Problem 1: Arrays and Asymptotic Notation (40 points)

1) Given the following code snippet.

```
boolean[][] array = new boolean[n][n];
for ( int i = 0; i < array.length; i++ ) {
  if ( array[i][i] == false ) {
    array[i][i] = true;
  }
 }
```

   a) (10 points) What is the exact number of array accesses in terms of n? Array access are reads and writes.
   There are 2n reads for each (array[i][i] == false): 1 read in the first dimension and 1 in the second dimension, the operation happens n times.
   There are n reads plus n writes for each (array[i][i] = true): 1 read in the first dimension and 1 write in the second dimension.
   Total is 2n (read) + n (read) + n (write) = 4n

   We also accept as correct n reads + n writes = 2n // 5 points if student answer is n

   b) (5 points) What is the Big O? O(n)

2) (10 points) The recitation problem *Merge Two Sorted Arrays* merges the two sorted parameter arrays. Assuming that m (first array number of items) is 3 and n (second array number of items) is 5 give one example of the best-case scenario, where merge executes the least number of if-conditional to merge the two arrays.
For example, one example of the worst-case scenario is when the first array is [2,5,9] and the second array is [1,3,4,6,8]. The call would be *merge([2,5,9], 3, [1,3,4,6,8], 5)*

```
// num1 and num2 are integer arrays that are already sorted in increasing
order. m and n are the number of items in the arrays, respectively.
// num1 length is m + n

public void merge(int[] nums1, int m, int [] nums2, int n) {
       int i = m - 1;
       int j = n - 1;
       int k = m + n - 1;
       while ( i >= 0 && j >= 0 ) {
           int one = nums1[i];
           int two = nums2[j];
           if ( one >= two ) {
               nums1[k] = one;
               k--;
               i--;
           } else {
               nums1[k] = two;
               k--;
               j--;
```

```
        }
    }

    while ( j >= 0 ) {
        nums1[k--] = nums2[j--];
    }
}
```

The best case is any example where the first/smallest element of one array is larger than the last/largest element of the other array.
[2,4,6,0,0,0,0,0] [7,8,9,11,13] or
[11,12,13,0,0,0,0,0] [1,2,3,4,5]

// we only ask for one example, 5 points if student doesn't give an example but describes the scenario

3) **(10 points)** Give the tilde and the big O notation for each of the functions below.

function	tilde	big O
$\frac{1}{3}n^4 + 2n^2 + 4$	~1/3n^4	O(n^4)
$4n^2 + 356\,n + 7893$	~4n^2	O(n^2)
$3n + 9$	~3n	O(n)
$2n^3 + n^2 + 4n$	~2n^3	O(n^3)
$\frac{1}{6}n^5 + 2n^4$	~1/6n^5	O(n^5)

1 point for each correct

4) Consider the following snippet of code.

```
int count = 0;
for ( int i = n; i >= 1; i = i/2 ) {
   for ( int j = 1; j <= i; j++ ) {
      count++;
  }
}
```

(5 points) What is the big O of the running time as a function of n? Count the operation “count++;” towards the running time. O(n)

2. **Stacks and Queues (45 points)**

2a. **(Stack Application – 15 points)** Consider the following function foo() that uses the stack s to complete a computation.

```
public String foo (int n) {
        Stack<Integer> s = new Stack<Integer>();
        String result = "";
        for ( int i = n; i > 0; i = i/2 )
                s.push( i % 2 );
        while ( !is.isEmpty() )
                result += s.pop();
        return result;
}
```

a) Find the output for n = 2, 4, 8, and 50

2 = 10 4 = 100 8 = 1000 50 = 110010

2b. (**Stack Test Client - 10 points**) Consider the following test client program for a stack.

```
Stack<String> myStack = new Stack<String>();

while (!StdIn.isEmpty()) {
     String s = readString();
     if (s.equals("-"))
         StdOut.print(myStack.pop());
     else
         myStack.push(s);
  }
```

(i) What is the output of the program if the standard input is : 1 2 3 4 5 - - - - -

Ans: 5 4 3 2 1

(ii) Using the characters 1, 2, 3, 4, 5, and – as input, suggest an input to the test client that does not produces the output 3 2 1 4 5

Ans: 1 2 3 -- 5 4 - - (there can be others) 3 _ 2 _ 1 _ 4 _ 5 _ or 1 2 3 --- 4 – 5 -

2c. **(Queue Application – 5 points)** Now consider a modified foo function from previous part 2a that uses a queue (instead of a stack)

```
public String foo2 (int n) {
        Queue<Integer> q = new Queue<Integer>();
        String result = “”;
        for ( int i = n; i > 0; i = i/2 )
                q.enqueue(i % 2);
        while ( !q.isEmpty() )
                result += q.dequeue();
        return result;
}
```

What is the value of foo2(50)?
Ans: 010011

2d**. Queue and Stack – (5 points)** Consider the following code fragment that processes a non-empty queue q.

```
Stack<String> s = new Stack<String>();
while ( !q.isEmpty() )
        s.push(q.dequeue()):
while ( !s.isEmpty() )
        q.enqueue(s.pop());
```

If the queue q contains: (front) 12 35 30 67 89 11 (back), meaning that 11 was the last item to be enqueued on the queue. Write the content in the queue after code fragment is executed. Clearly mark the front and back of the new queue.

Ans: (front) 11 89 67 30 35 12 (back)

2e. **(Max Stack – 10 pts)**

Suppose that you are asked to modify the stack class to add a new operation **max()** that returns the current maximum of the stack comparable objects. Assume that pop and push operations are currently implemented using array a as follows where item is a String and n is the size of the stack.

Note: if x and y are objects of the same type, use x.compareTo(y) to compare the objects x and y.

```
public void push ( String item ) { a[n++] = item; }
public String pop () { return a[--n]; }
```

Implement the max operation in two ways: (Part I) by writing a new method using array a, and (Part II) by updating push and pop methods to track max as the stack is changed.

Part I (5 points) Using O(1) space and O(n) running time.

public String max() {….

}

Ans: **O(n) time complexity for max operation (5 pts for this solution)**

```
public String max() {
            max=a[0];
            for (int i=1; i<top; i++)
                  if (a[i].compareTo(max)>0) max = a[i];
            return max;
}
```

Part II (5 points) Using O(n) space and O(1) run time. You may update the push and pop methods as needed.

public void push(String item) {

}

public String pop() {

}

public String max() {….

}

Ans. (there can be other solutions here)

O(n) space complexity and O(1) for each max operation

One idea is to maintain an extra stack/array, where the top element is always the max.

```
int[] b = new int[a.size];
/* O(n) additional space. The (top-1) element of the array b is the max
element at all times */

=====================================================================
public void push(String item) {
         if (this.isEmpty()) { a[top] = item; b[top++] = item;}
         else  {
               a[top] = item;
               if (item.compareTo(b[top-1])>0)  b[top] = item;
               else b[top] = b[top++ -1];
         }
}

=====================================================================
public String pop() {
      return a[--top];
}

=====================================================================
public String max() { return b[top-1];}
```

Problem 3: Linked List (45 points)

a) **(25 points)** Implement the following method to find the minimum value in a linked list. Assume the linked list holds only positive values. You may NOT use or implement helper methods - all your code must be implemented inside the given method. You may NOT use recursion.

```
public class Node {

    public int data;
    public Node next;
}

// Given a front pointer to the first node in the list.
// Returns the minimum value among all integers in the list.
// Input list could be empty, returns -1 if the given linked list is
empty.

public static int findMin(Node front) {

    // COMPLETE THIS METHOD
}
```

```
Sol:
public static int findMin(Node front) {

     if(front == null){
         return -1; //5 pts
     }


      int min = front.data;

     Node ptr = front.next; //5 pts for declaring a new variable to
iterate over all nodes

      while(ptr != null) {      //5 pts for checking whether ptr is
null or not

          if(ptr.data < min)    //3 pts for data comparison

               min = ptr.data;

          ptr = ptr.next;  //5 pts for moving on to the next

      }

      return min;               //2 pts for return

}
```

Solution can be different from the above, but the rubric will be the same.

b) (5 pts) What is Big O notation for the running time of the findMin method in part a? Given the reason process.

Sol:
O(n). 2 pts
All nodes must be visited to find the minimum value. 3 pts

c) (5 pts) Consider the linked list:
front -> 2 -> 5 -> 6 -> 1 -> null

What is the resulting list after the following code snippet is run:

```
Node prev = null;
Node ptr  = front;
while ( ptr.next != null ){
   prev = ptr;
   ptr = ptr.next;
}
Node newNode = new Node();
newNode.item = 2;
newNode.next = null;
prev.next = newNode;
```

Sol: front -> 2 -> 5 -> 6 -> 2 -> null

d) (5 pts) Read the following code snippet (same as part c) :

```
Node prev = null;
Node ptr  = front;
while ( ptr.next != null ){
   prev = ptr;
   ptr  = ptr.next;
}
Node newNode = new Node();
newNode.item = 2;
newNode.next = null;
prev.next = newNode;
```

Shall we have errors when running the code for some linked list? If so, what kind of linked list will trigger errors and what is the error?

Sol: yes. 2pts. If yes is given without empty list OR one node list OR NullPointerException, then 0 pts.
Empty list or one node list. 5 pts
NullPointerException. 3 pts