Práctica 1
Complejidad de algoritmos

Jesús Javier Quirante Pérez

Ingeniería Informática 2ºD

Índice

1. **Análisis de la complejidad experimental** **3**
2. **Código de la clase Analizador** **4**
3. **Implementación del código** **8**

1. Análisis de la complejidad experimental

Desde el punto de vista teórico, el análisis de un algoritmo nos permite determinar cuál es su complejidad teórica. En la práctica, nos encontramos también con muchos casos en los que determinar esta complejidad de un programa complejo de forma teórica es difícil, o incluso imposible si no disponemos del código fuente.

Aun así, la experimentación con distintos valores de entrada nos ofrece información sobre la posible complejidad de un programa. En esta práctica nos proponemos diseñar un programa en Java para medir experimentalmente la complejidad de otro programa ya compilado.

Para ello, se implementa un programa Analizador.java capaz de determinar de forma automática la complejidad experimental de la función f(long n) de distintas clases denominadas Algoritmo.class que se proporcionan ya compiladas.
A priori se sabe que la complejidad de las funciones es siempre una de las ocho que se muestran en la siguiente tabla.

La ejecución del programa java Analizador dará como resultado una de las siguientes etiquetas:

Complejidad	Salida de java Analizador
$\Theta(1)$	1
$\Theta(log(n))$	LOGN
$\Theta(n)$	N
$\Theta(nlog(n))$	NLOGN
$\Theta(n2)$	N2
$\Theta(n3)$	N3
$\Theta(2n)$	2N
$\Theta(n!)$	NF

2. Código de la clase Analizador

```
public class Analizador {

  // Utilizaremos n1 para medir las complejidades pequeñas
  private static final long [] n1 =
{1,2,3,4,5,6,7,8,9,10,12,14,16,18,20,22,24,26,28,30,40,50,60,70,80,90,100,300,500,700
,1000,3000,5000,7000,10000,30000,50000,70000,
100000,200000,300000,500000,700000,800000,850000,900000,950000,
1000000,1500000,2000000,2500000,3000000,3500000,5000000,7000000};

  // Con este vector diferenciaremos tiempos de complejidades como N2 y NLOGN
  private static final long [] n2 =
{1,2,3,4,5,6,7,8,9,10,12,14,16,18,20,22,24,26,28,30,40,50,60,70,80,90,100,300,500,700
,1000,3000,5000,7000};

  private static double [] tiempo = {};


  //Procedimiento para calcular la media.
  public static double media(double a[]){
    double media=0.0;

    for(int i=1;i<a.length;i++){ //cogemos a partir del segundo valor para más exacti-
tud
      media+=a[i];
    }

    media=media/a.length-1;

    return media;
  }

  public static boolean costeAlto(){
    boolean esGrande=true;
    double suma1 = 0.0;
    double suma2 = 0.0;
    double ratio= 0.0;

    /*
    Sumamos los 5 primeros tiempos y los dividimos entre los 5 últimos para establecer
un ratio.
```

```
 */
if(!esNF()){

  for(int i=1;i<=5;i++){
    suma1+=tiempo[i];
  }

  for(int i=10;i>5;i--){
    suma2+=tiempo[i];
  }

  ratio = suma2/suma1;

  // Si supera este ratio, quiere decir que la complejidad es superior a N
  if(ratio<1.7) return esGrande=false;

  else{
    if(ratio<4){
      Temporizador t = new Temporizador();

      for(int i=0;i<n2.length;i++){
        t.iniciar();
        Algoritmo.f(n2[i]);
        t.parar();
        tiempo[i] = t.tiempoPasado();
        t.reiniciar();
      }

      double media=media(tiempo);

      if(media<65000){

        System.out.println("NLOGN");

                          }else{
        System.out.println("N2");

      }

    }else if(ratio<8){

      System.out.println("N3");

    }else{
```

```
            System.out.println("2N");
          }

          return esGrande;
        }
      }else{

        System.out.println("NF");
      }

      return esGrande;
    }

    /*
    En este método vamos a coger algunos valores del array n1 para tener una
    idea a priori de la complejidad con valores pequeños, ya que, en otro caso,
    si fuese muy grande los tiempos serían muy grandes. A su vez, también
    comprobamos si es NF.
     */
    public static boolean esNF(){
      boolean NF=false;
      Temporizador t = new Temporizador();

      for(int i=0;i<12;i++){
        t.iniciar();
        Algoritmo.f(n1[i]);
        t.parar();
        tiempo[i]=t.tiempoPasado();

        if(i!=0){
          //Tras hacer pruebas, si la division de los tiempos es mayor que 10, la complejidad es NF
          if((t.tiempoPasado()/tiempo[i-1])>10){
            i = 12;
            NF=true;
          }
        }
        t.reiniciar();
      }
      return NF;
    }

    /* Vamos a dividir las complejidades en dos partes, pequenas y grandes,
```

```
    de tal forma que usaremos unos valores mas altos para complejidades altas
    y unos valores mas pequenos para complejidades mas pequenas.
  */

  public static void main(String arg[]) {

    tiempo = new double[80];
    double media = 0.0;
    Temporizador t = new Temporizador();

    if(!costeAlto()){

      for(int i=0;i<n1.length;i++){
        t.iniciar();
        Algoritmo.f(n1[i]);
        t.parar();
        tiempo[i] = t.tiempoPasado();
        t.reiniciar();
      }

      //Aquí manejamos las complejidades menores o iguales que N
      media = media(tiempo);

      if(media<140){

        System.out.println("1");

      }else if(media<10000){

        System.out.println("LOGN");

      }else{
        System.out.println("N");
      }
    }

  }
}
```

3. Implementación del código

En primer lugar, para abordar el problema del analizador, divido el problema en dos. Por un lado, las complejidades pequeñas y, por otro lado, las complejidades grandes. Para poder medir las complejidades pequeñas almaceno distintos valores en la variable n1, la cual me permite trabajar con números más grandes sin problemas. Para las complejidades grandes, tengo valores más pequeños almacenados en n2, como podemos ver en este fragmento de código:

```
// Utilizaremos n1 para medir las complejidades pequenas
private static final long [] n1 =
{1,2,3,4,5,6,7,8,9,10,12,14,16,18,20,22,24,26,28,30,40,50,60,70,80,90,100,300,500,700
,1000,3000,5000,7000,10000,30000,50000,70000,
100000,200000,300000,500000,700000,800000,850000,900000,950000,
1000000,1500000,2000000,2500000,3000000,3500000,5000000,7000000};

// Con este vector diferenciaremos tiempos de complejidades como N2 y NLOGN
private static final long [] n2 =
{1,2,3,4,5,6,7,8,9,10,12,14,16,18,20,22,24,26,28,30,40,50,60,70,80,90,100,300,500,700
,1000,3000,5000,7000};

private static double [] tiempo = {};
```

Además, será necesario almacenar los tiempos de las pruebas para los distintos valores que tomará el algoritmo, lo cual se almacenará en la variable *tiempo.*

Pasamos a la función main del programa, donde primero se inicializarán algunas variables como *media,* o el *temporizador t* .

```
public static void main(String arg[]) {

  tiempo = new double[80];
  double media = 0.0;
  Temporizador t = new Temporizador();
```

La variable *media* servirá para almacenar la media de los tiempos del algoritmo que queremos analizar. Existe una función para calcular la media, la cual consiste en recorrer los valores de tiempo y sumarlos para dividirlo entre el total de elementos y así obtener la media. No obstante, observé que el primer valor modificaba bastante la media y eso producía valores irreales a la hora de analizar la complejidad, por lo que siempre se descarta el primer valor, para ser más preciso.

```
public static double media(double a[]){
  double media=0.0;
```

```
  for(int i=1;i<a.length;i++){ //cogemos a partir del segundo valor para mas exactitud
    media+=a[i];
  }

  media=media/a.length-1;

  return media;
}
```

Volviendo a la función main, después de inicializar las variables, compruebo si el algoritmo con el que voy a tratar es de coste alto (N2, NLOGN, N3, 2N o NF) con la función costeAlto(). De esta forma es más sencillo, ya que puedo usar los valores de n1 o n2 una vez estime si será de coste alto o no.

Si no es de coste alto, utilizo n1 para darle valores a la función f de la clase algoritmo a través de un bucle for, de la siguiente forma:

```
for(int i=0;i<n1.length;i++){
  t.iniciar();
  Algoritmo.f(n1[i]);
  t.parar();
  tiempo[i] = t.tiempoPasado();
  t.reiniciar();
}
```

En este bucle utilizamos el temporizador para medir los tiempos de la función de la clase algoritmo y almacenarlos en la variable *tiempo*. Una vez terminado, hacemos la media de los resultados de tiempo y, dependiendo del resultado, estimaremos la complejidad del algoritmo. Estos “ratios” han sido deducidos a través de prueba y error, son unos valores estimados después de probar distintos algoritmos de cada complejidad:

```
media = media(tiempo);

if(media<140){

  System.out.println("1");

}else if(media<10000){

  System.out.println("LOGN");

}else{
```

```
  System.out.println("N");
}
```

De esta manera hemos conseguido tratar con las complejidades pequeñas.

La función costeAlto() inicializa las variables double suma1, suma2 y ratio, además de una variable booleana esGrande, que se usará más adelante.

```
public static boolean costeAlto(){
  boolean esGrande=true;
  double suma1 = 0.0;
  double suma2 = 0.0;
  double ratio= 0.0;
```

Lo primero que se comprueba es si es NF o no y, para ello, existe el método esNF(), que se explica más adelante. En el caso de que lo sea, se devolverá lo siguiente:

```
}else{

  System.out.println("NF");
}

return esGrande;
```

Donde esGrande devuelve true si al complejidad es alta.

Si deducimos que no es NF, sumamos los 5 primeros tiempos y los dividimos entre los 5 últimos para establecer un ratio, de la siguiente forma:

```
if(!esNF()){

  for(int i=1;i<=5;i++){
    suma1+=tiempo[i];
  }

  for(int i=10;i>5;i--){
    suma2+=tiempo[i];
  }

  ratio = suma2/suma1;
```

Al igual que antes, después de probar distintos algoritmos, si el ratio es menor que aproximadamente 1.7, el algoritmo no es de coste alto, por lo cual devuelve false y la complejidad se tratará en el Main.

Ahora, pasamos a tratar las distintas complejidades con if:

- Si el ratio es menor que 4, hacemos el mismo bucle for que con las complejidades pequeñas pero esta vez usamos n2 en lugar de n1. Hacemos la media de los valores de tiempo y, si la media es menor de 65000, la complejidad es NLOGN y, en otro caso, es N2.

```
- if(ratio<4){
    Temporizador t = new Temporizador();

    for(int i=0;i<n2.length;i++){
      t.iniciar();
      Algoritmo.f(n2[i]);
      t.parar();
      tiempo[i] = t.tiempoPasado();
      t.reiniciar();
    }

    double media=media(tiempo);

    if(media<65000){

      System.out.println("NLOGN");

                              }else{
      System.out.println("N2");

    }
```

- Si el ratio es menor que 8, la complejidad es N3

```
- }else if(ratio<8){

    System.out.println("N3");

  }
```

- En cualquier otro caso, la complejidad es 2N

```
- }else{

    System.out.println("2N");
  }
```

Por otro lado, la función esNF() devuelve True si es NF o False en caso contrario. En este método vamos a coger algunos valores del array n1 para tener una idea a priori de la complejidad con valores pequeños, ya que, en otro caso, si fuese muy grande los tiempos serían muy grandes. Al hacer la división de tiempos, si el resultado es mayor que 10, podemos determinar que la función es NF.

```
public static boolean esNF(){
  boolean NF=false;
  Temporizador t = new Temporizador();

  for(int i=0;i<12;i++){
    t.iniciar();
    Algoritmo.f(n1[i]);
    t.parar();
    tiempo[i]=t.tiempoPasado();

    if(i!=0){
      //Tras hacer pruebas, si la division de los tiempos es mayor que 10, la complejidad es NF
      if((t.tiempoPasado()/tiempo[i-1])>10){
        i = 12;
        NF=true;
      }
    }
    t.reiniciar();
  }
  return NF;
}
```

A modo de resumen, en la clase analizador divido las complejidades en dos grupos, grandes y pequeñas, hago una comprobación a priori para saber con qué tipo estoy tratando y, después, analizo los tiempos y a través de los ratios determino que tipo de complejidad es.

Es importante recalcar que los ratios son valores estimados y deducidos de realizar las pruebas con distintos algoritmos y hacer un debug del programa para comprobar los valores, por lo tanto, es probable que para ciertos algoritmos de una complejidad errónea.