Common Recurrences and their solutions.

label	recurrence	solution
A	`T(n) = T(n/2) + O(1)`	$O(\log n)$
B	`T(n) = T(n/2) + O(n)`	$O(n)$
C	`T(n) = 2T(n/2) + O(1)`	$O(n)$
D	`T(n) = 2T(n/2) + O(n)`	$O(n \log n)$
E	`T(n) = T(n-1) + O(1)`	$O(n)$
F	`T(n) = T(n-1) + O(n)`	$O(n^2)$
G	`T(n) = 2T(n-1) + O(1)`	$O(2^n)$

`TreeNode` and `ListNode` classes as used on this test. In some problems the type of the `info` field may change from `int` to `String` and *vice versa*

```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;

    TreeNode(int x){
        info = x;
    }
    TreeNode(int x,TreeNode lNode,
                      TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
}
```

```
public class ListNode {
    int info;
    ListNode next;
    ListNode(int val) {
        info = val;
    }
    ListNode(int val,
             ListNode link){
        info = val;
        next = link;
    }
}
```

Write your netid:

PROBLEM 1 : (***Honor Code (2 points)***)

Write your name here to acknowledge that you have read the rules for taking this exam and that you will follow them:

Write your name here to acknowledge that you will follow the Duke Community Standard:

PROBLEM 2 : (***Sorting ginorSt Selection (4 points)***)

For the following list of numbers, apply two passes of **selection sort** to them and show the result after each pass. A pass is one execution of the loop. Use the version of selection sort demoed in lecture that focuses on selecting the min.

```
15 7 14 3 6 21 5
```

Enter the result **after the first pass** here (be sure to put a blank between each adjacent pair of numbers):

Enter the result **after the second pass** here (be sure to put a blank between each adjacent pair of numbers):

PROBLEM 3 : (***Sorting ginorSt Bubble (4 points)***)

For the following list of numbers, apply two passes of bubble sort to them and show the result after each pass. A pass is one execution of the loop. Use the version of **bubble sort** demoed in lecture that focuses on bubbling the max.

```
15 7 14 3 6 21 5
```

Enter the result **after one pass** here (be sure to put a blank between each adjacent pair of numbers):

Enter the result **after the second pass** here (be sure to put a blank between each adjacent pair of numbers):

PROBLEM 4 : (***Sorting ginorSt Insertion (4 points)***)

For the following list of numbers, apply two passes of **insertion sort** to them and show the result after each pass. A pass is one execution of the loop. Use the version of insertion sort demoed in lecture.

```
15 7 14 3 6 21 5
```

Enter the result **after the first pass** here (be sure to put a blank between each adjacent pair of numbers):

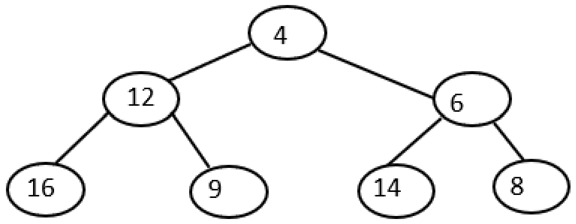
Enter the result **after the second pass** here (be sure to put a blank between each adjacent pair of numbers):

PROBLEM 5 : (***Priority Queues: not min-heap (2 points)***)

For this problem a min-heap is stored in an array but could be viewed visually as a binary tree.

Part A: (2 points)

Consider the following min-heap a student has drawn that is incorrect.



Which node does not follow the min-heap property for nodes?

Circle the correct answer:

`4    6    8    9    12`

PROBLEM 6 : (***Priority Queues: min-heap (8 points)***)

Part B: (8 points)

Draw the visual tree that results from inserting the following eight numbers into an empty min-heap in this order:

`20 6 9 3 7 2 5 10`

After your min-heap is complete answer these questions about the resulting min-heap. You will not submit the drawing.

1. What is the number at the root of the tree that represents the min-heap?

   Your answer:

2. List the numbers from the min-heap in the order they would appear in the array associated with this min-heap. Include a space between each pair of numbers. For example you might list them like this (this is not the correct answer, just an example of how to list them):

   `20 6 9 3 7 2 5 10`

   Your answer:

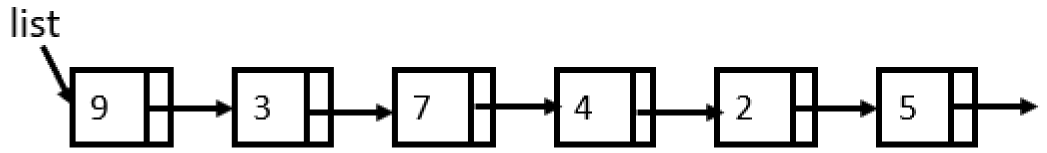
3. Apply the operation removeMin once to your min-heap.

   List the numbers from the resulting min-heap in the order they would appear in the array. Include a space between each pair of numbers.

   Your answer:

PROBLEM 7 : (***Linked Lists Tracing (7 points)***)

Consider the following linked list, and the code shown below.



Trace through the code to see what happens when mysteryList(list) is called, where list is the above linked list. Answer the questions below to give values at certain points during the execution of the code.

Remember the ListNode is defined on page 2 or here: bit.ly/201spring20e

```
1  public ListNode mysteryList(ListNode list) {
2
3      ListNode first = list;
4      while (list.next.next.next != null) {
5          list = list.next;
6      }
7      // Question 1
8      ListNode temp = list.next;
9      ListNode temp2 = first.next.next;
10     // Question 2
11     list.next = first.next;
12     first.next = temp;
13     list.next.next = temp.next;
14     temp.next = temp2;
15     // Question 3
16     return first;
17 }
```

1. What is the value of **list.info** when execution is on line 7?
2. What is the value of **temp2.info** when execution is on line 10?
3. Assume the original list that list is pointing to could be written out as: 9 3 7 4 2 5

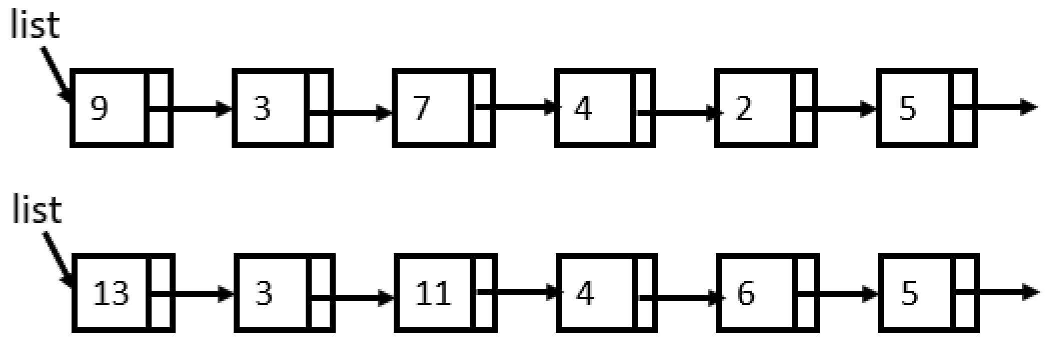
   Write out the complete linked list that first is pointing to when the execution is on line 15. Be sure to put a blank between each pair of numbers.

   The list **first** is pointing to is:

PROBLEM 8 : (***Linked Lists Every Other Non-Recursively(7 points)***)

Complete the method named **addEveryOther** that has two parameters, one of type **ListNode** named **list** and one of type **int** named **value**. This method adds **value** to every other node starting with the first node.

For example, consider the first linked list shown below. After the call `addEveryOther(list, 4)`, list would be pointing to the second linked list shown below, where 4 has been added to the first, third and fifth nodes.



This method should be written **non-recursively** and has been started for you. Note that the list could contain 0 or more nodes.

```
    public ListNode addEveryOther(ListNode list, int value) {
        ListNode first = list;
        if (list == null) {
            return null;
        }


        // Missing code



        return first;
    }
```

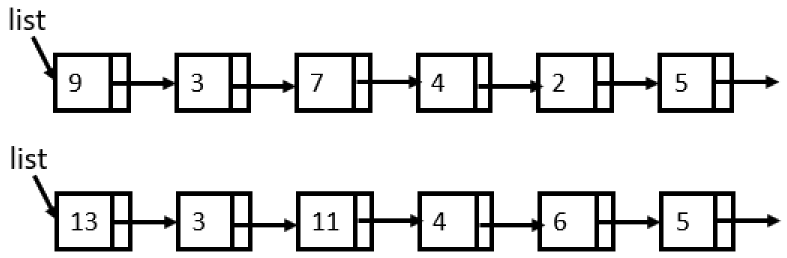
1. Write the missing code here:
2. What is the big-Oh running time of your method for a linked list with N nodes?

PROBLEM 9 : (***Linked Lists Every Other Recursively(7 points)***)

This is the same problem, but write the method **addEveryOther** recursively. Here is the statement problem again.

Complete the method named **addEveryOther** that has two parameters, one of type **ListNode** named **list** and one of type **int** named **value**. This method adds **value** to every other node starting with the first node.

For example, consider the first linked list shown below. After the call `list = addEveryOther(list, 4)`, list would be pointing to the second linked list shown below, where 4 has been added to the first, third and fifth nodes.



This method should be written **recursively**. Note that the list could contain 0 or more nodes.

```
public void addEveryOther(ListNode list, int value) {



     // body missing



}
```

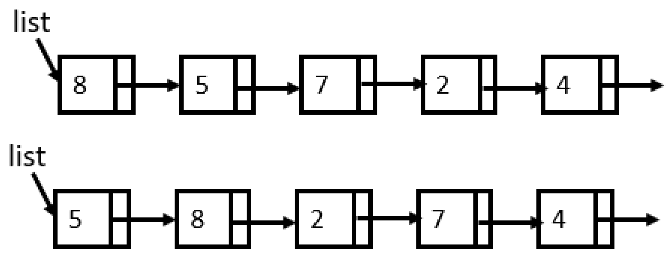
1. Write the missing code here:
2. Write the recurrence relation for this method. Assume T(n) is the time to solve this method with n nodes. T(n) =

PROBLEM 10 : (***Linked Lists Swapping Every Other (6 points)***)

Complete the method named **swapPairs** that has one parameter of type **ListNode** named **list**. This method swaps the values of every pair of nodes starting with the first pair, until the end of the list. Values are only swapped once, so the second pair to be swapped is the third and fourth nodes in the linked list. If there is an odd number of nodes, then the last node's value is unchanged. Note that the list could contain 0 or more nodes.

For example, consider the first linked list shown below. After the call `swapPairs(list)`, list would be pointing to the second linked list shown below, where the first two nodes swapped their values, and the second two nodes swapped their values. The last node is unchanged.

Note that you are not moving nodes, but changing the values in nodes.



This method can be written with or without **recursion**.

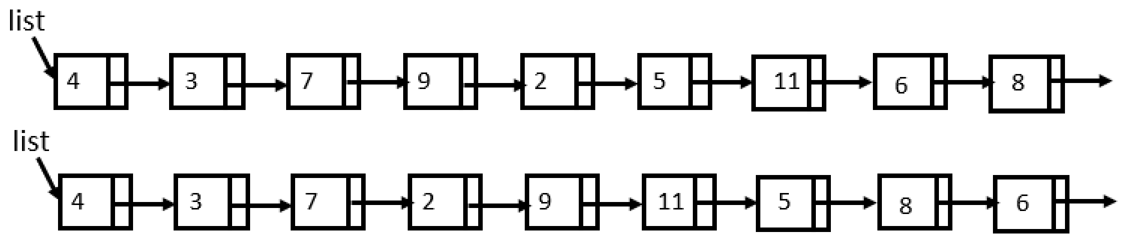
```
public void swapPairs(ListNode list) {

\\ missing code




}
```

1. Write the missing code here:

PROBLEM 11 : (***Linked Lists Swapping Every Other K (6 points)***)

Complete the method named **swapPairsK** that has three parameters, one of type **ListNode** named **list**, one of type **int** named **k**, and one of type **boolean** named **swap**. This method swaps the values of every pair of nodes starting with the first node whose value is greater than or equal to k. Starting with that node, every pair swaps values to the end of the list. If there are an odd number of nodes once swapping starts, then the last node will be unchanged. Note that the list could contain 0 or more nodes.

For example, consider the first linked list shown below. After the call `swapPairsK(list, 9, false)`, list would be pointing to the second linked list shown below. Note that the fourth node in the first list is the first node that has a value greater than or equal to 9. Starting with the fourth node, every other pair of nodes are swapped: 9 and 2 are swapped, 5 and 11 are swapped and 6 and 8 are swapped.



Hint: the boolean variable swap can be called as false and flipped to true once you find the first node greater than or equal to k.

This method should be written **recursively**. There should be no while/for loops in your code.

You may call the method swapPairs from Part A. Assume that method works as intended.

```
public void swapPairsK(ListNode list, int k, boolean swap) {

\\ missing code




}
```

Write the missing code here:

PROBLEM 12 : (***Binary Search Tree: 8 points***)

Draw a **binary search tree** on paper by inserting these numbers one at a time in the order shown into an empty tree. Do not use any balancing method (we have not talked about any such methods). Remember that each node in a binary search tree has nodes less than or equal to it to its left and nodes greater than it to its right.

```
20 6 9 3 7 2 5 10
```

Answer the following questions after you have drawn the binary search tree. You will not turn in the drawing.

1. What is the number at the root of your binary search tree?

   Your answer:

2. How many nodes in your tree have exactly one child?

   Your answer:

3. How many leaves are in your tree?

   Your answer:

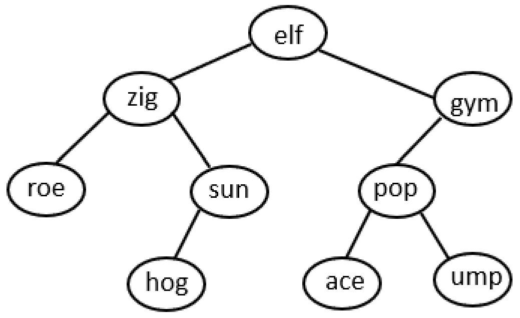
4. List the leaves in your tree from left to right. Include a blank between each pair of numbers.

   Your answer:

PROBLEM 13 : (***Regular Binary Tree: 5 points***)

Consider the following **binary tree** for the problems here. This same tree is used in all the problems for this part. It will be repeated for each question. Assume the info field in the TreeNode is of type String for this part.

Consider the following binary tree and code.

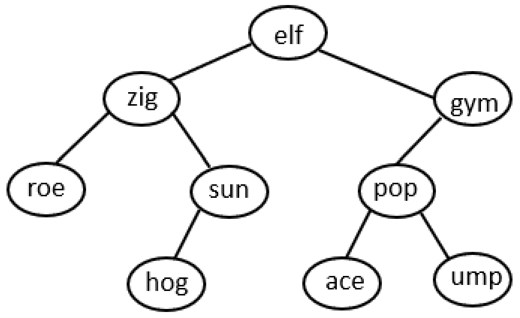


```
public void mystery(TreeNode root) {
    if (root != null) {
        System.out.printf("%d ", root.info);
        mystery(root.left);
        mystery(root.right);
    }
}
```

1. List the first six strings that are printed when this code executes. Be sure to include a blank between each pair of words.
2. What is the recurrence relation for this method if T(n) is the time it takes to execute this method for a balanced tree with n nodes?
3. What is the Big-Oh running time of this method for a balanced tree?

PROBLEM 14 : (***Regular Binary Tree: 3 points***)

Consider the following binary tree from before and code.

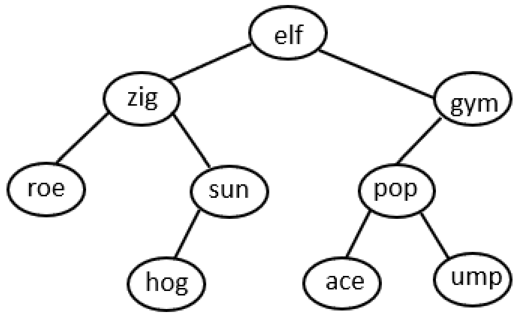


```
public void mystery2(TreeNode root) {
    if (root != null) {
        mystery2(root.right);
        mystery2(root.left);
        System.out.printf("%d ", root.info);
    }
}
```

List the first six strings that are printed when this code executes, in the order they would be printed. Be sure to include a blank between each pair of words.

PROBLEM 15 : (***Regular Binary Tree: 4 points***)

Consider the following binary tree from before and code.

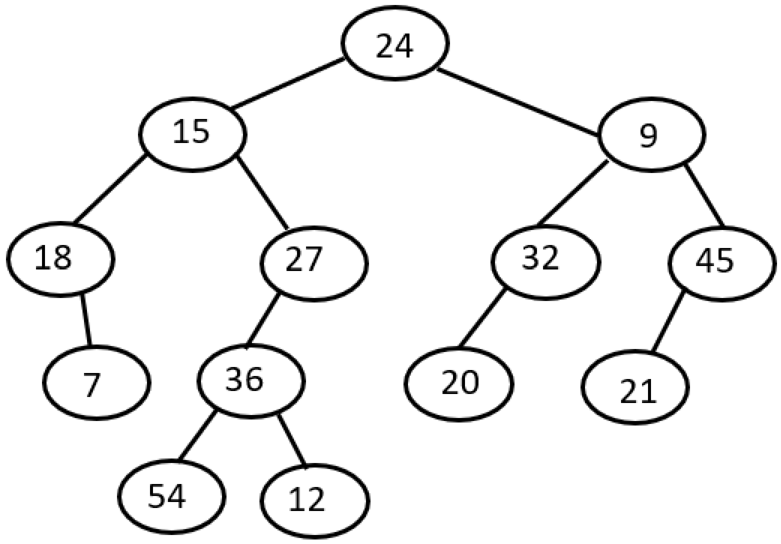


```
public void mystery3(TreeNode root) {
    if (root != null) {
        System.out.printf("%d ", root.info);
        mystery3(root.left);
        System.out.printf("%d ", root.info);
        mystery3(root.right);
    }
}
```

1. List the first eight strings that are printed when this code executes. Be sure to include a string between each pair of words.
2. What is the recurrence relation for this method if T(n) is the time it takes to execute this method for a balanced tree with n nodes?

PROBLEM 16 : (***Scratching up Trees Leftmost (5 points)***)

Consider the binary tree shown below.



Complete the method named **leftmost** that has one parameter of type **TreeNode** named **root**. This method returns a TreeNode (or rather a reference to) that is the leftmost node in the tree from root. For example, in the tree above in which root contains 24, the call leftmost(root) would return a reference to the TreeNode that contains 18. The call leftmost(root.right) would return a reference to the TreeNode that contains 20, and the call leftmost(root.left.right) would return a reference to the TreeNode that contains 54, as 54 is the leftmost node of 27.

For this problem you can assume the tree has 1 or more nodes. If the tree has only one node, then that node is the leftmost node.

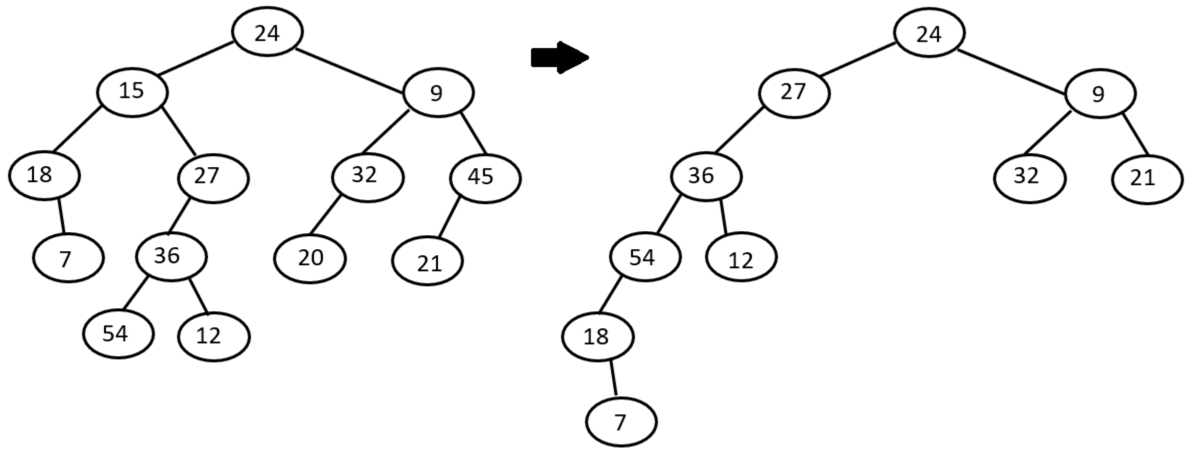
```
public TreeNode leftmost(TreeNode root) {


// body of code missing


}
```

Enter your code here:

What is the recurrence relation for this method if T(n) is the time it takes to execute this method for a balanced tree with n nodes?

PROBLEM 17 : (***Scratching up Trees Leftmost (11 points)***)

Consider the binary tree shown on the left, and the result of the call `removeNodes(root, 5)` shown on the right.



Complete the method named **removeNodes** that has two parameters, one of type **TreeNode** named **root**, and one of type **int** named **number**. This method returns the modifed tree in which all nodes that are a multiple of **number** have been removed.

For example, in the tree above on the left with root 24, the call `removeNodes(root,5)` would return the resulting tree on the right after removing the nodes that are a multiple of 5, which are: 15, 20 and 45.

If a node is a multiple of **number** there are three cases for removing it.

Case 1) The node is a leaf node, it is just removed. See node 20 in the example. It is a leaf node. It is removed in the tree on the right.

Case 2) The node has one child, then the node is removed and its child takes its place. See node 45. It is removed in the tree on the right.

Case 3) The node has two children. In this case the right child (and its subtree) takes the node's place. The left child (and its subtree) then becomes a left child of the leftmost node of the original node's right child. See node 15 in the example. Node 27 and its subtree is now the left subtree of 24. The subtree of 18 and 7 are now the left subtree of 54. Note that 54 was the leftmost node of 27 for the tree on the left. .

The code for **removeNodes** is started below with missing pieces. Write the code for the missing pieces in the questions below.

You can call the method **leftmost** from Part A. Assume it works as intended.

```
public TreeNode removeNodes(TreeNode root, int number) {
    if (root == null) return root;

    // PART 1 - Two recursive calls missing for subtrees


    if (root.info % number != 0) {
        return root;
    }

    // node must be removed - there are three cases

    // PART 2 - case if node is a leaf node


    // PART 3 - case if node has only one child


    // Part 4 - case if node has two children

}
```

Part 1: 3 pts

Give the code PART 1 for the two recursive calls for handling the subtrees.

Part 2: 2 pts

Give the code for PART 2 for removing the node that is a leaf node.

Part 3: 3 points

Give the code for PART 3 for removing the node that has exactly one child.

Part 4: 3 points

Give the code for PART 4 for removing the node that has two children.