Practica1macodecena.pdf

macodecena

Análisis y diseño de algoritmos

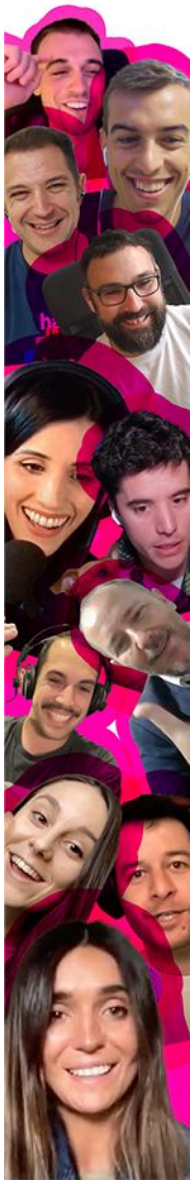
2º Grado en Ingeniería del Software

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

No sé en qué momento nos pareció buena idea lanzar nuestro podcast para estudiantes en verano.

escúchatelo o algo, así le digo a mi jefe que ha sido un éxito

```
public class Analizador {

	/*
	 * NOTA IMPORTANTE
	 *
	 * Esta clase se proporciona solamente para ilustrar el formato de
	 * salida que deberia tener la solucion a este ejericio.
	 * Esta clase debe modificarse completamente para cumplir
	 * mínimamente los requisitos de esta practica.
	 * Notese que ni siquiera esta completa......
	 */


	private static void algoritmo(int a) {
		// En esta funcion estan los "simuladores" de algoritmos. Han
servido de prueba para probar funciones de casi todas las complejidades.

		// FUNCION CONSTANTE:

//		int res = constante(a);


		// FUNCION LOGARITMICA:
//		int res = potencia(2, a);


		// FUNCION LINEAL:
//		int res = suma(a);


		// FUNCION CUASILINEAL:

//


		// FUNCION CUADRATICA:

//		int res = sumaDeSumas(a);


		// FUNCION CUBICA:

//		int res = cub(a);


		// FUNCION EXPONENCIAL:

//		double res = raiz(100,a);


	}

	private static int constante (int a) {
		// complejidad constante.
```



Wuolah @macodecena

```
	int suma = 0;
	for (int i=0; i<10; i++) {
		suma++;
	}
	return suma;
}

private static int potencia(int x, int n) {
	// complejidad logaritmica.
	if (n==0)
		return 1;
	else {
		if (n%2==0) {
			return (potencia(x*x, n/2));
		}
		else {
			return x * potencia(x*x, n/2);
		}
	}
}

private static int suma(int a) {
	// complejidad lineal.
	if (a==0)
		return 0;
	else {
		return a+suma(a-1);
	}
}

private static int sumaDeSumas(int a) {
	// complejidad cuadratica.
	if (a == 0)
		return 0;
	else {
		return suma(a) + sumaDeSumas(a-1);
	}
}

private static int cub (int a) {
	// complejidad cubica.
	int res = 0;
	for (int i=0; i<a; i++) {
		for (int j=0; j<a; j++) {
			for (int k=0; k<a; k++) {
				res++;
			}
		}
	}
	return res;
}

private static double raiz(double x, int n) {
	// complejidad exponencial.
	if (n==0)
```

Wuolah @macodecena

EL PRIMER NÚMERO QUE VEAS, SERÁ TU NOTA EN EL PRÓXIMO EXAMEN

O L G S N R W B F Q L Y Q E
S U T M W T C U A T R O O H
E P G R R R J S E A N L M R
A N G J E E P V Q T F N O L
Y R P E Y S P P M J G Z M L
M A T R I C U L A V A A F C
Y S Y C L G K K E F H X S L
V N M I U Y G A J J L Z C O
X U D O S R Q V Y N E O R Y
B E S A M K D I E S S C T B
S V I V O B H S V E C H G A
W E E E V T I J I I G O U J
N D T C I N C O J S Z F F P
E N E A U U N O J J O W S D

Comparte por rr.ss. la nota que hayas visto y etiquétanos

BeReal.

WUOLAH

Análisis y diseño de algoritmos

Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

```
                return 1;
            else {
                return (x/(raiz(x, n-1)) + (raiz(x, n-1)))/2;
            }
        }


        public static void main(String arg[]) {
            // las siguientes entradas variaran en funcion de la complejidad
que compruebe el programa en cada momento.
            int [] entradas1 =
{1,1,1,1,1,100000000,100000000,100000000,100000000,100000000}; //      1 / LOGN
            int [] entradas2 = {5,10,20,50,700,1000,1500,2000,4000,7000}; //
                                    N / NLOGN
            int [] entradas3 = {5,10,20,50,700,1000,1500,2000,4000,7000}; //
                                   N2
            int [] entradas4 = {4,8,15,20,24,30,40,50,60,70}; //
                                            N3
            int [] entradas5 = {1,2,3,5,7,9,10,12,14,16}; //
                                            2N / NF

            int n = 10;

            long [] tiempos1 = new long[10];
            long [] tiempos2 = new long[10];
            double [] ratio = new double[10];
            Temporizador t = new Temporizador();

            boolean encontrado = false;
            int cont = 5;       // contador decreciente.
            int comp = 0;       // almacenara el dato que indicara si la
complejidad es correcta, o si hay que probar con mayor/menor complejidad.
            String [] salidas = {"1", "LOGN", "N", "N2", "N3", "2N", "NF"}; //
NLOGN es un caso "especial" (explicado en la memoria)

            while (!encontrado) {
                for (int i = 0; i < n; i++) {
                    t.reiniciar();
                    t.iniciar();

                    switch (cont) {     // para usar las distintas entradas
en funcion de la complejidad que vayamos a comprobar.
                    // se almacena el tiempo que tarda el Algoritmo.class
                        case 1: {           // COMPROBAR SI ES 1 / LOGN
                            Algoritmo.f(entradas1[i]);
                            //algoritmo(entradas1[i]);
                        }
                        case 2: {           // COMPROBAR SI ES N / NLOGN
                            Algoritmo.f(entradas2[i]);
                            //algoritmo(entradas2[i]);
                        }
                        case 3: {           // COMPROBAR SI ES N2
                            Algoritmo.f(entradas3[i]);
                            //algoritmo(entradas3[i]);
                        }
```

Wuolah @macodecena

```
                        case 4: {          // COMPROBAR SI ES N3
                              Algoritmo.f(entradas4[i]);
                              //algoritmo(entradas4[i]);
                        }
                        case 5: {          // COMPROBAR SI ES 2N / NF
                              Algoritmo.f(entradas5[i]);
                              //algoritmo(entradas5[i]);
                        }
                  }

                  t.parar();

                  tiempos1[i] = t.tiempoPasado();

                  switch (cont) {    // para usar las distintas entradas
en funcion de la complejidad que vayamos a comprobar.
                  // se almacena el tiempo que tardan las funciones
"modelo" de cada complejidad.
                        case 1: {
                              if (i==0) {
                                    encontrado = true;
                                    if (!diferenciaMitades(tiempos1)) {
                                          // Las condiciones de la
cabecera equivalen a una complejidad constante.

      System.out.println(salidas[0]);
                                    }
                                    else {
                                          // La complejidad es
logaritmica.

      System.out.println(salidas[1]);
                                    }
                              }
                              break;
                        }
                        case 2: {
                              tiempos2[i] = lineal(entradas2[i]);
                              break;
                        }
                        case 3: {
                              tiempos2[i] = cuadratica(entradas3[i]);
                              break;
                        }
                        case 4: {
                              tiempos2[i] = cubica(entradas4[i]);
                              break;
                        }
                        case 5: {
                              tiempos2[i] = exponencial(entradas5[i]);
                              break;
                        }
                  }

                  ratio[i] = (double)tiempos2[i]/tiempos1[i];
```

Wuolah @macodecena

```
			}

			if (cont != 1)  {
				comp = comprobarComplejidad(ratio);


				if (comp == -1) { // El modelo probado no sirve. Es de orden de complejidad inferior. Seguimos buscando.
					cont--;
				}

				else if (comp == 0) { // El modelo probado es correcto. Encontrado.
					encontrado = true;
					System.out.println(salidas[cont]);
				}

				else { // El modelo probado no sirve. Es de 1 orden de complejidad superior. Encontrado.
					encontrado = true;
					if (cont == 2) {
						System.out.println("NLOGN");
					}
					else {
						System.out.println(salidas[cont+1]);
					}
				}
			}
		}
	}



	private static boolean diferenciaMitades(long[] tiempos1) {
		// devuelve true si el minimo elemento de la segunda mitad de tiempos1 es un 20% mayor que el minimo de la primera mitad.
		int l = tiempos1.length;

		long min1 = tiempos1[l/2];
		for (int i=0; i<l/2; i++) {
			if (tiempos1[i] < min1) {
				min1 = tiempos1[i];
			}
		}

		long min2 = tiempos1[l/2];
		for (int i=(l/2)+1; i<l; i++) {
			if (tiempos1[i] < min2) {
				min2 = tiempos1[i];
			}
		}

		return min2 >= 1.2*min1;
	}
```

```
	// la siguiente funcion devuelve:
	// -1 si el orden de complejidad del algoritmo es inferior al del modelo
usado.
	// 0 si el orden de complejidad del algoritmo coincide con el del modelo
usado.
	// 1 si el orden de complejidad del algoritmo es superior al del modelo
usado.
	private static int comprobarComplejidad(double[] ratio) {
		int res = 0;

		if (valoresNegativos0(ratio)) {
			res = -1;
		}

		else {
			if ((valoresCerc0(ratio)) ||
					(!valoresCrecientes(ratio))) {
				res = 1;
			}

			else {
				if (todosCrecientes(ratio)) {
					res = -1;
				}

				else if ((valoresCrecientes(ratio)) &&
						(valoresLejosMedia(ratio))) {
						res = -1;
				}
			}
		}

		return res;
	}


	private static boolean valoresNegativos0(double[] ratio) {
		// devuelve true si un 30% de los valores de ratio (o más) es
negativo o 0.

		int param = (int) (ratio.length*0.3);
		int cont = 0;

		for (double elem : ratio) {
			if (elem <= 0) {
				cont++;
			}
		}

		return cont > param;
	}

	private static boolean valoresCerc0(double[] ratio) {
```

Wuolah @macodecena

```
		// devuelve false si dos o mas valores de ratio son mayores que
0,001.

		int cont = 0;

		for (double elem : ratio) {
			if (elem > 0.001) {
				cont++;
			}
		}

		return cont < 2;
	}

	private static boolean valoresCrecientes(double[] ratio) {
		// devuelve true si el 70% de los valores de ratio (sin contar los
3 primeros) es mayor que el 70% de valores anteriores.

		int contj = 0, conti = 0;

		for (int i=3; i<ratio.length; i++) {
			for (int j=0; j<i; j++) {
				if (ratio[i] > ratio[j]) {
					contj++;
				}
			}
			if (contj >= Math.floor(i*0.7)) {
				conti++;
			}
			contj = 0;
		}

		return conti >= Math.floor((ratio.length-3)*0.7);
	}

	private static boolean todosCrecientes(double[] ratio) {
		// devuelve true si el 100% de los valores de ratio (sin contar el
primero) es mayor que el 100% de valores anteriores,
		// y si el ultimo valor de ratio es un 30% mayor que el valor de la
posicion 4 de ratio.

				int contj = 0, conti = 0;

				for (int i=1; i<ratio.length; i++) {
					for (int j=0; j<i; j++) {
						if (ratio[i] > ratio[j]) {
							contj++;
						}
					}
					if (contj == i) {
						conti++;
					}
					contj = 0;
				}
```

Wuolah @macodecena

```
                        return (conti == ratio.length-1) &&
(ratio[ratio.length-1] > 1.3 * ratio[4]);
	}

	private static boolean valoresLejosMedia(double[] ratio) {
		// devuelve true si el 20% de los valores de ratio distan en 1,5
unidades de su media, o bien si el ultimo valor cumple eso mismo.

		double media = 0;
		int cont = 0;

		int param = (int) (ratio.length*0.2);

		for (double elem : ratio) {
			if (elem > 0) {
				media += elem;
				cont++;
			}
		}
		media /= cont;
		cont = 0;

		for (double elem : ratio) {
			if ((abs(media-elem)) > 1.5) {
				cont++;
			}
		}

		return ((cont >= param) || (ratio[9]-1.5 > media));
	}

	private static double abs(double d) {
		// devuelve el valor absoluto de 'd'.

		double res = d;

		if (d < 0)
			res = -d;

		return res;
	}


	// Funciones modelo de lo que cabria esperar que tardara cada complejidad
en ejecutar.

	private static long lineal(int n) {
		// T(n) = n
		return n;
	}

	private static long cuasilineal(int n) {
		// T(n) = n*log(n)
		return n * (long)Math.log(n);
	}
```

Wuolah @macodecena

No sé en qué momento nos pareció buena idea lanzar nuestro podcast para estudiantes en verano.

escúchatelo o algo, así le digo a mi jefe que ha sido un éxito

```
	private static long cuadratica(int n) {
		// T(n) = n^2
		return n*n;
	}

	private static long cubica(int n) {
		// T(n) = n^3
		return n*n*n;
	}

	private static long exponencial(int n) {
		// T(n) = 2^n
		return (long)Math.pow(2, n);
	}
}
```