Mauricio Luque Jiménez 78004003D

Relación 1: Eficiencia

1. Probad que las siguientes afirmaciones son verdad:

1. 17 es $O(1)$

Para calcular la clase de eficiencia de un algoritmo, debemos seguir los siguientes pasos: quitar las constantes y fijarnos en la función $f(n)$ que resulte para dicho algoritmo. En este caso, al tratarse de una constante, no hay función $f(n)$, por lo que la clase de eficiencia es la menor posible (constante).

2. $\frac{n(n-1)}{2}$ es $O(n^2)$

En este caso, $n(n-1)$ es lo mismo que n^2-n. Si quitamos la constante ½, nos quedan dos monomios: n^2 y n. Al tratarse de notación $O(f(n))$, se tiene en cuenta el peor caso (por la regla de la suma) y la mayor clase de eficiencia, de manera que $n^2>n$.

3. $max(n^3, 10n^2)$ es $O(n^3)$

Este puede ser el caso de un algoritmo en el que haya dos partes que no se ejecuten simultáneamente (por ejemplo, un condicional que lleve a dos partes distintas de código), en cuyo caso se aplica la regla de la suma. Como $max(n^3, 10n^2)=n^3$, la clase de eficiencia será el orden de eficiencia de n^3, donde tenemos un caso similar al del apartado anterior.

4. $\log_2(n)$ es $O(\log_3(n))$

Una de las propiedades de los logaritmos es que $\log_a n=\frac{\log_b n}{\log_b a}$. Si tomamos $a=2$ y $b=3$, tenemos $\log_2 n=\frac{\log_3 n}{\log_3 2}$, donde $\log_3 2$ es una constante que no depende de n. Por lo tanto, como al calcular la clase de eficiencia se eliminan las constantes, nos encontramos una clase de eficiencia logarítmica, concretamente $O(\log_3 n)$.

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

1. $f(n)=13n^2+4n-73$

En este caso, si eliminamos las constantes, vemos que el orden de eficiencia es n^2, de manera que $k=2$.

2. $f(n)=\frac{1}{n+1}$

Buscamos una constante c tal que $\frac{1}{n+1}\leq cn^0$. En este caso, como queremos encontrar el k más pequeño y c debe ser positivo, probamos con $c=1$, de manera que tenemos $\frac{1}{n+1}\leq n^0$, lo cual es cierto para cualquier positivo $n\geq 1$. Así pues, siendo $n^0=1$, tenemos que el orden de eficiencia del algoritmo es $O(1)$.

3. $f(n)=\frac{1}{n-1}$

Este caso es igual al anterior, pero con $n^0=2$ porque en $n^0=1$ no existe la función, de manera que es válido para todo $n>1$. De todas maneras, al eliminar las constantes, nos queda que con k^0 el orden de eficiencia es $O(1)$.

4. $f(n)=(n-1)^3$

Desarrollando el polinomio, obtenemos n^3-3n^2+3n-1, donde podemos ver que el orden de eficiencia es n^3, por lo que $k=3$.

5. $f(n)=\frac{n^3+2n-1}{n+1}$

Como podemos ver en la imagen, el resultado del cociente es $n^2-n+3-\frac{4}{n+1}$, lo que nos lleva a una función con orden de eficiencia $O(n^2)$.

6. $f(n)=\sqrt{(n^2-1)}$

En este caso, buscamos una solución a $\sqrt{(n^2-1)}\leq cn$. Como en los apartados 2 y 3, probamos con $c=1$, de manera que, si transformamos $n=\sqrt{n^2}$, obtenemos $\sqrt{(n^2-1)}\leq\sqrt{n^2}$, lo cual es cierto para cualquier $n\geq 1$. Sin embargo, a diferencia de los apartados 2 y 3, esta función crece sin límite, por lo que no nos podemos quedar con la constante c como cota superior, sino que se necesita n. De esta manera, el k mas pequeño posible pasa a ser 1, y el orden de eficiencia pasa a ser $O(n)$.

3. Ordenar de menor a mayor lo siguientes órdenes de eficiencia:

$n, \sqrt{n}, n^3+1, n^2, n\log_2(n), 3^{\log_2(n)}, 3^n, 2^n+3^{n-1}, 20000, n+100, n2^n$

$$20000 < \sqrt{n} < n < n+100 < n\log_2(n) < 3^{\log_2(n)} < n^2 < n^3+1 < n2^n < 2^n+3^{n-1} < 3^n$$

- 20000: orden constantes
- $n, n+100, n\log_2(n)$: orden lineal
- n^2: orden cuadrático
- $\sqrt{(n)}, 3^{\log_2(n)}, n^3+1$: orden polinomial (ya que $\sqrt{(n)}=n^{1/2}$ y, por propiedades de logaritmos, $3^{\log_2(n)}=n^{\log_2 3}$)
- $n2^n, 2^n+3^{n-1}, 3^n$: orden exponencial

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.

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

Esto se podría dar en un algoritmo que contenga un *if-else* con dos bloques de código alternativos, es decir, si se aplica uno no se aplica el otro. De esta manera, por la regla de la suma, el orden de eficiencia es el máximo orden de eficiencia de ambos fragmentos, por lo que en este caso es $max(O(f(n)), O(f(n)))=O(f(n))$ y la afirmación es verdadera.

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

Esto es cierto, ya que $f(n)\geq 1$ a partir de cierto n_0 (puesto que estamos hablando de tiempos de ejecución, que siempre son positivos) y la notación O siempre mide el peor caso, es decir, es una cota superior. Por lo tanto, si $T_1\in O(f(n))$, como $O(f(n))<O(f^2(n)), O(f(n))\in O(f^2(n))$. Es decir, si está contenida en una cota superior, siempre va a estar igualmente contenida en una cota superior mayor.

3. $\frac{T_1(n)}{T_2(n)}\in O(1)$

Esto es falso ya que, como en el apartado anterior, estamos hablando de cotas superiores. Es decir, el peor caso está cubierto dentro de $O(f(n))$, pero eso no significa que siempre se dé el peor caso. Si, por ejemplo, si $T_2(n)=\frac{f(n)}{n}$ (cosa que está contemplada en $O(f(n))$ como $f(n)/n\leq f(n)$), $\frac{T_1(n)}{T_2(n)}=\frac{f(n)}{f(n)/n}=n\notin O(1)$

5. Considerar las siguientes funciones de n:

1. $f_1(n)=n^2$
2. $f_2(n)=n^2+100n$
3. $f_3(n)=n\ si\ n\ es\ impar\ , n^3\ si\ n\ es\ par$
4. $f_4(n)n\ si\ n\leq 100\ , n^3\ si\ n>100$

Indicar para cada par distinto i, j si $f_i(n)$ es $O(f_j(n))$

En primer lugar, hay que determinar el orden de eficiencia de cada función.

- $f_1(n)\in O(n^2)$
- $f_2(n)\in O(n^2)$
- $f_3(n)\in O(n^3)$
- $f_4(n)\in O(n^3)$

Cabe destacar que no se puede determinar el orden de eficiencia para $f_3(n)$, ya que hay tanto infinitos impares como infinitos pares, luego son dos órdenes de eficiencia simultáneos. Sin embargo, sí que en cualquier caso la función está contenida en n^3, ya que se trata del peor caso. En cambio, $f_4(n)\in O(n^3)$ porque hay un número finitos de casos para los que $f_4(n)\in O(n)$, pero a partir de $n=101$, la función siempre tiende a un orden exponencial.

Como $O(f_1(n))=O(f_2(n))$, $f_1(n)$ es $O(f_2(n))$ y viceversa. Así pues, las comparaciones posteriores se pueden realizar con una de las dos funciones o con las dos simultáneamente.

En el caso de $f_3(n)$, $f_1(n)\notin O(f_3(n))$, ya que hay infinitos casos en los que $O(f_3(n))=O(n)$, que no cubre $O(n^2)$. Sin embargo, tampoco se cumple al revés, ya que hay infinitos casos en los que $O(f_3(n))=O(n^3)$, por lo que $f_3(n) not\in O(f_1(n))$. Por otra parte, si la comparamos con $f_4(n)$, vemos que para $n>100$, $O(f_4(n))=O(n^3)$, que es el peor caso de $f_3(n)$ (cuando n par), por lo que $f_3(n)\in O(f_4(n))$. Sin embargo, al revés no se cumple, ya que si buscamos un número impar mayor que 100, $O(f_3(n))=O(n)$ mientras que $O(f_4(n))=O(n^3)$.

Por último, en el caso de $f_4(n)$, se ve muy fácilmente que $f_1(n), f_2(n)\in O(f_4(n))$, igual que el caso contrario no se cumple porque $O(n^2)<O(n^3)$.

6. Obtener usando la notación O-mayúscula la eficiencia de las siguientes funciones o trozo de código.

```
1.  void ejemplo(n){
2.  int i,j,k;
3.  for (int i=0;i<n;i++)
4.    for (int j=i+1;j<n;j++)
5.      for k=0;k<=j; k++
6.          Global+=k*i;
7.  }
```

En esta función, vemos un triple bucle anidado, lo que nos lleva a un orden de eficiencia $O(n^3)$

```
1.  for (int i=0;i<n;i++)
2.    If (i%2){
3.     for (int j=i;j<n;j++)
4.       x*=i;
5.     For (in j=0;j<i;j++)
6.      y*=j;
7.    }
```

En este caso, tenemos un bucle que dentro tiene otros dos bucles operando de forma independiente. Si analizamos en primer lugar el interior del bucle, vemos que, al no estar directamente relacionados, el orden de eficiencia de los dos bucles es el resultado de aplicar la regla de la suma, lo que nos lleva a $O(n)$. Si ahora tenemos en cuenta que eso está dentro de otro bucle, nos queda un orden de eficiencia cuadrático.

```
1.  int funcion1(int n){
2.    int s=0;
3.    for (int i=0;i<n;i++)
4.     s+=i;
5.    return s;
6.  }
7.  int funcion2(int n){
8.     int s=0;
9.     for (int i=0;i<n;i++)
10.       s+=funcion1(i);
11.    return s;
12. }
13. int main(){
14.    int k;
15.   cin>>k;
16.    cout<<funcion2(k);
17. }
```

Por último, en este caso tenemos una función que llama a otra función. En primer lugar, vemos que *funcion1* tiene un orden de eficiencia lineal, al igual que *funcion2* de forma independiente. Sin embargo, como una ejecución se ejecuta dentro de la otra, nos queda un orden de eficiencia cuadrático, porque realmente es como si dentro del bucle de *funcion2* se declarara un segundo bucle, lo que nos dejaría algo similar al segundo fragmento de código.