Actividades Extra Complejidad Ciclomática

1. *Búsqueda binaria*

Dado un array de números enteros positivos, ordenado de menor a mayor, podemos ejecutar una búsqueda binaria de un número concreto, de la siguiente manera:

```
10      private int array[];
11
12      public claseBusquedaBinaria(){
13          array = new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
14      }
15
16      public int busquedaBinaria(int numero) {
17
18          int umbral_inferior = 0;
19          int umbral_superior = array.length - 1;
20          int respuesta = -1;
21          int index;
22
23          while (umbral_inferior <= umbral_superior) {
24              index = (umbral_inferior + umbral_superior) / 2;
25
26              if (array[index] == numero) {
27                  respuesta = index;
28                  umbral_inferior = umbral_superior + 1;
29              } // fin if
30              else if (array[index] < numero) {
31                  umbral_inferior = index + 1;
32              } // fin else if
33              else {
34                  umbral_superior = index - 1;
35              } // fin else
36          } // fin while
37
38          return respuesta;
39
40      } //fin metodo
```

a) Diseñar pruebas de caja negra mediante una de las técnicas estudiadas, e implementarlas mediante JUnit.

b) Diseñar pruebas de caja blanca mediante la técnica de cobertura de caminos, incluyendo la representación del grafo, el cálculo de la complejidad ciclomática y los caminos básicos.

2. Suma de números primos

El siguiente código calcula la suma de los números primos hasta un número máximo (siendo precondición del método que el número máximo sea mayor o igual que 2).

Por ejemplo, la suma de números primos hasta 10 sería 2+3+5+7=17, y la suma de números primos hasta 12 sería 2+3+5+7+11=28.

```
8   public static int sumaPrimos(int numeroMaximo) {
9
10      int acumulador = 0;
11      int numero = 2;
12      int contador;
13      boolean primo;
14
15      do {
16          System.out.println(numero);
17          contador = 2;
18          primo = true;
19          while ((primo) && (contador != numero)) {
20              if (numero % contador == 0) {
21                  primo = false;
22              } // fin if
23              contador++;
24          } // fin while
25
26          if (primo) {
27              acumulador = acumulador + numero;
28          } // fin if
29
30          numero++;
31      } while (numero <= numeroMaximo); // fin do..while
32
33      return acumulador;
34
35  } //fin metodo
```

a) Diseñar pruebas de caja negra mediante una de las técnicas estudiadas, e implementarlas mediante JUnit.

b) Diseñar pruebas de caja blanca mediante la técnica de cobertura de caminos, incluyendo la representación del grafo, el cálculo de la complejidad ciclomática y los caminos básicos.

3. Ordenación burbuja

Dado un array de números enteros positivos, el método aplica el algoritmo de ordenación de burbuja, devolviendo *true* si el array ya estaba ordenador, y *false* si ha sido necesario ordenarlo:

```
14      private boolean burbuja(int[] arrayNumeros) {
15
16          boolean ordenado = true;
17          int elementoActual, elementoSiguiente;
18
19          for (int x = 0; x < arrayNumeros.length; x++) {
20              for (int y = 0; y < arrayNumeros.length - 1; y++) {
21                  elementoActual = arrayNumeros[y];
22                  elementoSiguiente = arrayNumeros[y + 1];
23
24                  if (elementoActual > elementoSiguiente) {
25                      // Intercambiar
26                      arrayNumeros[y] = elementoSiguiente;
27                      arrayNumeros[y + 1] = elementoActual;
28                      ordenado = false;
29                  }
30              }
31          }
32
33          return ordenado;
34      }
```

a) Diseñar pruebas de caja negra mediante una de las técnicas estudiadas, e implementarlas mediante JUnit.

b) Diseñar pruebas de caja blanca mediante la técnica de cobertura de caminos, incluyendo la representación del grafo, el cálculo de la complejidad ciclomática y los caminos básicos.