**Juan José Hernández Arenas**

**202259500**

**Opcional 2 de Programación Funcional y Concurrente**

**ENUNCIADO:**

La potenciación de enteros es una operación que involucra dos enteros positivos a y n y se trata de calcular $a^n$. Esta operación puede abordarse así:
**Solución ingenua:**
Multiplicar a por sí mismo n veces, para esto haga un vector con a repetido n veces y aplique la función `foldLeft` o `foldRight` para multiplicar los elementos
**Solución recursiva:** Asumamos que tenemos la función $f(a,n)=a^n$ el proceso es:

1. Si $n=0$ retornar 1
2. Si n es impar retornar $a*f(a,n-1)$
3. Si n es par retornar $f(a,n/2)*f(a,n/2)$

**INFORME:**

En las funciones establecidas, probamos con dos valores (153 y 80), todas las soluciones por lo que vemos, nos da igual valor, ya que vamos a operar lo mismo, pero de diferente forma. Observo que la solución ingenua me parece ser una forma más fácil de operar estas potencias, ya que no tiene demasiado código y se puede realizar de una manera sencilla.

```
Valor de a: 153
Valor de n: 80
Solucion ingenua: 1209767297

Solucion ingenua paralela: 1209767297
Solucion recursiva : 1209767297
Solucion recursiva paralela : 1209767297
```

Utilizando el benchmark medimos los tiempo de ejecución de cada función para poder compararlas, observo que la solución ingenua tiene un tiempo más rápido que la forma paralela, con una aceleración de 0,2231, por lo tanto, no es eficaz hacer el código para realizarlo de forma paralela (recordemos que esta comparación es con los números de a:153 y n:80).

```
Comparacion:

(Tiempo de solucion ingenua, tiempo de solucion ingenua paralela, aceleracion)
Unable to create a system terminal
(0.0659,0.2953,0.22316288520149)
```

Comparamos la solución recursiva con los mismos valores de a y n, y se examina que la forma paralela sigue sin ser tan efectiva, ya que es demasiada rápida la forma normal, con un tiempo de 0,0084, en cambio la forma paralela tiene un tiempo de 0,4918 con la aceleración de 0,0170.

```
(Tiempo de solucion recursiva, tiempo de solucion recursiva paralela, aceleracion)
(0.0084,0.4918,0.01708011386742578)
```

Hacemos de nuevo pero con diferentes valores ( a:61 y n:53), y observo que la paralela sigue siendo la forma más lenta de hacer estas potencias, recordemos que la paralelización sirve dependiendo del tamaño, y con este tamaño no es efectivo.

```
Valor de a: 61
Valor de n: 53
Solucion ingenua: -1224803571
Solucion ingenua paralela: -1224803571
Solucion recursiva : -1224803571
Solucion recursiva paralela : -1224803571

Comparacion:
(Tiempo de solucion ingenua, tiempo de solucion ingenua paralela, aceleracion)
Unable to create a system terminal
(0.0983,0.2796,0.3515736766809728)
(Tiempo de solucion recursiva, tiempo de solucion recursiva paralela, aceleracion)
(0.0085,0.1825,0.04657534246575343)
```

Realicé 6 test, utilizando la forma ingenua dos veces y la ingenua paralela una vez, lo mismo con la recursiva que utilicé dos pruebas y la forma recursiva paralela solo una vez.

En el primer test se ponen dos valores (a:10 y n:3) y comprobamos que da como resultado el valor de 100 llamando a la función. En el segundo tenemos dos valores diferentes, los cuales son 6 y 7, y nos da un resultado de 279936. Por último, en la prueba paralela, utilizamos el 8 y 5, que al hacer la operación da como resultado 32768.

```
    test( testName = "testSolucionIngenua"){
        val obj = new Opcional()
        val a = 10
          val n=3
        assert( condition = 1000 == obj.solucionIngenua(a,n))
    }

    test( testName = "testSolucionIngenua2") {
        val obj = new Opcional()
        val a = 6
        val n = 7
        assert( condition = 279936 == obj.solucionIngenua(a, n))
    }
    test( testName = "testSolucionIngenuaPar") {
        val obj = new Opcional()
        val a = 8
        val n = 5
        assert( condition = 32768 == obj.solucionIngenuaPar(a,n))
    }
```

En las pruebas de la solución recursiva se escribieron diferentes valores, en el primer test a es igual a 2 y n es igual a 25, con un resultado de 33554432. En la segunda prueba, los valores son 13 y 5 con su resultado de371293. Y en el tercero los valores son 105 y, con un resultado de 1157625.

```
test( testName = "testSolucionRecursiva") {
    val obj = new Opcional()
    val a = 2
    val n = 25
    assert( condition = 33554432== obj.solucionRecursiva(a,n))
}
test( testName = "testSolucionRecursiva2") {
    val obj = new Opcional()
    val a = 13
    val n = 5
    assert( condition = 371293 == obj.solucionRecursiva(a, n))
}
test( testName = "testSolucionRecursivaPar"){
    val obj = new Opcional()
    val a = 105
    val n = 3
    assert( condition = 1157625 == obj.solucionRecursivaPar(a,n))
}
```

Por último, comprobamos que las pruebas salieron correctas.

```
✓ Tests passed: 6 of 6 tests – 80 ms

> Task :app:compileJava NO-SOURCE
> Task :app:compileScala
> Task :app:processResources NO-SOURCE
> Task :app:classes
> Task :app:compileTestJava NO-SOURCE
> Task :app:compileTestScala
> Task :app:processTestResources NO-SOURCE
> Task :app:testClasses
> Task :app:test
BUILD SUCCESSFUL in 15s
3 actionable tasks: 3 executed
4:28:51 p. m.: Execution finished ':app:test --tests "opcional.OpcionalTest"'.
```