



---

# TECNICAS DE PROGRAMACION AVANZADAS

---

## PRACTICA 1



VICTOR PEREZ PEREZ

19/04/2021

Nºexp: 21923658

## Tabla de contenido

<b>Ejercicio 1</b> .....	2
<b>Función: esIdentidad_v1</b> .....	2
Explicación del problema y solución propuesta .....	2
Código .....	2
Tiempo y Orden de ejecución .....	4
<b>Función: esIdentidad_v2</b> .....	5
Solución propuesta .....	5
Código .....	5
Tiempo y Orden de ejecución .....	6
<b>Función: esIdentidad_v3</b> .....	7
Solución propuesta .....	7
Código .....	7
Tiempo y Orden de ejecución .....	9
<b>Mejor versión para comprobar las matrices</b> .....	11
<b>Ejercicio 2</b> .....	12
Solución propuestas .....	12
<b>Función Divide y Vencerás</b> .....	12
Tiempo y Orden de ejecución .....	13
<b>Función con estrategia iterativa tradicional</b> .....	14
Tiempo y Orden de ejecución .....	15
Comparación .....	16
<b>Ejercicio 3</b> .....	17

# Ejercicio 1

## Función: esIdentidad\_v1

### Explicación del problema y solución propuesta

Tras estudiar los pasos que sigue la función, me he dado cuenta de que el error está a la hora de almacenar el valor de la variable identidad. La función compara todos los números, pero al final, solo devuelve el resultado de la última posición. Es decir, no importa como sea la matriz, porque si el último elemento es igual a 1, la función nos devuelve true (matriz identidad)

Para solucionar esto, he creado un bucle if, al que entrara mientras todas las posiciones recorridas, coincidan con las posiciones de la matriz identidad correspondientes. Si encontramos una posición errónea, ya no comprobaremos más posiciones, y cuando acabemos de recorrer el array, devolveremos el valor de la variable identidad.

### Código

```
public static boolean esIdentidad_v1 (int[][] matriz) {
    boolean identidad = true;
    for (int fila = 0; fila < matriz.length; fila++) {
        for (int col = 0; col < matriz[fila].length; col++) { //
            if(identidad == true) {
                if (fila==col) {
                    if (matriz[fila][col] == 1)
                        identidad = true;
                    else
                        identidad = false;
                }
                else {
                    if (matriz[fila][col] == 0)
                        identidad = true;
                    else
                        identidad = false;
                }
            }
        }
    }
    return identidad;
}
```

```

public static boolean esIdentidad_v1 (int[][] matriz) {
    boolean identidad = true;

    for (int fila = 0; fila < matriz.length; fila++) {
        for (int col = 0; col < matriz[fila].length; col++) {
            if(identidad == true) {
                if (fila==col) {
                    if (matriz[fila][col] == 1)
                        B identidad = true;
                    else
                        identidad = false;
                }
            }
            else {
                if (matriz[fila][col] == 0)
                    A identidad = true;
                else
                    identidad = false;
            }
        }
    }
    return identidad;
}

```

Diagram illustrating the execution flow of the `esIdentidad_v1` method:

- The flow starts at the beginning of the method (G).
- It enters the first `for` loop (F).
- It enters the second `for` loop (E).
- It enters the `if(identidad == true)` block (D).
- It enters the `if (fila==col)` block (C).
- It enters the `if (matriz[fila][col] == 1)` block (B).
- It enters the `if (matriz[fila][col] == 0)` block (A).
- It exits the `if (matriz[fila][col] == 0)` block (A).
- It exits the `if (matriz[fila][col] == 1)` block (B).
- It exits the `if (fila==col)` block (C).
- It exits the `if(identidad == true)` block (D).
- It exits the second `for` loop (E).
- It exits the first `for` loop (F).
- It exits the method (G).

## Tiempo y Orden de ejecución

$$T(A) = T(B)$$

$$T(A) = 1 + \text{Max}(1, 1) = 2$$

$$T(C)$$

$$T(C) = 1 + \text{Max}(T(A), T(B)) = 1 + \text{Max}(2, 2) = 3$$

$$T(D)$$

$$T(D) = 1 + \text{Max}(T(C)) = 1 + \text{Max}(3) = 4$$

$$T(E)$$

$$\begin{aligned} T(E) &= 1 + \sum_{i=0}^{n-1} (1 + 4 + 1) + 1 = 2 + 6 \sum_{i=0}^{n-1} 1 = \\ &= 2 + 6(n-1-0+1) = 6n + 2 \end{aligned}$$

$$T(F)$$

$$\begin{aligned} T(F) &= 1 + \sum_{i=0}^{n-1} (1 + (6n + 2) + 1) + 1 = 2 + (6n + 4) \sum_{i=0}^{n-1} 1 = \\ &= 2 + (6n + 4)(n-1-0+1) = 6n^2 + 4n + 2 \end{aligned}$$

$$T(G)$$

$$T(G) = 2 + 6n^2 + 4n + 2 + 1$$

$$\text{Tiempo de ejecución} = 6n^2 + 4n + 5$$

$$\text{Orden de ejecución} = O(n^2)$$

## Función: esIdentidad\_v2

### Solución propuesta

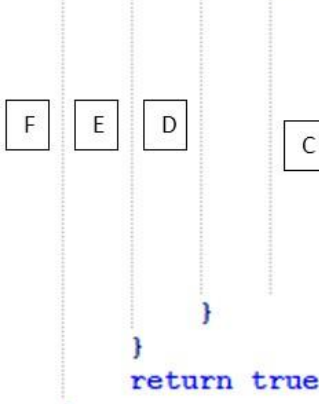
En esta función, vamos a recorrer el array comprobando cada posición. En el momento en el que encontremos una posición que no coincida con la posición de la matriz identidad correspondiente, saldremos de la función retornando un false. Si conseguimos recorrer la matriz entera, sin encontrar datos en posiciones incorrectas, retornaremos true.

Esta función, también se puede hacer con dos bucles while, recorriendo las filas una a una y actualizando el valor de la columna cada vez que acabamos de recorrer la fila.

### Código

```
public static boolean esIdentidad_v2 (int[][] matriz) {  
    for (int fila = 0; fila < matriz.length; fila++) {  
        for (int col = 0; col < matriz[fila].length; col++) {  
            if(fila==col) {  
                if(matriz[fila][col] != 1) {  
                    return false;  
                }  
            }else {  
                if(matriz[fila][col] != 0) {  
                    return false;  
                }  
            }  
        }  
    }  
    return true;  
}
```

```
public static boolean esIdentidad_v2 (int[][] matriz) {  
    for (int fila = 0; fila < matriz.length; fila++) {  
        for (int col = 0; col < matriz[fila].length; col++) {  
            if(fila==col) {  
                if(matriz[fila][col] != 1) {  
                    B  
                    return false;  
                }  
            }else {  
                if(matriz[fila][col] != 0) {  
                    A  
                    return false;  
                }  
            }  
        }  
    }  
    return true;  
}
```



## Tiempo y Orden de ejecución

$$T(A) = T(B)$$

$$T(A) = 1 + \text{Max}(1) = 2$$

$$T(C)$$

$$T(C) = 1 + \text{Max}(T(A), T(B)) = 3$$

$$T(D)$$

$$\begin{aligned} T(D) &= 1 + \sum_{i=0}^{n-1} (1 + 3 + 1) + 1 = 2 + 4 \sum_{i=0}^{n-1} 1 = \\ &= 2 + 4(n-1-0+1) = 2 + 4n \end{aligned}$$

$$T(E)$$

$$\begin{aligned} T(E) &= 1 + \sum_{i=0}^{n-1} (1 + (2 \cdot 4n) + 1) + 1 = 2 + (4n+4) \sum_{i=0}^{n-1} 1 = \\ &= 2 + (4n+4)(n-1-0+1) = 4n^2 + 4n + 2 \end{aligned}$$

$$T(F)$$

$$T(F) = 4n^2 + 4n + 2 + 1$$

$$\text{Tiempo de ejecución} \rightarrow 4n^2 + 4n + 3$$

$$\text{Orden de ejecución} \rightarrow O(n^2)$$

## Función: esIdentidad\_v3

### Solución propuesta

Para poder aplicar la estrategia de Divide y Vencerás, la función necesita tres parámetros, el array o matriz, un inicio y un fin. Para poder realizar la estrategia sin modificar las cabeceras propuestas, desde la función dada en el ejercicio, llamaremos a otra función auxiliar. Esta función auxiliar tiene como parámetros la matriz, un inicio y un fin, cumpliendo así los requisitos necesarios para aplicar la estrategia pedida.

Esta función auxiliar, va a dividir la matriz en filas, para poder comprobar si cada una de estas coincide con la fila correspondiente de la matriz identidad. Si alguna de las dos mitades obtenidas al aplicar DyV, no coincide con la matriz identidad, devolveremos false, en caso contrario, devolveremos true.

Además, en el caso base llamamos a una función para realizar la comprobación de la fila correspondiente.

### Código

```
public static boolean esIdentidad_DyV (int[][] matriz) {
    return esIdentidad_DyV2(matriz, 0, matriz.length-1);
}

private static boolean esIdentidad_DyV2 (int[][] matriz, int inicio,
int fin) {
    if(inicio == fin) {
        return comprobar(matriz, inicio);
    }else {
        int mitad = (inicio + fin)/2;
        boolean x = esIdentidad_DyV2(matriz, inicio, mitad);
        boolean y = esIdentidad_DyV2(matriz, mitad+1, fin);
        if(x == false) {
            return false;
        }else if (y == false) {
            return false;
        }else {
            return true;
        }
    }
}

private static boolean comprobar (int[][] matriz, int inicio) {
    if(matriz[inicio][inicio] == 1) {
        for(int i=0; i<matriz.length; i++) {
            if(i!= inicio) {
                if(matriz[inicio][i] != 0) {
                    return false;
                }
            }
        }
    }else {
        return false;
    }
    return true;
}
```



```

public static boolean esIdentidad_DyV (int[][] matriz) {
    return esIdentidad_DyV2(matriz, 0, matriz.length-1);
}

```

Funciones auxiliares

```

private static boolean esIdentidad_DyV2 (int[][] matriz, int inicio, int fin) {
    if(inicio == fin) {
        return comprobar(matriz, inicio);
    } else {
        int mitad = (inicio + fin)/2;
        boolean x = esIdentidad_DyV(matriz, inicio, mitad);
        boolean y = esIdentidad_DyV(matriz, mitad+1, fin);

        if(x == false) {
            return false;
        } else if (y == false) {
            return false;
        } else {
            return true;
        }
    }
}

```

```

private static boolean comprobar (int[][] matriz, int inicio) {
    if(matriz[inicio][inicio] == 1) {
        for(int i=0; i<matriz.length; i++) {
            if(i!= inicio) {
                if(matriz[inicio][i] != 0) {
                    return false;
                }
            }
        }
    } else {
        return false;
    }
    return true;
}

```

## Tiempo y Orden de ejecución

$T(A)$

$$T(A) = 1 + \text{Max}(1) = 2$$

$T(B)$

$$T(B) = 1 + \text{Max}(2) = 3$$

$T(C)$

$$\begin{aligned} T(C) &= 1 + \sum_{i=0}^{n-1} (1 + 3 + 1) + 1 = 2 + 5 \sum_{i=0}^{n-1} 1 = \\ &= 2 + 5(n - 1 - 0 + 1) = 5n + 2 \end{aligned}$$

$T(D)$

$$T(D) = 1 + \text{Max}(5n + 2, 1) = 5n + 3$$

$T(E)$

$$T(E) = 5n + 3 + 1 = 5n + 4$$

Tiempo de ejecución  $\rightarrow 5n + 4$

Caso base

$$T(C) = 1 + 5n + 4 = 5n + 5$$

$$C_0 = 5n + 5 \quad n^{u_0} = n \rightarrow k_0 = 1$$

Caso general

$$T(A) = 1 + \text{Max}(1, 1, 1) = 2$$

$$T(B) = 2 + 4 + 2 + 3 = 11$$

$$C_1 = 11 \quad n^{u_1} = 1 \rightarrow k_1 = 0$$

$a$  = llamadas en cada invocación = 2

$b$  = valor por el que reduce = 2

$$\text{Tiempo de ejecución} \rightarrow T(n) \begin{cases} (5n + 5)n \\ 2 \cdot T(n/2) + 11 \end{cases}$$

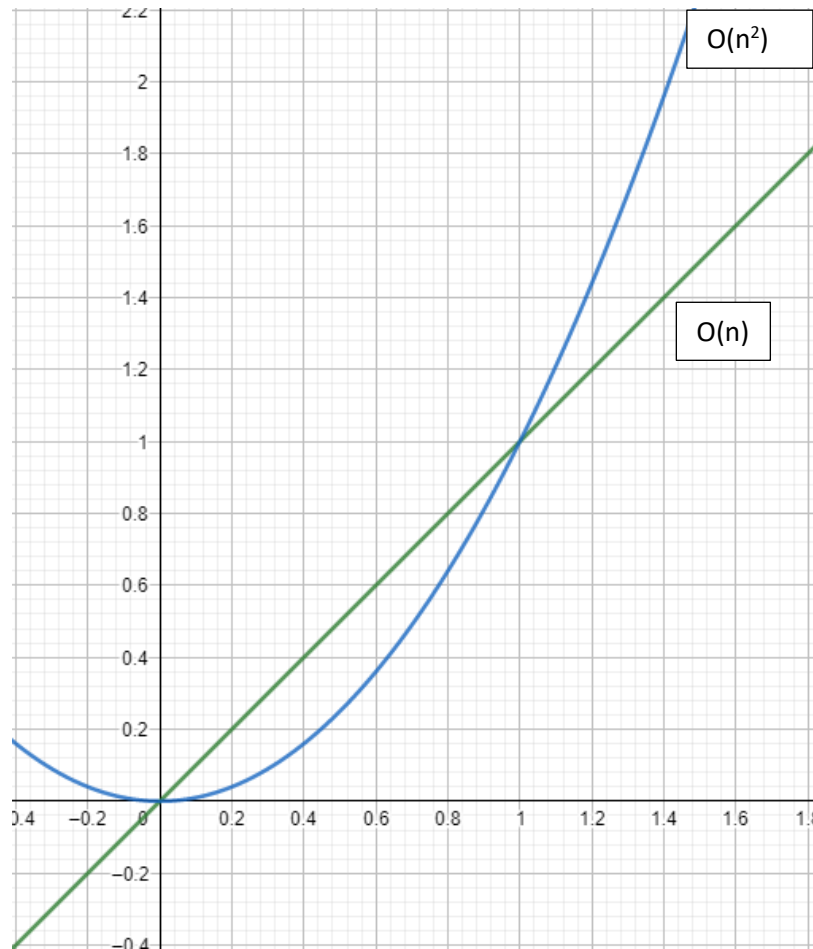
$$\text{Orden de ejecución} \quad a > b^{u_1} \rightarrow 2 > 1$$

$$O(n^{\log_2 2}) = O(n)$$

## Mejor versión para comprobar las matrices

Aunque la función `esIdentidad_DyV` es la más eficaz, no podemos conseguir nada si no implementamos además funciones auxiliares. Estas son necesarias ya que para realizar la estrategia de Divide y Vencerás, es necesario pasarle como parámetros un inicio y un fin.

Sin embargo, esta función tiene un tiempo y orden de ejecución menor, por lo que podemos decir que es la mejor versión para comprobar si la matriz es o no la identidad.



## Ejercicio 2

### Solución propuestas

Aplicando la estrategia DyV vamos a conseguir dividir el array en elementos individuales. Sumaremos estos elementos de dos en dos e iremos calculando las medias correspondientes. Al final, obtendremos dos medias, la de la parte izquierda(x) y la de la parte derecha(y), las cuales iremos sumando hasta obtener la media correspondiente a nuestro array.

Aplicando una estrategia iterativa, recorreremos el array con un bucle while, en el cual almacenaremos en una variable suma, la suma de los elementos que hemos recorrido hasta llegar a la última posición. Después, dividimos el valor de la variable suma entre la longitud del array, obteniendo la media.

### Función Divide y Vencerás

```
/**
 * Funcion a la cual le pasamos el array, su inicio y su fin.
 * Dividiremos el array tantas veces hasta que obtener un solo
 * elemento, este se lo sumaremos
 * a su correspondiente e iremos calculando la media de estos.
 * De esta manera obtendremos la media
 * de cada la parte izquierda (x) y la parte derecha (y) y
 * las sumaremos, consiguiendo la media del array
 *
 * @param array, inicio, fin
 * @return
 */
public static float media(int []array, int inicio, int fin) {
    if(inicio == fin) {
        return array[inicio];
    }else {
        int mitad = (inicio + fin)/2;
        float x = media(array, inicio, mitad);
        float y = media(array, mitad+1, fin);
        return (x+y)/2;
    }
}

public static float media(int []array, int inicio, int fin) {
    if(inicio == fin) {
        A return array[inicio];
    }else {
        B int mitad = (inicio + fin)/2;
        float x = media(array, inicio, mitad);
        float y = media(array, mitad+1, fin);
        return (x+y)/2;
    }
}
```



## Tiempo y Orden de ejecución

Caso base ( $C_0 n^{k_0}$ )

$T(A)$

$$T(A) = 1 + (1 + 1) = 3$$

$$C_0 = 3 \quad n^{k_0} = 1 \rightarrow k_0 = 0$$

Caso general ( $a T(n/b) + C_1 n^{k_1}$ )

$$a = n^{\circ} \text{ llamadas recursivas} = 2$$

$$b = n^{\circ} \text{ por el que divido} = 2$$

$T(B)$

$$T(B) = 4 + 2 + 3 + 2 = 11$$

$$C_1 = 11 \quad n^{k_1} = 1 \rightarrow k_1 = 0$$

$$T(n) = \begin{cases} 3 \\ 2 \cdot T(n/2) + 11 \end{cases}$$

$$a = 2, \quad b = 2, \quad k_1 = 0$$


$$2 > 2^0 \rightarrow 2 > 1$$

$$\text{orden de ejecución} \quad O(n^{\log_2 2}) = O(n)$$

## Función con estrategia iterativa tradicional

```
/**
 * En esta función tenemos dos variables inicializadas a cero,
 * que nos indicaran la posición en la que nos encontramos
 * y donde almacenaremos la suma de los elementos recorridos
 *
 * Recorremos la lista con un bucle while. Una vez recorrida
 * y obtenida la suma de todos los elementos, devolvemos esa
 * suma dividida entre la longitud del array, es decir,
 * devolvemos la media del array
 *
 * @param m1
 * @return
 */
public static float media2(int []m1) {
    float suma = 0;
    int posicion = 0;
    while(posicion < m1.length) {
        suma = suma + m1[posicion];
        posicion++;
    }
    return suma/m1.length;
}
```

```
public static float media2(int []m1) {
    float suma = 0;
    int posicion = 0;
    while(posicion < m1.length) {
        suma = suma + m1[posicion];
        posicion++;
    }
    return suma/m1.length;
}
```



## Tiempo y Orden de ejecución

$T(A)$

$$T(A) = \sum_{i=0}^{n-1} (1+3) + 1 = 1 + 4(n-1-0+1) = 4n+1$$

$T(B)$

$$T(B) = 2 + 2 + (4n+1) + 2 = 4n+7$$

Tiempo de ejecución  $\rightarrow 4n+7$

Orden de ejecución  $\rightarrow O(n)$



## Comparación

Para realizar el ejercicio de una forma más sencilla, rápida y eficaz, deberíamos implementar la función iterativa, ya que solo tenemos un bucle para recorrer el array entero y donde realizamos solo la suma de los elementos. Mientras que si utilizamos la estrategia Divide y Vencerás, vamos a realizar más divisiones hasta sumar los elementos uno a uno, de una forma menos eficaz.

```
Array = {1,2,3,4,5,6,7,8}
Usando Divide y Vencerás :
    4.5

Usando forma iterativa :
    4.5
```

Para realizar la comprobación de si la longitud del array es una potencia de dos, he creado la siguiente función. Esta recorrerá el array, dividiendo entre dos su tamaño. Si encuentra algún número cuyo modulo 2 sea distinto de 0, significa que ese número no es potencia de dos (cualquier número que sea potencia de 2, su modulo siempre va a ser 0, hasta que lleguemos a la operación 2/2, donde sería 1) .

```
public static boolean comprobarPotencia(int longitud) {
    while(longitud > 1) {
        if(longitud%2!=0) {
            return false;
        }
        longitud = longitud/2;
    }
    return true;
}
```

### Ejercicio 3

Expresión del Doble Hashing:  $H(k) = H1(k) + (c \cdot H2(k))$

$$H1(k) = k \bmod N$$

$$H2(k) = 5 - (k \bmod 3)$$

$$H(k,c) = k \bmod N + (c \cdot (5 - (k \bmod 3)))$$

Además, el factor de carga no debe superar el 80%.

Cada vez que insertemos un elemento, vamos a calcular el factor de carga, para poder saber en qué momento tenemos o no que redimensionar la tabla.

- Inicio

Posiciones ocupadas = 2

Posiciones totales = 7

Factor de carga = 28.57%

Posicion	0	1	2	3	4	5	6
Valor			9			2	

- Insertamos primer elemento (44)

$$H(44,0) = 44 \bmod 7 + (0 \cdot (5 - (44 \bmod 3))) = 2 + (0 \cdot 3) = 2 \rightarrow \text{posición ocupada}$$

$$H(44,1) = 44 \bmod 7 + (1 \cdot (5 - (44 \bmod 3))) = 2 + (1 \cdot 3) = 5 \rightarrow \text{posición ocupada}$$

$$H(44,2) = 44 \bmod 7 + (2 \cdot (5 - (44 \bmod 3))) = 2 + (2 \cdot 3) = 8 \rightarrow \text{posición fuera de rango} \rightarrow 8 \bmod 7 = 1 \rightarrow \text{posición libre}$$

Posiciones ocupadas = 3

Posiciones totales = 7

Factor de carga = 42.85%

Posicion	0	1	2	3	4	5	6
Valor		44	9			2	

- Insertamos el segundo elemento (6)

$$H(6,0) = 6 \bmod 7 + (0 \cdot (5 - (6 \bmod 3))) = 6 + (0 \cdot 5) = 6 \rightarrow \text{posición libre}$$

Posiciones ocupadas = 4

Posiciones totales = 7

Factor de carga = 57.14%

Posicion	0	1	2	3	4	5	6
Valor		44	9			2	6

- Insertamos el tercer elemento (43)

$H(43,0) = 43 \bmod 7 + (0 * (5 - (43 \bmod 3))) = 1 + (0 * 4) = 1 \rightarrow$  posición ocupada  
 $H(43,1) = 43 \bmod 7 + (1 * (5 - (43 \bmod 3))) = 1 + (1 * 4) = 5 \rightarrow$  posición ocupada  
 $H(43,2) = 43 \bmod 7 + (1 * (5 - (43 \bmod 3))) = 1 + (2 * 4) = 9 \rightarrow$  posición fuera de rango  $\rightarrow 9 \bmod 7 = 2 \rightarrow$  posición ocupada  
 $H(43,3) = 43 \bmod 7 + (1 * (5 - (43 \bmod 3))) = 1 + (3 * 4) = 13 \rightarrow$  posición fuera de rango  $\rightarrow 13 \bmod 7 = 6 \rightarrow$  posición ocupada  
 $H(43,4) = 43 \bmod 7 + (1 * (5 - (43 \bmod 3))) = 1 + (4 * 4) = 17 \rightarrow$  posición fuera de rango  $\rightarrow 17 \bmod 7 = 3 \rightarrow$  posición libre

Posiciones ocupadas = 5

Posiciones totales = 7

Factor de carga = 71.42%

Posicion	0	1	2	3	4	5	6
Valor		44	9	43		2	6

- Insertamos el cuarto elemento (12)

Podemos observar, que si insertamos este elemento, el factor de carga sería igual a 85.71%, superando el límite del enunciado. Para evitar que esto ocurra, vamos a redimensionar la tabla.

Tamaño de la tabla \* 2 = 14  $\rightarrow$  siguiente número primo = 17 (nuevo tamaño)

Insertamos en la nueva tabla los elementos de la tabla obtenida anteriormente:

Posicion	0	1	2	3	4	5	6
Valor		44	9	43		2	6

Insertamos en la nueva tabla los elementos en orden, y después seguimos insertando los que nos pide el enunciado

- Insertamos el primer elemento (44)

$H(44,0) = 44 \bmod 17 + (0 * (5 - (44 \bmod 3))) = 10 + (0 * 3) = 10 \rightarrow$  posición libre

Posiciones ocupadas = 1

Posiciones totales = 17

Factor de carga = 5.88%

Posicion	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valor											44						

- Insertamos el segundo elemento (9)

$$H(9,0) = 9 \bmod 17 + (0 \cdot (5 - (9 \bmod 3))) = 9 + (0 \cdot 5) = 9 \rightarrow \text{posición libre}$$

Posiciones ocupadas = 2

Posiciones totales = 17

Factor de carga = 11.76%

Posicion	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valor										9	44						

- Insertamos el tercer elemento (43)

$$H(43,0) = 43 \bmod 17 + (0 \cdot (5 - (43 \bmod 3))) = 9 + (0 \cdot 4) = 9 \rightarrow \text{posición ocupada}$$

$$H(43,1) = 43 \bmod 17 + (1 \cdot (5 - (43 \bmod 3))) = 9 + (1 \cdot 4) = 13 \rightarrow \text{posición libre}$$

Posiciones ocupadas = 3

Posiciones totales = 17

Factor de carga = 17.64%

Posicion	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valor										9	44			43			

- Insertamos el cuarto elemento (2)

$$H(2,0) = 2 \bmod 17 + (0 \cdot (5 - (2 \bmod 3))) = 2 + (0 \cdot 3) = 2 \rightarrow \text{posición libre}$$

Posiciones ocupadas = 4

Posiciones totales = 17

Factor de carga = 23.53%

Posicion	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valor			2							9	44			43			

- Insertamos el quinto elemento (6)

$$H(6,0) = 6 \bmod 17 + (0 \cdot (5 - (6 \bmod 3))) = 6 + (0 \cdot 5) = 6 \rightarrow \text{posición libre}$$

Posiciones ocupadas = 5

Posiciones totales = 17

Factor de carga = 29.41%

Posicion	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valor			2				6			9	44			43			

Ahora, comenzamos con los elementos que nos faltaban del enunciado

- Insertamos el sexto elemento (12)

$$H(12,0) = 12 \bmod 17 + (0 \cdot (5 - (12 \bmod 3))) = 12 + (0 \cdot 5) = 12 \rightarrow \text{posición libre}$$

Posiciones ocupadas = 6

Posiciones totales = 17

Factor de carga = 35.29%

Posicion	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valor			2				6			9	44		12	43			

- Insertamos el séptimo elemento (23)

$$H(23,0) = 23 \bmod 17 + (0 \cdot (5 - (23 \bmod 3))) = 6 + (0 \cdot 3) = 6 \rightarrow \text{posición ocupada}$$

$$H(23,1) = 23 \bmod 17 + (1 \cdot (5 - (23 \bmod 3))) = 6 + (1 \cdot 3) = 9 \rightarrow \text{posición ocupada}$$

$$H(23,2) = 23 \bmod 17 + (2 \cdot (5 - (23 \bmod 3))) = 6 + (2 \cdot 3) = 12 \rightarrow \text{posición ocupada}$$

$$H(23,3) = 23 \bmod 17 + (3 \cdot (5 - (23 \bmod 3))) = 6 + (3 \cdot 3) = 15 \rightarrow \text{posición libre}$$

Posiciones ocupadas = 7

Posiciones totales = 17

Factor de carga = 41.17%

Posicion	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valor			2				6			9	44		12	43		23	

La tabla final nos quedaría:

Posicion	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valor			2				6			9	44		12	43		23	

