Doble Grado en Ingeniería Informática y Matemáticas
Relaciones de Estructuras de Datos. Eficiencia

J. Fdez-Valdivia

Departamento de Ciencias de la Computación e I.A. ETS Ingeniería Informática
Universidad de Granada. 18071 Granada. Spain.
Email: jfv@decsai.ugr.es

Resumen

En este documento se presenta relación de problemas de análisis de eficiencia que se propone para la asignatura de estructura de datos de segundo curso del Doble Grado en *Ingeniería Informática y Matemáticas* .

1. Análisis de la eficiencia

Problema 1.1 Probar que las siguientes sentencias son verdad:

- 17 **es** $O(1)$
- $\frac{n(n-1)}{2}$ **es** $O(n^2)$ **y** $\Omega(n^2)$ **(es decir,** $\Theta(n^2)$**).**
- $max(n^3, 10n^2)$ **es** $O(n^3)$
- $log_2 n$ **es** $\Theta(log_3 n)$

Problema 1.2 Encontrar el entero k **más pequeño tal que** $f(n)$ **es** $O(n^k)$ **en los siguientes casos:**

1. $f(n) = 13n^2 + 4n - 73$
2. $f(n) = 1/(n+1)$
3. $f(n) = 1/(n-1)$
4. $f(n) = (n-1)^3$
5. $f(n) = (n^3 + 2n - 1)/(n+1)$
6. $f(n) = \sqrt{n^2 - 1}$

Problema 1.3 Ordenar de menor a mayor los siguientes órdenes de eficiencia:
n, $\sqrt{n}$, $n^3 + 1$, n^2, $nlog_2(n^2)$, $nlog_2log_2(n^2)$, $3^{log_2(n)}$, 3^n, 2^n, $2^n + 3^{n-1}$, 20000, $n + 100$, $n2^n$

Problema 1.4 Supongamos que $T_1(n) \in O(f(n))$ **y** $T_2(n) \in O(f(n))$**. Razonar la verdad o falsedad de las siguientes afirmaciones:**

a.- $T_1(n) + T_2(n) \in O(f(n))$

b.- $T_1(n) \in O(f^2(n))$

c.- $T_1(n)/T_2(n) \in O(1)$

Problema 1.5 Considerar las siguientes funciones de n:

$f_1(n) = n^2$

$f_2(n) = n^2 + 1000n$

$$f_3(n) = \begin{cases} n & si\ n\ es\ impar \\ n^3 & si\ n\ es\ par \end{cases}$$

$$f_4(n) = \begin{cases} n & si\ n \leq 100 \\ n^3 & si\ n > 100 \end{cases}$$

Indicar para cada par distinto i y j si $f_i(n)$ es $O(f_j(n))$ y si $f_i(n)$ es $\Omega(f_j(n))$.

Problema 1.6 Demostrar que si $f(n) \in O(g(n))$ y $g(n) \in O(h(n))$ entonces $f(n) \in O(h(n))$

Problema 1.7 Demostrar que $O(f(n)) \in O(g(n))$ sii $f(n) \in O(g(n))$ y $g(n) \in O(f(n))$.

Problema 1.8 Demostrar la siguiente jerarquía de órdenes de complejidad:

$$O(1) \subset O(log(n)) \subset O(n) \subset O(n^2) \subset O(2^n) \subset O(n!)$$

Problema 1.9 *Obtener usando la notación O-mayúscula la eficiencia del siguiente trozo de código:*

```
for (i=0;i<n;i++)
   for (j=0;j<n;j++) {
      C[i][j]= 0;
      for (k=0;k<n;k++)
         C[i][j]+=A[i][k]*B[k][j];
   }
```

Problema 1.10 Obtener usando la notación O-mayúscula la eficiencia de la siguiente función:

```
void ejemplo(int n)
{
   int i, j, k;

   for (i = 1; i < n; i++)
      for (j = i+1; j <= n; j++)
         for (k = 1; k <= j; k++)
            Global += k*i;
}
```

Problema 1.11 Obtener usando la notación O-mayúscula la eficiencia del siguiente trozo de código:

```
for (i=0;i<n;i++)
   if (i%2) {
      for (j=i;j<n;j++)
         x* =i;
      for (j=1;j<i;j++)
         y* =j;
   }
```

Problema 1.12 Suponer que el parámetro n en la siguiente función es una potencia positiva de 2, es decir, $n = 2, 4, 8, 16, \ldots$ Dar la fórmula que expresa el valor de la variable *cont* en

términos del valor de n cuando la función termina.

```
void ejemplo(int n)
{
  int x, cont;

  cont = 0;
  x = 2;
  while (x < n)
  {
    x = 2*x;
    cont++;
  };
  printf(''%d\n'', cont);
}
```

Problema 1.13 Estimar el peor caso de tiempo de ejecución para la siguiente función. Emplear la notación O-mayúscula

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

Problema 1.14 Dada la siguiente función:

```
int E(int n)
{
  if (n == 1)
    return n;
  else
    return (E(n/2) + 1);
}
```

1. **¿Cuál es el valor que devuelve la función?**
2. **Obtener una expresión para el peor caso de tiempo de ejecución de la función.**

Problema 1.15 Resolver la recurrencia siguiente en función de k y s:

$$T(n) = k \times T(n-1) + s^2 \quad n > 1 \quad k, s > 1 \quad T(1) = 1$$

Problema 1.16 Considerando que el máximo de un vector es el máximo de dos valores:

1. **El máximo de la primera mitad.**
2. **El máximo de la segunda mitad.**

Implementar una función recursiva para calcular el máximo de un vector y estudiar su eficiencia.

Problema 1.17 Resolver la recurrencia:

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

Problema 1.18 Implementar una función recursiva que resuelva el problema de las *Torres de Hanoi* y estudiar la eficiencia en función de la altura de la torre.

Problema 1.19 Suponiendo que la función *rectangle* tiene un tiempo de ejecución constante, calcular la eficiencia de la siguiente función fractal en función de "n", es decir, del número de niveles de profundidad.

```
/**
* @brief Dibuja un fractal usando la función rectange
* @param n Número de niveles del fractal
* @param cx Coordenada x del centro del rectángulo
* @param cy Corrdenada y del centro del rectángulo
* @param t Longitud del lado del cuadrado
*/
void fractal (int n, int cx, int cy, int t)
{
  if (n>0) {
    rectangle(cx-t/2,cy-t/2,cx+t/2,cy+t/2);
    fractal (n-1,cx-t/2,cy-t/2,t/4);
    fractal (n-1,cx+t/2,cy+t/2,t/4);
    fractal (n-1,cx-t/2,cy+t/2,t/4);
    fractal (n-1,cx+t/2,cy-t/2,t/4);
  }
}
```