Ejercicio 2.

Queremos obtener números enteros a partir de otros dos n, k (siendo $n>k>1$), según la siguiente definición:

$$exam(n,k) = \begin{cases} n+k, & si\ k \leq 2 \\ n*k + exam(n,k/2) - exam(n,k/3), & e.o.c. \end{cases}$$

Se proponen dos implementaciones a la anterior definición:

```
long exam1(int n, int k) {
    if (k <= 2) {
        return n + k;
    } else {
        long p = 0;
        for (int i = 1; i <= k; i++) {
             p += n;
        }
        return p + exam1(n, k/2) – exam1(n, k/3);
    }
}
```

```
long exam2(int n, int k) {
    return exam2(n, k, new HashMap<>());
}
long exam2(int n, int k, Map<Pair<Integer, Integer>, Long> m) {
    Long res = m.get(Pair.of(n, k));
    if(res==null) {
        if (k <= 2) {
           res = n + k + 0L;
        } else {
           res = n*k + exam2(n, k/2, m) - exam2(n, k/3, m);
        }
        m.put(Pair.of(n, k), res);
    }
    return res;
}
```

SE PIDE:

1) Indique razonadamente el tamaño del problema y la clase de recursividad de ambos métodos.

2) Analice el orden de complejidad de ambos métodos. Indique para cada método el tipo de complejidad dentro de la jerarquía de órdenes.

3) Implemente un método iterativo para la definición anterior siguiendo el esquema de transformación *bottom-up* y justifique su complejidad.

SOLUCIÓN:

1) El tamaño del problema es *k* para ambos métodos, dado que es el parámetro que cambia de valor en las llamadas recursivas. La clase de recursividad del primero es múltiple sin memoria. La del segundo, múltiple con memoria.

2) Para el primer método tenemos:

$$T(k) = \sum_{i=1}^{k} 1 + T\left(\frac{k}{2}\right) + T\left(\frac{k}{3}\right) = k + T\left(\frac{k}{2}\right) + T\left(\frac{k}{3}\right);$$

No podemos calcular el orden exacto luego debemos acotarlo:

$\Omega(T(k)) = 2T(k/3) + k \rightarrow \Omega(T(k)) = k;$ [Caso B: 2,3,1,0]

$O(T(k)) = 2T(k/2) + k \rightarrow O(T(k)) = k \log k;$ [Caso B: 2,2,1,0]

El primer método es por tanto cuasi-lineal con cotas: $k <_\infty \Theta(T(k)) <_\infty k \log_2 k$

Para el segundo método la complejidad se reduce al número de casos distintos. Podemos aproximar el orden generando la secuencia de dichos casos:

$$\{k\} \overset{1}{\rightarrow} \left\{\frac{k}{2}, \frac{k}{3}\right\} \overset{2}{\rightarrow} \left\{\frac{k}{4}, \frac{k}{6}, \frac{k}{6}, \frac{k}{9}\right\} \overset{3}{\rightarrow} \left\{\frac{k}{8}, \frac{k}{12}, \frac{k}{12}, \frac{k}{18}, \frac{k}{12}, \frac{k}{18}, \frac{k}{18}, \frac{k}{27}\right\} \overset{4}{\rightarrow} \dots;$$

Si eliminamos los casos que se repiten en cada nivel nos quedaría:

$$\{k\} \overset{1}{\rightarrow} \left\{\frac{k}{2^1}, \frac{k}{3^1}\right\} \overset{2}{\rightarrow} \left\{\frac{k}{2^2}, \frac{k}{6}, \frac{k}{3^2}\right\} \overset{3}{\rightarrow} \left\{\frac{k}{2^3}, \frac{k}{12}, \frac{k}{18}, \frac{k}{3^3}\right\} \overset{4}{\rightarrow} \left\{\frac{k}{2^4}, \frac{k}{24}, \frac{k}{36}, \frac{k}{54}, \frac{k}{3^4}\right\} \overset{5}{\rightarrow} \dots;$$

Por tanto, los valores de k para las llamadas recursivas de profundidad i son $\frac{k}{2^i}, \frac{k}{2^{i-1}3^1}, \frac{k}{2^{i-2}3^2}, \dots, \frac{k}{3^i}$; es decir, i+1 valores distintos. La profundidad de los casos base estará entre $log_3 k$ y $log_2 k$, por lo que el número de llamadas distintas estará entre $\sum_{i=0}^{log_3 k}(i+1)$ y $\sum_{i=0}^{log_2 k}(i+1)$, de forma que:

$$\Omega\big(T(k)\big) = \sum_{i=0}^{log_3 k}(i+1) \cong (log_3 k)^2 \rightarrow \Omega\big(T(k)\big) = (\log k)^2$$
$$O\big(T(k)\big) = \sum_{i=0}^{log_2 k}(i+1) \cong (log_2 k)^2 \rightarrow O\big(T(k)\big) = (\log k)^2$$

Luego, $\Theta(T(k) = (\log k)^2$

3) La transformación *bottom-up* a iterativo sería:

```
public static long exam3B(int n, int k) {
    long[] ls = new long[k+1];
    for (int i = 0; i <= k; i++) {
            if(i<3)
                    ls[i]=n+i;
            else
                    ls[i] = n*i + ls[i/2] - ls[i/3];
    }
    return ls[k];
}
```

La complejidad en este caso es lineal, $T(k) = \sum_{i=0}^{k} 1 \rightarrow \Theta\big(T(k)\big) = k;$