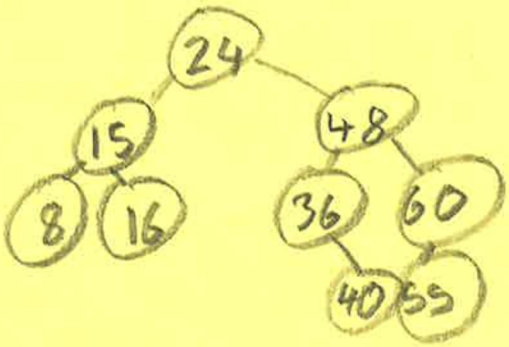


add(15)





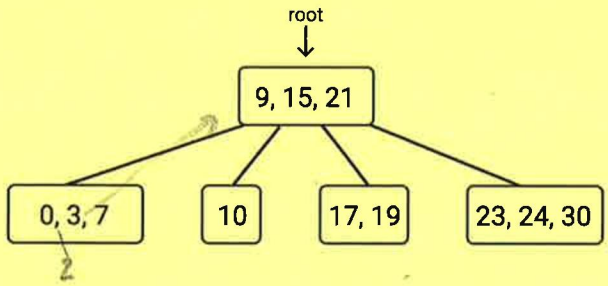
remove(12)



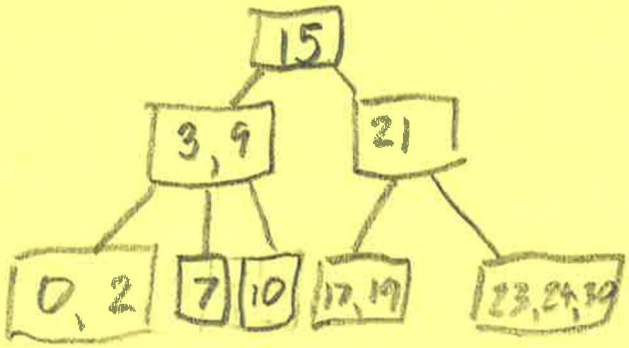
7) 2-4 Tree - Diagramming [10 points]

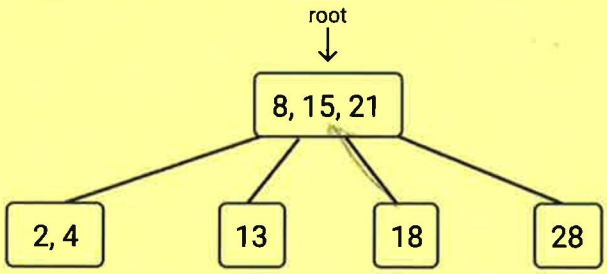
Given the following initial 2-4 trees in the left column below, perform the stated operation. Draw the resulting 2-4 tree in the right column. If you want, you can draw multiple steps (**circle the final step if you do so**). Follow the implementation taught in the **1332 module videos and live lectures**.

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

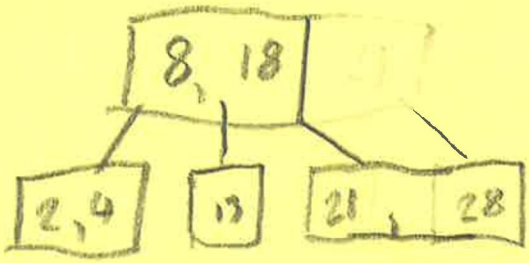


add(2)





remove(15)

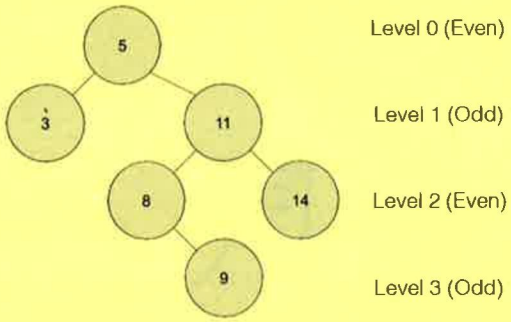


8) BSTs - Coding [15 points]

Goal: Given the following **BST** class, implement the **sumOddLevels()** method that sums all elements that are on odd levels. (The root is on level 0, its children are on level 1, etc.) For example, for a tree of size 1, return 0 since it only has level 0, and no elements on odd levels.

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). Your code should be as efficient as possible.

Example:



sum odd levels: 3 + 11 + 9= 23

Hint: You need to keep track of what level you are on to know if it is even or odd.

```
public class BinarySearchTree {
    private class Node {
        public int data;
        public Node left;
        public Node right;
        public Node(int data, Node left, Node right) { ... }
    }
    private Node root;
    // implementation omitted

    /**
     * Returns the sum of all elements on odd levels in the tree.
     * Should be recursive.
     * @return sum of elements of odd levels
     */
    public int sumOddLevels() {
```

return: recurse (root)

```
private int getHeight(Node node) {
    if (root == null)
        return 0
    return getHeight(left)
}

private int recurse(Node node) {
    if (node == null) {
        return 0;
    }
    if (getHeight(node) % 2 == 0) {
        return recurse(left) + recurse(right);
    else
        return data + recurse(left) + recurse(right)
    }
```

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

9) Deque - Coding [7 points]

Goal: Given the following **LinkedDeque** class, implement the **removeLast()** method.

Requirements: Since Node is an inner class, **you should access its fields directly** (e.g. `node.data`) instead of using getters/setters. **Your code should be as efficient as possible.** You may **not** assume any other method in the LinkedDeque class is implemented.

```
public class LinkedDeque<T> {

    private class Node<T> {
        public T data;
        public Node<T> prev;
        public Node<T> next;
        public Node() { ... }
        public Node(T data, Node<T> prev, Node<T> next) { ... }
    }

    private Node<T> head;
    private Node<T> tail;
    int size;

   /**
    * Removes and returns the last element of the deque.
    *
    * @returns the data formerly located at the back of the deque
    * @throws java.lang.NoSuchElementException if deque is empty
    */
    public T removeLast() {
        // YOUR CODE HERE, USE THE NEXT PAGE IF NEEDED
```

```
if (size == 0) {
    throw new NoSuchElementException("Deque Empty");
}
if (size == 1) {
    T dataR = head.data;
    head = null;
    tail = null;
    size = 0;
    return dataR
}
```

```
tail = tail.prev;
}
T dataR = tail.data
tail = tail.prev;
tail.next = null;

size = size - 1

return dataR
```

```
    } // END OF METHOD

} // END OF CLASS
```

10) Heap - Coding [6 points]

Goal: Fill out the code inside the while loop for the **isMaxHeap()** method. This method will take an array as input, and will need to determine whether the array is a representation of a valid MaxHeap or not. Assume that **index 0 is left empty**, that the size variable is accurate (that is, there are exactly **size** non-null entries in the array), that the array is **contiguous** (that is, there are no null spots in between elements), and that the array contains **no duplicate values** (not counting null). You may also assume that the passed in array is not null.

Requirements: You **must** use the code we have provided. Your code should be as efficient as possible. Do **not** attempt to use any methods from the MaxHeap class other than those you write yourself, doing so may result in large deductions. You are allowed to write and use your own helper methods. However, since **isMaxHeap()** is a static method, any helper method you write should also be static, and must include the **<T extends Comparable<? super T>>** before the return type.

Example:

- If the array is **[null, 10, 8, 5, 7, 6, 4, 2, null]**, you would return true.
- If the array is **[null, 9, 4, 8, 5, 3, 1, null]**, you would return false.

NOTE: While the examples here only use integers, your code **must work with GENERIC data**, as per the method header.

```
public class MaxHeap<T extends Comparable<? super T>> {
    // implementation omitted.
    // Do NOT use any other methods or instance variables from the MaxHeap class.

    /**
    * Determines whether the passed in array is a MaxHeap or not.
    *
    * @param arr the 1-indexed, contiguous array to check.
    * @param size the number of non-null elements in the array.
    * @return true if the array represents a valid MaxHeap, false if it does not.
    */
    public static <T extends Comparable<? super T>> boolean isMaxHeap(T[] arr,
int size) {
        int i = size;
        while (i > 1) {
            // HINT: Think about the relationship between
            // the node at i and its parent.
            // YOUR CODE HERE
                if ( arr[i].compareTo(arr[i/2]) > 0 ) {
                    return false
                }
```

j = j - 1

```
    } // END OF WHILE LOOP
    return true;
} // END OF METHOD
```

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

11) Bonus - World Cup Prediction [1 point]

For 1 extra point, please completely fill in **exactly one** bubble corresponding to the nation that you think is going to win the FIFA World Cup this year ⚽. If you aren't sure who to select, choose ANYONE.

- ○ Brazil
- ○ Belgium
- ○ Argentina
- ○ France
- ○ England
- ○ Spain
- ○ Netherlands
- ○ Portugal
- ○ Denmark
- ○ Germany
- ○ Croatia
- ○ Mexico
- ○ Uruguay
- ○ Switzerland
- ● USA
- ○ Ghana
- ○ Senegal
- ○ Wales
- ○ Iran
- ○ Serbia
- ○ Morocco
- ○ Japan
- ○ Poland
- ○ South Korea
- ○ Tunisia
- ○ Costa Rica
- ○ Australia
- ○ Canada
- ○ Cameroon
- ○ Ecuador
- ○ Qatar
- ○ Saudi Arabia

This page is blank beyond your Wildest Dreams.

(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)