UNLP
Facultad de Informática
Algoritmos y Estructuras de Datos
Curso 2015

Tiempos de Ejecución

Ejercicio 1

Sean dos algoritmos alg1 y alg2 que realizan la misma de tarea de calcular el promedio de edad de un grupo de personas que solicitó el pasaporte en el año 2011. Cada uno de estos algoritmos está codificado de forma diferente, por lo cual, el tiempo de ejecución para el peor caso de alg1 es $T_1(n) = 10.000n$ mientras que el de alg2 es $T_2(n) = n^2$.

¿Con que cantidad de personas alg1 es mejor que alg2? ¿Y a la inversa? Fundamente su respuesta.

Ejercicio 2

Dadas las siguientes funciones ordenarlas según sus velocidades de crecimiento.

a. $T_1(n) = n*\log_2(n)$
b. $T_2(n)= (1/3)n$
c. $T_3(n) =2n+n^2$
d. $T_4(n) = (3/2)n$
e. $T_5(n) = (\log_2(n))^2$
f. $T_6(n) = \log_2(n)$
g. $T_7(n) = n+\log_2(n)$
h. $T_8(n) = n^{1/2}$
i. $T_9(n) = 3^n$

Para poder ordenarlas, utilice el graficador de funciones que se encuentra en http://fooplot.com/.

Dado que el graficador solo trabaja con logaritmos en base 10 (log o $\log_{10}$) utilice el cambio de base para convertir los logaritmos (tal como se explica en: http://es.wikipedia.org/wiki/Logaritmo#Cambio_de_base). Básicamente:

$$Log_a x = \frac{Log_b x}{Log_b a} \Leftrightarrow a \neq 1, b \neq 1$$

Ejercicio 3

Determinar si las siguientes sentencias son verdaderas o falsas, justificando la respuesta.

a. 3^n es de $O(2^n)$
b. $n/\log_2(n)$ es de $O(\log_2(n))$
c. $n^{1/2} + 10^{20}$ es de $O(n^{1/2})$
d. $n+ \log_2(n)$ es de $O(n)$
e. Si p(n) es un polinomio de grado k, entonces p(n) es $O(n^k)$. Pista: Si $p(n)=a_k n^k + ... + a_1 n + a_0$, utilizar como constante a $M = |a_k| + ... + |a_1| + |a_0|$
f. $\begin{cases} 3n+17, n<100 \\ 317, n \geq 100 \end{cases}$ tiene orden lineal
g. $\begin{cases} n^2, n \leq 100 \\ n, n>100 \end{cases}$ tiene orden cuadrático
h. 2^{n+1} es de $O(2^n)$
i. 2^{2n} es de $O(2^n)$

Ejercicio 4

Se necesita generar una permutación random de los n primeros números enteros. Por ejemplo [4,3,1,0,2] es una permutación legal, pero [0,4,1,2,4] no lo es, porque un número está duplicado (el 4) y otro no está (el 3). Presentamos tres algoritmos para solucionar este problema. Asumimos la existencia de un generador de números random, ran_int (i,j) el cual genera en tiempo constante, enteros entre i y j inclusive con igual probabilidad. También suponemos el mensaje swap () que intercambia dos datos entre si. Si bien los tres

algoritmos generan permutaciones random legales, tenga presente que por la forma en que utilizan la funcion ran_int algunos de ellos podrían no terminar nunca, mientras otro sí.

```
public class Ejercicio4 {
	private static Random rand = new Random();

	public static int[] randomUno(int n) {
		int i, x = 0, k;
		int[] a = new int[n];
		for (i = 0; i < n; i++) {
			boolean seguirBuscando = true;
			while (seguirBuscando) {
				x = ran_int(0, n - 1);
				seguirBuscando = false;
				for (k = 0; k < i && !seguirBuscando; k++)
					if (x == a[k])
						seguirBuscando = true;
			}
			a[i] = x;
		}
		return a;
	}

	public static int[] randomDos(int n) {
		int i, x;
		int[] a = new int[n];
		boolean[] used = new boolean[n];
		for (i = 0; i < n; i++) used[i] = false;
		for (i = 0; i < n; i++) {
			x = ran_int(0, n - 1);
			while (used[x]) x = ran_int(0, n - 1);
			a[i] = x;
			used[x] = true;
		}
		return a;
	}

	public static int[] randomTres(int n) {
		int i;
		int[] a = new int[n];
		for (i = 0; i < n; i++) a[i] = i;
		for (i = 1; i < n; i++) swap(a, i, ran_int(0, i - 1));
		return a;
	}

	private static void swap(int[] a, int i, int j) {
		int aux;
		aux = a[i]; a[i] = a[j]; a[j] = aux;
	}

	/** Genera en tiempo constante, enteros entre i y j con igual probabilidad.
	 */
	private static int ran_int(int a, int b) {
		if (b < a || a < 0 || b < 0) throw new IllegalArgumentException("Parametros
invalidos");
		return a + (rand.nextInt(b - a + 1));
	}

	public static void main(String[] args) {
		System.out.println(Arrays.toString(randomUno(1000)));
		System.out.println(Arrays.toString(randomDos(1000)));
		System.out.println(Arrays.toString(randomTres(1000)));
	}
}
```

a. Determinar que algoritmos podrían no terminar nunca. Tenga presente, que para esos algoritmos, el peor de los casos, sería el que no terminen nunca.

b. Calcular el tiempo de ejecución para el / los algoritmos que sí terminan.

Ejercicio 5

Para cada uno de los algoritmos presentados:

a. Expresar en función de n el tiempo de ejecución

b. Establecer el orden de dicha función usando notación big-Oh.

En el caso de ser necesario tenga presente que:

$$\sum_{i=1}^{n} i^4 = \frac{n(n+1)\ \ (6n^3+9n^2+n-1)}{30}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)\ \ (2n+1)}{6}$$

1.

```
public static void uno (int n)  {
       int  i, j, k ;
       int [] [] a, b, c;
       a = new int [n] [n];
       b = new int [n] [n];
       c = new int [n] [n];
       for ( i=1; i<=n-1; i++) {
              for ( j=i+1; j<=n; j++) {
                     for ( k=1; k<=j; k++)  {
                            c[i][j] = c[i][j]+ a[i][j]*b[i][j];
                     }
              }
       }
}
```

2.

```
public static void dos (int n){
       int i, j, k, sum;
       sum = 0;
       for ( i=1; i<=n; i++) {
              for ( j=1; j <= i*i; j++) {
                     for ( k=1; k<= j; k++)  {
                            sum = sum + 1;
                     }
              }
       }
}
```

Ejercicio 6

Para cada uno de los algoritmos presentados calcule el T(n).

1.

```
int c = 1;
while ( c < n ) {
    algo_de_O(1);
    c = 2 * c;
}
```

2.

```
int c = n;
while ( c > 1 ) {
algo_de_O(1);
       c = c / 2;
}
```

3.

```
int x=1;
for (int i = 1; i < n; i = i+4)
       for (int j = n; j > 1; j = |j/4|)
              for (int k = 1; k < n; k = k*2)
                     x = x+1;
```

Ejercicio 7

En complejidad computacional, se dice que el orden de ejecución de un algoritmo es el orden de la cantidad de veces de la instrucción más utilizada del mismo.

a. Dado el siguiente algoritmo, determine el valor de la variable suma.

```
private int suma(int n){
       int suma=1;
       for (int i=1;i<=n;i++)
              for (int j=1;j<=i;j++)
                     suma = suma + 1;
       return suma;
}
```

b. Determine el T(n) del algoritmo previo.

c. Dado el siguiente algoritmo, determine el valor de la variable producto.

```
private int producto(int n){
       int producto=1;
       for (int i=1;i<=n;i++)
              for (int j=1;j<=i;j++)
                     producto= 2*producto
       return producto;
}
```

Ejercicio 8

Para cada uno de los siguientes fragmentos de código, determine en forma intuitiva el orden de ejecución:

`for(int i = 0; i< n; i++)` `  sum++;`	`for(int i = 0; i< n; i+=2)` `  sum++;`

`for(int i = 0; i< n; i++)`
`  for(int j = 0; j< n; j++)`
`  sum++;`
	`for(int i = 0; i< n; i++)`
	`  for(int j = 0; j< n*n; j++)`
	`  sum++;`

Ejercicio 9

a. Exprese la función del tiempo de ejecución de cada uno de los siguientes algoritmos, resuélvala y calcule el orden.

b. Comparar el tiempo de ejecución del método 'rec2' con el del método 'rec1'.

c. Implementar un algoritmo más eficiente que el del método rec3 (es decir que el T(n) sea menor).

```
package estructurasdedatos;

public class Recurrencia {
       static public int rec2(int n){
              if (n <= 1)
                     return 1;
              else
                     return (2 * rec2(n-1));
       }
```

```
        static public int rec1(int n){
                if (n <= 1)
                        return 1;
                else
                        return (rec1(n-1) + rec1(n-1));
        }

        static public int rec3(int n){
                if  ( n == 0 )
                        return  0;
                else {
                        if ( n == 1 )
                                return 1;
                        else
                                return (rec3(n-2) * rec3(n-2));
                }
        }

        static public int potencia_iter(int x, int n){
                int potencia;
                if  (n == 0)
                        potencia = 1;
                else {
                        if  (n == 1)
                                potencia  =  x;
                        else{
                                potencia  = x;
                                for (int  i  = 2 ; i <=  n ; i++) {
                                        potencia *= x ;
                                }
                        }
                }
                return  potencia;
        }

        static public int potencia_rec( int x, int n){
                if( n == 0 )
                        return 1;
                else{
                        if( n == 1)
                                return x;
                        else{
                                if ( (n  %  2 ) == 0)
                                        return  potencia_rec (x * x, n / 2 );
                                else
                                        return potencia_rec (x * x, n / 2) * x;
                        }
                }
        }
}
```

Ejercicio 10

Considerar la estrategia mergesort, que permite ordenar un arreglo recursivamente. Se divide el vector en dos partes, se ordena con mergesort cada una de ellas, y luego se combinan ambas partes ordenadas.

a. Define una clase OrdenArreglo en el paquete estructurasdedatos.utiles e implemente el método mergesort como un método de clase.

b. Hallar la función del tiempo de ejecución del algoritmo planteado.

Ejercicio 11

Dado el siguiente método, plantear y resolver la función de recurrencia:

```
int funcion(int n) {
      int x = 0;
      if (n <= 1)
            return 1;
      else {
            for (int i = 1; i < n; i++) {
                  x = 1;
                  while (x < n) {
                        x = x * 2;
                  }
            }
            return funcion(n/2) + funcion(n/2);
      }
}
```

Ejercicio 12

La siguiente clase llamada Fibonacci, contiene el cálculo del número de Fibonacci de dos maneras: recursiva e iterativa (como se vio en la teoría). El método **main**, lo calcula para n=10, e imprime el tiempo que tarda cada implementación.

Ejécutelo para los siguientes valores de n=5, 10, 15, 20, 25, 30, 35, 40, 45

```
//Fibonacci.java
public class Fibonacci {
      public static int fibRecursivo(int n) {
            if (n < 2) {
               return n;
            }
            else {
                return fibRecursivo(n-1)+fibRecursivo(n-2);
            }
      }

      public static int fibIterativo(int n) {
            int prev1=0, prev2=1;
            for(int i=0; i<n; i++) {
                int savePrev1 = prev1;
                prev1 = prev2;
                prev2 = savePrev1 + prev2;
            }
            return prev1;
      }

      public static void main(String[] args) {
         long tiempoInicio = System.currentTimeMillis();

         System.out.print(fibIterativo(10));
         long totalTiempo = System.currentTimeMillis() - tiempoInicio;
          System.out.printf("El tiempo de demora del método Iterativo es : %d  miliseg"
                            ,totalTiempo);

         tiempoInicio = System.currentTimeMillis();

         System.out.print(fibRecursivo(10));
         totalTiempo = System.currentTimeMillis() - tiempoInicio;
          System.out.printf("El tiempo de demora del método Recursivo es : %d  miliseg"
                            ,totalTiempo);

      }
}
```

Ejercicio 13

a.- Considerando que un algoritmo requiere ***f(n)*** operaciones para resolver un problema y la computadora procesa 100 operaciones por segundo.

Si *f(n)* es:

i. $\log_{10} n$
ii. $\sqrt{n}$

Determine el tiempo en segundos requerido por el algoritmo para resolver un problema de tamaño n=10000.

b.- Suponga que Ud. tiene un algoritmo ALGO-1 con un tiempo de ejecución exacto de $\mathbf{10}n^2$. ¿En cuánto se hace más lento ALGO-1 cuando el tamaño de la entrada n aumenta:…….?

i. El doble
ii. El triple

Ejercicio 14

Resolver las siguientes recurrencias

1.

$$T(n) = \begin{cases} 2, n = 1 \\ T(n-1) + n, n \geq 2 \end{cases}$$

2.

$$T(n) = \begin{cases} 2, n = 1 \\ T(n-1) + \frac{n}{2}, n \geq 2 \end{cases}$$

3.

$$T(n) = \begin{cases} 1, n = 1 \\ 4T\left(\frac{n}{2}\right) + n^2, n \geq 2 \end{cases}$$

4.

$$T(n) = \begin{cases} 1, n = 1 \\ 8T\left(\frac{n}{2}\right) + n^3, n \geq 2 \end{cases}$$

Anexo – Ejercicios Parciales

Ejercicio 1

Dado el siguiente fragmento de código:

```
public static int recu(int[] array, int count, int len) {
    if (len == 0)
        return 0;
    else
    if (array[len - 1] == count)
        return 1 + recu(array, count, len - 1);
    else
        return recu(array, count, len - 1);
}

public static void countOverlap(int[] arrayA, int[] arrayB) {
    int count = 0, calc = 0, tam = 0;
    if (arrayA.length == arrayB.length) {
        tam = arrayA.length;
        for (int i = 0; i < arrayA.length; i++)
            for (int j = 0; j < arrayB.length; j++)
                if (arrayA[i] == arrayB[j]) {
                    count++;
                    calc = calc + recu(arrayA, count, tam)
                            + recu(arrayB, count, tam);
                }
    }
    System.out.println("count:" + count + "-calc:" + calc);
}
```

a.- Calcular el T(n) para el peor caso, detallando los pasos seguidos para llegar al resultado.

b.- Calcular el O(n) de la función del punto anterior justificando usando la definición de big-OH.

Ejercicio 2

Dada la siguiente recurrencia:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{4}\right) + \sqrt{n}, & n \geq 2 \end{cases}$$

a.- Calcular analíticamente el T(n), detallando los pasos seguidos para llegar al resultado.

b.- Calcular el O(n) justificando usando la definición de big-OH

Ejercicio 3

Dado el siguiente algoritmo:

```
public static void ejercicio1 (int n){
      int i, j, k, sum;
      sum = 0;
      for (i=n; i>1; i--) {
        for ( j=1; j < i*i; j++) {
          for ( k=i; k<j; k+=2)
                  sum = sum + 1;
        }
      }
    }
```

a.- Calcular el T(N).

b.- Calcular el O(N) justificando por definición.

Ejercicio 4

Dado el siguiente algoritmo,

```
for ( i=1; i<=n; i++) {
       for ( j=i^2; j>=1; j--) {
              for ( k=i; k<=j; k++)  {
                     ** operacion de orden 1**;
                     }
              }
       }
}
```

Pistas:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4} \qquad \sum_{i=1}^{n} i^4 = \frac{n(n+1)(6n^3+9n^2+n-1)}{30}$$

a.- Calcular el T(n)

b.- Calcular el O(n) justificando debidamente.

Ejercicio 5

Dado el siguiente segmento de código,

```
j = 1;
while (j <= n) {
    for (i = n*n; i >=1; i = i-3)
          x=x+1;
    j = j*2;
}
```

a.- Calcular analíticamente el T(n), detalle los pasos seguidos para llegar al resultado.

b.- Calcular el O(n) justificando usando la definición de big-OH

Ejercicio 6

Dada la siguiente recurrencia,

$$T(n) = \begin{cases} (4/5)C\log_5(n) & n = 1 \\ (Cn)/5 + 5T(n/5) & n > 1 \end{cases}$$

a.- Calcular el T(n) resolviendo la recurrencia, detallando los pasos seguidos para llegar al resultado.

b.- Calcular el O(n) justificando usando la definición de big-OH.

Ejercicio 7

Dada el siguiente algoritmo,

a.- Calcular su T(n), detallando los pasos seguidos para llegar al resultado.

b.- Calcular su O(n) justificando usando la definición de big-OH.

```
public static void uno(int n) {
    for (int i = 1;  i <= n;  i ++)
         for (int j = 1;  j<= i;  j++)
              algo_de_O(1);
    if (n>1)
          for (int i = 1;  i <= 4;  i++)
               uno(n div 2);

}
```

Ejercicio 8

Dada el siguiente algoritmo,

a.- Calcular su T(n), detallando los pasos seguidos para llegar al resultado.

b.- Calcular su O(n) justificando usando la definición de big-OH.

```
public static void main(String[] args) {
    int sum = 0, sum1 = 0, sum2 = 0;
    for (double i = 1; i <= n; i+=0.5) {
       ++sum1;
       for (double j = 1; j <= sum1; j++) {
           sum2++;
           for (double k = j; k <= sum1; k++)) {
               sum += sum1 + sum2;
           }
       }
    }
}
```

Ejercicio 9

```
static public int recursivo(int n){
       if (n == 1)
             return  1;
       else
             return (n * recursivo (n-1));
}
```

a) Calcule el Tiempo de ejecución
b) Determine el Orden por definición
c) Indique que valor retorna el algoritmo y compárelo con a.