Print Full Name Key Andrew ID Key

Tree Problem Code (15 points)

Show the exact output of the following program. Note the comments in the code. The output begins being produced in the main routine in the second class (Tree.java).

```
package btdiameter;

public class TreeNode {

   private int data;
   private TreeNode lc,rc;

   public TreeNode(TreeNode lc, int data, TreeNode rc){
      this.data = data;
      this.lc = lc;
      this.rc = rc;
   }

   @Override
   public String toString() {
      String LC = "";
      String RC = "";
      if (lc == null) LC = "||--";
      else LC = "<--";
      if (rc == null) RC = "--||";
      else RC = "-->";
      String s = LC + data + RC;
      return s;
   }

   public int getData() {
      return data;
   }
   public TreeNode getLc() {
      return lc;
   }

   public TreeNode getRc() {
      return rc;
   }

   public void setData(int data) {
      this.data = data;
   }

   public void setLc(TreeNode lc) {
      this.lc = lc;
   }
   public void setRc(TreeNode rc) {
      this.rc = rc;
   }
}
```

(0,7)
(0,1), (1,7) (7,8), (8,6), (6,5), (5,4), (4,3)
15
(3,2)

```
package btdiameter;
public class Tree {

   private TreeNode tree;

   public Tree() {
      tree = null;
   }
   public void inOrderTraversal(TreeNode t) {
      if(t != null) {
         inOrderTraversal(t.getLc());
         System.out.println(t);
         inOrderTraversal(t.getRc());
      }
   }

   public void inOrderTraversal(){
      inOrderTraversal(tree);
   }

   public void reverseOrderTraversal(TreeNode t) {
      if(t != null) {

         reverseOrderTraversal(t.getRc());
         System.out.println(t);
         reverseOrderTraversal(t.getLc());
      }
   }

   public void reverseOrderTraversal(){
      reverseOrderTraversal(tree);
   }
```

```
public void insert(int value) {
   TreeNode y = null;
   TreeNode x = tree;
   TreeNode z = new TreeNode(null, value, null);
   while(x != null) {
      y = x;
      if (z.getData() < x.getData()) {
         x = x.getLc();
      }
      else {
         x = x.getRc();
      }
   }
   if(y == null) {
      tree = z;
   }
   else {
      if (z.getData() < y.getData()) {
         y.setLc(z);
      }
      else {
        y.setRc(z);
      }
   }
   z.setLc(null);
   z.setRc(null);
}


public boolean contains(int v) {

   TreeNode x = tree;
   while(x != null) {
      if (v == x.getData()) return true;

      if (v < x.getData()) {
         x = x.getLc();
      }
      else {
         x = x.getRc();
      }
   }
   return false;
}
```

```
public int height(TreeNode t) {
   if( t == null) return -1;
   int heightOnLeft = height(t.getLc());
   int heightOnRight = height(t.getRc());

   if (heightOnLeft > heightOnRight) {
      return heightOnLeft + 1;
   }
   else {
      return heightOnRight + 1;
    }
}

public int height() {
   return height(tree);
}

/* The diameter of a tree is the length of the longest path
  between any two nodes in the tree. This algorithm measures
  path length in terms of the number of edges and not the number
  of nodes on the path.
*/
public int diameter(TreeNode root) {
   if (root == null) return 0;

   int lheight = height(root.getLc());
   int rheight = height(root.getRc());

   /* get the diameter of left and right subtrees */
   int ldiameter = diameter(root.getLc());
   int rdiameter = diameter(root.getRc());

   return Math.max(lheight + 1 + rheight + 1, Math.max(ldiameter, rdiameter));

}
public int diameter() {
   return diameter(tree);
}
```

```
    public static void main(String[] args) {

      Tree t = new Tree();

      System.out.println("a. The initial height is " + t.height());     /* a. show output */

      t.insert(34);
      t.insert(100);
      t.insert(300);
      t.insert(5);
      t.insert(2);
      t.insert(1);

      t.inOrderTraversal();    /* b. show output */

      t.reverseOrderTraversal(); /* c. show output */

      if(t.contains(300)) System.out.println("d. Found 300");    /* d. show this output   */
      else System.out.println("d. No 300 found");               /* d. or this output */

      System.out.println("e. The height is " + t.height());        /* e. show output */

      System.out.println("f. Diameter == " + t.diameter());        /* f. show output */

      t.insert(6);
      t.insert(7);
      t.insert(8);
      t.insert(9);
      System.out.println("g. Diameter == " + t.diameter());        /* g. show output */

      t.insert(0);
      t.insert(-1);
      t.insert(-2);
      System.out.println("h. Diameter == " + t.diameter());        /* h. show output */
    }
}
```

1) Show the output created by a., b., and c. here. (3 Points)

a. The inorial height is -1 ✓

11 --- 1 ---11
<--- 2 ---11
<--- 5 ---11
<--- 34 --->
11--- 100 ---> ✓
11--- 300 ---11

11 300 11
11 100 --->
<--- 34 --->
<--- 5 --- 11
<--- 2 --- 11 ✓
11--- 1 --- 11

2) Show the output of d. and e. here: (5 Points)

Found 300 ✓
The height is 3 ✓

3) Show the output of f., g., and h. here: (7 Points)

Diameter :: 5
Diameter :: 7 ✓
Diameter :: 9

OFF BY 1 error = -1

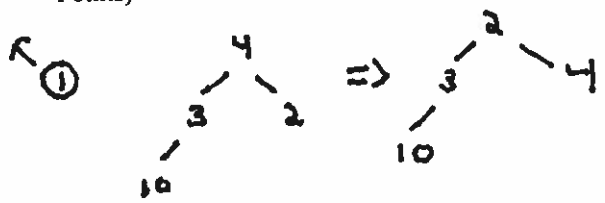
Heaps (15 points)

4) Insert the following numbers into a min heap. Draw a new tree for each heap insertion. (5 Points) -2

10, 1,2,3,4 10

5) What is the height of the tree that you drew in question 4? (2 Points) 2

6) Perform a single delete operation on the heap that you drew in question 4. Draw the resulting tree. (3 Points)

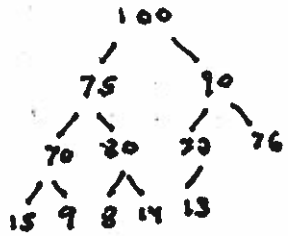


Points on page = 23

Key

7) Consider the following max heap implemented in an array. It is not quite correct. To make it a max heap exactly one swap must occur. What two numbers need to be swapped in order to make this a max heap? (5 points)

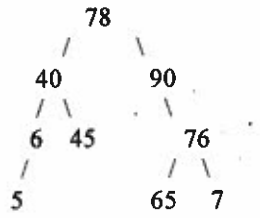
100
75
90
70
80
72
76
15
9
8
14
13



75 AND 80

Binary Trees (16 points)

8. Parts (a), (b), and (c) refer to the following binary tree:



(a) List the data that would be accessed by a pre-order traversal on the given tree by writing out the values in the nodes as they would be accessed, separated by commas. (3 points)

78, 40, 6, 5, 45, 90, 76, 65, 7

(b) List the data that would be accessed by an in-order traversal on the given tree by writing out the values in the nodes as they would be accessed, separated by commas. (2 points)

5, 6, 40, 45, 78, 90, 65, 76, 7

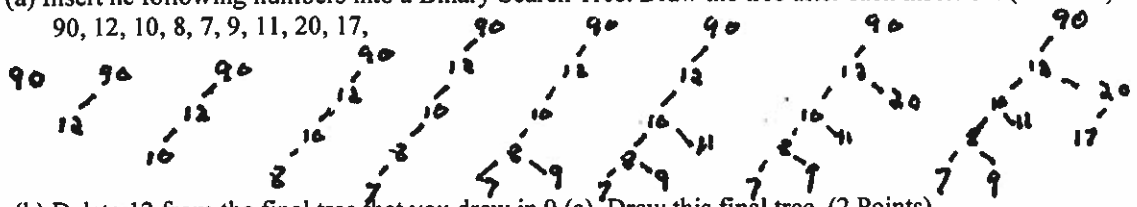
(c) List the data that would be accessed by a post-order traversal on the given tree by writing out the values in the nodes as they would be accessed, separated by commas. (2 points)

5, 6, 45, 40, 65, 7, 76, 90, 78

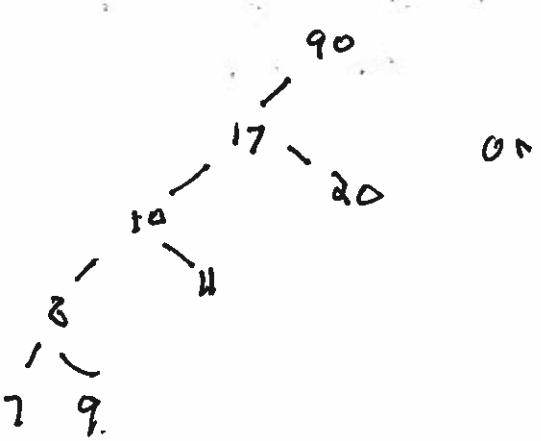
(d) In general, if a binary tree is perfectly balanced (unlike the tree pictured here) and complete with height h, how many leaves, in terms of h, will the tree have? (2 points) 2^h Note, this tree has a perfectly flat bottom.

(e) In general, if a binary tree is perfectly balanced (unlike the tree pictured here) and complete with exactly k leaves. What is the height (in terms of k) of this tree? (2 points) $\log_2 K$ Note, this tree has a perfectly flat bottom.

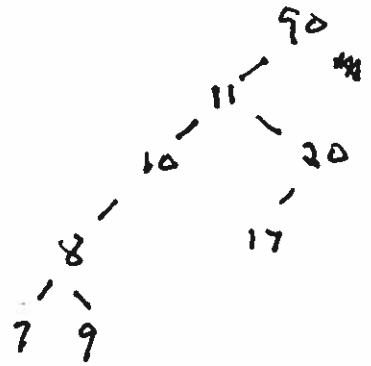
9. (a) Insert he following numbers into a Binary Search Tree. Draw the tree after each insertion. (3 Points)
90, 12, 10, 8, 7, 9, 11, 20, 17,



(b) Delete 12 from the final tree that you drew in 9 (a). Draw this final tree. (2 Points)



or



Project Questions (18 points)

10. Recall the Merkle-Hellman cryptosystem and the Merkle tree that we worked with in Project 1, the spell checker application and the dynamic programming exercise in Project 2, and the calculator problem from Project 3.

The Merkle-Hellman cryptosystem in Project 1 was based on the subset sum problem which is known to be NP-Complete. The problem itself can be described as follows: given a set of numbers *X* and a number k, is there a subset of *X*, which sums to k?

(a) Suppose X = {4, 16, 3, 9, 2, 8, 5} and k = 41. Is there a subset of X which sums to k?
Yes Yes/No (2 points) 28 13 28 + 13 = 41

(b) Suppose Alice sends messages to Bob encrypted with Bob's Merkle-Hellman public key. Circle the one statement that is true? (2 Points)

1. Bob decrypts with a super increasing sequence.
2. Alice encrypts with a super increasing sequence.
3. Alice decrypts with a super increasing sequence.
4. Bob decrypts with a set such as X in part a.
5. Alice decrypts with a set such as X in part a.

(c) Write a method in Java that returns true if there is a subset of X which sums to k and false otherwise. The method signature and pre-conditions are provided. Note, the array x is a super increasing sequence. (6 points)

```
public static boolean subSetSum(int[] x, int n, int k) {
// pre: x is a super increasing and x[0] < x[1] <x[2] <...<x[n-1].
// pre: n is the size of x.
// pre: all integers involved are small enough that overflow is not of concern
// post: returns true if some subset of x sums to k, false otherwise.
FOR (i = N-1; i >= 0; i--){
    IF (X[i] <= K) K = K - X[i];
    IF (K == 0) return true;
}
return false
```

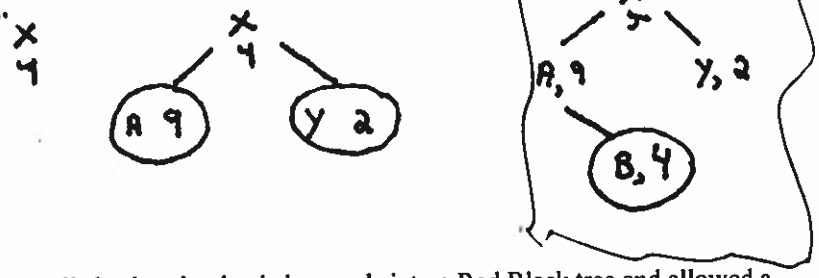
(d) What is the worst case run time complexity of your code in part (c)? Use Big Theta.
(1 Points) $\Theta(N)$

(e) In project 1 we wrote a recursive function that computed probabilities of a World Series win. Briefly describe the run time performance of this function. (1 Point)

IT WAS EXPONENTIAL TIME.
IT WAS NOT POLYNOMIAL TIME.
IT WAS TOO SLOW FOR LARGE N.

(f) In Project 3, we wrote a calculator that processed RPN expressions and used a Red Black Tree. Draw what the Red Black Tree would look like after the following user interaction. Circle RED nodes and leave BLACK nodes un-circled. (4 Points)

X 4 =
A 9 =
Y 2 =
B Y 2 + =



(g) In Project 2 we wrote a spell checker that loaded n words into a Red Black tree and allowed a user to make queries against the tree. We wrote a lookup method that checked if a word was present. Which of the following is true of the best-case lookup method? Circle all correct answers. (2 Points)

1. It ran in O(LogN)
2. It ran in O(1)
3. It ran in Ω(N^2)
4. It ran in Θ(N)
5. It ran in Θ(LogN)

ONE POINT deducted for each wrong answer. Maximum deduction is 2 points.

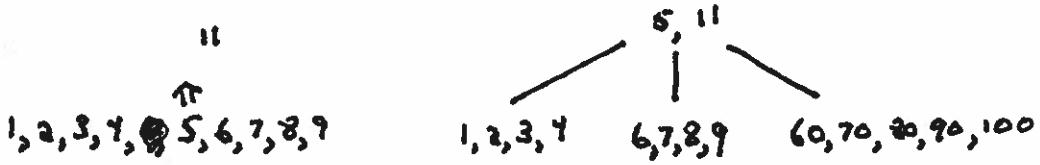
Balanced Trees (15 points)

11. Consider the following B-Tree with a minimum of four and a maximum of eight.

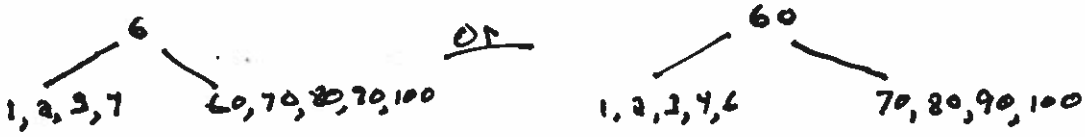
11

1,2,3,4,6 60,70,80,90,100

(a) Redraw the tree after inserting 5,7,8,9. (3 points)



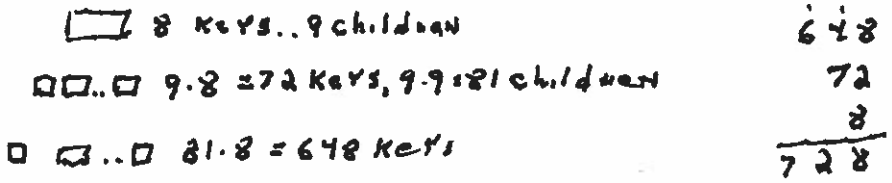
(b) Delete the value 11 from the B-Tree shown above. Begin work from the original tree, not the tree with the values added in Part a. There are two possible answers. Either is fine. (2 points)



(c) Consider again the original tree of height of 1. What is the maximum number of keys that this type of tree (min = 4, max=8) could hold with a height of 1? (1 points). 80

8 keys
9 · 8 = 72 keys in children
72 + 8 = 80 keys total

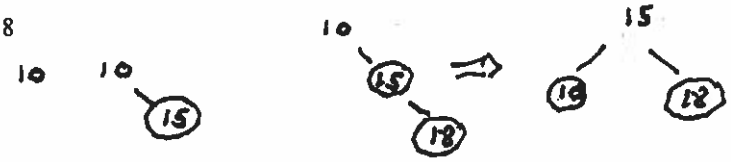
(d) Consider again the original tree of height of 1. What is the maximum number of keys that this type of tree (min = 4, max=8) could hold with a height of 2? (2 points). 728



12. Red Black Trees

(a) Insert the following numbers, one by one, into a Red-Black Tree. Show the tree after each insertion. red vertices should be circled and black vertices should appear without circles. (5 points)

10, 15, 18

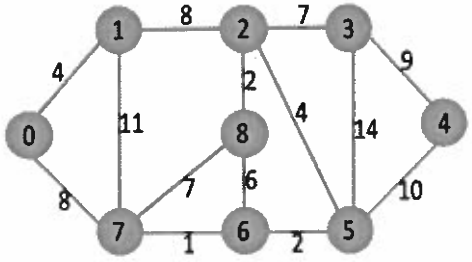


(b) What is the runtime complexity of an inorder traversal of a Red Black Tree ? Use Big Theta notation. (1 Point) $\Theta(N)$ visit each node once

(c) What is the best-case runtime complexity of a Red Black Tree lookup operation? Use Big Theta notation. (1 point) $\Theta(1)$ hit the root

Graph Algorithms(14 points)

13. (a) What is the longest path from node 0 to node 2 in the graph immediately below? Your path must be a list of ordered pairs. For example, you might answer with this list of ordered pairs: (0,1), (1,2). That answer would be a wrong answer but the syntax is correct. Cycles are not permitted. A vertex may be visited no more than once.
(1 point) (0,1), (1,7), (7,8), (8,6), (6,5), (5,4), (4,3), (3,2)



(b) Question 13. (a) was an easy problem for you to solve. But the problem of finding the longest path is considered NP-Hard. Why was your solution easy to come up with? (2 Points) Please answer with one sentence printed neatly and clearly. It involved a small graph.

(c) Dijkstra computes the shortest path. Draw the contents of the distance array for each iteration of Dijkstra's Algorithm as it works on the graph on the last page (Graph 1). The initial state is given. Mark the node to be selected next to the left of the array (note how 1 is marked to the left of the first array.) Fill in each array cell working downward. That is, complete the first row before the second row, etc. Place your answers inside the space provided on the last page. (6 Points)

(d) Draw an adjacency list representation for the graph shown on the last page (Graph 1). Again, show your answer on the last page provided. (2 points)

(e) The graph shown on the last page (Graph 1) is a simple, directed graph. It contains no loops or multiple edges. What is the maximum number of edges that this graph (Graph 1) can contain? Your answer should be an exact number. Answer here. (1 Point) 30

(f) Name an appropriate and fast algorithm to determine whether or not there is a path from one vertex to another in a directed or undirected graph. BFS or DFS (2 points)

Queues (7 points)

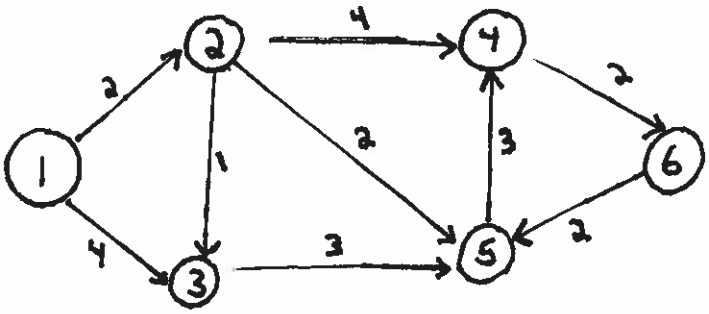
14. Write a Queue class that is designed to process simple integers. Your Queue class will be based on a singly linked list. It will provide two methods – void addAtRear(int x) and int removeFromFront(int x). Please use good syntax (not perfect but good), indentation and comments.

```
CLASS Queue {

    LIST L;

    Queue ( ) {
        L = New LIST()
    }

    addAtRear (INT x) {
        L.AddAtEnd (x)
    }

    INT removeFromFront() {
        return L.removeFromFront()
    }

}
```

```
CLASS LIST {

    Node h, t;

    LIST() { h = t = NULL; }

    Void AddAtEnd (INT x) {
        IF (h == NULL) {
            h = New Node(x, NULL)
            t = h;
        }
        else {
            t.NEXT = New Node(x, NULL)
            t = t.NEXT;
        }
    }

    // pre: h != NULL
    INT removeFromFront() {
        INT d = h.data
        h = h.NEXT
        return d
    }
```

Longer versions, showing
The list implementation now
Fine but not required.

Graph 1



Key

13 C.

	1	2	3	4	5	6
1	0	?	?	?	?	?
2	0	2	4	?	?	?
3	0	2	3	6	4	?
5	0	2	3	6	4	?
4	0	2	3	6	4	8
6	0	2	3	6	4	8

6

13 D.

1 — 2 — 3
2 — 3 — 4 — 5
3 — 5
4 — 6
5 — 4
6 — 5

2

6.5 = 30