✕ Define an overloaded constructor for MyLinkedList that creates a LinkedList from a given ArrayList. A sample use of the constructor is shown below. 0/1

```
public static void main(String[] args) {
  ArrayList<Integer> alist = new ArrayList<>(List.of(10, 40, 20, 30));
  MyLinkedList<Integer> llist = new MyLinkedList<>(alist);
  llist.add(50);
  System.out.println(alist); // [10, 40, 20, 30]
  System.out.println(llist); // [10, 40, 20, 30, 50]
}
```

Choose a sequence of code from below to form the overloaded constructor. NO SPACE, NUMBER or SYMBOL in your answer. A sample answer with the correct format is abcdefghi. All lines are used exactly once.

```
  public MyLinkedList(ArrayList<E> list) {
```

a: else

b: size++;

c: head = newNode;

d: tail = newNode;

e: for (E e: list) {

f: if (head == null)

g: tail.next = newNode;

h: Node<E> newNode = new Node<>(e);

i: }

```
  }
```

hfcadebgi ✕

Correct answers

ebhfcagdi

ehbfcagdi

ehfcagbdi

ehfcagdbi

✕ Define a countOdd method that counts the number of odd integers in an array. A sample use of the countOdd method is shown below. 0/1

```
public static void main(String[] args) {
  int[] array = {1, 2, 3, 5, 7, 8};
  System.out.println (countOdd(array, 0)); // Output is 4
}
```

Choose a sequence of code from below to form the countOdd. NO SPACE, NUMBER or SYMBOL in your answer. A sample answer with the correct format is abcdef. All lines are used exactly once.

```
public static int countOdd (int[] array, int n) {
a:  return 0;
b:  return 0 + countOdd (array, n+1);
c:  return 1 + countOdd (array, n+1);
d:  else
e:  if (n == array.length)
f:  else if (array[n] % 2 == 0)
}
```

eafcdb ✕

Correct answer

eafbdc

1/1

Which of the following statements are TRUE? Choose TWO.

```
protected void inorder (TreeNode<E> parent) {
1:  if (parent == null)
2:    return;
3:  inorder (parent.left);
4:  System.out.print (parent.element + " ");
5:  inorder (parent.right);
}
```

- [ ] Line 4 has a recursive call.
- [ ] Line 1 is a recursive case.
- [ ] Line 2 has a recursive call.
- [x] Line 5 has a recursive call.
- [x] Line 1 is a base case.
- [ ] Line 3 does not have a recursive call.

The insert method for binary search trees is provided below (same as lecture). 0/1

```
public boolean insert(E e) {
  if (root == null)
    root = new TreeNode<>(e);
  else {
    TreeNode<E> parent = null;
    TreeNode<E> current = root;
    while (current != null) {
      if (e.compareTo(current.element) < 0) {
        parent = current;
        current = current.left;
      }
      else if (e.compareTo(current.element) > 0) {
        parent = current;
        current = current.right;
      }
      else
        return false;
    }
    if (e.compareTo(parent.element) < 0) // THISLINE
      parent.left = new TreeNode<>(e);
    else
      parent.right = new TreeNode<>(e);
  }
  size++;
  return true;
}
```

What is the value of "current" variable when the code reaches THISLINE?

- (●) root ✗
- ( ) true
- ( ) null
- ( ) parent
- ( ) false

Correct answer

- (●) null

✓ Which of the following statements about binary search trees are TRUE? Choose TWO. 1/1

- [ ] Inorder traversal prints the root first.
- [x] Postorder traversal prints the root last. ✓
- [ ] Inorder traversal prints the root last.
- [ ] Postorder traversal prints the root first.
- [x] Preorder traversal prints the root first. ✓
- [ ] Preorder traversal prints the root last.

✓ Which of the following statements can determine whether a LinkedList is empty or not? Choose TWO. 1/1

- [x] head == null ✓
- [ ] head == 0
- [x] size == 0 ✓
- [ ] size == null