Data Structure Assignment 2 - Ruth Dooley 19300753

Question 1

C. 46, 34, 42, 23, 52, 33

Question 2

Maximum chain length: 3. In slot one the keys 28, 19, 10 chain

Minimum chain length: 0. In slot 0, 4, 7 there are 0 keys (Empty slot)

Average chain length: Can be calculated by $\frac{0+3+1+1+0+1+2+0+1}{9} = 1$

A. 3,0,1

Question 3

B. $i_3 \bmod 10$

Question 4

*Base case highlighted in q4-6

```
Class Node:
        Node left
        Node right
        data

        Node (data){
                data = data
                right = null
                left = null
        }
Class BinaryTreeMirror:
        isSymmetric(node1, node2)
                if (node1 and node 2 are both null)
                        return true
                if ((node 1 and node 2 are not null) and (isSymmetric(node1.left, node2.right)) and
        (isSymmetric(node1.right, node2.left)))
                        return true

        isSymmetric (root)
                if (root.left, root.right)
                        return true


Class Main:
        BinaryTreeMirror myTree
        //Set it up like example
        myTree.root = new Node(1);
        myTree.root.left = new Node(3);
        myTree.root.left.left = new Node(7);
        myTree.root.left.right = new Node(9);
        myTree.root.left.right.left = new Node(15);
```

```
        myTree.root.right = new Node(5);
        myTree.root.right.left = new Node(11);
        myTree.root.right.left.right = new Node(17);
        myTree.root.right.right.right = new Node(13);

        if (myTree.isSymmetric(root))
            print("symmetric");
        else
            print(" not symmetric");
```

Question 5

```
Class Node:
        Node left
        Node right
        data

        Node (data){
                data = data
                right = null
                left = null
        }

Class BinaryTreeMirrorShape:
        Node root;

        mirrorSym (){
                root = mirror(root)
        }

        mirrorSym (node){
                if (node isnull)
                        return node;

    Node left = mirrorSym(node.left);
    Node right = mirrorSym(node.right);

    node.left = right;
    node.right = left;

    return node;
        }

  order(){
    order(root);
  }
```

```
    order(node) {
        if (node isnull)
            return;

        order(node.left);
        print(node.data + " ");
        order(node.right);
    }

Class Main:
        BinaryTreeMirrorShape myTree
        //Set it up like example
        myTree.root = new Node(2);
        myTree.root.left = new Node(1);

        myTree.root.right = new Node(9);
        myTree.root.right.left = new Node(3);
        myTree.root.right.left.right = new Node(6);

        myTree.root.right.left.right.left = new Node(5);
        myTree.root.right.left.right.left.left = new Node(4);
        myTree.root.right.left.right.right = new Node(7);
        myTree.root.right.left.right.right.right = new Node(8);

        myTree.mirror();
        print("Inorder traversal of binary tree is : ");
        myTree.inOrder();
```

Question 6

```
Class BinaryDistance:
        Class Node:
                Node left
                Node right
                data

                Node (data){
                        data = data
                        right = null
                        left = null
                }


        //Global variables
        x = -1;
        y = -1;
        distance = 0;

        level (node, data, level){
```

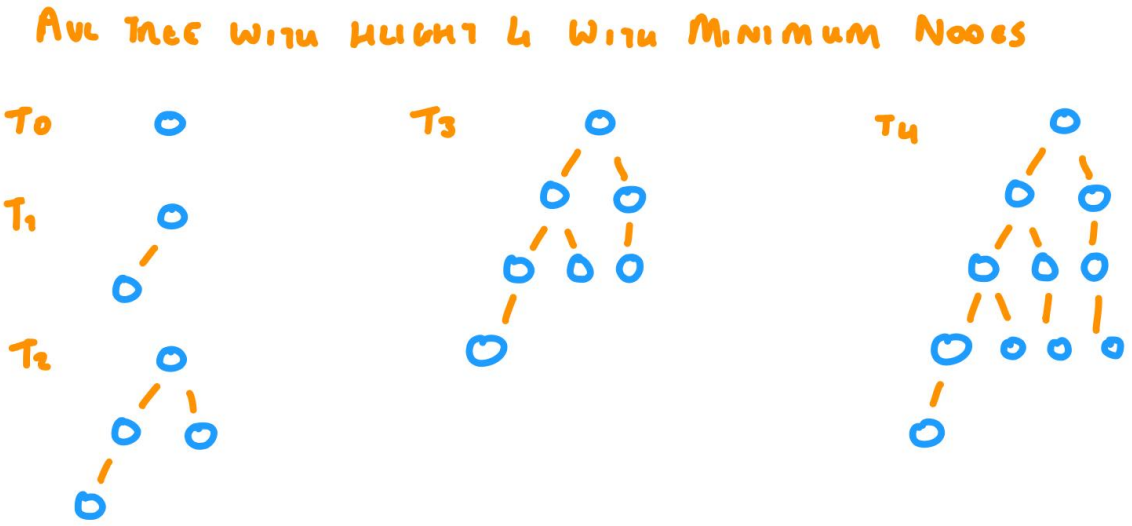
```
            //Base case
            if (node is equal null)
                        return -1;

            if (node.data is equal data)
                        return level;

             int l = level (node.left, data, level + 1);
            return (l != -1)? l : level(node.right, data, level + 1);
}

distanceHelper (root, node1, node2, level){
            //Base case
            if (root is equal null)
               return null;

            if (root.data is equal node1)
                        x = level;
                        return root;

            if (root.data is equal node2)
                        y = level;
                        return root;

            Node leftPath = distanceHelper (root.left, node1, node2, level + 1);
            Node rightPath = distanceHelper (root.right, node1, node2, level + 1);


            if (leftPath is not null && rightPath is not null){
                  distance = (x + y) - 2 * level;
                  return root;
            }

             return (leftPath is not null)? leftPath : rightPath;
}

dist (root, node1, node2){
            x = -1;
             y = -1;
   distance = 0;
   Node distanceHelper = distance helper (root, node1, node2, 1);

   if (x is not -1 && y is not -1)
      return distance;

   if (x is not-1)
      distance = level(distanceHelper, node2, 0);
      return distance;
```

```
            if (y is not -1)
               distance = level(distanceHelper, node1, 0);
               return distance;

            return -1;
         }

//Set up like the question
public static void main(String[] args) {

   Node root = new Node(44);
   root.left = new Node(17);
   root.left.left = new Node(8);
   root.left.right = new Node(32);
   root.left.right.left = new Node(28);
   root.left.right.left.left = new Node(21);
   root.left.right.left.right = new Node(29);

   root.right = new Node(88);
   root.right.left = new Node(65);
   root.right.left.left = new Node(54);
   root.right.left.right = new Node(82);
   root.right.left.right.left = new Node(76);
   root.right.left.right.left.right = new Node(80);
   root.right.right = new Node(97);
   root.right.right.left = new Node(93);

   dist (root, 54, 93)); // = 4
   dist (root, 21, 80)); // = 9
}
```

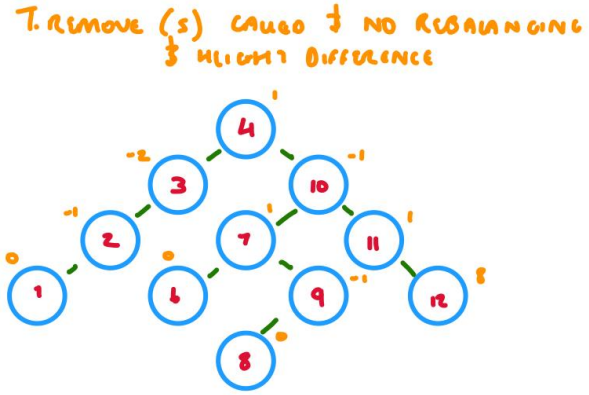
Question 7

The maximum number of nodes in an AVL tree with the given height of h is 2^h. This is due to each node branching exactly twice.

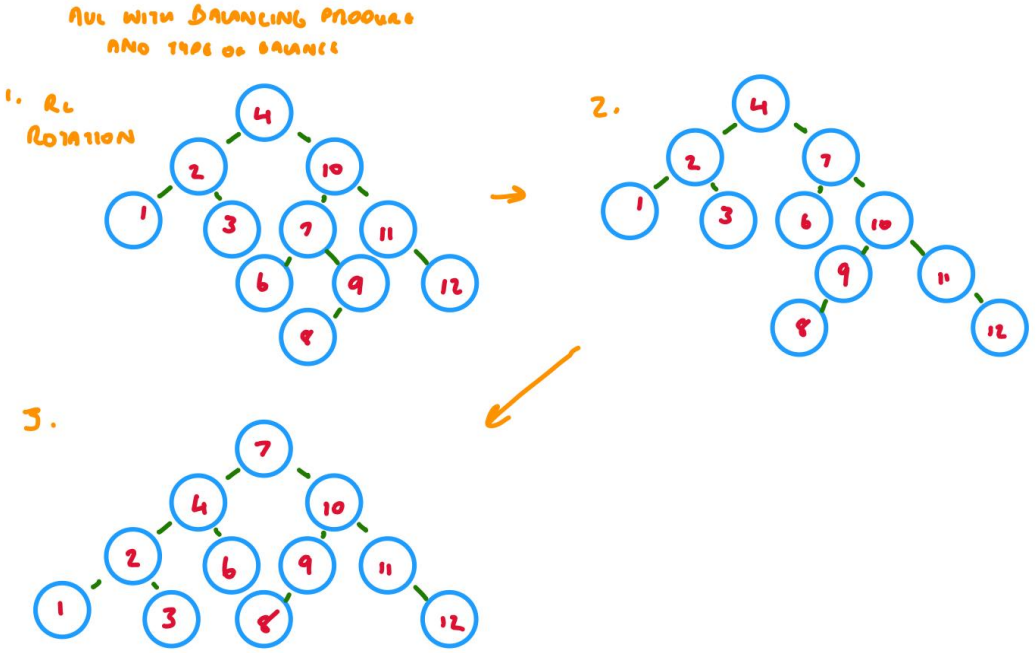
Question 8



Question 9



(a)



(b)