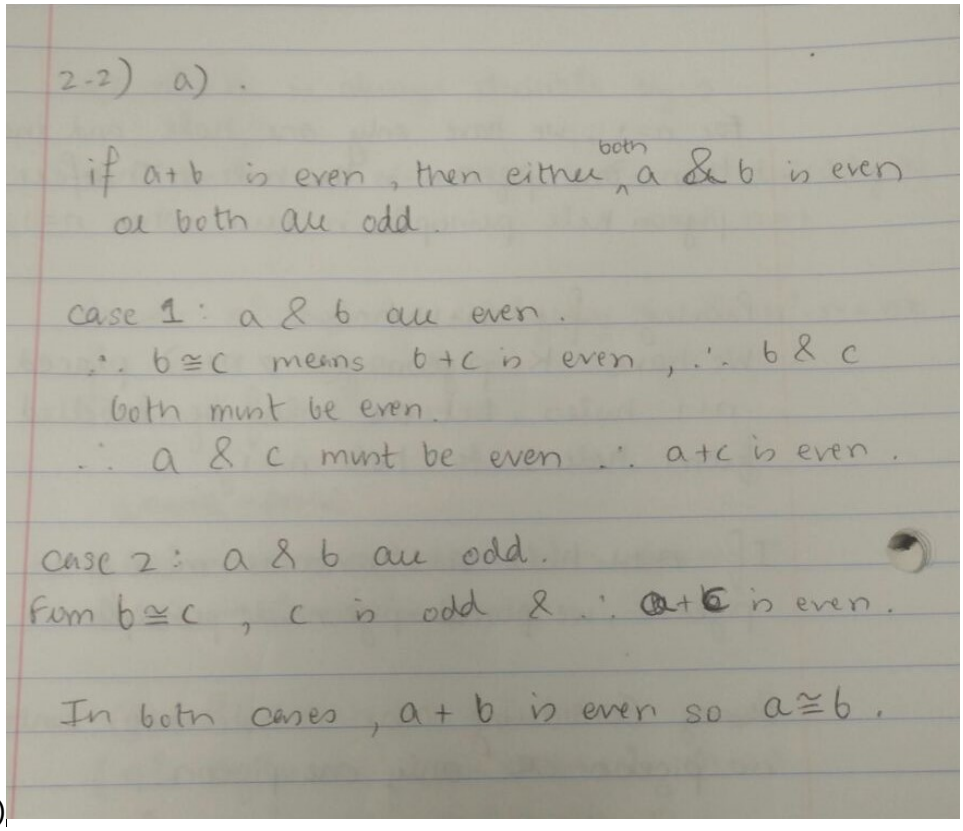
1.8 Imagine that you have been assigned to implement a simple sequential search on an array. The problem is that you want the search to be as general as pos- sible. This means that you need to support arbitrary record and key types. Describe ways to generalize the search function to support this goal. Consider the possibility that the function will be used multiple times in the same program, on differing record types. Consider the possibility that the func- tion will need to be used on different keys (possibly with the same or different types) of the same record. For example, a student data record might be searched by zip code, by name, by salary, or by GPA.

Vague question. AI said professor will post more information.

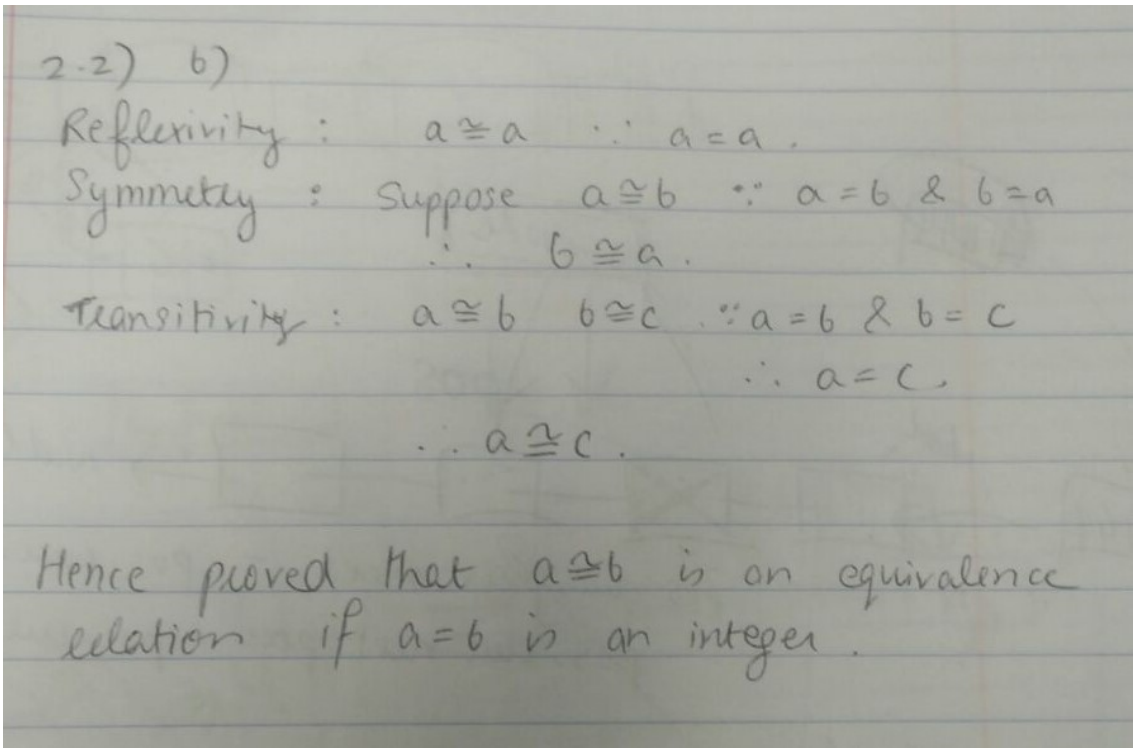
I would use a hashmap to store all the data and then do a search depending on the user input.

2.2 For each of the following relations, either prove that it is an equivalence relation or prove that it is not an equivalence relation. DONE
(a) For integers a and b, a ~ b if and only if a + b is even.
(d) For nonzero rational numbers a and b, a ~ b if and only if a=b is an integer.

a)



b.)



2.2) b)

Reflexivity : $a \cong a \quad \because a = a$.

Symmetry : Suppose $a \cong b \quad \because a = b \ \& \ b = a$

$\therefore b \cong a$.

Transitivity : $a \cong b \quad b \cong c \quad \because a = b \ \& \ b = c$

$\therefore a = c$.

$\therefore a \cong c$.

Hence proved that $a \cong b$ is an equivalence relation if $a = b$ is an integer.

2.3 State whether each of the following relations is a partial ordering, and explain why or why not.

2.3]

a) is father of.

No. if a is father of b
& b is father of c
a is not father of c
Not transitive.

b) is Ancestor of.

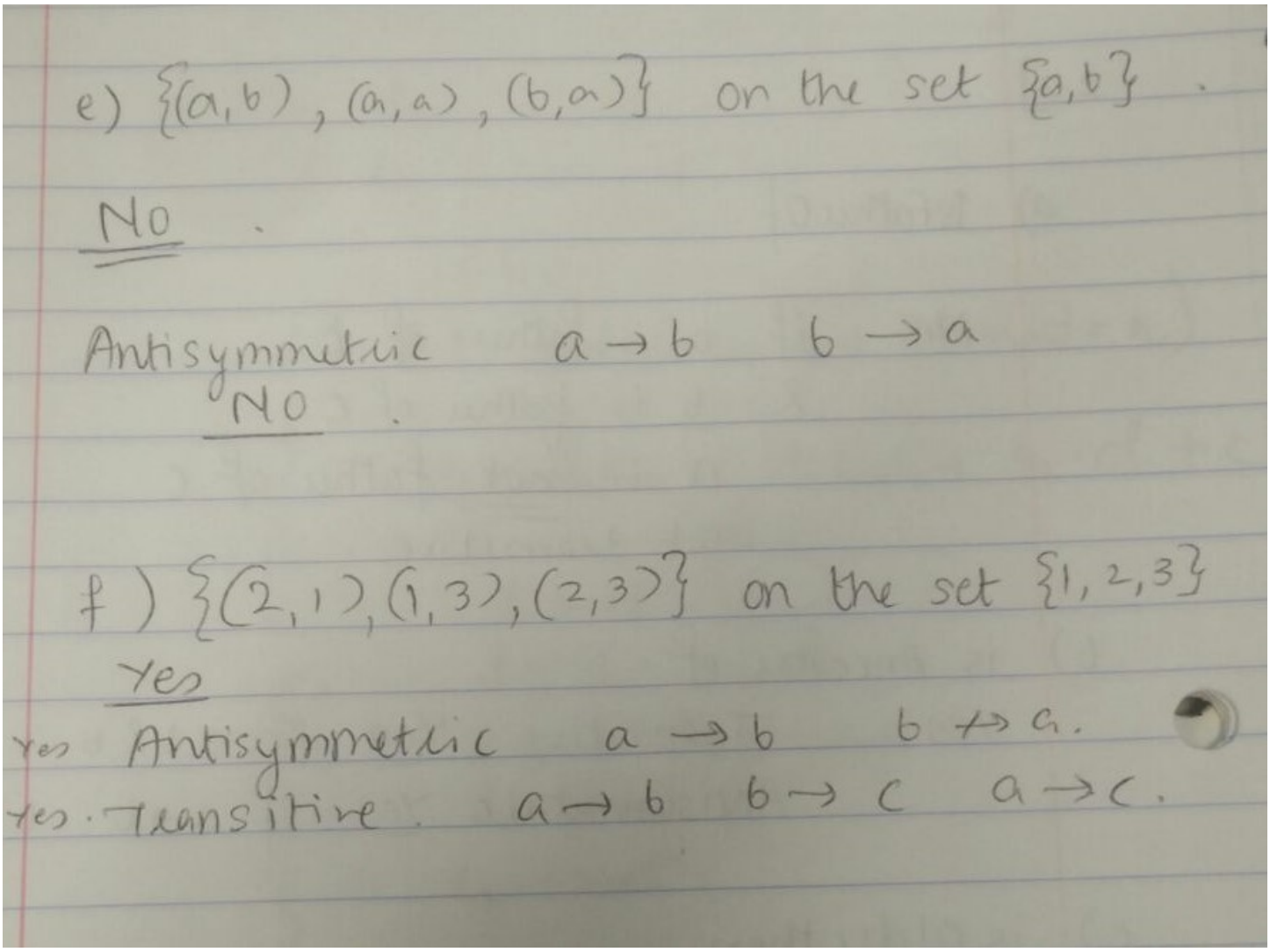
Yes. Transitive Yes if $a \to b$ & $b \to c$ $a \to c$.
Antisymmetric Yes if $a \to b$ $b \not\to c$.

c) is older than

Yes. Transitive Yes if $a \to b$ & $b \to c$ $a \to$
Antisymmetric Yes if $a \to b$ $b \not\to c$.

d) is Sister of.

No. if $a \to b$ & $b \to c$ then $a \not\to c$.
Transitive. No
if $a \to b$ & $b \to c$ or $b \not\to c$.
Antisymmetric Maybe depends.



2.5 Define an ADT for a set of integers (remember that a set has no concept of duplicate elements, and has no concept of order). Your ADT should consist of the functions that can be performed on a set to control its membership, check the size, check if a given element is in the set, and so on. Each function should be defined in terms of its input and output.

Implemented a linkedlist with following functions :-
Add a new node
Delete a node
Find a node
Traverse the linked list

Refer to the below .java files for code :-
LinkedList.java
intNode.java
LinkedListTest.java

2.15 Write a function to print all of the permutations for the elements of an array containing n distinct integer values.

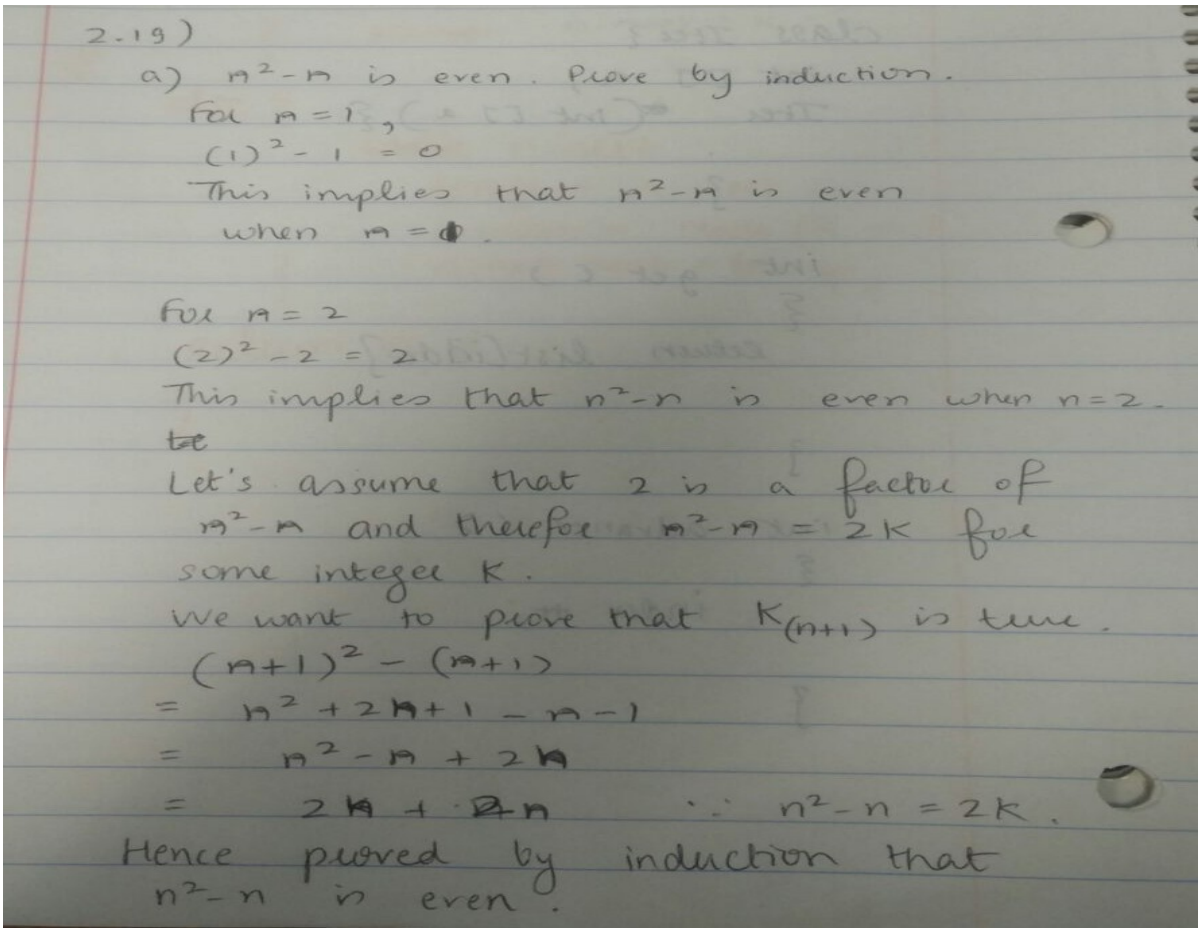
```
    public static void permute(int a [], int start, int end){
            if(start==end){
            System.out.println(Arrays.toString(a));
            count++;
        }
        else{
        for(int i=start;i<=end;i++){
            swap(a, start, i);
            permute(a, start+1, end);
            swap(a, start, i);
        }
        }

    }

    public static void swap(int a [], int i, int j){
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

2.19 (a) Use induction to show that n^2 - n is always even.
(b) Give a direct proof in one or two sentences that n^2 - n is always even.
(c) Show that n^3 - n is always divisible by three.
(d) Is n^5 - n always divisible by 5? Explain your answer.

a.)



b.)c.)

6) direct proof for n^2-n is even.

n^2-n

$n(n-1)$

n and $n-1$ are consecutive numbers.

Therefore, either of them must be even.

Hence, n^2-n is even.

c) n^3-n is divisible by 3 Prove.

For $n=1$: $(1)^3-1=0$ $\therefore n^3-n$ is divisible by 3 when $n=1$.

Let's assume n^3-n is true for all values of n. & $n^3-n=3k$ for some integer k.

$\therefore$ for $(n+1)$: $(n+1)^3-(n+1)$

$n^3+3n^2+3n+1-n-1$

$(n^3-n)+3n^2+3n$

$3k+3n^2+3n$ $\because n^3-n=3k$.

even even even

$\therefore$ Since all parts are multiple of 3.

Hence proved, n^3-n is divisble by 3.

d.)

d) $n^5 - n$ is always divisible by 5.

For $n = 1$: $(1)^5 - (1) = 0$ $\therefore n^5 - n$ is divisibly by 5 when $n = 1$.

Assume $n^5 - n$ is divisibly by 5 & $n^5 - n = 5k$ for some integer K.
Proving for $n+1 \Rightarrow$

~~$((n+1)^5 - (n+1))$~~
~~$= (n^5 + 5n^4 + 10n^3 + 10n^2 + 5n + 1) - (n+1)$~~
~~$= n^5 + 5n^4 + 10n^3 + 10n^2$~~

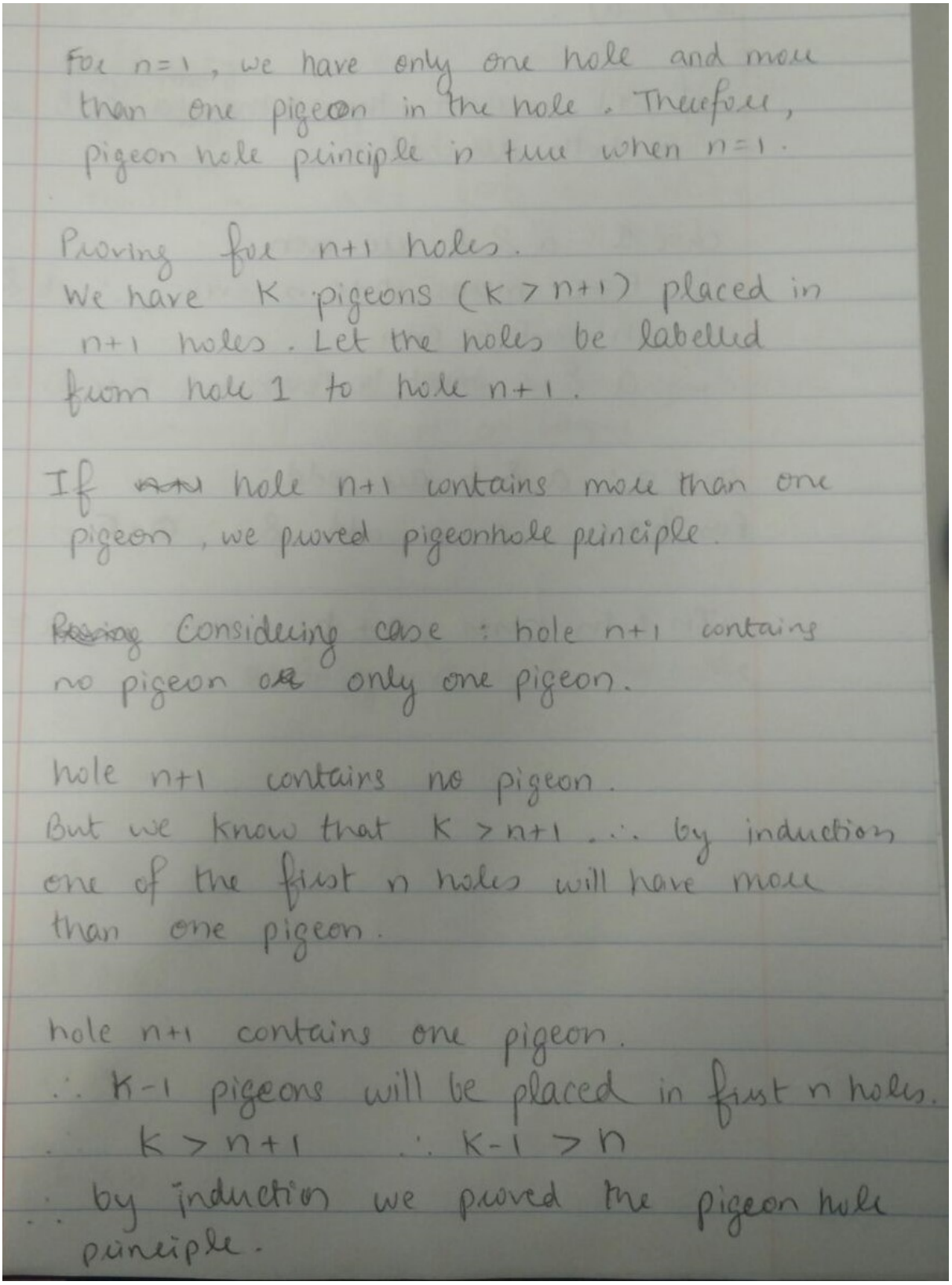
$(n+1)^5 - (n+1)$
$(n^5 + 5n^4 + 10n^3 + 10n^2 + 5n + 1) - (n+1)$
$n^5 + 5n^4 + 10n^3 + 10n^2 + 5n - n$
$(n^5 - n) + 5n^4 + 10n^3 + 10n^2 + 5n$.
$5K + 5n^4 + 10n^3 + 10n^2 + 5n$.

Since all parts are multiple of 5,
$n^5 - n$ is divisible by 5 ~~when n~~ for $n+1$.

Hence proved by induction.

2.30 The following theorem is called the Pigeonhole Principle.
Theorem 2.10 When n + 1 pigeons roost in n holes, there must be some hole containing at least two pigeons.

(b) Prove the Pigeonhole Principle using mathematical induction.



4.7 Write a function to merge two linked lists. The input lists have their elements in sorted order, from lowest to highest. The output list should also be sorted from lowest to highest. Your algorithm should run in linear time on the length of the output list.

```
public static LNode merge(LNode n1, LNode n2)

{ //Check if the current node in list 1 is empty

  if (n1 == null) return n2;

//Check if the current node in list 2 is empty

  if (n2 == null) return n1;

  LNode node;

//Check if current node data in list 1 is less than current node data in list 2

  if (n1.data < n2.data)

  {  node = n1;

   //Do a recursive call to compute next smallest node

    node.next = merge(n1.next, n2) }

  else

  { node = n2;

    node.next = merge(n1, n2.next); }

  return node; }
```

4.15 A palindrome is a string that reads the same forwards as backwards. Using only a fixed number of stacks and queues, the stack and queue ADT functions, and a fixed number of **int** and **char** variables, write an algorithm to determine if a string is a palindrome. Assume that the string is read from standard input one character at a time. The algorithm should output **true** or **false** as appropriate.

```
public static boolean isPal(String a) {
    Stack l = new Stack();
    Queue q = new LinkedList();
    String s;
    for (int i = 0; i < a.length(); i++) {
        s = "" + a.charAt(i);
        l.push(s);
        q.add(s);
    }

    while (!q.isEmpty()) {
        if (!q.remove().equals(l.pop()))
            return false;
    }

    return true;

}
```

4.3 Use singly linked lists to implement integers of unlimited size. Each node of the list should store one digit of the integer. You should implement addition, subtraction, multiplication operations.

Addition

```
public LNode add(LNode l1, LNode l2) {
      int carry =0;

       LNode nHead = new LNode(0);
       LNode p1 = l1, p2 = l2, p3=nHead;

       //Check if either of the node is empty
       while(p1 != null || p2 != null){

          if(p1 != null){
             carry = carry + p1.data;
             p1 = p1.next;
          }

          if(p2 != null){
             carry = carry + p2.data;
             p2 = p2.next;
          }

          p3.next = new LNode(carry%10);
          p3 = p3.next;
          carry = carry/10;
       }

       if(carry==1)
          p3.next=new LNode(1);

       return nHead.next;
   }
```

Subtraction

1.) Calculate sizes of given two linked lists.
2.) If sizes not are same, then append zeros in smaller linked list.
3.) If size are same, then follow below steps:
    a.)Find the smaller valued linked list.
    b.)One by one subtract nodes of smaller sized linked list from larger size. Keep track of borrow while subtracting.