PRÁCTICA 3

Ángela López López UO270318

SUSTRACCIÓN 1:

N	sustraccion1 (ms)
1	0,00000031
2	0,0000033
4	0,00000534
8	0,00001024
16	0,00002101
32	0,00004168
64	0,00008748
128	0,00017095
256	0,0003334
512	0,00064115
1024	0,00132218
2048	0,00386248
4096	0,008441
8192	0,016894
16384	0,034219
32768	0,070041

Esta clase implementa un método con esquema por Sustracción, cuyos parámetros son a=1, b=1 y k=0;

Como a=1 se aplica la relación O(n^{k+1}), por lo que su complejidad temporal es O(n).
El gasto de pila es O(n) por lo que la pila se desborda

```java
public class Sustraccion1 {
    static long cont;

    public static boolean rec1(int n) {
        if (n <= 0)
            cont++;
        else {
            cont++; // O(1)=O(n^0)
            rec1(n - 1); //a=1, b=1
        }
        return true;
    }
```

SUSTRACCIÓN 2:

N	sustraccion2 (ms)
1	0,000017
2	0,000016
4	0,000015
8	0,000079
16	0,000218
32	0,000852
64	0,003861
128	0,014582
256	0,053999
512	0,141
1024	0,556
2048	2,234
4096	8,757
8192	34,982
16384	139,082
32768	563,173

Esta clase implementa un método con esquema por Sustracción, cuyos parámetros son a=1, b=1 y k=1;

Como a=1 se aplica la relación O(n^{k+1}), por lo que su complejidad temporal es O(n^2).
El gasto de pila es O(n) por lo que la pila se desborda

```java
public static boolean rec2(int n) {
    if (n <= 0)
        cont++;
    else {
        for (int i = 0; i < n; i++)
            cont++; // O(n) k=1
        rec2(n - 1); //a=1, b=1
        for (int i = 0; i < n; i++)
            cont++; // O(n) k=1
    }
    return true;
}
```

SUSTRACCIÓN 3:

N	sustraccion3 (ms)
18	0,063
19	0,219
20	0,234
21	0,281
22	0,347
23	1,917
24	1,963
25	2,654
26	3,257
27	16,974
28	19,703
29	23,881
30	28,771

Esta clase implementa un método con esquema por Sustracción, cuyos parámetros son a=2, b=1 y k=0;

Como a<1 se aplica la relación O($a^{n/b}$), por lo que su complejidad temporal es O(2^n).

El gasto de pila es O(n) pero no se desborda porque mucho antes el tiempo de ejecución se hace intratable

```java
public static boolean rec3(int n) {
    if (n <= 0)
        cont++;
    else {
        cont++; // O(1) k=0
        rec3(n - 1);
        rec3(n - 1); //a=2, b=1
    }
    return true;
}
```

SUSTRACCIÓN 4:

N	sustraccion4 (ms)
18	0,047
19	0,203
20	0,266
21	0,39
22	0,53
23	1,797
24	2,252
25	3,557
26	4,714
27	16,097
28	20,888
29	32,116
30	42,368

Esta clase implementa un método con esquema por Sustracción, cuyos parámetros son a=3, b=2 y k=1;

Como a<1 se aplica la relación O($a^{n/b}$), por lo que su complejidad temporal es O($3^{n/2}$).

El gasto de pila es O(n) pero no se desborda porque mucho antes el tiempo de ejecución se hace intratable

```java
public static boolean rec4(int n) {
    if (n <= 0)
        cont++;
    else {
        for (int i = 1; i < n; i++) {
            cont++; // O(n) k=1
        }
        rec4(n - 2);
        rec4(n - 2);
        rec4(n - 2); // a=3 b=2
    }
    return true;
}
```

DIVISIÓN 1:

N	division1 (ms)
1	0
2	0,000015
4	0,000016
8	0,000047
16	0,000093
32	0,000157
64	0,000312
128	0,000641
256	0,001262
512	0,002457
1024	0,004843
2048	0,009647
4096	0,019121
8192	0,038194
16384	0,076407
32768	0,151933

Esta clase implementa un método con esquema por División, cuyos parámetros son a=1, b=3 y k=1;

Como a< b^k se aplica la relación O(n^k), por lo que su complejidad temporal es O(n).

El gasto de pila es O(log n) por lo que por lo que por mucho que crezca n no se desbordará.

```
public static boolean rec1 (int n)
{
    if (n<=0)
        cont++;
    else
    {
        for (int i=1;i<n;i++) cont++ ;  //O(n)  k=1
        rec1 (n/3); //a=1 b=3
    }
    return true;
}
```

DIVISIÓN 2:

N	division2 (ms)
1	0
2	0,000015
4	0,000032
8	0,000047
16	0,000094
32	0,000265
64	0,000438
128	0,001187
256	0,002075
512	0,005486
1024	0,00957
2048	0,024486
4096	0,042955
8192	0,109141
16384	0,190312
32768	0,482271

Esta clase implementa un método con esquema por División, cuyos parámetros son a=2, b=2 y k=1;

Como a= b^k se aplica la relación O($n^k logn$), por lo que su complejidad temporal es O($nlogn$).

El gasto de pila es O(log n) por lo que por lo que por mucho que crezca n no se desbordará.

```
public static boolean rec2 (int n)
{
    if (n<=0) cont++;
    else
    {
        for (int i=1;i<n;i++) cont++ ;  //O(n) k=1
        rec2 (n/2);
        rec2 (n/2); //a=2 b=2
    }
    return true;
}
```

DIVISIÓN 3:

N	division3 (ms)
1	0,000016
2	0,000016
4	0,000016
8	0,000047
16	0,000047
32	0,000203
64	0,000234
128	0,000859
256	0,001142
512	0,003188
1024	0,003742
2048	0,012984
4096	0,015447
8192	0,051045
16384	0,061173
32768	0,204931

Esta clase implementa un método con esquema por División, cuyos parámetros son a=2, b=2 y k=0;

Como a> b^k se aplica la relación O($n^{log_n b^a}$), por lo que su complejidad temporal es O(n).

El gasto de pila es O(log n) por lo que por lo que por lo que por mucho que crezca n no se desbordará.

```
public static boolean rec3 (int n)
{
    if (n<=0)
        cont++;
    else
    {
        cont++ ; // O(1)  k=0
        rec3 (n/2);
        rec3 (n/2);//a=2 b=2
    }
    return true;
}
```

DIVISIÓN 4:

N	division4 (ms)
1	0
2	0,00015
4	0,00016
8	0,00016
16	0,00031
32	0,00141
64	0,00312
128	0,01078
256	0,04903
512	0,15592
1024	0,64269
2048	2,19426
4096	9,2295
8192	37,554
16384	155,71
32768	608,082

Esta clase implementa un método con esquema por División, cuyos parámetros son a=4, b=3y k=2;

Como a< b^k se aplica la relación O(n^k), por lo que su complejidad temporal es O(n^2).

No se desborda.

```
public static boolean rec4(int n) {
    if (n <= 0)
        cont++;
    else {
        for (int i = 1; i < n; i++) {
            for (int j = 1; j < i; j++) {
                cont++; // O(n^2) k=2
            }
        }
        rec4(n / 3);
        rec4(n / 3);
        rec4(n / 3);
        rec4(n / 3); // 4 subproblemas b=3
    }
    return true;
}
```

TÍO GILITO

Nos piden encontrar una moneda falsa entre la gran riqueza que posee el tío Gilito. Para ello se utiliza una balanza (ya que la moneda falsa pesa menos).

GILITO1: La clase Gilito1 calcula la energía media consumida en aplicar el algoritmo de ordenación para n monedas, siendo n un parámetro de entrada.
Cada vez que se pesa aumenta la energía consumida.

- El método **gilito1** devuelve la posición donde se encuentra la moneda falsa. Para ello evalúa cada par de posiciones (0 y 1, 2 y 3, 4 y 5…) hasta que encuentre donde está la moneda falsa. De esta forma, si la cantidad de monedas es impar y se han evaluado todas se sabe que la última en la falsa.

En caso de que la moneda falsa esté en el primer par solo se debe pesar una vez, por lo que la energía consumida es 1 y la complejidad O(1).
En caso de que la moneda falsa sea última moneda hay que usar la balanza n/2 veces, por lo que la complejidad es O(n).
En caso de que la moneda falsa esté en la media de todas las posiciones se gasta aproximadamente n/4 por lo que la complejidad es O(n).

Se evalúan todos los casos (desde que la moneda falsa sea la primera a la última) y se calcula su energía media consumida.

GILITO1TIEMPOS: Se calcula el tiempo para el caso peor. En este caso, el peor caso es que la moneda falsa sea la última pues tendríamos que evaluar todas.

GILITO2: La clase Gilito2 minimiza el gasto energético de Gilito1. Para ello se hace un divide y vencerás.

- Si es impar deja sin evaluar la última moneda del grupo.
- Se subdivide en 2 grupos el resto de monedas.
- Si el grupo izquierdo pesa menos, sabemos que la moneda falsa está en él y se sigue evaluando recursivamente por este camino. Si el grupo derecho pesa menos se sigue evaluando por este camino. Si pesan lo mismo sabemos que la moneda falsa es la última que dejamos sin evaluar en esa iteración.

GILITO2TIEMPOS: Se calcula el tiempo para el caso peor. En este caso, el peor es que la moneda falsa sea la primera pues tendríamos que evaluar todas.

N	gilito1 (wH)	gilito2 (wH)	gilito1tiempos (ms)	gilito2tiempos (ms)
100	25	5	0,0001781	0,0000781
200	50	6	0,0003645	0,0001297
400	100	7	0,000716	0,0002388
800	200	8	0,0014352	0,0004803
1600	400	9	0,0028747	0,0009507
3200	800	10	0,005803	0,0018969
6400	1600	11	0,0119847	0,0038037

Gilito2 (el caso medio) tiene O(logn) wH frente al O(n) que tiene gilito1 y sus tiempos en el caso peor mejoran los de gilito1.