```
public class Q4{

    /*
     * For this question I implemented my own version of a linked list that could be used for this problem.
     *
     * Colin Gallacher
     **/


    /*
     * Inner node class
     * Contains the methods and fields required by the linked list in this problem
     */
    class Node{

        public int value;
        public Node next;

        public Node(){
            value = 0;
            next = null;

        }

        public Node(int value_in, Node link){
            value = value_in;
            next = link;


        }


        public void setNext(Node next_link){

            next = next_link;
        }

        public Node getNext(){

            return next;
        }

        public void setData(int data_in){

            value = data_in;

        }

        public int getData(){

            return value;
        }


    }

    /*
     * Inner singly linked list class implements the methods associated with the singly linked list ADT
     */
    class SinglyLinkedList{

        protected Node start;
        protected Node end;
        protected int size;

        public SinglyLinkedList(){
            start = null;
        end = null;
        size = 0;

    }

    public boolean add(int value, int position){

        Node iter = start;
        Node newNode = new Node(value,null);


     if (size == 0){

         start = newNode;
         newNode.setNext(end);
         size++;

     }
     else if (position == 0){

         newNode.setNext(start);
         start = newNode;
         size++;

     }



     else if (position <= size && position >= 0){

            for(int i =1; i<=position; i++){


                if(i == position){


                    Node temp = iter.getNext();
                    iter.setNext(newNode);
                    newNode.setNext(temp);

                }

                iter=iter.getNext();

            }
            size++;
            return true;
    }

        return false;
    }

    public boolean addToStart(int value){


        return add(value,0);

    }

    public boolean addToEnd(int value){

        return add(value, size);
    }

    public void printList(){

        Node temp = start;
        System.out.println("Size: " + size);

```

```java
        System.out.print("[ ");
        for(int i = 0; i<size; i++){

        System.out.print(temp.value + "  ");
        temp=temp.getNext();

        }

        System.out.println(" ]");

    }



    /*
     * This is the iterative call to count the size of the singly linked list
     *
     */

    public int CountIterative(){

        int count=0;

        Node temp = start;

        while(temp!=null){

            temp = temp.getNext();
            count++;


        }


        return count;

    }

    /*
     * This is the recursive call to count the size of the singly linked list
     *
     */

    public int CountRecursive(Node n){


        int count;

        if(n == null){
            return 0;

        }

        count =1;

        if(n.getNext()!= null){

             count = CountRecursive(n.getNext())+1;
        }

        return count;

    }


}



```

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

        Q4 outer = new Q4();

        Q4.SinglyLinkedList SLL =  outer.new SinglyLinkedList();

        //Build a list and test the function methods

        SLL.addToStart(5);
        SLL.addToStart(4);

        SLL.addToEnd(7);
        SLL.add(2, 1);
        SLL.add(7, 2);
        SLL.add(9, 4);
        SLL.add(10, 3);
        SLL.add(4,  5);
        SLL.add(2, 1);
        SLL.add(7, 2);
        SLL.add(9, 4);
        SLL.add(10, 3);
        SLL.add(4,  5);
        SLL.add(4,  5);
        SLL.add(2, 1);
        SLL.add(7, 2);
        SLL.add(9, 4);
        SLL.add(10, 3);
        SLL.add(4,  5);


        //Print the list

        SLL.printList();


        // Question 4 problems.

        int count = SLL.CountIterative();

        System.out.println("Iterative call count: " + count);

        count = SLL.CountRecursive(SLL.start);

        System.out.println("Recursive call count: " + count);

    }

}


OUTPUT:


Size: 19
[ 4  2  7  10  2  4  9  7  10  2  4  4  9  7  10  5  4  9  7   ]
Iterative call count: 19
Recursive call count: 19
```