1. (2 points each, 20 points total) Java Mechanics and algorithmic analysis. For parts A – G what is the output of the code fragment? Consider each piece of code in isolation. Put your answer directly after each fragment. If an error would occur answer "syntax error" or "runtime error" depending on what type of error it would be.

A.
```java
int x = 5;
int y = 11;
double a = x * y;
double b = 1.5;
b = a / x + b + y + y / x;
System.out.println(b);
```

B.
```java
Color c1 = new Color(100, 25, 200, 100);
// previous line of code creates a new Color object.
Color c2;
c2 = c1;
System.out.println(c2 == c1);
c2 = new Color(100, 25, 200, 100);
System.out.println(c1 == c2);
```
C. int t = 2;
   int s = 37;
   int j = 33;
   for(int i = 1; i < s; i += 5)
       t += j % i;
   System.out.println(t);

D. int x = 27;
   int y = 14;
   int[] intList = new int[ y % x - 3];
   for(int i = 0; i < intList.length; i++)
       { if(i % 2 == 0)
           intList[i] = i * i;
       else
           intList[i] = i + i + 1;
       }
   int t = 0;
   for(int i = 0; i < intList.length; i++)
       System.out.print(intList[i] + " ");

E. For part E methods one and two are in the same class. Assume method two has just been
called with the following parameters:

   someObject.two(7, 12);

public int one(int a, int b)
{ int temp = a;
   a = b;
   b = temp;
   System.out.println(a + " " + b);
   return a + b;
}

public void two(int a, int b)
{ int x = 5;
   a = x * 2;
   b += 3;
   System.out.println( one(b,a) + " " + a + " " + b); }
F. For part F consider the following class:

```java
class Bad {
    public int one;
    public int two;
}
```

The following two methods both appear in another class. Assume the following method call has just been made:

```java
someObject.go();
```

```java
class Bad {
    public void helper(Bad b) {
        int temp = b.one;
        b.one = b.two;
        b.two = temp;
        System.out.println(b.one + " " + b.two);
    }

    public void go() {
        Bad bb = new Bad();
        bb.one = 20;
        bb.two = 5;
        System.out.println(bb.one + " " + bb.two);
        helper(bb);
        System.out.println(bb.one + " " + bb.two);
    }
}
```
G. Part G uses the same class, Bad, from part F. The following two methods both appear in another class. Assume the following method call has just been made:

```java
someObject.anotherGo();
```

```java
public void anotherHelper(Bad b2)
{
    b2.two = 12;
    System.out.println(b2.one + " " + b2.two);
    b2 = new Bad();
    b2.one = 19;
    b2.two = 13;
    System.out.println(b2.one + " " + b2.two);
}
```

```java
public void anotherGo()
{
    Bad b2 = new Bad();
    b2.one = 1;
    b2.two = 8;
    System.out.println(b2.one + " " + b2.two);
    anotherHelper(b2);
    System.out.println(b2.one + " " + b2.two);
}
```

For parts H - J of question 1 look at the code skeleton provided. State the Big O algorithmic execution time as a function of n. You can assume that the portions of the loops not shown require only constant execution time.

**H.**
```java
for(int i = 0; i < n; i++)
{
    ...
    for(int j = n; j >= 0; j--)
    {
        ...
    }
}
```

**I.**
```java
for(int i = 0; i < n; i++)
{
    for(int j = 0; j < i; j++)
    {
        ...
    }
}
```

**J.**
```java
for(int i = n; i > 0; i = i / 2)
{
    ...
}
```
2. (20 points) Implement a class to meet the following requirements. A color can be described by the amount of red, green, and blue in the color. Assume the values of red, green, and blue can vary from 0 to 255 inclusive and must be integer values. Create a class that tracks this information about a color. Include a default constructor and a constructor with parameters for the amount of red, green, and blue. Also include an equals method, a toString method, and a method that returns true if the color is a pure color, meaning one of the three component values is 255 and the others are both 0. Do not use any other classes besides the String class when creating this class. Write out the class as it would have to appear in the Color.java file for the class with the necessary instance variables, 2 constructors, and 3 methods specified above.
3. (20 points) This question deals with the Hand class from assignment 3. Consider the following implementation of the Hand class.

```java
public class Hand
{
    private Card[] myCards;

    private int iMyNumCards;
    // methods and constructors not shown
}
```

You may use the following methods from the Card class for this question, but you may not use any other Java classes, other methods from the Hand class, or other methods from the Card class except those listed.

```java
public class Card
{
    // default constructor
    public Card()
    {
    }

    // copy constructor
    public Card(Card c)
    {
    }

    // returns true if the suits and values of both cards are equal
    public boolean equals(Card rhs)
    {
    }

    // accessor for suit
    public int getSuit()
    {
    }

    // accessor for value
    public int getValue()
    {
    }

    // other methods and instance variables not shown
}
```

Write the following method for the Hand class:

```
/*
   a mutator that removes all cards from the hand that match the suit and / or the value of the parameter c.
   post: getNumCards() = old getNumCards - number of cards that match the suit, the value, or both the suit and value of the Card c.
*/

public void removeMatchingCards(Card c)

// complete the method on the next page
public void removeMatchingCards(Card c)
/* note to implementer. Do not leave "holes" in myCards. For example, if there are four cards left in the Hand after removing all the matching cards those four cards will occupy the first 4 elements of myCards. There may be extra capacity beyond these first 4 elements, but that is okay. Also myCards is not kept in sorted order.

Example of remove. If the hand starts with the following cards (s = spades, d = diamonds, c = clubs, h = hearts)

2s 5h 2d As 7h As Ad 7d Kc and the card sent in is Ace of Spades, (As), the hand would now contain the following cards:

5h 2d 7h 7d Kc (all aces and spades removed)
*/
4. (25 points) This question involves the analysis of DNA sequences. **Please note that although this question deals with DNA it is very different from the assignment you did on DNA.** Implement a method that looks for matches in a single strand of DNA. That is find whether any pattern of length m bases is ever repeated in the array of values. m will be a small number (3-10)

Example (assume first base is in element 0 of an array):

gtatctgctatcgtctatg

If m were 3 then the following patterns are repeated in the above base

```
tat starting at indexes 1, 8, and 15
atc starting at indexes 2 and 9
tct starting at indexes 3 and 13
cta starting at indexes 7 and 14
```

The bases are stored in an array of characters and passed as a parameter to the method. The other parameter is an integer that specifies exactly how long of a match to look for. The method shall return a String containing the indexes of where all matches start. All matches are placed in the String that is returned, separated by a single space. The matches do not need to be in any particular order even if the indexes are for different matches. For the example the method could return a String such as:

```
"1 8 15 2 9 3 13 7 14"
```

*You may not use any other Java classes or methods in writing this method other than the instance constant length for arrays, the String concatenation operator and a String variable to hold the result. In particular do not change the array of chars to a String and use the String class to process it. (But of course you can create new arrays of primitives if needed.)*

```java
// dna contains the bases, m is the size of the matches to look for
public String findMatches(char[] dna, int m)

/* complete the method on the next page to initialize your result use

String result = "";

when you find another index where a match occurs you will need to add the index to result using something like:

result += index + " ";
*/
```
// dna contains the bases, m is the size of the
// matches to look for

public String findMatches(char[] dna, int m)
5. (15 points) The Sieve of Eratosthenes is a method for finding prime numbers. A prime number is any integer that is evenly divisible only by itself and 1. The Sieve algorithm works as follows.
Create an array of booleans and initialize them all to true. Array elements with prime subscripts or indexes will remain true, all other array elements will be set to false. Starting with the element with index 2 loop through the remainder of the array and set every element whose index is a multiple of 2 to false (4, 6, 8, 10…). Go to the next element in the array. If it is true then that number is prime and you must loop through all multiples of the index of that element and set them to false. So, 3 is the next index after 2 and its element holds true so it must be a prime number. Loop through to set all the indexes that are multiples of 3 to false (6, 9, 12, 15…). Note, some of these are already false and are simply set to false again. Do not try to remove this inefficiency. If an element is false the index for that element is not a prime number and there is nothing to do except go on to the next element.

Implement a method using the Sieve of Eratosthenes algorithm as described above to find all the prime numbers between 2 and N where N is specified as a parameter to the method. Print out all the primes you find using System.out.print and System.out.println. The only Java methods you may use are the print, println, and Math.sqrt(int). This is the only problem on the test where the efficiency of your code will be graded. Essentially deviating from the algorithm as described above will also cause you to lose points.

/* finds and prints prime numbers using the Sieve of Eratosthenes method. 
   pre: limit > 2
   post: prints out all prime numbers p such that 2 <= p <= limit */
public void Sieve(int limit)