Problem 1:

```
add r0, #256, r9          ; global int g = 256

 p:
        add, r26, r27, r16
        sll r16, #2, r1
        ret r25 , 0
        sub r1, #1, r1

q:

        add r9, r0, r10
        callr r25, p
        sub r0, r26, r11
        ret r25, 0
        add r0, r0, r0



f:
        add r0, r26, r16
        sub r16, r0, r0{C}
        jle Elf1                ;if(n>0){
        add r0, r0, r0

        sub r16, #1, r17        ;       tmp = n-1
        callr r25, f            ;       tmp2 = f(tmp)
        add r0, r17, r10        ; tmp as 1st param
        add r0, r16, r10        ; n as 1st param
        callr r25, multiply     ;       assuming existince of multiply          function.
        add r0, r1, r11         ; tmp2 as 2nd param.
        ret r25, 0                      ;       return n*(f(tmp))
        add r0, r0, r0

Elf1:
        ret r25, 0
        add r0, #1, r1
```

Problem 2:

		Number of procedure calls	**Maximum window depth**	**Number of window overflows**	**Number of window underflows**
Register	**6**	172233	511	83911	83911
window	**8**	172233	511	82950	82950
numbers	**16**	172233	511	79229	79229

Problem 3:
Running ackermann(3,6) 10 times in Java yielded an average time of 5,327,627 nanoseconds. The difficulties of timing this program in such a high level language are that the compiler will make all sorts of optimizations unseen to the programmer, which might not reflect the true run time of the algorithm if programmed in a lower level language. You are also relying on other high level methods and classes to accurately behave in recording the time spent running the program which does not guarantee an accurate result.

Ackermann Program:
```
public class Ackerman {

        static int count = 0;
        final static int NUM_WINDOWS = 16;
        static int numOver = 0;
        static int numUnder = 0;
        static int cwp = 0;
        static int maxDepth = 0;
        public static void main(String[] args) {

                final long startTime = System.nanoTime();
                ackermann(3,6,0);
                final long endTime = System.nanoTime();
                System.out.println("total calls: " + count);
                System.out.println("Number of overflows: " + numOver);
                System.out.println("Number of underflows: " + numUnder + "\ncurrent cwp: " +
cwp + "\nMaxDepth: " + maxDepth + "\nTime took: " + (endTime-startTime));
        }
        //avg time = 5,327,627 nanoseconds.

        public static void checkOverFlow(){
                count++;
                cwp++;
```

```
			if(cwp > NUM_WINDOWS){
				numOver++;
				cwp--;
			}
		}

		public static void checkMaxDepth (int depth){
			if(depth > maxDepth)
				maxDepth = depth;
		}
		public static void checkUnderFlow(){
			cwp--;
			if(cwp < 0){
				cwp++;
				numUnder++;
			}
		}

		public static int ackermann(int x, int y, int depth) {
			depth++;
			checkMaxDepth(depth);
			checkOverFlow();
			if (x == 0) {
				checkUnderFlow();
				return y+1;
			}
			else if (y == 0) {
				int result = ackermann(x-1, 1, depth);
				checkUnderFlow();
				return result;
			}
			else {
				int param2 = ackermann(x, y-1, depth);
				int result = ackermann(x-1, param2, depth);
				checkUnderFlow();
				return result;
			}
		}

	}
```