Midterm 2 - S1

● Graded

Student

Sara Huston

Total Points

29 / 30 pts

Question 1

Q1a

1.5 / 1.5 pts

✔ **– 0 pts** Correct

– 1 pt Incorrect

– 0.5 pts Invalid explanation

Question 2

Q1b

1.5 / 1.5 pts

✔ **– 0 pts** Correct

– 1 pt Incorrect complexity

– 0.5 pts Incorrect/missing explanation

Question 3

Q1c

1.5 / 1.5 pts

✔ **– 0 pts** Correct

– 1.5 pts Incorrect

Question 4

Q1d

1.5 / 1.5 pts

✔ **– 0 pts** Correct

– 1.5 pts Incorrect/Blank

– 0.75 pts Only mentions one disadvantage

Question 5

Q2

Resolved **5.5** / 6 pts

– **0 pts** Correct

– **0.25 pts** Portions of Part I incorrect - correct answer is:
A: Perfect
B: Full
C: None
D: CBT
E: None

– **0.25 pts** Portions of Part II incorrect - correct answer is:
A: No
B: Yes
C: No
D: No
E: Yes

– **0.25 pts** Portions of Part III incorrect - correct answer is:
A: Max
B: No
C: No
D: Min
E: No

✔ – **0.5 pts** Portions of Part IV incorrect - correct answer is:
A: No: Not a BST
B: Yes: BF = 1-2 = -1
C: No: Not a BST
D: No: Not a BST
E: No: BF = 3-1 = -2

– **6 pts** Unanswered

– **1.5 pts** Part of question unanswered

Regrade Request **Submitted on: Apr 07**

A heap must be a CBT. Thus, since C is not a CBT, it cannot be a heap.

Agreed, returning credit for C

Reviewed on: Apr 14

Question 6

Q3a

1.75 / 1.75 pts

✔ – **0 pts** Correct

– **0.75 pts** Some Errors.

– **1.75 pts** Incorrect/Missing.

Question 7

Q3b

1.75 / 1.75 pts

✔ **– 0 pts** Correct

– 0.75 pts Minor Error.

– 1.75 pts Incorrect.

Question 8

Q3c

1 / 1.5 pts

– 0 pts Correct

✔ **– 0.5 pts** +-2 in couting.

– 1 pt +-4 in counting.

– 1.5 pts Incorrect/Missing.

Question 9

Q4a

1.5 / 1.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Minor mistakes or didn't show all changes

– 1 pt Major mistakes or incorrect final heap

– 1.5 pts Blank or significantly incorrect

Question 10

Q4b

1.5 / 1.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Minor mistakes or didn't show all changes

– 1 pt Major mistakes or incorrect final heap

– 1.5 pts Blank or significantly incorrect

Question 11

Q4c

2 / 2 pts

✔ **– 0 pts** Correct

– 2 pts Click here to replace this description.

– 0.5 pts Right answer; incomplete/wrong justification.

– 1 pt Wrong answer but fully attempted.

– 1.5 pts Wrong answer but attempted.

– 1 pt Correct answer but no justification.

Question 12

Q5

5 / 5 pts

✔ **– 0 pts** Correct

– 5 pts No answer.

– 2.5 pts Improper while loop condition, wrong declaration and initilization of variables, however logic inside of while loop is correct.

– 4 pts Incorrect while loop and first case returns null instead of -1.

– 0.5 pts Almost completely correct, the while condition needs to make sure that ptr does not equal null as well.

– 5 pts Code has many errors and not remotely close to answer.

– 3.75 pts Does not traverse through the queue to find the index of where the value occurs.

– 0.5 pts Almost correct, your cart node needs to start at head, _prevNode, refers to the node it is pointing to next.

– 0.5 pts Almost correct, Small, simple syntactic errors.

– 3 pts No while loop to traverse through the queue, logic is almost correct but has errors as well.

– 3 pts Initial if condition is correct since _head is an object and cannot be compared with a primitive type, Has a good traversal however there are several errors in the logic inside

– 1 pt Almost correct, has some syntactic errors and fails to increment I in the while loop.

– 0.25 pts Small syntactic errors.

– 0.5 pts 1 or 2 small logical errors.

– 2 pts Incorrect initialization and declaration of variables, many syntactic errors, incorrect return type, does not check if ptr ever equals null.

– 2.5 pts Improper traversal condition, has many logical errors, does not return the correct location (location being the "index" of the node that has that value). However has legitimate attempt at the question.

– 1 pt Ptr needs to start at _head, then you changed it to _tail, while loop needs to take care of the case when ptr is null as well.

– 4 pts Incorrect return type in the if condition and returns the size of the next node, incorrect check in the if condition, and no traversal to find the location of the value.

– 1.5 pts Forgets to initialize a ptr to a node, so that traversal is proper.

– 1.5 pts Does not traverse properly in the for loop, head node must go to its next node and keep checking if its value equals the value given in the argument.

– 0.5 pts While loop must check if curr != null

– 3.5 pts Does not use a traversal method to find the location of the value and incorrect return types

– 0.5 pts Logic is correct, however traversal is backwards.

– 4 pts Condition in the traversal is incorrect, returns the wrong data type, and has an infinite while loop.

– 4 pts No traversal, incorrect return types.

– 3 pts Did not finish the problem but logic thus far is correct.

– 1 pt Does not have proper return type.

- **– 0.5 pts** No location variable
- **– 2 pts** Inner while loop does not traverse properly, meaning the entire while loop segment does not traverse properly.
- **– 0 pts** Click here to replace this description.

Question 13

Q6 **3** / 3 pts

✔ **– 0 pts** Correct

- **– 0.6 pts** incorrectly starting with list of all items
- **– 0.6 pts** incorrect bubbledown at i=3
- **– 0.6 pts** incorrect bubbledown at i=2
- **– 0.6 pts** incorrect bubbledown at i=1
- **– 0.6 pts** incorrect bubbledown at i=0
- **– 1.2 pts** Incorrect heap building order
- **– 3 pts** incorrect or blank
- **– 0.6 pts** building process (order) not clearly shown

66 3/26/2025

COMP 210 – Data Structures and Analysis

Spring 2025, Section 1

Midterm-2 Exam

Date: March 27th, 2025 Student's Full Name: Sara Huston

Points: 30 Duration: 70 minutes, Pages: 8 Student ID: 730459812

NOTE: PLEASE ANSWER ALL QUESTIONS GIVEN IN THIS QUESTION PAPER IN THE SPACES GIVEN.

Question No.	Points allocated
1	6
2	6
3	5
4	5
5	5
6	3
Total Points	**30**

Question 1. [6 points]

a) Does a circular implementation of a Queue using an ArrayList improve the worst-case time complexities? Briefly explain.

NO the circular implementation of a queue will help reduce ammortized time complexity of a deque to O(log N) but in general worst case is still O(N) because the Arraylist has to resize. in worst case # copy over all elements

b) What is the *worst-case* time complexity of an *insert* operation in a Heap with N elements that is stored as an *ArrayList*? Briefly explain why.

O(N) the ArrayList has to resize to a larger Arraylist so it has to copy all N elements over to the new arraylist.

c) What is the Big-O relationship between the height (h) and the number of nodes (N) in a completely **unbalanced** Binary Search Tree?

h = O(?) ⟶ O(N)

N

d) Mention 2 advantages of implementing a Heap as an ArrayList (vs. using pointers)?

(1) Can access the ith element in constant O(1) time as opposed to O(log N) time. which helps other operations.

(2) No storage space allocated to pointers. An Arraylist can be more efficient for storage. Even with the sometimes unused space of an Arraylist, storing a pointers takes ~2N extra space which is less than the average ×1.5 Arraylist size increases.

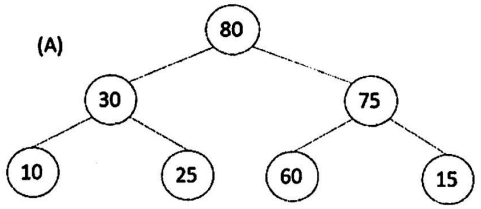
Question 2. [6 points]

For each one of the trees below, fill the table below to identify if each one of them is:

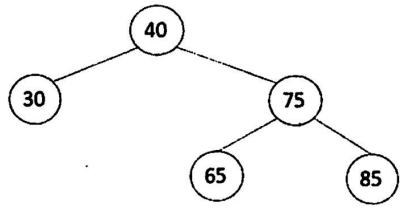
i) Either Perfect or Full or Complete Binary Tree (CBT) or none of the above? (Options: Perfect/Full/CBT/None)
ii) If it's a Binary Search Tree (BST) or not? (Options: Yes/No)
iii) If it's a Heap? If yes, is it a min or max Heap (Options: No, or Min or Max)
iv) If it's an AVL or not? Specify the reason, if necessary by calculating the Balance Factor. (Options: Yes/No, along with reasoning)

Tree	(i) Perfect/Full/ CBT/None	(ii) BST?	(iii) Heap?	(iv) AVL?
A	Perfect	NO	Max	yes. BF = 0 ≤ 1
B	Full	yes	NO	yes. BF = 1 ≤ 1
C	None	NO	NO	yes. BF = 0 ≤ 1
D	CBT	NO	Min	yes. BF = 1 ≤ 1
E	None	yes	NO	No. BF = 2 ≥ 1

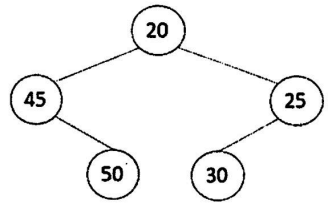
(A)



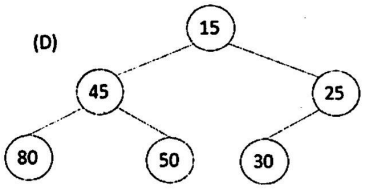
(B)



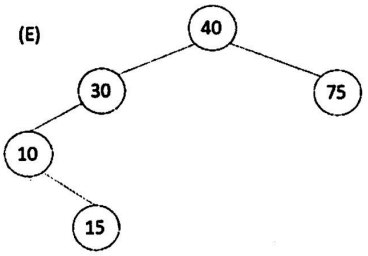
(C)



(D)

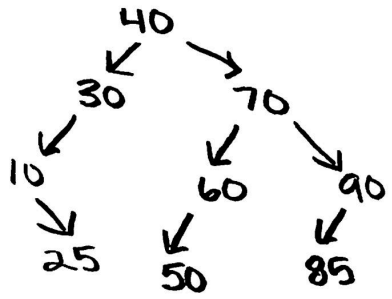


(E)

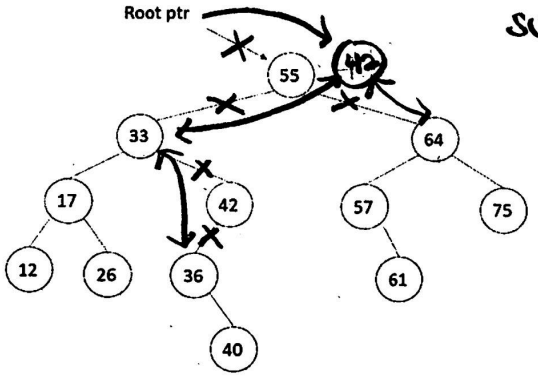


Question 3. [5 Points]

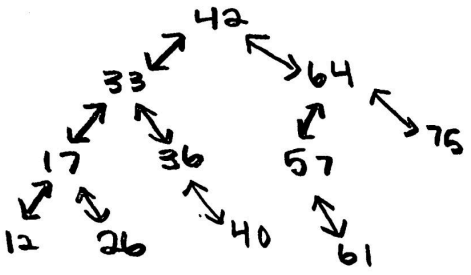
a) Draw the BST that results after inserting the following integers in the order given:
Insert: 40, 30, 70, 60 50, 90, 85, 10, 25.



b) Consider the *Binary Search Tree* shown below. Show the resulting tree after a Delete(55) operation. Assume that a successor is selected from the **left** subtree Clearly show the steps involved.



SUCCESSOR = 42

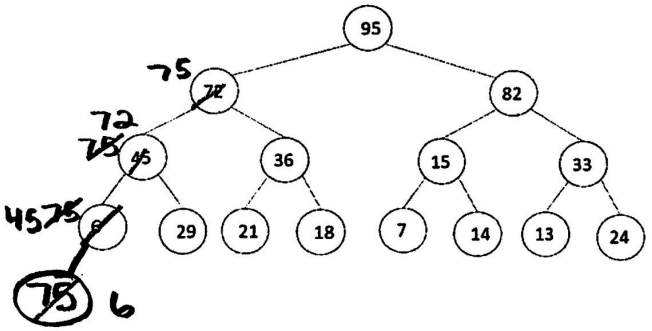


c) Assume that the tree has a root pointer as shown, and each node has a left, right and parent pointer, and that the above *delete* operation moves the **complete node** and not just the *value* of the successor. Indicate in the final tree (above) with **arrows**, all the pointers whose values would have to be changed as a result of the above deletion? Give a count of how many such pointers were changed?

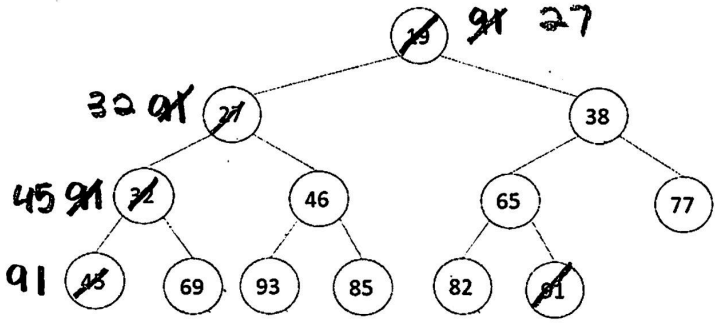
7

Question 4. [5 Points]

a) Consider the **Max-Heap** shown below. Show the heap that results after an **enqueue(75)** operation (mark **all** the changes in each step).



b) Consider the **Min-Heap** shown below. Show the heap that results after dequeuing the *minimum* priority (mark all the changes in each step).



c) Given that the heap in (b) above has N elements and uses an ArrayList data structure, what is the ***Amortized*** (average) complexity of a dequeue operation. Justify based on each of the steps involved in the dequeue operation.

Find last element to swap with highest priority (to keep CBT intact): O(1) because ArrayList can find last elem in constant time

Bubble down operation: O(log N) proportional to height of tree which is balanced so ~log N.

Question 5. [5 Points]

Consider the following implementation of a Queue class (as discussed in lectures). Complete the code for the method '**search(T *value*)**', that searches for the first occurrence of ***value*** starting at the head of the queue and returns its location in the queue, with 0 being the head of queue, or returns -1 if ***value*** is not found.

Note: You can compare two ***values*** using the **equals()** method, e.g. **value1.equals(value2)** would return a boolean, true or false

Queue.java

```
package mt2Queue;
class Node<T> {
    T _value;
    Node<T> _prevNode;
}

// First In First Out (FIFO)
public class Queue<T> {
    private Node<T> _head;   // head of the queue
    private Node<T> _tail;   // tail of the queue
    private int _size;       // size of queue

    // inserts a node to the tail of the queue
    public void enqueue(T value){
        Node<T> node = new Node<>();
        node._value = value;
        node._prevNode = null;
        if (_size == 0){
            _head = node;
        }
        else {
            _tail._prevNode = node;
        }
        _tail = node;
        _size++;
    }
```

```
28      // dequeues from the head of the queue and returns the node
29      public Node<T> dequeue(){
30          Node<T> tempNode = _head;
31          if (_head == null){
32              return(null);
33          }
34          else if (_size == 1) {
35              _head = null;
36              _tail = null;
37          }
38          else {  // _size > 1
39              _head = _head._prevNode;
40          }
41          _size--;
42          tempNode._prevNode = null;  // reset before returning
43          return (tempNode);
44      }
45      public int getSize() { return (_size); }
```

1 → 2 → 3

2 → 3

```
// search for the first occurrence of value starting at head and returns its location in the queue,
// starting with 0 being the head of queue, or -1 if not found.
public int search (T value){
    // Your code here:
      Node<T> ptr = _head;

      for (int i=0; i<_size; i++){
          if (ptr._value.equals(value){
              return (i);
          }
          ptr = ptr._prevNode
      }

    return (-1);
}
```

1 2 3

0 1 2

Question 6. [3 Points]

Given the following array of unsorted integers:

[15, 45, 90, 20, 75, 10, 50, 60]

Use the given Efficient ***Build-Heap*** algorithm to show how to construct a ***maxHeap*** (in O(N) time).

Efficient Heap Build:

- Start with list of all items.
- Set i to parent index of last node.
- While not done:
  - ***bubbleDown*** at i
  - Decrement i