There are 7 problems on the exam. The first and last are mandatory, and you may eliminate any one of problems 2 – 6 by drawing an X through them. Problem 1 is worth 10 points, and all other problems are worth 18 points. Please write in pen if possible. Circle answers when they occur in the midst of a bunch of work. If you need more room, use the back of the sheet and tell me this on the front sheet.

**Problem One.** (True/False – MANDATORY) Write True or False to the left of each statement.  [False statements in red]

1. If a class A implements an interface B, then all the public methods in A must also be specified in B. [A must satisfy AT LEAST the methods in B, but could do extra ones.]

2. In Java you may redefine a variable inside its scope, as long as the new declaration is in a for loop. [nope]

3. If Stack is a generic data type, then you can declare a new stack of ints using:
   ```java
   Stack<Integer> S = new Stack<Integer>();
   ```
   [Not counted]

4. An overloaded operator in Java is one that can be used for more than one type of operand.

5. Mergesort is difficult to implement “in place” (i.e., without using an extra array).

6. When performing a widening conversion you must cast the value to the new type or else you will get an error. [widening conversions cause no problems.]

7. If a class BigInt contains a method getLength() and you see an expression
   ```java
   BigInt.getLength()
   ```
   then you know that getLength() must be declared as static inside the class BigInt.

8. The code in the box will print out a 6.

9. In the Boolean expression ((A || B) && C), if the expression A evaluates to false, then regardless of the value of B, the expression C will not be evaluated. [if B is true, then C must be evaluated.]

10. I have put my name on the top of this exam. 😊

    ```java
    int x = 5;
    ++x;
    System.out.println(x++);
    ```
**Problem Five.** For each of the following algorithms or problems, state the $\Theta(...N...)$ estimate for the worst case; if a problem is listed, assume you use the best known algorithm that we studied this semester.

(A) Finding the average of the numbers in a linked list.  $\Theta( N )$

(B) Finding the largest element in an ordered array.  $\Theta( 1 )$

(C) Finding a number in an ordered array.  $\Theta( \log(N) )$

(D) Insertion Sort on an array that is already sorted in the correct order.  $\Theta( N )$

(E) Inserting a new number into an unordered linked list, assuming you don’t care about duplicates in the list.  $\Theta( 1 )$

(F) Inserting a new number into an unordered linked list, assuming you may NOT insert duplicates into the list.  $\Theta( N )$

(G) Inserting a new number into an ordered array.  $\Theta( N )$

(H) Finding a number in an ordered linked list.  $\Theta( N )$

(I) Finding the largest three elements in an unordered array and printing them out in order.  $\Theta( N )$

(J) Mergesort on an array that is already sorted in the correct order.  $\Theta( N \log(N) )$
Problem Four. Consider this linked list:

Apply the following code to this linked list, and (A) draw a picture of the data structure that results, including what the variables head1, p, q, and head2 point to and (B) give a one-sentence description of what this method does.

```java
public Node whatDoesItDo(Node head) {
    if( head == null || head.next == null )
        return null;
    Node p = head;
    Node q = p.next;
    Node head2 = q;
    while( q.next != null ) {
        p.next = q.next;
        q.next = p.next.next;
        p = p.next;
        q = q.next;
        if(q == null)
            return head2;
    }
    p.next = null;
    return head2;
}

Node head2 = whatDoesItDo( head1 );
```

(A) Draw the data structure

headA --> 3 --> 4 --> 5 --> .

headB --> 1 --> 2 --> .

(B) Describe in one sentence what it does.

The method “unzips” the list into two lists, putting every other node in another list (starting from position 1, odd numbered nodes in headA and even-numbered nodes in headB.
**Problem Six.** Consider the program shown below.

This problem is similar to one from the Part A homeworks. Suppose a number of insert() and getMax() operations are performed on a MaxQueue M. The put() operations put the integers 0 through 9 onto M in order, and the getMax() operations print the return value. That is, the sequence looks like this:

```java
insert(0);
repeat 0 or more times: System.out.print(getMax() + " ");
insert(1);
repeat 0 or more times: System.out.print(getMax() + " ");
insert(2);
repeat 0 or more times: System.out.print(getMax() + " ");
....
insert(9);
repeat 0 or more times: System.out.print(getMax() + " ");
```

For each of the following sequences, state whether or not they could occur as a result of such a sequence of MaxQueue operations.

a. 3 2 1 0 4 5 8 6 7 9  NO
b. 9 8 7 6 5 4 2 3 1 0  NO
c. 3 2 1 0 4 5 9 8 7 6  YES
d. 0 1 3 2 4 6 5 7 9 8  YES
**Problem Two.** This is about **Mergesort** and has two parts.

(A) Perform Mergesort on the following array, according to the algorithm presented in lecture: (i) show the 2D matrix of values that would be produced as you perform the algorithm (as shown in class), (ii) give the exact number of comparisons between elements in the array that were necessary, and (iii) give the exact number of moves of elements that were necessary.

<table>
<thead>
<tr>
<th># Comps</th>
<th># Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 8 7 4 6 2 1 3</td>
<td>4 8</td>
</tr>
<tr>
<td>5 8 4 7 2 6 1 3</td>
<td>6 8</td>
</tr>
<tr>
<td>4 5 7 8 1 2 3 6</td>
<td>6 8</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>6 8</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>16 24</strong></td>
</tr>
</tbody>
</table>

(B) Give an example of a sequence of 8 numbers (two sorted lists of 4 each, to be sorted as if in the last stage of the mergesort algorithm), in which merging the two lists would take the **minimum** number of comparisons.

1 2 3 7 4 5 6 8  (one list has largest, other has second-largest)

(C) In general, if you have a sequence of N numbers (N an even number) consisting of two sorted sublists of size N/2 each, what is the **minimum** number of comparisons necessary to merge the two sublists?

1 2 3 4 5 6 7 8  (every number in one list is larger than all in the other list)

(D) Give an example of a sequence of 8 numbers (two sorted lists of 4 each, to be sorted as if in the last stage of the mergesort algorithm), in which merging the two lists would take the **maximum** number of comparisons.

N - 1  (generalization of (A) -- last comparison is done with 2 numbers left)

(E) In general, if you have a sequence of N numbers (N an even number) consisting of two sorted sublists of size N/2 each, what is the **maximum** number of comparisons necessary to merge the two sublists?

N / 2  (generalization of (B) -- only numbers in smaller list must be compared)
Problem Two. How many times does the following code print out “X!” overall? Express your answer as a function of N, in terms of $\Theta(...N...)$.

You must write the $\Theta(...N...)$ estimate for each individual for loop first, then figure out the estimate for the three separate nested loops and then give the overall result for the whole page. Show all work! Partial credit will be given for the parts of your answer!

```java
for( int j = 1; j < 1000*N; j = j + 2 ) {
    for(int k = 10*N; k >= 1; k = k / 2) {
        System.out.println("Hi there!");
    }  // Subtotal: $\Theta(N*\log(N))$
}

for( int j = -N; j < 1000; ++j ) {
    for( int m = 2; m < log(N); ++m ) {
        System.out.println("Hi there!");
    }  // Subtotal: $\Theta(N*(\log(N)+1)) = \Theta(N*\log(N)+N) = \Theta(N*\log(N))$
}

for( int i = 7; i < N-1; i = i + 2 ) {
    for( int k = N; k >= 1; k = k / 2 ) {
        for( int j = 1; j < 1000; j = j * 2 ) {
            System.out.println("Hi there!");
        }  // Subtotal: $\Theta(N*\log(N)*1) = \Theta(N*\log(N))$
    }
}
```

Total: $\Theta( N*\log(N) + N*\log(N) + N*\log(N) ) = \Theta( N*\log(N) )$

Your answer: The whole page prints out “Hi there” $\Theta( N*\log(N) )$ times.
Problem Seven. (MANDATORY) In this problem you will add a new method to a stack, which will in effect allow it to be used as a priority queue. You must fill in the missing code in the template below:

```java
public class MaxStack {
    private int[] A = new int[100];
    private int next = 0;
    public int size() {
        return next;
    }
    public void push(int n) {
        A[next++] = n;
    }
    public int pop() {
        return A[--next];
    }

    // Remove and return the maximum number in the
    // stack, leaving all other numbers in the same
    // order. I.e., if stack is 1 2 5 1 4 (top to R)
    // then getMax() returns 5 and leaves 1 2 1 4.
    // If stack is empty return Integer.MIN_VALUE.
    public int getMax() {
        int maxIndex = 0;
        for (int i = 1; i < next; ++i) {
            if (A[i] > A[maxIndex]) {
                maxIndex = i;
            }
        }
        int result = A[maxIndex];
        for (int i = maxIndex+1; i < next; ++i) {
            A[i-1] = A[i];
        }
        --next;
        return result;
    }
}
```

Here is an alternate way of doing it:

```java
public int getMax() {
    MaxStack S = new MaxStack();
    int max = Integer.MIN_VALUE;
    while (size() != 0) {
        int n = this.pop();
        if (n > max) {
            max = n;
        }
        S.push(n);
    }
    while (S.size() != 0) {
        int n = S.pop();
        if (n < max) {
            // this will remove all the instances of the max
            this.push(n);
        }
    }
    return max;
}
```