Universidad de Valladolid
Departamento de Informática

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

Grado en Ingeniería Informática / Grado en Estadística

Estructuras de Datos y Algoritmos

Hoja de Problemas – Curso 20/21

Análisis de Algoritmos

1. Un programador ha implementado un algoritmo que tarda, en su antiguo ordenador de 100 Mhz, una milésima de segundo en resolver un problema de tamaño $n = 20$. ¿Cuánto tardaría en resolver un problema de tamaño $n = 30$ en los siguientes casos?

   a) El algoritmo es de orden $O(n^3)$

   b) El algoritmo es de orden $O(2^n)$

   c) El algoritmo es de orden $O(n!)$

   ¿Por qué factor debería multiplicar la velocidad de su ordenador para que volviera a tardar una milésima de segundo? **Nota**: 1 año = 31.557.600 segundos.

2. ¿Es posible que un algoritmo de orden $O(n^2)$ se ejecute más rápidamente que un algoritmo de orden $O(n)$ en los siguientes casos?

   a) Para un tamaño $n = 100$.

   b) Para un determinado valor de la entrada.

   c) Para el mejor caso del primer algoritmo.

   d) Para todos los valores de la entrada.

   e) Si se ejecutan en máquinas distintas y podemos elegir aquella en que se ejecuta el primer algoritmo.

3. Al medir experimentalmente un algoritmo se encuentra que nunca emplea más de $k_1 \cdot n^3$ operaciones ni menos de $k_2 \cdot n^2$, siendo n el tamaño de los datos de entrada y k_1 y k_2 constantes. ¿Cuál sería la forma más adecuada de informar sobre estos datos usando notación O-grande?

   - El algoritmo es $O(n^2)$
   - El algoritmo es $O(n^3)$
   - El algoritmo es $O(n^{2.5})$
   - El algoritmo es $O(n^2 + n^3)$

4. Analizar cuál sería el orden de complejidad del algoritmo que usamos habitualmente para multiplicar dos números enteros. Se supondrá que los dos números tienen el mismo número de dígitos, n, y que este valor define el tamaño de la entrada. Contar únicamente las operaciones de producto y suma de dígitos individuales.

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

5. Analizar el número de operaciones en las que interviene un elemento del vector en el siguiente algoritmo (ordenación por selección):

```
static void ordenar(double[] v) {
  int n = v.length;
  double tmp;
  for(int i = 0; i < n-1; i++) {
    int min := i;
    for(int j = i+1; i < n; j++) {
      if(v[j] < v[min]) { min = j; }
    }
    tmp = v[i];
    v[i] = v[min];
    v[min] = tmp;
  }
}
```

Encontrar las fórmulas exactas para el peor y el mejor caso, identificar cuáles son los tipos de entradas que producen esos casos y repetir el análisis utilizando notación asintótica.

6. Calcular el número de operaciones de incremento de *x* que realizan los siguientes fragmentos de programas, teniendo en cuenta que sólo deseamos conocer exactamente el término de mayor crecimiento, y el resto de términos se pueden expresar usando notación asintótica:

```
for(int i = 0; i < n; i++) {
  for(int j = 0; j < n; j++) {
    if(j = i) { x++; }
  }
}
```

```
for(int i = 0; i < n; i++) {
  for(int j = n; j > i; j--) {
    for(int k = i; k < j; k++) {
      x++;
    }
  }
}
```

```
for(int i = 0; i < n; i++) {
  for(int j = 0; j < i*i; j++) {
    for(int k = 0; k < j*j; k++) {
      x++;
    }
  }
}
```

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

7. Calcular el orden de complejidad de las siguientes funciones recursivas (contar sólo las operaciones producto):

```
public int f1(int n) {
  if(n < 1) { return 1; } else
    { return f1(n-3)*f1(n-3); }
}
```

```
public int f2(int n) {
  if(n < 1) { return 1; } else
    { return f2(n/2) + f2(n/2)*f2(n/2); }
}
```

```
public int f3(int n) {
  if(n < 1) { return 1; }
  else {
     int x = 1;
     for(int i = 0; i < n; i++) { x = x*n; }
     return x + f3(n-1);
  }
}
```

```
public int f4(int n) {
  if(n < 1) { return 1; }
  else {
     int x = 1;
     for(int i = 0; i < n*n; i++) { x = 2*x; }
     return x*f4(n/2);
  }
}
```

8. Demostrar si los siguientes enunciados son ciertos o falsos:
   a) ¿$\log 3^n \in O(n)$?
   b) ¿$2^{n+1} \in O(2^n)$?
   c) ¿$3^n \in O(2^n)$?
   d) ¿$2n^2 - n + 1 \in O(n)$?
   e) ¿$\log(n!) \in O(n \log n)$?

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

9. Calcular el orden de complejidad respecto al tiempo (suponiendo que deseamos contar las operaciones suma) y al espacio de la siguiente función:

```
public int f(int n) {
  if(n < 1) { return 1; }
  else {
     int x = n;
     for(int i = 0; i < n; i++) { x++; }
     return f(n/2)*f(n/2) + f(n/3) + x;
  }
}
```

10. Al analizar la eficiencia de una operación de inserción sobre una estructura de datos se encuentra que a veces la inserción tarda un tiempo $O(n^3)$ pero eso implica que las siguientes n inserciones tardarán un tiempo de $O(1)$. Analizar el comportamiento del algoritmo en el peor y mejor caso, el caso promedio y el tiempo amortizado.

11. El problema de evaluar la expresión a^b donde a es un valor real y b un entero positivo se puede resolver mediante un bucle donde se evalúe b veces el producto de a, o bien es posible aplicar la siguiente fórmula en un esquema divide y vencerás:

$$a^b = \begin{cases} \left(a^{b \text{ div } 2}\right)^2 & \text{si } b \text{ es par} \\ a \cdot \left(a^{b \text{ div } 2}\right)^2 & \text{si } b \text{ es impar} \end{cases}$$

Desarrollar algoritmos para ambas soluciones y evaluar el orden de complejidad de cada uno tomando como tamaño de la entrada el número de bits de b.

12. El problema del **producto de enteros grandes** consiste en calcular el producto de enteros de precisión arbitraria (representados como arrays de n bytes). El número que representa el producto consistirá en un array de cómo mucho $2n$ bytes. Este resultado se debe calcular usando como operación elemental el producto de bytes individuales.

- Crear un esquema de algoritmo, basado en el método clásico de multiplicación, que resuelva el problema anterior y evaluar su eficiencia.
- El algoritmo "divide y vencerás" de multiplicación de enteros se basa en la idea siguiente: Supongamos que los dos valores a multiplicar, x e y, se representan por el mismo número de bits, n, y que este valor se puede dividir por 2, $n = 2 \cdot m$. Se puede dividir cada número, representado como una secuencia de bytes, en dos "mitades" de la siguiente forma:

$$x = x_1 \cdot 256^m + x_0$$
$$y = y_1 \cdot 256^m + y_0$$

El producto de x por y se calcula como:

$$x \times y = 256^n \cdot x_1 \times y_1 + 256^m \cdot (x_1 \times y_0 + x_0 \times y_1) + x_0 \times y_0$$

Es decir, consiste en 4 productos de números de tamaño (número de bytes) mitad que el original (los productos por potencias de 2 consisten en realidad en desplazamientos de los dígitos en el array que representa los números). Evaluar el tiempo que requiere una evaluación recursiva según el esquema anterior.

- El algoritmo de Karatsuba da una vuelta de tuerca al método anterior y propone calcular recursivamente el producto usando la siguiente fórmula:

$$x \times y = 256^n \cdot x_1 \times y_1 + 256^m \cdot ((x_1 + x_0) \times (y_1 + y_0) - x_1 \times y_1 - x_0 \times y_0) + x_0 \times y_0$$

Evaluar el algoritmo y comprobar si supone una mejora respecto al método anterior.

13. Supongamos que tenemos una función que escribe números enteros mayores que cero por pantalla usando el siguiente algoritmo:

```
static char[] Digitos = ['0','1','2','3','4','5','6','7','8','9'];

public void escribe_numero(int x) {
  if(x > 0) {
     System.out.print(Digitos[x % 10]);
     escribe_numero(x / 10);
  }
}
```

Si analizamos el orden de este algoritmo vemos que el número de operaciones de resta y división que se realizan es proporcional al logaritmo en base 10 de x (el número de veces que se puede dividir x por 10 hasta que valga cero). Ya que $\log_{10} x = 3.32 \cdot \log_2 x$ obtenemos que el tiempo es proporcional al número de bits de x.

Supongamos que estamos trabajando con enteros de precisión arbitraria, y por lo tanto ahora x se representa como un array de n bits. En este caso las operaciones de resta y cociente no son elementales y supondremos que tienen el siguiente coste:

- x % 10, x / 10 : $O(n)$
- x % $10^{n/2}$, x / $10^{n/2}$: $O(n^{3/2})$

Analizar el orden del algoritmo anterior en esta nueva situación, expresándolo en función de n (número de bits de x). Crear un algoritmo *divide y vencerás* para este mismo problema y analizar su orden.

Universidad de Valladolid
Departamento de Informática

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

Algoritmos de búsqueda y ordenación

14. El algoritmo de búsqueda ternaria es similar a la búsqueda binaria pero se diferencia en que en cada etapa el subvector se divide en **tres** partes iguales (en lugar de dos) y se examinan los dos elementos que separan estas tres partes para decidir si se ha encontrado el valor o, en caso contrario, elegir una de estas tres partes para continuar la búsqueda en ella.

    Analizar cuál sería la complejidad de este algoritmo para las operaciones en que interviene un elemento del vector (se puede suponer que el tamaño del vector es una potencia de tres) y comprobar si mejora o no el orden de la búsqueda binaria.

15. Dado un vector ordenado de enteros, $v_1...v_n$, donde no existen elementos repetidos, crear un algoritmo que determine si existe algún elemento igual a su índice (es decir, si $v_i = i$) en un tiempo mejor que $O(n)$.

16. Se dispone de una matriz cuadrada, M, de n x n elementos con la propiedad de que cada fila y cada columna de la matriz están ordenadas de menor a mayor (es decir, $\forall i, j : \mathrm{M}[i, j] < \mathrm{M}[i+1, j]$ y $\forall i, j : \mathrm{M}[i, j] < \mathrm{M}[i, j+1]$).

    Diseñar un algoritmo que permita detectar si la matriz contiene un determinado valor, x. El objetivo es conseguir un algoritmo lo más eficiente posible.

17. Se dispone de un vector que contiene n elementos, todos ellos iguales. ¿Cuál sería la eficiencia del algoritmo de ordenación rápida aplicado a este vector?

18. Encontrar el orden de los enteros 1,2,3,4,5,6,7,8,9 que provoca el peor caso para el algoritmo de ordenación rápida. ¿Y si modificamos el algoritmo de forma que se escoja como pivote el elemento medio (intercambiándole con el primero antes de efectuar la partición)?

19. Se desea ordenar en un tiempo de $O(n)$ una estructura de datos que almacena n elementos representando uno de tres colores posibles (rojo, blanco o azul). Los colores pueden compararse entre sí, siendo rojo < blanco < azul.

    - Crear un algoritmo que resuelva el problema suponiendo que la estructura es un vector.
    - Crear un algoritmo que resuelva el problema suponiendo que la estructura es un TAD donde las únicas operaciones posibles son **color(i)**, que devuelve el color i-ésimo e **intercambia(i,j)**, que intercambia los colores i y j.

20. Dado un vector A[1..n] de enteros (todos distintos) y un valor entero x, diseñar un algoritmo que imprima **todos** los pares (i,j) que cumplan que $i < j$ y A[i]*A[j] = x en un tiempo $O(n \log n)$.