1. **Recursive Tracing**, 15 points. Consider the following method:

**public static int mystery(int x, int y) {**

**if (x < 0) {**

**return -mystery(-x, y);**

**} else if (y < 0) {**

**return -mystery(x, -y);**

**} else if (y < x) {**

**return 0;**

**} else {**

**return 1 + mystery(x, y - x);**

**}**

**}**

For each call below, indicate what value is returned:

Method Call Value Returned

mystery(10, 18) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(5, 12) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(2, 10) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(4, -15) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

mystery(-3, -20) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}Top of Form

Bottom of Form

2. **Recursive Programming**, 15 points. Write a method writeNums (ONLY ONE METHOD FOR THIS PROBLEM) that takes an integer n as a parameter and that writes the first n integers starting with 1 to System.out in sequential order and separated by commas. All output should be on the current line of output.

For example, the following calls:

writeNums(5);

System.out.println(); // to complete the line of output

writeNums(12);

System.out.println(); // to complete the line of output

should produce the output:

1, 2, 3, 4, 5

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

You must exactly reproduce the format of the examples above. Your method

should throw an IllegalArgumentException if passed a value less than 1.

You may NOT use a while loop, for loop or do/while loop to solve this problem; you must use recursion.

Write your solution to writeNums below.

3. Suppose we are performing a binary search on a sorted array called numbers initialized as follows:

// index 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

int[] numbers = {-5, -1, 0, 3, 9, 14, 19, 24, 33, 41, 56, 62, 70, 88, 99};

int index = binarySearch(numbers, 37);

Write the indexes of the elements that would be examined by the binary search (the mid values in our algorithm's code) and write the value that would be returned from the search. Assume that we are using the binary search algorithm shown in the textbook.

Top of Form

|  |  |  |
| --- | --- | --- |
| indexes examined |  |  |

|  |  |  |
| --- | --- | --- |
| value returned | For these problems, assume that we are using the standard ListNode class:  public class ListNode {  public int data; // data stored in this node  public ListNode next; // link to next node in the list    public ListNode() {  this(0, null);  }  public ListNode(int data) {  this(data, null);  }  public ListNode(int data, ListNode next) {  this.data = data;  this.next = next;  }  }  And that we are writing methods for a class called LinkedIntList that has a  single data field of type ListNode called front:  public class LinkedIntList {  private ListNode front;  <methods>  }  *In solving these problems, you may not call any other methods of the class.*  1. Linked Lists. (15 points) For this problem, let’s assume that our linked list is a sorted list of integers. Write a method removeDuplicates that removes all repetitions of a particular value. For example, if the list contains the following values:  (2, 3, 3, 4, 4, 4, 5, 8, 8, 8, 9)  The method call:  list.removeDuplicates();  would remove all duplicates occurrences of the values 3, 4, 8, and 9 from the current list, yielding:  (2, 3, 4, 5, 8, 9) |  |

Linked Lists, 10 points.

4. Write a method **removeMin** that could be added to the **LinkedIntList** class that removes the smallest element value from the linked list. Suppose a LinkedIntList variable named list stores the following elements:

**[8, 4, 7, 2, 9, 4, 5, 3]**

If you made the call of list.removeMin();, the list would then store the elements:

**[8, 4, 7, 9, 4, 5, 3]**

If the list has more than one occurrence of the minimum value, the first occurrence is the one that should be removed. For example, [8, 10, 13, 8, 8, 14, 9, 8, 11] becomes [10, 13, 8, 8, 14, 9, 8, 11]. If the list is empty, you should throw a NoSuchElementException.

Obey the following restrictions in your solution:

* **Do not call any other methods on the LinkedIntList object, such as add, remove, or size.**
* **Do not create new ListNode objects** (though you may have as many ListNode variables as you like).
* Do not use other data structures such as arrays, lists, queues, etc.

Assume that you are adding this method to the LinkedIntList class

   Write your solution to **removeMin** below.

Stacks and Queues:

5. Write a method called isSorted that takes a stack of integers and returns true if the stack is sorted and false otherwise. A stack is considered sorted when its integers are in non-decreasing order (i.e. increasing order with duplicates allowed) when read from bottom to top.

So, a sorted stack has its smallest integer on the bottom and its largest integer on the top. A stack that contains fewer than two integers is sorted by definition. For example, suppose that a variable called s stores the following sequence of values:

bottom [-12, 0, 1, 8, 8, 8] top

then a call on isSorted(s) should return true. If s had instead contained the following values:

bottom [-9, 10, 43, 24, 97] top

then a call on isSorted(s) should return false, because 24 is less than 43. You may use one Queue as auxiliary storage to solve this problem. You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. Your method must restore the stack so that it stores the same sequence of values after the call as it did before.

You have access to the following two methods and may call them as needed to help you solve the problem:

public static void s2q(Stack s, Queue q) { ... } //changes a stack to a queue

public static void q2s(Queue q, Stack s) { ... } //changes a queue to a stack

6. Write a method collapse that takes a Stack of integers as a parameter and that collapses it by replacing each successive pair of integers with the sum of the pair. For example, suppose a stack stores this sequence of values:

bottom (7, 2, 8, 9, 4, 13, 7, 1, 9, 10) top

Assume that stack values appear from bottom to top. In order words, 7 is on the bottom, with 2 on top of it, with 8 on top of it, and so on, with 10 at the top of the stack. The first pair should be collapsed into 9 (7 + 2), the second pair should be collapsed into 17 (8 + 9), the third pair should be collapsed into 17 (4 + 13) and so on to yield:

bottom (9, 17,17, 8, 19) top

As before, stack values appear from bottom to top. If the stack stores an odd number of elements, the final element is not collapsed. For example, the sequence:

bottom (1, 2, 3, 4, 5) top

would collapse into:

bottom (3, 7, 5) top, with the 5 at the top of the stack unchanged.

Use one queue as auxiliary storage to solve this problem. Do not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. You also may not solve the problem recursively.

In writing your method, assume that you are using the Stack and Queue interfaces. Your method should take a single parameter: the stack to collapse.

5. **Sets** (10 points) Write a static method **numInCommon** that takes two Lists of integers as parameters and returns the number of unique integers that occur in both lists. Use one or more **Sets** as storage to help you solve this problem.

For example, if one list contains the values [3, 7, 3, -1, 2, 3, 7, 2, 15, 15] and the other list contains the values [-5, 15, 2, -1, 7, 15, 36], your method should return 4 (because the elements -1, 2, 7, and 15 occur in both lists).