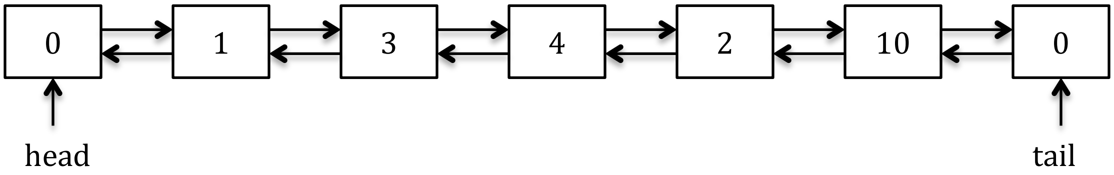
CS272 Midterm 2
Close book; Time: 11:45am-1:00pm, Nov. 20, 2014 Points: 100pts

Banner ID: ______________________ Name: ________________________ Score: _____

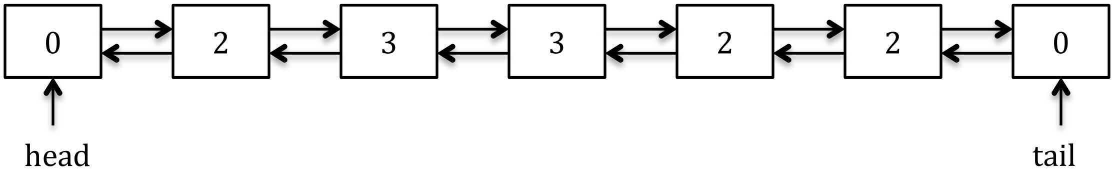
Q1. (27pts) Given the classes *DNode* and *DoublyLinkedListDummy*,
1.1) (7pts) implement the **removeFromLast** method.

```
public class DNode<E>
{
    public E data;                  //The value for the node, which is of generic type
    public DNode<E> next = null;    //The next node of the current one
                                    //For the last node of a list, this is either null
                                    //or is a reference to the dummy tail node
    public DNode<E> prev = null;    //The previous node of the current one
                                    //For the first node of a list, this is either null
                                    //or is a reference to the dummy head node
    public DNode(){;}
}

public class DoublyLinkedListDummy<E> {
    //The actual list nodes in a doubly linked list (with dummy nodes) should not
    //include the dummy head and tail nodes
    public int manyItems;       //The number of actual list nodes in the list
    public DNode<E>  head;      //The head of a doubly linked List, a dummy node, NOT null
    public DNode<E>  tail;      //The tail of a doubly linked List, a dummy node, NOT null

    public DoublyLinkedListDummy ( ){ //Constructor
        head = new DNode<E>();
        tail = new DNode<E>();
        head.next = tail;
        tail.prev=head;
        manyItems = 0;
    }

    // Remove the last actual node (i.e., the node that the tail points to).
    public void removeFromLast(){









    }
    public DoublyLinkedListDummy<E> f(int p1, int p2)
    {
        //make sure that p1>=0 and p2<manyItems and p1<=p2
        DoublyLinkedListDummy<E> myList = new DoublyLinkedListDummy<E>();
        DNode<E> cursor = head.next;
        int i = 0;
        while(cursor!=tail && i < p1){
                cursor = cursor.next;
                i++;
        }

        while(cursor!=tail && i < p2){
                DNode<E> newnode = new DNode<E>();
                newnode.data = cursor.data;
                newnode.prev = myList.tail.prev;
                newnode.next = myList.tail;
                myList.tail.prev.next = newnode;
                myList.tail.prev = newnode;
                (myList.manyItems)++;
                i++;
                cursor = cursor.next;
        }
        return myList;
    }
}
```

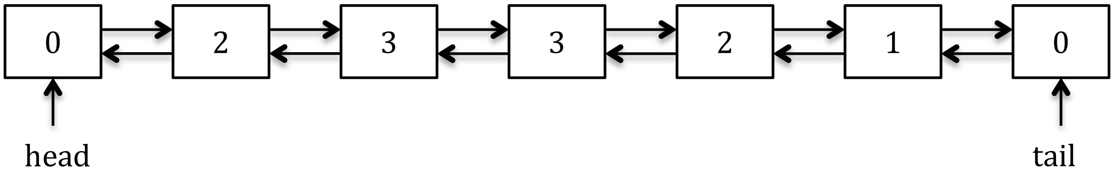
1.2) (10 pts) Given the function f(), show the result of running f(2,4) on a given list shown as follows.



1.3) (10 pts) Design and implement a function to get ONE value that occurs most frequently in a doubly linked list.
For example, given the following doubly linked list, the value 2 is returned because 2 occurs 3 times in this list.



If the following doubly linked list is given, then you can return either 2 or 3 because both of them occur two times in the list.



}

Q2. (20 pts) Answer the following questions by utilizing the *SNode* class given below.
2.1) (10 pts) Finish the ***pop*** method for the class *LinkStack*.
2.2) (10 pts) Implement an O(1) ***enqueue*** method for the class *LinkedQueue*.

```
public class SNode <E>{
    public E data;
    public SNode<E> next = null;
    public SNode(){; }
}

public class LinkStack<E> {
    public SNode<E> top;
    public LinkStack()                  {top = null;}

    public void push(E e) {//Insert data to the stack
        SNode<E> newtop = new SNode<E>();
        newtop.data = e;
        newtop.next = top;
        top = newtop;
    }

    public E pop() {







    }
}

public class LinkedQueue<E> {

    public SNode<E> rear = null;                        //the rear of a queue
    public SNode<E> front = null;                       //the front of a queue
    public LinkedQueue(){; }

    public void enqueue(E e) {










    }
}
```

Q3. (45pts) Given the classes *AVLNode* and *AVL* as follows, please
3.1) (10pts) Finish the ***getLeftHeight*** and ***getRightHeight*** methods in the *AVLNode* class.
3.2) (10 pts) Finish the ***searchNonRecursion*** method in the *AVL* class.

```
class AVLNode{
        public int data;            //the element value for this node
        public AVL left;            //the left child of this node
        public AVL right;           //the right child of this node
        public int height;          //height of the tree rooted at this node

        public AVLNode()            {data = 0; left = new AVL(); right = new AVL(); height = 1;}
        public AVLNode(int initData)   {data = initData; left = new AVL();right = new AVL();
        height = 1;}

        public void setHeight(){//Set the height of the tree rooted at this node
                this.height = 1+Math.max(getLeftHeight(), getRightHeight());
        }

        public int getLeftHeight(){// Get the height of the left subtree




        }

        public int getRightHeight(){// Get the height of the right subtree




        }
}

public class AVL {
        public AVLNode          root; //instance variable to denote the root of the AVL tree
        public AVL()            {root = null;}

        public AVLNode searchNonRecursion(int e){










        }
}
```

3.2) (13 pts) Given the following function in the AVL class.

```
public boolean f (int e)
{
        if(root==null){
                root = new AVLNode(e);
                return true;
        }else if (e==root.data){
                return false;
        }else if(e<root.data){
                return (root.left.f(e));
        }else{
                return (root.right.f(e));
        }
}
```

Please draw the *tree* structure after running the following code:

```
AVL tree = new AVL();
tree.f(10);
tree.f(5);
tree.f(20);
tree.f(3);
tree.f(15);
tree.f(15);
tree.f(25);
tree.f(22);
```

Result tree:

Is the result tree a valid AVL tree? Justify your answer.

3.4) (10 pts) Given an AVL tree with *n* nodes, derive an equation to represent the height of the AVL tree *h* as a function of *n* such that *h* < *c f(n)* where *f(n)* is a logarithmic function of *n* and *c* is a constant.

Q4. (10 pts) Recursive thinking.
Given the following function ***f***:

```
public static void f(int[] A, int i,int j)
{
        if(i<j){
                int tmp = A[i];
                A[i]=A[j];
                A[j]=tmp;

                f(A,i+1,j-1);
        }
}
```

What is the output after running the following 6 lines of code?

```
    int[] A = new int[]{1,2,3,4,5};
    f(A,0,A.length-1);
    for(int i=0;i<A.length;i++){
        System.out.print(A[i]+" ");
    }
    System.out.println("");
```

Result: ____

========================== END ==============================