Ejemplos de análisis de algoritmos

Abraham Sánchez López
Grupo MOVIS
FCC/BUAP

Preguntas I

- ¿Cuál es la recurrencia para el peor de los casos de Quicksort y cuál es la complejidad del tiempo en el peor de los casos?
  a. La recurrencia es $T(n) = T(n-2) + O(n)$ y la complejidad del tiempo es $O(n^2)$.
  b. La recurrencia es $T(n) = T(n-1) + O(n)$ y la complejidad del tiempo es $O(n2)$.
  c. La recurrencia es $T(n) = 2T(n/2) + O(n)$ y la complejidad del tiempo es $O(n \log n)$.
  d. La recurrencia es $T(n) = T(n/10) + T(9n/10) + O(n)$ y la complejidad del tiempo es $O(n \log n)$.

- Supongamos que tenemos un algoritmo $O(n)$ que encuentra la media de un arreglo desordenado. Ahora considera una implementación de Quicksort, donde primero encontramos la media con el algoritmo anterior, luego usamos la media como pivote. ¿Cuál será la complejidad del peor caso de este Quicksort modificado?
  a. $O(n^2 \log n)$
  b. $O(n^2)$
  c. $O(n \log n \log n)$
  d. $O(n \log n)$

Preguntas II

- ¿Cuál de las siguientes afirmaciones no es cierta sobre los algoritmos de ordenamiento basados en comparación?
  a. La complejidad mínima de tiempo posible de un algoritmo de ordenamiento basado en comparación es O(n log n) para un arreglo de entrada aleatoria.
  b. Cualquier algoritmo de ordenamiento basado en comparación se puede implementar de forma estable utilizando la posición como un criterio cuando se comparan dos elementos.
  c. La ordenación por conteo (counting sort) no es un algoritmo de ordenamiento basado en comparación.
  d. El heap sort no es un algoritmo de ordenamiento basado en comparación.
- Cual es el tiempo de complejidad de fun()?
  a. $\Theta(n)$
  b. $\Theta(n2)$
  c. $\Theta(n \log n)$
  d. $\Theta(n \log n \log n)$

```
int fun(int n)
{
  int count = 0;
  for (int i = 0; i < n; i++)
     for (int j = i; j > 0; j--)
        count = count + 1;
  return count;
}
```

Preguntas III

- Cual es la complejidad en tiempo de la siguiente función?
  a. $O(n)$
  b. $O(n^2)$
  c. $O(n \log n)$
  d. $O(n (\log n)^2)$

```
void fun(int n, int arr[])
{
    int i = 0, j = 0;
    for(; i < n; ++i)
        while(j < n && arr[i] < arr[j])
            j++;
}
```

- En una competencia, se observan cuatro funciones diferentes. Todas las funciones usan un solo ciclo for y dentro del ciclo for, se ejecutan el mismo conjunto de declaraciones. Considera los siguientes ciclos for:
- Si *n* es el tamaño de la entrada (positiva), ¿qué función es más eficiente (si la tarea a realizar no es un problema)?
  a. A
  b. B
  c. C
  d. D

```
A) for(i = 0; i < n; i++)

B) for(i = 0; i < n; i += 2)

C) for(i = 1; i < n; i *= 2)

D) for(i = n; i > -1; i /= 2)
```

Preguntas IV

- ¿Qué significa cuando decimos que un algoritmo *X* es asintóticamente más eficiente que *Y*?
  a. X será una mejor opción para todas las entradas.
  b. X será una mejor opción para todas las entradas, excepto posiblemente pequeñas entradas.
  c. X será una mejor opción para todas las entradas, excepto posiblemente entradas grandes.
  d. Y será una mejor opción para pequeñas entradas.
- En la siguiente función, sea $n \geq m$. ¿Cuántas llamadas recursivas se hacen por esta función?
  a. $\Theta(\log n)$
  b. $\Omega(n)$
  c. $\Theta(\log \log n)$
  d. $\Theta(\sqrt{n})$

```
int gcd(n,m)
{
  if (n%m ==0) return m;
  n = n%m;
  return gcd(m, n);
}
```

Ejemplo 1, I

- Qué valor devuelve la siguiente función? Proporcionar la respuesta como función en términos de *n* y calcular su complejidad.

```
function valor(n : int) return r : int
var i, j, k: int
   r = 0
   for  i = 1 to n -1
      for  j = i + 1 to n
         for  k = 1 to j
            r = r + 1
         end
      end
   end
```

- Si escribimos las sumas sucesiva que realizan los tres ciclos anidados, como fórmula matemática, obtenemos:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{j} 1$$

Ejemplo 1, II

- Para conocer el valor explícito de estas sumas en función de *n*, se procede desde la sumatoria más interna, simplificando paso a paso cada una de las tres sumas.
- Primeramente, $j \geq 1 + 1 \geq 1 + 1 = 2$ y es obvio que:

$$\sum_{k=1}^{j} 1 = j$$

- A continuación, dado que $1 \leq i \leq n - 1$, el rango de *j* entre $i + 1$ y *n* no es vacío, y si recordamos la fórmula de la sumatoria

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

- Nos permite calcular

$$\sum_{j=i+1}^{n} \sum_{k=1}^{j} 1 = \sum_{j=i+1}^{n} j = \sum_{j=1}^{n} j - \sum_{j=1}^{i} j = \frac{n(n+1)}{2} - \frac{i(i+1)}{2}$$

Ejemplo 1, III

- Y finalmente, recordemos otra de las fórmulas de las sumatorias:

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

- Para poder calcular, $n \geq 2$

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j} 1 = \sum_{i=1}^{n-1}\left(\frac{n(n+1)}{2} - \frac{i(i+1)}{2}\right) = (n-1)\frac{n(n+1)}{2} - \sum_{i=1}^{n-1}\frac{i(i+1)}{2} =$$

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

$$= \frac{(n-1)n(n+1)}{2} - \frac{1}{2}\frac{(n-1)n(2n-1)}{6} - \frac{1}{2}\frac{(n-1)n}{2} = \frac{(n-1)n}{2}\left(n+1-\frac{2n-1}{6}-\frac{1}{2}\right) =$$

$$= \frac{(n-1)n}{2}\frac{4(n+1)}{6} = \frac{(n-1)n(n+1)}{3} = \frac{n(n^2-1)}{3} \in \Theta(n^3).$$

Ejemplo 1, IV

- Obtener *O y* Ω *para el algoritmo de la multiplicación de matrices cuadradas.*

```
for i= 1 to n
  for j=1 to n
     suma=0
     for k=1 to n
        suma=suma+a[i,k]b[k,j]
     end
     c[i,j]=suma
  end
end
```

- Podemos observar que siempre se ejecutan las mismas instrucciones, independientemente de la entrada ➔ O(*f*) = Ω (*f*) = Θ(*f*).

Ejemplo 2, II

$$t(n) = c_1 + \sum_{i=i}^{n} \left(c_2 + \sum_{j=1}^{n} \left(c_3 + \sum_{k=1}^{n} c_4 \right) \right) =$$

$$c_1 + \sum_{i=i}^{n} \left(c_2 + \sum_{j=1}^{n} \left(c_3 + c_4 n \right) \right) =$$

$$c_1 + \sum_{i=i}^{n} \left(c_2 + c_3 n + c_4 n^2 \right) = c_1 + c_2 n + c_3 n^2 + c_4 n^3$$

$$t \in \Theta(n^3)$$

Ejemplo 3

- Dado el siguiente código:

```
cont=0
for i=1 to n
   for j=1 to i-1
      if a[i] < a[j]
         cont=cont+1
      endif
   endfor
endfor
```

$$t(n) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} \frac{1}{2} = \sum_{i=2}^{n} \frac{i-1}{2} = \frac{n^2 - n}{4}$$

$$t(n) \in \Theta(n^2)$$

- Calcular el orden exacto del número promedio de veces que se ejecuta la instrucción cont=cont+1.

- Como los valores del arreglo *a* no se modifican a lo largo de la ejecución del programa, se puede considerar que estos valore están uniformemente distribuidos, y que en cada comparación, la probabilidad de que sea verdad es de ½.

Ejemplo 4, I

- Obtener O, Ω y Θ del siguiente algoritmo.

```
i=0
a[n+1]=x
repeat
   i=i+1
until a[i]=x
```

- Las dos primeras asignaciones se ejecutan siempre, también se ejecuta siempre una vez el cuerpo del ciclo y la comprobación final.
- En el mejor caso (que es cuando $a[1]=x$) se ejecutan 4 instrucciones que consideramos elementales.
- En el peor caso (cuando x no está en el arreglo), tanto el cuerpo del ciclo como la comprobación se ejecutan $n+1$ veces, por lo que se ejecutan $2n+4$ instrucciones.
- En el caso promedio, si consideramos que x está en el arreglo con probabilidad p(y por lo tanto que no está con probabilidad $1-p$).

Ejemplo 4, II

- Si suponemos que *x* está en cualquiera de los *n* lugares del arreglo, tendremos una probabilidad de *p*/n de que esté en el lugar *i* con *i* entre 1 y *n*.
- Si está en el lugar *i*, el número de operaciones sería 2*i*+2 y por lo tanto, tendríamos que el número promedio de operaciones es:

$$\sum_{i=1}^{n}\left(\frac{p}{n}(2+2i)\right)+(4+2n)(1-p)=$$

$$(3+n)p+(4+2n)(1-p)=$$

$$3p+np+4+2n-4p-2np=$$

$$2n+4-(n+1)p$$

Ejemplo 5, I

- Calcular el tiempo de ejecución y el orden exacto del siguiente algoritmo:

```
p=0
for i=1 to n
   p=p+i*i
   for j=1 to p
     escribir(a[p,j])
   endfor
endfor
```

- Sería interesante revisar como varían las variables i, j y p.

Con i=1, p=1, j varía entre 1 y 1

Con i=2, p=$1+2^2$, j varía entre 1 y $1+2^2$

Con i=3, j varía entre 1 y $1+2^2+3^2$

En general, para un cierto i, j varía entre 1 y $1+2^2+3^2+\ldots+i^2$

Ejemplo 5, II

- Si suponemos que todas las operaciones elementales consumen un tiempo unitario.
- *j* toma el valor 1 *n* veces, los siguientes valores, del 2 al $1+2^2$, *n*-1 veces, los siguientes valores *n*-2 veces, …
- De esta manera, el número de operaciones elementales sería:

$$n+(n-1)2^2+(n-2)3^2+\ldots+(n-(n-1))n^2=$$

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

$$\approx n\int_0^n(i^2+2i+1)di-\int_0^n(i^3+2i^2+i)di=$$

$$=\frac{n^4}{12}+\frac{n^3}{3}+\frac{n^2}{2}$$

El orden exacto es $\Theta(n^4)$

Solución ecuación de recurrencia

- Consideremos la siguiente ecuación en recurrencia:

$$t(n) - 3t(n\text{-}1) - 4t(n\text{-}2) = 0$$

cuando $n > 2$, con condiciones iniciales $t(0) = 0$ y $t(1) = 1$.

- La ecuación característica es $x^2 - 3x - 4 = 0$, que tiene las soluciones -1 y 4, por lo que tenemos:

$$t(n) = c_1(-1)^n + c_2\, 4^n$$

- Considerando las condiciones iniciales:

$$c_1 + c_2 = 0$$

$$-\, c_1 + 4\, c_2 = 1$$

- Donde $c_1 = -1/5$ y $c_2 = 1/5$, para que finalmente la solución sea

$$t(n) = 1/5(4^n - (-1)^n)$$

Ejemplo de ecuación en recurrencia, I

- Resolver

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

$$\begin{aligned} T(n) &= 2T(n-1) + 2n - 1 = 2(2T(n-2) + 2(n-1) - 1) + 2n - 1 \\ &= 2^2 T(n-2) + 2^n (n-1) - 2 + 2n - 1 \\ &= 2^2 (2T(n-3) + 2(n-2) - 1) + 2^n (n-1) - 2 + 2n - 1 \\ &= 2^3 T(n-3) + 2^3 (n-2) - 2^2 (n-1) - 2 + 2n - 1 \\ &= 2^3 (2T(n-4) + 2(n-3) - 1) + 2^3 (n-2) - 2^2 + 2^2 (n-1) - 2 + 2n - 1 \\ &= 2^4 T(n-4) + 2^4 (n-3) - 2^3 + 2^3 (n-2) - 2^2 + 2^2 (n-1) - 2 + 2n - 1 \end{aligned}$$

Ejemplo de ecuación en recurrencia, II

$$= 2^4 T(n-4) + \sum_{j=0}^{3} 2^{j+1}(n-j) - \sum_{j=0}^{3} 2^j$$

- Caso *i*

$$= 2^i T(n-i) + \sum_{j=0}^{i-1} 2^{j+1}(n-j) - \sum_{j=0}^{i-1} 2^j$$

$$= 2^i T(n-i) + (2n-1)\sum_{j=0}^{i-1} 2^j - 2\sum_{j=0}^{i-1} j2^j$$

- El caso base es cuando $n - i = 1, \Rightarrow i = n - 1$

$$T(n) = 2^{n-1} + (2n-1)\sum_{j=0}^{n-2} 2^j - 2\sum_{j=0}^{n-2} j2^j$$

- Se pueden resolver las dos sumatorias, con la fórmula telescópica.

Ejemplo de ecuación en recurrencia, III

$$\sum_{j=0}^{n-2} 2^j = 2^{n-1} - 1 \qquad \sum_{j=0}^{n-2} j2^j = (n-3)2^{n-1} + 2$$

$$T(n) = 2^{n-1} + (2n-1)(2^{n-1} - 1) - 2((n-3)2^{n-1} + 2)$$

$$= (1 + 2n - 1 - 2n + 6)2^{n-1} - (2n-1) - 4$$

$$= 3 \cdot 2^n - 2n - 3$$

$$T(n) \in \Theta(2^n)$$

Ejemplo 2 de ec. en recurrencia, I

- Resolver

$$T(n) = 4T(n/2) + n^2, \text{ si } n > 4, \ n \text{ potencia de } 2$$

$$T(1) = 1 \text{ y } T(2) = 8$$

$$n = 2^k \ (k = \log n)$$

$$\Rightarrow T(2^k) = 4T(2^{k-1}) + 2^{2k}$$

- Sea

$$t_k = T(2^k),$$

$$t_k = 4t_{k-1} + 4^k, \ t_k - 4t_{k-1} = 4^k$$

ec. no homogénea, su ec. característica es:

$$(x-4)^2 = 0$$

$$t_k = c_1 4^k + c_2 k 4^k$$

$$T(2^k) = c_1 4^k + c_2 k 4^k = c_1 2^{2^k} + c_2 k 2^{2^k}$$

Ejemplo 2 de ec. en recurrencia, II

- Como

$$n = 2^k,$$

$$\Rightarrow T(n) = c_1 n^2 + c_2 n^2 \log n$$

- Se debe resolver el sistema de ecuaciones, utilizando los valores base de la ecuación en recurrencia, para encontrar los valores de c_1 y c_2.

$$\left.\begin{array}{l} c_1(1)^2 + c_2(1)^2 \log 1 = 1 \\ c_1(2)^2 + c_2(2)^2 \log 2 = 8 \end{array}\right\} \begin{array}{l} c_1 = 1 \\ c_2 = 1 \end{array} \Rightarrow$$

$$T(n) = n^2 + n^2 \log n$$

$$T(n) \in \Theta(n^2 \log n)$$

Ejemplo 6, I

- Calcular el orden exacto de la siguiente función:

```
funcion1(n:integer):integer
Inicio
   if n=1
      return 1
   else
      return (2*funcion1(n-1)+1)
   end
Fin
```

- En el caso base, n=1 se ejecuta la comprobación del if y el return. Por lo que el tiempo es constante, lo llamamos a.
- Cuando no estamos en el caso base, se hace una llamada con n-1 y el resto de las operaciones toman un tiempo constante b.
- La ecuación en recurrencia sería:

$$t(n) = \begin{cases} a & \text{si } n = 1 \\ b + t(n-1) & \text{si } n > 1 \end{cases}$$

Ejemplo 6, II

- Es necesario resolver la ecuación en recurrencia, en este caso utilizamos el método iterativo:

$$t(n) = b + t(n-1) = 2b + t(n-2) = \ldots = ib + t(n-i)$$

- Para el caso base, $n - i = 1 \Rightarrow i = n - 1$

$$\therefore\ t(n) = (n-1)b + a = bn + a - b$$

- Por lo tanto

$$t(n) \in \Theta(n)$$

Ejemplo 7, I

- ¿Cuántas multiplicaciones realizan los siguientes algoritmos para calcular el factorial?

a) function Factorial1(x:nat): f:nat

```
var
    y:nat
Inicio
    y=x; f=1;
    While y > 1 do
        f=f*y; y=y-1;
    fin
Fin
```

b) function Factorial2(x:nat):f:nat

```
Inicio
    if x ≤ 1 entonces f=1
      else f=x*Factorial2(x-1)
    fin
```

- En el caso iterativo, el algoritmo realiza una multiplicación (f*y) en cada iteración del ciclo, el número total de multiplicaciones (en cualquier caso) coincide con el número de pasadas por el ciclo.
- La variable y decrece en uno en cada pasada, inicia valiendo x y termina en 1.

Ejemplo 7, II

- Por lo tanto, el número total de multiplicaciones coincide con x-1, donde x es el número cuyo factorial que se desea calcular.
- En el caso recursivo, tenemos que plantear una recurrencia M(x) que representa el número de multiplicaciones.
- Consideremos M(x) el número de multiplicaciones que realiza el algoritmo Factorial2 cuando se llama con el argumento x.
- En el caso base, cuando $x \leq 1$, no se realiza ninguna multiplicación, por lo tanto M(x)=0.
- En el caso recursivo, se hace una llamada recursiva con argumento x-1 y después una multiplicación más, por lo tanto, para $x > 1$, $M(x) = M(x-1) + 1$.
- Es decir:

$$M(x) = \begin{cases} 0 & x \leq 1 \\ M(x-1)+1 & x > 1 \end{cases}$$

Ejemplo 7, III

- Aplicando el método iterativo:

$$
\begin{aligned}
(1)\quad & M(x) = M(x-1) + 1 \\
(2)\quad & = M(x-2) + 1 + 1 = M(x-2) + 2 \\
(3)\quad & = M(x-3) + 1 + 2 = M(x-3) + 3 \\
(i)\quad & = M(x-i) + i \\
(x-1)\quad & = M(1) + x - 1 = x - 1
\end{aligned}
$$

- Por lo tanto, el algoritmo recursivo realiza exactamente el mismo número de multiplicaciones que la versión iterativa.