**Ejercicios segunda actividad del portafolio.**

**Asignatura: ESTRUCTURAS DE DATOS Y ALGORITMOS 15GIIN**

**Resolución de ejercicios teórico-prácticos sobre Recursión y Análisis de algoritmos**

**Calificación: 10 puntos (5% de la nota final)**

**Instrucciones:**

- **Deberá resolver los ejercicios (1 y 3) ó (2 y 3)**, una de las dos posibilidades
- Deberá enviar al profesor un archivo comprimido (.zip, .rar) con la siguiente información:
  - el nombre cumpla con la especificación siguiente: ACT2_NOMBRE_PRIMERAPELLIDO_SEGUNDOAPELLIDO. Por ejemplo: ACT2_OSCAR_MEZA_HOUTTEMAN.zip
  - Sus respuestas las dará en un archivo PDF
  - El código del programa en JAVA (si procede) deberá enviarlo en un archivo de texto (o .java) de manera que el profesor pueda luego ejecutarlo. Coloque todas las clases e interfaces en un solo archivo.

**Fecha de entrega del enunciado por parte del profesor: 15 de Noviembre 2022**
**Fecha de entrega de la solución por parte del alumno: 29 de Noviembre 2022**
**Recuerde que de entregar después del 29 de Noviembe, se corregirá la actividad sobre 8 y no sobe 10 puntos**

**Preguntas:**

**PUEDE ESCOGER ENTRE LA PREGUNTA 1 ó LA 2 (RESPONDER SÓLO UNA). LA PREGUNTA 3 ES OBLIGATORIO RESPONDERLA.**

1. Un programa tarda 0,5 milisegundos para el tamaño de entrada 100. ¿Cuántas veces más tardará para un tamaño de entrada 1000 si el tiempo de ejecución es el siguiente?:

   a. lineal

   b. O(N log N) (suponga el logaritmo en base 10). La respuesta puede ser una estimación

   c. cuadrático

   d. cúbico

   Nota: Suponga que los términos de orden inferior son insignificantes de cada una de las funciones (lineal, cuadrática, n log n y cúbica) y "veces más" significa, por ejemplo, que si tarda 3 veces más para una entrada de 1000 entonces el programa tardará 0.5 * 3 milisegundos para una entrada de 1000

   (2 puntos)

2. Hacer un programa recursivo en JAVA para determinar **st(n,m) para n>=m>=1**, donde:

   a. st(n,m) = st(n-1,m-1) + m*st(n-1,m)
   b. st(n,1)=1 y st(n,n)=1.

   Si lo desea, haga también un programa no recursivo para determinar **st(n,m),** no es obligatorio.

   (2 puntos)

3. La clase **EvaluarPolinomio** dada más abajo, evalúa de tres formas distintas un polinomio **pol** de grado **n** en un valor **x**. Los coeficientes del polinomio se guardan en un arreglo **pol** con **(n+1)** elementos y donde los coeficientes se guardan desde la potencia más alta **n** hasta la potencia 0, así **pol[0]** es el coeficiente del polinomio para la potencia $x^n$, **pol[1]** para la potencia $x^{n-1}$, y así sucesivamente.

   **Pregunta**: Dar la complejidad **exacta** en función de **n** (el grado del polinomio) del tiempo en peor caso de los métodos "calculaHorner", "calculaConPotencia" y "calculaConPotencia1". **Debe en cada caso justificar detalladamente sus respuestas**. (8 puntos).

   **Nota**: De matemáticas discretas conocemos los siguientes resultados que pueden ser útiles en sus respuestas:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} \log i \leq (n \log n)$$

```
import java.util.Scanner;
```

```java
/* Evaluación de un polinomio por tres programas distintos */

public class EvaluarPolinomio {

    private static double calculaHorner(double [ ] pol,
            double x, int n) {
      /* Aplica la fórmula de Horner para evaluar el polinomio
      pol de grado n en el valor de la variable x */

        double resultado = 0;
        for (int i = 0; i <= n; i++) {
            resultado = (resultado * x) + pol[i];
        }
        return resultado;
    }

    private static double calculaConPotencia(double [ ] pol,
            double x, int n) {
      /* Aplica la fórmula del polinomio para evaluar
      pol de grado n en el valor de la variable x
      las potencias i-esimas de x se calculan multiplicando
       i veces x */

      int i=n; double suma=0.0;
        while (i >= 0) {
            suma += pol[n-i]*potencia(x, i);
            i--;
         }
        return suma;
    }
    private static double calculaConPotencia1(double [ ] pol,
            double x, int n) {
      /*   Aplica la fórmula del polinomio para evaluar
         * el polinomio pol de grado n en el valor de
         * la variable x
         * las potencias i-esimas de x se calculan con
         * un algoritmo mejorado potencia1*/

        int i=n; double suma=0.0;
        while (i >= 0) {
            suma += pol[n-i]*potencia1(x, i);
            i--;
         }
        return suma;
    }

    public static double potencia(double x,int i) {
      double resultado = 1.0;
      for(int j=0;j<i;j++)
            resultado*=x;
      return resultado;
    }

    public static double potencia1(double x,int i) {

      if( i == 0 ) return 1;
```

```
        if( i == 1 ) return x;
        if( i%2 == 0 )return potencia1( x * x, i / 2 );
        else return potencia1( x * x, i / 2 ) * x;
    }

    public final static void main(String[] args) {
        try (Scanner reader = new Scanner(System.in)) {
            System.out.println("Introduce el grado del polinomio: ");
            int n = reader.nextInt();
            double[ ] pol= new double[n+1];
            System.out.println("Introduce coeficientes "
                    + "polinomio de mayor a menor grado: ");
            for(int i=0;i<=n;i++) {
             pol[i] = reader.nextDouble();
            }
            System.out.println("Introduce el valor x: ");
            double x = reader.nextDouble();

            double resultado;
            resultado = calculaHorner(pol, x, n);
            System.out.println(String.format("Resultado: %10.2f",
                    resultado));
            resultado = calculaConPotencia(pol, x, n);
            System.out.println(String.format("Resultado: %10.2f",
                    resultado));
            resultado = calculaConPotencia1(pol, x, n);
            System.out.println(String.format("Resultado: %10.2f",
                    resultado));
        } catch(Exception e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    }
}
```