CMPUT 115 Section B3
Term test 1

February 6, 2001

Instructions:

- This is a closed book, no notes exam.
- Try to put all of your answers in the space provided.
- The backs of pages can be used for rough work.
- Be sure to write your student id number on each internal page.
- Please do not open the exam until you are instructed to do so.
- Good luck.

First Name:

Last Name:

1. [2 Marks] What output will the following program produce?

```
class G {
  public void f( Object obj ) {
   System.out.println( "G.f()" );
  }
}

class H extends G {
  public void f( String str ) {
   System.out.println( "H.f()" );
  }

  public static void main(String args[]) {
    H h = new H();
    h.f( "B" );
    h.f( new Integer(4) );
  }
}
```

2. [3 Marks] Implement the `insertElementAt` method of the Vector class. There is space on the next page to put your answer.

3. [4 Marks] Implement a method for the Vector class called:

   `insertVectorAt( Vector v, int index )`

   This method should insert each element from `v` into the receiver vector starting at position `index`. For example if a vector `v1` contains the objects:

   `[ a, b, c, d, e, f ]`

   and a vector `v2` contains the objects:

   `[ x, y, z ]`

   then after the call `v1.insertVectorAt(v2,2)` the vector `v1` should contain the objects:

   `[ a, b, x, y, z, c, d, e, f ]`

   Make your implementation as *efficient* as possible. There is space on the next page to put your answer.

```
public class Vector {
  protected Object elementData[]; // the data
  protected int elementCount;     // # of elements in vector


  public void insertElementAt(Object obj, int index) {
  // pre: 0 <= index <= size()
  // post: inserts new value in vector with desired index,
  //       moving elements from index to size()-1 to right
```

```
  public void insertVectorAt( Vector v, int index ) {
  // pre:  v is not null and index is a valid index into this vector.
  // post: each element in v is inserted into this vector starting at
  //       position index.
```

4. [1 Marks] What is the difference between the size and the capacity of a Vector?

5. [2 Marks] Which vector growth strategy is the best? Why?

6. [3 Marks] What is the **worst-case** time complexity of the Vector class's `removeElement` method? Express your answer as a function of **n** where **n** is the size of the vector. Be as *exact* as possible and *explain* your answer.

7. [3 Marks] Circle true or false for each of the following:

a) $n^2 + n = O(2^n)$ T F

b) $n\log_2 n = O(n)$ T F

c) $n^2 + n^4 = O(n^5)$ T F

8. [2 Marks] Consider the following method.

```
public static int f( int n ) {
  if ( n <= 1 ) return n;
  else if (n%3 == 0) return f(n/3) + f(n-1);
  else return 2 * f(n-1);
}
```

What is returned by the call `f(7)` to the above method? Recall that `%` is the remainder operator (for example: `9%4` is `1`).

9. [2 Marks] The quick sort implementation we studied in class has a worst case time-complexity of $O(n^2)$, why is this? How could the implementation be improved to avoid this?

10. [3 Marks] The source code for `mergeSort`:

```
public class Sorter {
  protected Comparable[] data;

  protected void merge( Comparable[] d1, Comparable[] d2,
                        int low, int middle, int high ) {
  // pre: d1[middle..high] and d2[low..middle-1] are ascending
  // post: d1[low..high] contains all values in ascending order
   int ri = low;
   int ti = low;
   int di = middle;
   while ( ti < middle && di <= high ) {
         if (d1[di].compareTo(d2[ti]) < 0) d1[ri++] = d1[di++];
         else d1[ri++] = d2[ti++];
   }

   while ( ti < middle ) d1[ri++] = d2[ti++];
  }

  protected void mergeSort( Comparable[] d1, Comparable[] d2,
                            int low, int high ) {
   int n = high-low+1;
   int middle = low + n/2;

   if (n<2) return;
   for ( int i = low; i < middle; i++ ) d2[i] = d1[i];

   mergeSort( d2, d1, low, middle-1 );
   mergeSort( d1, d2, middle, high );
   merge(d1,d2,low,middle,high);
  }

  public void mergeSort() {
  // post: objects in data[] in ascending order
   mergeSort(data,new Comparable[data.length],0,data.length-1);
  }
}
```

How many calls to `compareTo` are made when `mergeSort` is called on an array constructed as follows (place your answer to the right of the code):

```
Comparable[] data = new Comparable[8];
data[0] = new Integer(12);
data[1] = new Integer(6);
data[2] = new Integer(1);
data[3] = new Integer(-5);
data[4] = new Integer(34);
data[5] = new Integer(-5);
data[6] = new Integer(8);
data[7] = new Integer(3);
```