Algorítmica

Los Del DGIIM, `losdeldgiim.github.io`

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Algorítmica

Los Del DGIIM, `losdeldgiim.github.io`

José Juan Urrutia Milán
Arturo Olivares Martos

Granada, 2023-2024

Índice general

Introducción

El siguiente documento no es sino el mero resultado proveniente de la amalgación proficiente de un cúmulo de notas, todas ellas tomadas tras el transcurso de consecutivas clases magistrales, primordialmente —empero, no exclusivamente— de teoría, en simbiosis junto con una síntesis de los apuntes originales provenientes de la asignatura en la que se basan los mismos. Como motivación para la asignatura, introducimos a continuación un par de problemas que sabremos resolver tras la finalización de esta:

Ejercicio (Parque de atracciones)**.** Disponemos de un conjunto de atracciones

$$A_1, A_2, \dots, A_n$$

Para cada atracción, conocemos la hora de inicio y la hora de fin. Podemos proponer varios retos de programación acerca de este parque de atracciones:

1. Seleccionar el mayor número de atracciones que un individuo puede visitar.
2. Seleccionar las atraciones que permitan que un visitante esté ocioso el menor tiempo posible.
3. Conocidas las valoraciones de los usuarios, $val(A_i)$, seleccionar aquellas que garanticen la máxima valoración conjunta en la estancia.

Ejercicio. Una empresa decide comprar un robot que deberá soldar varios puntos (n) en un plano. El software del robot está casi terminado pero falta diseñar el algoritmo que se encarga de decidir en qué orden el robot soldará los n puntos. Se pide diseñar dicho algoritmo, minimizando el tiempo de ejecución del robot (este depende del tiempo de soldadura que es constante más el tiempo de cada desplazamiento entre puntos, que depende de la distancia entre ellos). Por tanto, deberemos ordenar el conjunto de puntos minimizando la distancia total de recorrido.

Nociones de conceptos

A lo largo de la asignatura, será común ver los siguientes conceptos, los cuales aclararemos antes de empezar la misma:

- Instancia: Ejemplo particular de un problema.
- Caso: Instancia de un problema con una cierta dificultad.

Generalmente, tendremos tres casos:

- El mejor caso: Instancia con menor número de operaciones y/o comparaciones.
- El peor caso: Instancia con mayor número de operaciones y/o comparaciones.
- Caso promedio. Normalmente, será igual al peor caso.

Para notar la eficiencia del peor caso usaremos $O(\cdot)$, mientras que para el mejor caso, $\Omega(\cdot)$.

Diremos que un algoritmo es *estable* en ordenación si, dado un criterio de ordenación que hace que dos elementos sean iguales en cuanto a orden, el orden de stos vendrá dado por el primero se que introdujo en la entrada.

Ejemplo. Dado el criterio de que un número es menor que otro si es par, ante la instancia del problema: $1, 2, 3, 4$. La salida de un algoritmo de ordenación estable según este criterio será:

$$2, 4, 1, 3$$

Sin embargo, un ejemplo de salida que podría dar un algoritmo no estable sería:

$$4, 2, 3, 1$$

Los datos se encuentra ordenados pero no en el orden de la entrada.

Algoritmos de ordenación

A continuación, un breve reapso de algoritmos de ordenación:

- Burbuja es el peor algoritmo de ordenación.
- Si tenemos pocos elementos, suele ser más rápido un algoritmos simple como selección o inserción. Entre estos, selección hace muchas comparaciones y pocos intercambios, mientras que inserción hace menos comparaciones y más intercambios. Por tanto, ante datos pesados con varios registros, selección será mejor que inserrción.
- Cuando se tienen muchos elementos, es mejor emplear un algoritmo de ordenación del orden $n \log(n)$.

1. Eficiencia de Algoritmos

La asignatura se centrará en eficiencia basada en el tiempo de ejecución (no en la eficiencia en cuanto espacio, memoria usada por el programa). Para calcular la eficiencia de un algoritmo, tenemos tres métodos:

- Método empírico: donde se mide el tiempo real.
- Método teórico: donde se mide el tiempo esperado.
- Método híbrido: tiempo teórico evitando las constantes mediante resultados empíricos.

Proposición 1.1 (Principio de Invarianza)**.** *Dadas dos implementaciones $I1$, $I2$ de un algoritmo, el tiempo de ejecución para una misma instancia de tamaño n, $T_{I1}(n)$ y $T_{I2}(n)$, no diferirá en más de una constante multiplicativa. Es decir, $\exists K > 0$ que verifica:*

$$T_{I1}(n) \leqslant K \cdot T_{I2}(n)$$

Por este teorema, podremos despreciar las constantes. En un principio, se asumiriá que operaciones básicas como sumas, multiplicaciones, ... serán de tiempo constante, salvo excepciones (por ejemplo, multiplicaciones de números de 100000 dígitos).

Definición 1.1 (Notación O)**.** Se dice que un algoritmo A es de orden $O(f(n))$, donde f es una función $f : \mathbb{N} \to \mathbb{R}^+$, cuando existe una implementación del mismo tamaño cuyo tiempo de ejecución $T_A(n)$ es menor igual que $K \cdot f(n)$, donde K es una constante real positiva a partir de un tamaño grande n_0. Formalmente:

$$A \text{ es } O(f(n)) \Longleftrightarrow \exists K \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} \mid T_A(n) \leqslant K \cdot f(n) \quad \forall n \geqslant n_0$$

La notación O nos permite conocer cómo se comportará el algoritmo en términos de eficiencia en instancias del caso peor del problema. Como mucho, sabemos que el algoritmo no tardará más de $K \cdot f(n)$ en ejecutarse en el peor de los casos.

Al decir que el algoritmo A es de orden $O(f(n))$, decimos que siempre podemos encontrar una constante positiva K que para valores muy grandes del caso n (a partir de un n_0), el tiempo de ejecución del algoritmo siempre será inferior a $K \cdot f(n)$:

$$T_A(n) \leqslant K \cdot f(n)$$

Ejemplos de órdenes de eficiencia son:

- Constante, $O(1)$.

- Logarítmico, $O(\log(n))$.
- Lineal, $O(n)$.
- Cuadrático, $O(n^2)$.
- Exponencial, $O(a^n)$.
- $\vdots$

Proposición 1.2 (Principio de comparación)**.** *Para saber si dos órdenes $O(f(n))$ y $O(g(n))$ son equivalentes o no, aplicamos las siguientes reglas:*

$$O(f(n)) \equiv O(g(n)) \Longleftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = K \in \mathbb{R}^+$$
$$O(f(n)) > O(g(n)) \Longleftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$
$$O(f(n)) < O(g(n)) \Longleftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

Entendiendo que un órden es menor que otro si es mejor, es decir, más rápido en el caso asintótico.

Ejemplo. Si tenemos dos algoritmos A y B con órdenes de eficiencia $O(n^2)$ y $O((4n+1)^2+n)$ respectivamente, tratamos de ver qué algoritmos es más eficiente:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2}{(4n+1)^2+n)} = \lim_{n\to\infty} \frac{n^2}{(16n^2+1+2\cdot 4n\cdot 1)+n} = \lim_{n\to\infty} \frac{1}{16}$$

Gracias a la Proposición 1.2, tenemos que los algoritmos A y B son equivalentes.

Ejemplo. En esta ocasión, tenemos a dos algoritmos A y B con órdenes de eficiencia de $O(2^n)$ y $O(3^n)$, respectivamente.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} \left(\frac{2}{3}\right)^n = 0$$

Por la Proposición 1.2, A es más eficiente que B.

Ejemplo. El algoritmo A tiene una eficiencia $O(n)$ y el algoritmo B tiene una eficiencia de $O(n\log(n))$. Buscamos cuál es más eficiente.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n}{n\log(n)} = \lim_{n\to\infty} \frac{1}{\log(n)} = 0$$

Por lo que A es más eficiente que B, por la Proposción 1.2.

Ejemplo. Disponemos de dos algoritmos, A y B con órdenes de eficiencia $O((n^2+29)^2)$ y $O(n^3)$ respectivamente. Intuimos que B es más eficiente que A pero queremos probarlo.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(n^2+29)^2}{n^3} = \infty$$

Gracias a la Proposción 1.2, hemos probado lo que esperábamos; B es más eficiente que A.

Ejemplo. Se quiere probar que $O(\log(n))$ es más eficiente que $O(n)$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n}{\log(n)} = \lim_{n\to\infty} \frac{10^n}{n} = \infty$$

Por la Proposción 1.2, lo acabamos de probar.

Ejemplo. Se quiere dar un ejemplo de que el órden de eficiencia de los logaritmos es equivalente sin importar la base de este. Podemos ver qué sucede con $O(\log_2(n))$ y con $O(\log_3(n))$:

$$\lim_{n\to\infty} \frac{\log_3(n)}{\log_2(n)} = \lim_{n\to\infty} \frac{\ln(2)}{\ln(3)}$$

Por lo que ambos algoritmos tienen el mismo órden de eficiencia.

Definición 1.2 (Notación Ω)**.** Se dice que un algoritmo A es de orden $\Omega(f(n))$, donde f es una función $f : \mathbb{N} \to \mathbb{R}^+$, cuando existe una implementación del mismo tamaño cuyo tiempo de ejecución $T_A(n)$ es mayor igual que $K \cdot f(n)$, donde K es una constante real positiva a partir de un tamaño grande n_0. Formalmente:

$$A \text{ es } \Omega(f(n)) \iff \exists K \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} \mid T_A(n) \geqslant K \cdot f(n) \quad \forall n > n_0$$

La notación Ω nos permite conocer cómo se comportará el algoritmo en términos de eficiencia en instancias del caso mejor del problema. Como poco, sabemos que el algoritmo no t ardará menos de $K \cdot f(n)$ en ejecutarse, en el mejor de los casos.

Definición 1.3 (Notación Θ)**.** Se dice que un algoritmo A es de orden exacto $\Theta(f(n))$, donde f es una función $f : \mathbb{N} \to \mathbb{R}^+$, cuando existe una implementación del mismo tamaño cuyo tiempo de ejecución $T_A(n)$ es igual que $K \cdot f(n)$, donde K es una constante real positiva a partir de un tamaño grande n_0. En este caso, el algoritmo es simultáneamente de orden $O(f(n))$ y $\Omega(f(n))$.

$$A \text{ es } \Theta(f(n)) \iff \exists K \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} \mid T_A(n) = K \cdot f(n) \quad \forall n > n_0$$

Propiedades

A continuación, vemos algunas propiedades de las notaciones anteriormente vistas, todas ellas demostradas en el Ejercicio 3.1.1:

Reflexiva

$$f(n) \in O(f(n))$$

También se da para las notaciones Ω y Θ.

Simétrica

$$f(n) \in \Theta(g(n)) \iff g(n) \in \Theta(f(n))$$

Suma Si $T_1(n) \in O(f(n))$ y $T_2(n) \in O(g(n))$. Entonces:

$$T_1(n) + T_2(n) \in O(\text{máx}(f(n), g(n)))$$

Producto Si $T_1(n) \in O(f(n))$ y $T_2(n) \in O(g(n))$. Entonces:

$$T_1(n) \cdot T_2(n) \in O(f(n) \cdot g(n))$$

Regla del máximo

$$O(f(n) + g(n)) = \max(O(f(n)), O(g(n)))$$

Regla de la suma

$$O(f(n) + g(n)) = O(f(n)) + O(g(n))$$

Regla del producto

$$O(f(n) \cdot g(n)) = O(f(n)) \cdot O(g(n))$$

Puede suceder que el tamaño del problema no depende de una única variable n, sino de varias. En estos casos, se analiza de igual forma que en el caso de una variable, pero con una función de varias variables. Conocida una función $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+$:

$$A \text{ es } O(f(n,m)) \Longleftrightarrow \exists K \in \mathbb{R}^+ \mid T_A(n,m) \leqslant K \cdot f(n,m) \quad \forall n,m \in \mathbb{N}$$

Ejemplo. El órden de eficiencia del algoritmo canónico (el que todos conocemos) de suma de matrices $n \times m$ es de órden $O(n \cdot m)$.

1.1. Análisis de algoritmos

El primer paso a la hora de determinar la eficiencia de un algoritmo es identificar qué parámetro determina el tamaño del problema (n). Posteriormente, tenemos que tener claro como se analiza cada estructura del código:

1. Operaciones elementales.
2. Secuencias de sentencias.
3. Sentencias condicionales.
4. Sentencias repetitivas.
5. Llamadas a funciones no recursivas.
6. Llamadas a funciones recursivas.

Sentencias simples u operaciones elementales

Son aquellas instrucciones cuya ejecución no depende del tamaño del caso, como por ejemlo:

- Operaciones matemáticas básicas (sumas, multiplicaciones, ...).
- Comparaciones.
- Operaciones booleanas.

Su tiempo de ejecución está acotado superiormente por una constante. Su órden es $O(1)$.

Secuencias de sentencias

Constan de la ejecución de secuencias de bloques de sentencias:

```
Sentencia_1;
Sentencia_2;
// etc
Sentencia_r;
```

Suponiendo que cada sentencia i tiene eficiencia $O(f_i(n))$, la eficiencia de la secuencia se obtiene mediante las reglas de la suma y del máximo:

$$O(f_1(n) + f_2(n) + \cdots + f_r(n)) = \max\left[O(f_1(n)), O(f_2(n)), \ldots, O(f_r(n))\right]$$

Ejemplo. Un ejemplo que puede parecer confuso es el siguiente:

```
if(n == 10){
    for(int i = 0; i < n; i++){
        cout << "Hola" << endl;
    }
}
```

En este caso, se trata de un código de orden $O(1)$, ya que es para un valor fijo de n, 10. No depende por tanto del tamaño.

1.1.1. Sentencias condicionales

Constan de la evaluación de una condición y la ejecución de un bloque de sentencias. Puede ejecutarse la *Sentencia1* o la *Sentencia2*, en función de la veracidad o falsedad de la condición:

```
if(condicion){
    Sentencia_1;
}else{
    Sentencia_2;
}
```

Peor caso

El órden de eficiencia del peor caso (notación O) viene dado por:

$$O(\text{estructura condicional}) = \max\left[O(\text{condicion}), O(\text{Sentencia1}), O(\text{Sentencia2})\right]$$

Demostración. Como justificación para la fórmula, démonos cuenta de que la ejecución de la estructura condicional es igual a una de las siguientes secuencias de instrucciones (dependerá de la condición la ejecución de una o de otra):

```
bool a = condicion;        bool a = condicion;
Sentencia_1;               Sentencia_2;
```

La notación O trata de buscar el órden del mayor tiempo de ejecución, por lo que buscaremos la secuencia que más tarde de las dos:

$$O(\text{estructura condicional}) = \max\left[O(\text{Secuencia1}), O(\text{Secuencia2})\right]$$

Usando la relga para secuencias de instrucciones vista anteriormente, podemos expresar cada órden como:

$$\begin{aligned} O(\text{Secuencia1}) &= \max\left[O(\text{condicion}), O(\text{Sentencia1})\right] \\ O(\text{Secuencia2}) &= \max\left[O(\text{condicion}), O(\text{Sentencia2})\right] \end{aligned}$$

Por lo que:

$$\begin{aligned} O(\text{estructura condicional}) &= \max\left[O(\text{Secuencia1}), O(\text{Secuencia2})\right] = \\ &= \max\left[\max\left[O(\text{condicion}), O(\text{Sentencia1})\right],\right. \\ &\qquad\quad \left.\max\left[O(\text{condicion}), O(\text{Sentencia2})\right]\right] = \\ &= \max\left[O(\text{condicion}), O(\text{Sentencia1}), O(\text{Sentencia2})\right] \end{aligned}$$

□

Mejor caso

El órden de eficiencia del mejor caso (notación Ω) viene dado por:

$$\Omega(\text{estructura condicional}) = \Omega(\text{condicion}) + \min\left[\Omega(\text{Sentencia1}), \Omega(\text{Sentencia2})\right]$$

Demostración. Buscamos expresar el órden de ejecución en el mejor caso. Al igual que en la justificación anterior, el bloque condicional es equivalante a seleccionar una de dos secuencias:

```
bool a = condicion;        bool a = condicion;
Sentencia_1;               Sentencia_2;
```

Nos damos cuenta de que la condición siempre se ejecuta y luego se ejecuta una sentencia. Como estamos en el mejor caso, seleccionamos la sentencia que tarde menos tiempo en ejecutarse. Por esto, tenemos que el órden de ejecución es la suma del órden de la condición más el mínimo del órden de las dos sentencias. □

Ejemplo. Supongamos el siguienbte código:

```
if(n % 2 == 1){
    cout << "Es impar";
}else{
    // código de órden n
}
```

Aplicamos las fórmulas anteriormente vistas, sabiendo que la condición y la salida son de órden $O(1)$ al ser sentencias simples:

$$\begin{aligned} O(\text{estructura condicional}) &= \max\left[O(\text{condicion}), O(\text{Sentencia1}), O(\text{Sentencia2})\right] \\ &= \max\left[O(1), O(1), O(n)\right] = O(n) \end{aligned}$$

$$\begin{aligned} \Omega(\text{estructura condicional}) &= \Omega(\text{condicion}) + \min\left[\Omega(\text{Sentencia1}), \Omega(\text{Sentencia2})\right] \\ &= \Omega(1) + \min\left[\Omega(1), \Omega(n)\right] = \Omega(1) \end{aligned}$$

1.1.2. Sentencias repetitivas

Constan de la evaluación de uan condición y la ejecución de un bloque de sentencias, mientras que dicha condición se cumpla. Tienen la siguiente forma:

```
Mientras(condicion){
    BloqueSentencias;
}
```

Suponiendo que:

- El bloque de sentencias tiene eficiencia $f(n)$.
- La evaluación de la condición tiene eficiencia $g(n)$.
- El bucle se ejecuta $h(n)$ veces.

Entonces, la eficiencia será:

$$O(\text{estructura repetitiva}) = O(g(n) + h(n) \cdot (g(n) + f(n)))$$

Demostración. La condición se comprueba al menos una vez, por ello se suma. Cada iteración tiene un costo de $g(n) + f(n)$. El bucle realiza $h(n)$ iteraciones. □

Ejemplo. Dado el siguiente código, calcular su eficiencia:

```
while(n > 0){
    cout << n;
    n--;
}
```

- Evaluación de la condición: $g(n) = 1$.
- Bloque de sentencias: $f(n) = \max(O(1), O(1)) = 1$.
- Repeticiones: $h(n) = n$.

$$\begin{aligned} O(\text{estructura repetitiva}) &= g(n) + h(n) \cdot (g(n) + f(n)) \\ &= 1 + n \cdot (1 + 1) = 2 \cdot n + 1 \Rightarrow O(2 \cdot n + 1) \end{aligned}$$

Aplicando la regla del máximo: $O(2 \cdot n + 1) = \max(O(2 \cdot n), O(1)) = O(2 \cdot n)$. Simplificando la constante: La secuencia repetitiva es $O(n)$.

Bucles for

Constan de la inicialización de una variable, comprobación de una condición y actualización de la variable. Se ejecutará un bloque de sentencias mientras que la condición se cumpla:

```
for(Inicialización; Condición; Actualización){
    BloqueSentencias;
}
```

Suponiendo que:

- El bloque de sentencias tiene eficiencia $f(n)$.
- La evaluación de la condifición tiene eficiencia $g(n)$.
- El bucle se ejecuta $h(n)$ veces.
- La actualización tiene eficiencia $a(n)$.
- La inicialización tiene eficiencia $i(n)$.

La eficiencia de una estructura de este tipo viene dada por:

$$O(\text{for}) = O(i(n) + g(n) + h(n) \cdot (g(n) + f(n) + a(n)))$$

Demostración. La inicialización tiene lugar una vez. La condición se comprueba al menos una vez, por ello se suma. Cada iteración tiene un costo de $g(n)+f(n)+a(n)$. El bucle realiza $h(n)$ iteraciones. □

Ejemplo. Dado el siguiente código, calcular su eficiencia:

```
while(n > 0){
    for(int i = 1; i <= n; i*=2){
        cout << i;
    }
    n--;
}
```

Comenzamos analizando el bucle interno:

- Inicialización: $O(1)$.
- Condición: $O(1)$.
- Actualización: $O(1)$.
- Bloque de sentencias: $O(1)$.
- Veces que se ejecuta: $O(\log_2(n))$.

Eficiencia del bucle:

$$O(1) + O(1) + O(\log_2(n)) \cdot (O(1) + O(1) + O(1)) = O(\log_2(n))$$

Ahora, analizamos el bucle externo:

- Condición: $O(1)$.
- Bloque de sentencias: $O(\log_2(n) + 1)$.
- Veces que se ejecuta: $O(n)$.

Eficiencia del bucle:

$$O(1) + O(n) \cdot (O(1) + O(\log_2(n) + 1)) = O(n \cdot \log_2(n))$$

Despreciando la base del logaritmo:

$$O(n \log(n))$$

1.1.3. Funciones no recursivas

La eficiencia de la función per se se calcula como una secuencia de sentencias o bloques. Por otra parte, la eficiencia de la llamada a la función depende de si sus parámetros de entrada dependen o no del tamaño del problema. Esto se entenderá mejor en los siguientes ejemplos. Dada la siguiente función (que usaremos en varios ejemplos):

```
bool esPrimo(int valor){
    double tope = sqrt(valor);
    for(int i = 2; i <=tope; i++){
        if(valor % i == 0)
            return false;
    }

    return true;
}
```

El cuerpo de la función per se tiene efieciencia $O(\sqrt{n})$, ya que:

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

Ejemplo.
```
for(int i = 1; i<n; i++){
    if(esPrimo(1234567))
        cout << i;
}
```

El tiempo en calcular `esPrimo(1234567)` es constante, luego es de eficiencia $O(1)$. Se repite n veces, luego la eficiencia del bucle es de $O(n)$.

Ejemplo.
```
for(int i = 1; i<n; i++){
    if(esPrimo(i))
        cout << i;
}
```

En este caso, el tiempo es:

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

Demostraremos que $T(n) \in O(n\sqrt{n})$ usando el Criterio de Stolz. Para ello, calculamos el cociente de términos consecutivos:

$$\left\{\frac{T(n)-T(n-1)}{n\sqrt{n}-(n-1)\sqrt{n-1}}\right\} = \left\{\frac{\sqrt{n-1}}{n\left(\sqrt{n}-\sqrt{n-1}\right)+\sqrt{n-1}}\right\} =$$

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

$$= \left\{\frac{1}{\frac{\left(\frac{1}{\sqrt{1-\frac{1}{n}}}-1\right)}{\frac{1}{n}}+1}\right\} \xrightarrow{(*)} \frac{1}{\frac{1}{2}+1} = \frac{1}{\frac{3}{2}} = \frac{2}{3}$$

Por el Criterio de Stolz, tenemos que:

$$\lim_{n\to\infty} \frac{T(n)}{n\sqrt{n}} = \frac{2}{3} \in \mathbb{R}^+$$

Por tanto, tenemos que la eficiencia es de orden $T(n) \in O(n\sqrt{n})$. En el proceso, en $(*)$, hemos aplicado que:

$$\lim_{n\to\infty} \frac{\left(\frac{1}{\sqrt{1-\frac{1}{n}}}-1\right)}{\frac{1}{n}} \overset{L'Hôpital}{=} \lim_{n\to\infty} \frac{-\frac{1}{1-\frac{1}{n}} \cdot \frac{1}{2\sqrt{1-\frac{1}{n}}} \cdot \frac{1}{\cancel{n^2}}}{-\frac{1}{\cancel{n^2}}} = \frac{1}{2}$$

Ejemplo.
```
for(int i = 1; i<2000; i++){
    if(esPrimo(i))
        cout << i;
}
```

El tiempo de ejecución no depende de n, siempre tarda lo mismo (es decir, es constante), luego es de orden $O(1)$.

Ejemplo.
```
for(int i = n; i>0; i/=2){
    if(esPrimo(i))
        cout << i;
}
```

Intuitivamente, vemos que la eficiencia es de $O(\log(n)\sqrt{n})$, ya que repite $\log(n)$ veces una orden de eficiencia $O(\sqrt{n})$.

Ejemplo.
```
int LlamarV(int *s, int N){
    for(int i = N-1; i > 0; i = i/2)
        V[i] = V[i]-1;
    return V[0];
}

void Ejemplo(int *v, int N){
    for(int i=0; i<N; i++){
        v[i] = (i*2+20-4*i)/N;
        v[i] = LlamarV(v, N-1)*LlamarV(v, N-2);
    }
}
```

La función `LlamarV` tiene un tiempo de ejecución de $O(\log(n))$. El bucle de `Ejemplo` se repite n veces, luego es de orden $O(n\log(n))$.

Ejemplo.
```
void ejemplo1(int n){
    int i, j, k;
    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];
            }
        }
    }
}
```

Tiene eficiencia $O(n^3)$.

Ejemplo.
```
bool esPalindromo(char v[]){
    bool pal = true;
    int inicio = 0, fin = strlen(v)-1;

    while((pal) && (inicio<fin)){
        if(v[inicio] != v[fin])
            pal = true;
        inicio++;
        fin--;
    }

    return pal;
}
```

Se trata de un algoritmo de orden $O\left(\frac{n}{2}\right) = O(n)$.

Ejemplo.
```
void F(int num1, int num2){
    for(int i = num1; i<= num2; i*=2){
        cout << i << endl;
    }
}
```

Tenemos un algoritmo de dos variables, con $n = num2 - num1$. Se repite $\log(n)$ veces, luego eficiencia:

$$O(\log(n)) = O(\log(num2 - num1))$$

Ejemplo.
```
void F(int* v, int num, int num2){
    int i = -1, j = num2;

    while(i <= j){
        do{
            i++; j--;
        }while(v[i]<v[j]);
    }
}
```

Es un algoritmo que no funciona correctamente en todos los cacos, en un caso puede no llegar a terminar.

1.1.4. Funciones recursivas

Se expresa como una ecuación en recurrencias y el orden de eficiencia es su solución. Primero, suponemos que hay un caso base que se encarga de conocer la solución al problema en un caso menor. Si la solución al caso base la podemos manipular para dar una solución a un caso mayor, tenemos el problema resuelto.

Para solucionar la ecuación en recurrencias, tratamos de buscar la expresión general de la ecuación y luego podremos resolverla, tal y como se elustra en los siguientes ejemplos. Sin emargo, como se apreciará en el Ejemplo 1.1.4, no resuelve todos los algoritmos recursivos. Para ello, deberemos avanzar en teoría.

Ejemplo (Factorial)**.**

```
unsigned long factorial(int n){
    if(n<=1) return 1;
    else return n*factorial(n-1);
}
```

La eficiencia de este algoritmo viene dada por la función $T(n)$:

$$\begin{cases} T(n) = T(n-1) + 1 \\ T(0) = T(1) = 1 \end{cases}$$

Tratamos de dar una expresión general a esta función:

$$\begin{aligned} T(n) &= T(n-1) + 1 = T(n-2) + 2 = T(n-3) + 3 \\ &= T(n-k) + k \quad \forall k \in \mathbb{N} \end{aligned}$$

Tomando $k = n - 1$:

$$T(n) = T(n-(n-1)) + (n-1) = T(1) + n - 1 = \cancel{1} + n - \cancel{1} = n$$

Por lo que $T(n) \in O(n)$.

Ejemplo.

```
int algoritmo(int n){
    if(n <= 1){
        int k = 0;
        for(int = 0; i < n; i++){
            k += k*i;
        }
        return k;
    }else{
        int r1, r2;
        r1 = algoritmo(n-1);
        r2 = algoritmo(n-1);
        return r1*r2;
    }
}
```

La eficiencia viene dada por $T(n)$:

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

Tratamos de resolver la ecuación:

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

$$\begin{aligned} T(n) &= 2T(n-1) + 1 = 2[2T(n-1) + 1] + 1 = 2^2T(n-2) + 2 + 1 = \dots \\ &= 2^kT(n-k) + \sum_{i=0}^{k-1} 2^i \quad \forall k \in \mathbb{N} \end{aligned}$$

Para $k = n-1$:

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

Ejemplo. Plantedada la siguiente función que define el tiempo de ejecución de un algoritmo, se pide determinar el orden de eficiencia del algoritmo:

$$\begin{cases} T(n) = T(n-2) + 1 \\ T(1) = 1 \\ T(0) = 1 \end{cases}$$

Pasamos a resolver el ejercicio:

$$\begin{aligned} T(n) &= T(n-2) + 1 \\ T(n-2) &= T(n-4) + 1 \\ T(n-4) &= T(n-6) + 1 \end{aligned}$$

$$\begin{aligned} T(n) &= T(n-4)+1+1 = T(n-6)+3 = T(n-8)+4 = T(n-10)+5 \\ &= T(n-2k)+k \quad \forall k \in \mathbb{N} \end{aligned}$$

Para $k = \dfrac{n}{2}$ si n es par:

$$T(n) = T\left(n - \frac{\cancel{2}n}{\cancel{2}}\right) + \frac{n}{2} = T(0) + \frac{n}{2} = 1 + \frac{n}{2} \in O(n)$$

Para $k = \dfrac{n-1}{2}$ si n es impar:

$$T(n) = T\left(n - \frac{\cancel{2}(n-1)}{\cancel{2}}\right) + \frac{n-1}{2} = T(1) + \frac{n-1}{2} = 1 + \frac{n-1}{2} \in O(n)$$

Ejemplo (Fibonacci)**.** Dada la función del tiempo de ejecución de Fibonacci:

$$\begin{cases} T(n) = T(n-1) + T(n-2) + 1 \\ T(1) = T(0) = 1 \end{cases}$$

Se pide calcular la expresión de la función para determinar su orden de eficiencia.

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

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 = T(n-2) + T(n-3) + 1 + T(n-3) + T(n-4) + 1 + 1 = \\ &= T(n-3) + T(n-4) + 1 + 2[T(n-4) + T(n-5) + 1] + 1 + T(n-5) + T(n-6) + 1 \end{aligned}$$

En resumen, este método no es útil para resolver este problema.

1.2. Ecuación característica

Se usa para resolver ecuaciones recurrentes que salen en análisis de eficiencia de algoritmos. Vamos a ver dos etapas, que corresponden con cómo se solucionan:

- Ecuaciones lineales homogéneas de coeficientes constantes.
- Ecuaciones lineales no homogéneas de coeficientes constantes.

En análisis de algoritmos nunca tendremos ecuaciones homogéneas, ya que es normal tener un $+1$ por sentencias constantes, usuales en código.

1.2.1. Ecuaciones homogéneas

Raíces distintas

Dada una ecuación del estilo:

$$T(n) = a_1T(n-1) + a_2T(n-2) + \cdots + a_kT(n-k)$$

Tratamos de buscar la solución a la ecuación. Realizamos el cambio de variable $T(n) = x^n$:

$$x^n = a_1x^{n-1} + a_2x^{n-2} + \cdots + a_kx^{n-k}$$

$$\begin{aligned} x^n - a_1x^{n-1} - a_2x^{n-2} - \cdots - a_kx^{n-k} &= 0 \\ &= x^{n-k}(x^n - a_1x^{n-1} - a_2x^{n-2} - \cdots - a_k) \end{aligned}$$

$$p(x) = (x^k - a_1x^{k-1} - a_2x^{k-2} - \cdots - a_k) = 0$$

Se trata de un polinomio de grado k. Suponiendo que $r_1, \ldots, r_k$ son las raíces del polinomio (todas distintas), definimos:

$$t_n = \sum_{i=1}^{k} c_i r_i^n$$

Ejemplo (Fibonacci)**.**

$$T(n) = T(n-1) + T(n-2)$$

Se pide determinar el orden de eficiencia de $T(n)$.

$$\begin{aligned} x^n - x^{n-1} - x^{n-2} &= 0 \\ x_2 - x - 1 &= 0 \end{aligned}$$

El polinomio característico es:

$$p(x) = x^2 - x - 1$$

Buscamos raíces:

$$x = \frac{1 \pm \sqrt{1+1}}{2}$$

Luego:

- $r_1 = \dfrac{1+\sqrt{5}}{2}$
- $r_2 = \dfrac{1-\sqrt{5}}{2}$

$$t_n = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$$

Con $c_1, c_2 \in \mathbb{R}$.
Y se tiene que :

$$T(n) \in O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$

Raíces con multiplicidad distinta de 1

En caso de que las raíces no sean todas distintas, nuestro polinomio es del estilo:

$$p(x) = (x - r_1)^{m_1}(x - r_2)^{m_2} \cdots (x - r_s)^{m_s}$$

Si r_i es una raíz con multiplicidad m_i, tenemos que:

$$t_n = \sum_{i=1}^{s} \left(\sum_{j=0}^{m_i - 1} c_{ij} r_i^n n^j \right)$$

Ejemplo. Dada la función de tiempo de un algoritmo:

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

Se quedaría:

$$t_n - 5t_{n-1} + 8t_{n-2} - 4t_{n-3} = 0$$

Sustituyo $t_n = x^n$:

$$x^n - 5x^{n-1} + 8x^{n-2} - 4x^{n-3} = 0$$

Y al dividir entre x^{n-3}:

$$x^3 - 5x^2 + 8x - 4 = 0$$

Con raíces:

$$p(x) = (x-2)^2(x-2)$$

Entonces, t_n sería:

$$t_n = c_{11} \cdot 1^n + c_{21} 2^n + c_{22} n 2^n$$

Luego $t_n \in O(n2^n)$.

Ejemplo.

$$T(n) = 2T(n-1) - T(n-2)$$

Sustituimos $T(n) = x^n$:

$$x^n - 2x^{n-1} + x^{n-2} = 0$$
$$(x^2 - 2x + 1)x^{n-2} = 0$$

$$p(x) = x^2 - 2x + 1 = (x-1)^2$$

Luego:

$$t_n = c_{10} 1^n + c_{11} 1^n n$$

Con $c_{10}, c_{11} \in \mathbb{R}$.

1.2.2. Ecuciones no homogéneas

Trabajaremos con ecuaciones del estilo:

$$a_0T(n) + a_1T(n-1) + \cdots a_kT(n-k) = b_1^n p_1(n) + b_2^n p_2(x) + \cdots$$

Con b_i constantes y $p_i(n)$ es un polinomio de grado d_1.

Calculamos el polinomio con la fórmula:

$$p(x) = p_h(x) \prod_{i=1}^{s} (x - b_i)^{i+1}$$

Donde p_h es la ecuación característica de la ecuación homogénea.

De donde extraemos todas sus raíces: $r_1, \ldots, r_k$.

$$t_n = c_1\sqrt{2}^n + c_2(-\sqrt{2})^n + c_3 \cdot 1^n$$

Con c_1, c_2, c_3 constantes.

Ejemplo (Fibonacci)**.**

$$T(n) = 2T(n-1) + T(n-2) + 1$$

Sustituimos $T(n) = t_n$ y buscamos como expresar $1 = b^n p(n)$:

$$t_n - t_{n-1} - t_{n-2} = 1 = 1^n \cdot n^0$$

$$\left(x - \sqrt{2}\right)\left(x + \sqrt{2}\right)(x-1)^1$$

Luego:

$$t_n = c_1\sqrt{2}^n + c_2(-\sqrt{2})^n + c_3 \cdot 1^n$$

Con c_1, c_2, c_3 constantes, luego $T(n) \in O(\sqrt{2}^n)$.

Ejemplo.

$$T(n) = T(n-1) + n$$

Sustituimos:

$$t_n - t_{n-1} = n = 1^n \cdot n^1$$

$$p(x) = (x-1)(x-1)^2 = (x-1)^3$$

Luego:

$$t_n = c_1 1^n + c_2 n 1^n + c_3 n^2 1^n \Rightarrow T(n) = O(n^2)$$

Ejemplo.

$$T(n) - T(n-1) = n + 3^n$$

En este caso, nos podríamos quedar simplemente con la fórmula:

$$T(n) - T(n-1) = 3^n$$

Y el orden de eficiencia sería el mismo.

Lo resolvemos de ambas formas:

$$p(x) = (x-1)(x-3)$$
$$t_n = c_1 1^n + c_3 3^n \Rightarrow T(n) \in O(3^n)$$

Si decidimos mantener la n:

$$p(x) = (x-1)(x-1)^2(x-3) = (x-1)^3(x-3)$$

$$t_n = c_1 1^n + c_2 n 1^n + c_3 3^n \Rightarrow T(n) \in O(3^n)$$

1.2.3. Cambio de variable

Hay problemas que el método anterior no nos permite solucionar. Para ello, hacemos un cambio de variable. No olvidemos cambiarlo al final.

Ejemplo.
$$T(n) = 2T\left(\frac{n}{3}\right) + 1$$

Hacemos el cambio de variable: $n = 3^k$:

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

$$t_k - 2t_{k-1} = 1$$
$$p(x) = (x-2)(x-1)$$

$$t_k = c_1 2^k + c_2 1^k$$

Con c_1, c_2 constantes

Luego:

$$\begin{aligned} t_n &= c_1 2^{\log_3(n)} + c_2 \\ &\overset{(*)}{=} c_1 n^{\log_3(2)} + c_2 \Rightarrow T(n) \in O\left(n^{\log_3 2}\right) \end{aligned}$$

Donde en $(*)$ he usado que:

$$a^{\log b} = b^{\log a}$$

Ejemplo.
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Haciendo el cambio: $n = 2^k$:

$$t_k - 2t_{k-1} = 2^k k^0$$

$$p(x) = (x-2)(x-2) = (x-2)^2$$
$$t_k = c_1 2^k + c_2 k 2^k$$

Deshago el cambio: $k = \log_2(n)$

$$t_n = c_1 n + c_2 n \log_2(n) \in O(n \log(n))$$

Ejemplo.

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

Haciendo el cambio: $n = 2^k$:

$$t_k - 2t_{k-1} = (2^k)^2 k^0 = (2^2)^k = 4^k$$

$$p(x) = (x-2)(x-4)$$

$$t_k = c_1 2^k + c_2 k 4^k = c_1 2^k + c_2 (2^k)^2$$

Deshago el cambio: $k = \log_2(n)$

$$t_n = c_1 n + c_2 n^2 \in O(n^2)$$

Ejemplo.

$$T(n) = aT\left(\frac{n}{6}\right) + n^c$$

Con $a, b, c \in \mathbb{R}$. Se pide calcular la eficiencia del algoritmo en función de a, b, c.

Haciendo el cambio: $n = 6^k$:

$$t_k = at_{k-1} + (b^k)^c = at_{k-1} + (b^c)^k$$
$$t_k - at_{k-1} = (b^c)^k$$

Con polinomio:

$$p(x) = (x-a)(x-b^c)$$

- Si $a = b^c$:

$$p(x) = (x-b^c)^2 = (x-a)^2$$
$$t_k = c_1 a^k + c_2 k a^k = c_1 (b^c)^k + c_2 n (b^c)^k$$
$$t_n = c_1 n^c + c_2 n^c \log_b n \Rightarrow T(n) \in O\left(n^c \log_b(n)\right)$$

- Si $a \neq b^c$:

$$p(x) = (x-b^c)(x-a)$$
$$t_k = c_1 (b^c)^k + c_2 a^k$$
$$t_n = c_1 n^c + c_2 a^{\log_b(n)} \Rightarrow c_1 n^c + c_2 n^{\log_b(a)}$$

  - Si $a > b^c$: $T(n) \in O\left(n^{\log_b a}\right)$
  - Si $a < b^c$: $T(n) \in O(n^c)$

1.2.4. Cambio de recorrido o rango

Hay casos en los que los coeficientes no son constantes:

$$T(n) = nT(n-1)$$
$$T(n) = T^2(n-1)$$
$$T(n) = nT\left(\frac{n}{2}\right)$$

Ejemplo.

$$T(n) = T^2(n-1)$$

Sea $V(n) = \log_2 T(n)$:

$$\begin{gathered}
\log_2(T(n)) = \log_2\left(T^2(n-1)\right) = 2\log_2 T(n-1) \\
V(n) = 2V(n-1) \\
V_n = 2V_{n-1} \\
t_n - 2t_{n-1} = 0 \\
p(x) = (x-2) \\
v_n = c_1 2^n \\
t_n = 2^{v_n} = 2^{(c_1 2^n)} = 4^n
\end{gathered}$$

Teorema 1.3 (Teorema Maestro)**.** *Dada una función que mide el tiempo de ejecución de un algoritmo del estilo:*

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Entonces:

- *Si* $\exists \varepsilon > 0 \mid f(n) \in O\left(n^{\log_b(a-\varepsilon)}\right) \Rightarrow T(n) \in \Theta\left(n^{\log_b(a)}\right)$
- *Si* $\exists k \geqslant 0 \mid f(n) \in \Theta\left(n^{\log_b a} \log^k(n)\right) \Rightarrow T(n) \in \Theta\left(n^{\log_b(a)} \log^{k+1}(n)\right)$
- *Si* $\left\{\begin{array}{c} \exists \varepsilon > 0 \mid f(n) \in \Omega\left(n^{\log_b(a+\varepsilon)}\right) \\ \wedge \\ af\left(\frac{n}{b}\right) \leqslant cf(n) \text{ con } c < 1 \end{array}\right\} \Rightarrow T(n) \in \Theta(f(n))$

Ejercicio. Se pide determinar el orden de eficiencia de los siguientes algoritmos, conociendo las funciones que nos dan sus tiempos a partir de n:

1. $T(n) = 4T(n-1) - 4T(n-2)$
2. $T(n) = 3T(n-1) - 3T(n-2) + T(n-3)$
3. $T(n) = 2T(n-1) + n + n^2$
4. $T(n) = 5T(n-1) - 6T(n-2) + 4 \cdot 3^n$
5. $T(n) = 2T\left(\frac{n}{2}\right) + \log_2(n)$
6. $T(n) = T\left(\frac{n}{2}\right) \cdot T^2\left(\frac{n}{4}\right)$
7. $T(n) = aT\left(\frac{n}{b}\right) + n^k$

2. Algoritmos Divide y Vencerás

Dado un problema P, lo dividimos en subproblemas que han de ser del mismo tipo: $P = \{P_i\}_{i \in I}$. Encontramos las soluciones de todos estos problemas: $S_i \mid i \in I$ y con estas, las juntamos de forma adecuada para obtener la solución al problema P: $S = \{S_i\}_{i \in I}$. Notemos que podemos hacerlo de forma recursiva, podemos aplicar la técnica de divide y vencerás a los subproblemas. Debemos fijar un caso base, donde el problema sea suficientemente pequeño como para resolverlo.

Ejemplos de algoritmos que implementan la búsqueda binaria son Quicksort, Mergesort, Búsqueda Binaria, ...

Problemas como el de las Torres de Hanoi pueden resolverse mediante esta técnica de resolución de problemas.

Tratamos cuestiones importantes:

- Cómo descomponer un problema en subproblemas.
- Cómo resolver subproblemas.
- Cómo combinar las soluciones.
- ¿Mereció la pena aplicar esta técnica?

Para responder al último punto observemos el siguiente ejemplo, que trata un problema de forma genérica.

Ejemplo. Supongamos un problema P de tamaño n que sabemos que puede resolverse con un algoritmo (básico) A, con:

$$t_A(n) \leqslant cn^2$$

Dividimos P en 3 subproblemas de tamaño $\frac{n}{2}$, isneod cada uno de ellos del mismo tipo que A y consumiendo un tiempo lineal la combinación de sus soluciones: $t(n) \leqslant dn$. Tenemos, por tanto, un nuevo algoritmo B, Divide y Vencerás, que consumirá un tiempo de:

$$t_B(n) = 3t_A\left(\frac{n}{2}\right) + t(n) \leqslant 3t_A\left(\frac{n}{2}\right) + dn \leqslant \left(\frac{3c}{4}\right) n^2 + dn$$

B tiene un tiempo de ejecución mejor que el algoritmo A, ya que reduce la constante oculta.

Si volvemos a reducir el tamaño del subproblema, obtenemos un nuevo algoritmo C, que tendría un tiempo:

$$t_c(n) = \begin{cases} t_A(n) & \text{si } n \leqslant n_0 \\ 3t_C\left(\dfrac{n}{2}\right) + t(n) & \text{si } n > n_0 \end{cases}$$

C es mejor en eficiencia que los algoritmos A y B ($t_c(n) \leqslant bn^{1{,}58}$).

Al valor n_0 se le denomina umbral y al algoritmo que se ejecuta debajo de este umbral, `ad hoc` son fundamentales para que la técnica funcione bien.

Requisitos lógicos para usar Divide y Vencerás:

- El problema se tiene que poder reducir a subproblemas del mismo tipo.
- Los subproblemas han de poder resolverse de manera independiente.
- Las soluciones de los subproblemas no deben afectar entre sí, no debe existir solapamiento entre subproblemas.
- Ha de poderse juntar las soluciones de los subproblemas para resolver el problema.

Una forma matemática de entender el punto 3 es decir que los subproblemas deben formar una partición del problema.

Para que la técnica sea eficiente:

- Selección cuidadosa de cuando usar el algoritmo `ad hoc`.
- El número de subproblemas debe ser razonablemente pequeño (para no reducir mucho la eficiencia).
- Los subproblemas deben tener el menor tamaño posible.

Normalmente, al aplicar Divide y Vencerás nos encontraremos con eficiencias del estilo:

$$T(n) = \begin{cases} t(n) & \text{si } n \leqslant n_0 \\ IT\left(\frac{n}{b}\right) + G(n) & \text{si } n > n_0 \end{cases}$$

Donde I es el número de subproblemas, $\frac{n}{b}$ el tamaño de estos y:

$$G(n) = D(n) + C(n)$$

Con $D(n)$ el tiempo de dividir el problema en subproblemas y $C(n)$ el tiempo de combinación. $t(n)$ era el tiempo del algoritmo `ad hoc`.

Ejemplo. Dado un vector de elementos, determinar la posición que ocupa el máximo del mismo.

Proponemos como solución el siguiente código:

```
int Maximo(int* v, int inf, int sup){
    if(sup == inf) return v[inf];
    else{
        int med = (inf+sup)/2;
        int izda = Maximo(v, inf, med);
        int dcha = Maximo(v, med+1, sup);

        return max(izda, dcha);
    }
}
```

Con un tiempo de ejecución de:

$$T(n) = 2T\left(\frac{n}{2}\right) + 1 \in O(n)$$

Otra solución sería:

```
int Maximo2(int* v, int inf, int sup){
    if(sup == inf) return v[inf];
    else{
        int izda = Maximo2(v, inf, sup);

        return max(izda, sup);
    }
}
```

En este caso:

$$T(n) = T(n-1) + 1 \in O(n)$$

Los dos del mismo orden, pero encontramos un problema entre uno y otro: el número de llamadas recursivas:

- `Maximo2` Realiza n llamadas recursivas, pero el tamaño de la pila ocupado es de n.

- `Maximo` Realiza n llamadas recursivas, pero el tamaño de la pila es $\log n$.

El número de los elementos de la pila de `Maximo` puede razonarse dibujándolo con un árbol.

Por tanto, el primer algoritmo que planteamos era mejor ya que usa menos pila (evitando su desbordamiento).

Observación. Notemos que este ejemplo ha sido un caso instructivo, es más eficiente resolverlo de la forma canónica que aplicando Divide y Vencerás.

3. Relaciones de Problemas

3.1. La eficiencia de los algoritmos

Ejercicio 3.1.1. Demostrar las siguientes propiedades:

a) $k \cdot f(n) \in O(f(n)), \qquad \forall k > 0.$

Hemos de ver que existe una constante $c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$ tal que $k \cdot f(n) \leqslant c \cdot f(n)$ para todo $n \geqslant n_0$. En este caso, podemos tomar $c = k$ y $n_0 = 1$ y se tiene que $k \cdot f(n) \leqslant k \cdot f(n)$ para todo $n \in \mathbb{N}$.

b) $n^r \in O(n^k)$ si $0 \leqslant r \leqslant k$.

Hemos de ver que existe una constante $c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$ tal que $n^r \leqslant c \cdot n^k$ para todo $n \geqslant n_0$.

Como $0 \leqslant r \leqslant k$, entonces $n^r \leqslant n^k$ para todo $n \in \mathbb{N}$, por lo que podemos tomar $c = 1$ y $n_0 = 1$.

c) $O(n^k) \subset O(n^{k+1})$.

Sea $f(n) \in O(n^k)$; es decir, existe una constante $c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$ tal que $f(n) \leqslant c \cdot n^k$ para todo $n \geqslant n_0$. Hemos de ver que $f(n) \in O(n^{k+1})$; es decir, que existe una constante $c' \in \mathbb{R}^+$, $n'_0 \in \mathbb{N}$ tal que $f(n) \leqslant c' \cdot n^{k+1}$ para todo $n \geqslant n'_0$.

Tomando $c' = c$ y $n'_0 = n_0$, se tiene que $f(n) \leqslant c \cdot n^k \leqslant c \cdot n^{k+1}$ para todo $n \geqslant n_0$, por lo que $f(n) \in O(n^{k+1})$.

d) $n^k \in O(b^n) \quad \forall b > 1, k \geqslant 0.$

Hemos de ver que existe una constante $c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$ tal que $n^k \leqslant c \cdot b^n$ para todo $n \geqslant n_0$. Tomando $c = 1$, tenemos que dicho valor de n_0 existe, ya que:

$$\lim_{n \to \infty} \frac{n^k}{b^n} = 0$$

e) $\log_b n \in O(n^k) \quad \forall b > 1, k > 0.$

Hemos de ver que existe una constante $c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$ tal que $\log_b n \leqslant c \cdot n^k$ para todo $n \geqslant n_0$. Tomando $c = 1$, tenemos que dicho valor de n_0 existe, ya que:

$$\lim_{n \to \infty} \frac{\log_b n}{n^k} = 0$$

f) Si $f(n) \in O(g(n))$ y $h(n) \in O(g(n))$, entonces $f(n) + h(n) \in O(g(n))$.

Tenemos que:

$$f(n) \in O(g(n)) \Longrightarrow \exists c_1 \in \mathbb{R}^+,\ n_1 \in \mathbb{N} \text{ tal que } f(n) \leqslant c_1 \cdot g(n) \quad \forall n \geqslant n_1,$$
$$h(n) \in O(g(n)) \Longrightarrow \exists c_2 \in \mathbb{R}^+,\ n_2 \in \mathbb{N} \text{ tal que } h(n) \leqslant c_2 \cdot g(n) \quad \forall n \geqslant n_2.$$

Tomando $c = c_1 + c_2$ y $n_0 = \text{máx}\{n_1, n_2\}$, se tiene que:

$$f(n) + h(n) \leqslant c_1 \cdot g(n) + c_2 \cdot g(n) = (c_1 + c_2) \cdot g(n) \quad \forall n \geqslant n_0,$$

g) Si $f(n) \in O(g(n))$, entonces $f(n) + g(n) \in O(g(n))$.

Por el primer apartado, sabemos que $g(n) \in O(g(n))$. Por tanto, usando el apartado anterior, se tiene que $f(n) + g(n) \in O(g(n))$.

h) *Reflexividad*: $f(n) \in O(f(n))$.

Se tiene de forma directa por el primer apartado tomando $k = 1$.

i) *Transitividad*: Si $f(n) \in O(g(n))$ y $g(n) \in O(h(n))$, entonces $f(n) \in O(h(n))$.

Tenemos que:

$$f(n) \in O(g(n)) \Longrightarrow \exists c_1 \in \mathbb{R}^+,\ n_1 \in \mathbb{N} \text{ tal que } f(n) \leqslant c_1 \cdot g(n) \quad \forall n \geqslant n_1,$$
$$g(n) \in O(h(n)) \Longrightarrow \exists c_2 \in \mathbb{R}^+,\ n_2 \in \mathbb{N} \text{ tal que } g(n) \leqslant c_2 \cdot h(n) \quad \forall n \geqslant n_2.$$

Por tanto, tomando $c = c_1 \cdot c_2$ y $n_0 = \text{máx}\{n_1, n_2\}$, se tiene que:

$$f(n) \leqslant c_1 \cdot g(n) \leqslant c_1 \cdot c_2 \cdot h(n) = c \cdot h(n) \quad \forall n \geqslant n_0,$$

j) *Regla de la suma*: Si $T1(n)$ es $O(f(n))$ y $T2(n)$ es $O(g(n))$, entonces:

$$T1(n) + T2(n) \in O(\text{máx}\{f(n), g(n)\}).$$

Tenemos que:

$$T1(n) \in O(f(n)) \Longrightarrow \exists c_1 \in \mathbb{R}^+,\ n_1 \in \mathbb{N} \text{ tal que } T1(n) \leqslant c_1 \cdot f(n) \quad \forall n \geqslant n_1,$$
$$T2(n) \in O(g(n)) \Longrightarrow \exists c_2 \in \mathbb{R}^+,\ n_2 \in \mathbb{N} \text{ tal que } T2(n) \leqslant c_2 \cdot g(n) \quad \forall n \geqslant n_2.$$

Tomando $c = \text{máx}\{c_1, c_2\}$ y $n_0 = \text{máx}\{n_1, n_2\}$, se tiene que:

$$T1(n) + T2(n) \leqslant c_1 \cdot f(n) + c_2 \cdot g(n) \leqslant c \cdot \text{máx}\{f(n), g(n)\} \quad \forall n \geqslant n_0,$$

k) *Regla del producto*: Si $T1(n)$ es $O(f(n))$ y $T2(n)$ es $O(g(n))$, entonces:

$$T1(n) \cdot T2(n) \in O(f(n) \cdot g(n)).$$

Tenemos que:

$$T1(n) \in O(f(n)) \Longrightarrow \exists c_1 \in \mathbb{R}^+,\ n_1 \in \mathbb{N} \text{ tal que } T1(n) \leqslant c_1 \cdot f(n) \quad \forall n \geqslant n_1,$$
$$T2(n) \in O(g(n)) \Longrightarrow \exists c_2 \in \mathbb{R}^+,\ n_2 \in \mathbb{N} \text{ tal que } T2(n) \leqslant c_2 \cdot g(n) \quad \forall n \geqslant n_2.$$

Tomando $c = c_1 \cdot c_2$ y $n_0 = \text{máx}\{n_1, n_2\}$, se tiene que:

$$T1(n) \cdot T2(n) \leqslant c_1 \cdot f(n) \cdot c_2 \cdot g(n) = c \cdot f(n) \cdot g(n) \quad \forall n \geqslant n_0,$$

Ejercicio 3.1.2. Expresar, en notación $O(\cdot)$, el orden que tendrí un algoritmo cuyo tiempo de ejecución fuera $f_i(n)$, donde:

1. $f_1(n) = n^2$

   En este caso se tiene que $f_1(n) \in O(n^2)$.

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

   En este caso, por la regla de la suma, se tiene que $f_2(n) \in O(n^2)$.

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

   En este caso, como $n^3 \geqslant n$ para todo $n \geqslant 1$, se tiene que $f_3(n) \in O(n^3)$.

4. $f_n4(n) = \begin{cases} n & \text{si } n \leqslant 100 \\ n^3 & \text{si } n > 100 \end{cases}$

   En este caso, como se trata de comportamientos asintóticos, se tiene que $f_4(n) \in O(n^3)$.

5. $f_5(n) = (n-1)^3$

   En este caso, por la regla de la suma, se tiene que $f_5(n) \in O(n^3)$.

6. $f_6(n) = \sqrt{n^2 - 1}$.

   En este caso, como $\sqrt{n^2-1} \leqslant n$ para todo $n \geqslant 1$, se tiene que $f_6(n) \in O(n)$.

7. $f_7(n) = \log(n!)$

   Por el Criterio de Stolz, se tiene que:

$$\left\{\frac{\log(n!)}{n\log n}\right\} = \left\{\frac{\log(n+1)}{(n+1)\log(n+1) - n\log n}\right\} = \left\{\frac{1}{n+1-n\cdot\frac{\log(n)}{\log(n+1)}}\right\} \to \frac{1}{n+1-n} = 1$$

   Por tanto, tenemos que $\log(n!) \in O(n\log n)$.

8. $f_8(n) = n!$

   Claramente, $f_8(n) \in O(n!)$.

Ejercicio 3.1.3. Usando la notación $O(\cdot)$, obtener el tiempo de ejecución de las siguientes funciones:

1. Código Fuente 1 (`ejemplo1`).

```
1  void ejemplo1 (int n)
2  {
3      int i, j, k;
4
5      for (i = 0; i < n; i++)
6          for (j = 0; j < n; j++)
7          {
8              C[i][j] = 0;
9              for (k = 0; k < n; k++)
10                 C[i][j] += A[j][k] * B[k][j];
11         }
12 }
```

Código fuente 1: Función del Ejercicio 3.1.3 apartado 1.

En este caso, el tiempo de ejecución de la linea 10 lo podemos acotar por una constante, sea esta c. Entonces, tenemos que el tiempo de ejecución de la función `ejemplo1` es:

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left(1 + \sum_{k=0}^{n-1} c\right) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (1 + c \cdot (n - 1 - 0 + 1)) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (1 + c \cdot n) =$$
$$= n^2 \cdot (1 + c \cdot n) \in O(n^3)$$

Se trata del algoritmo de multiplicación de matrices, cuyo tiempo de ejecución es $O(n^3)$.

2. Código Fuente 2 (`ejemplo2`).

```
1  long ejemplo2 (int n)
2  {
3      int i, j, k;
4      long total = 0;
5
6      for (i = 0; i < n; i++)
7          for (j = i+1; j <= n; j++)
8              for (k = 1; k <= j; k++)
9                  total += k*i;
10
11     return total;
12 }
```

Código fuente 2: Función del Ejercicio 3.1.3 apartado 2.

En este caso, el tiempo de ejecución de la linea 9 lo podemos acotar por una constante, sea esta c. Entonces, tenemos que el tiempo de ejecución de la función `ejemplo2` es:

$$\begin{aligned}
T(n) &= 1 + \sum_{i=0}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j} c = 1 + \sum_{i=0}^{n-1}\sum_{j=i+1}^{n} c\cdot(j-1+1) = 1 + \sum_{i=0}^{n-1}\sum_{j=i+1}^{n} c\cdot j = \\
&= 1 + \sum_{i=0}^{n-1} c\cdot\left(\frac{n(n+1)}{2} - \frac{i(i+1)}{2}\right) = 1 + cn\cdot\frac{n(n+1)}{2} - c\cdot\sum_{i=0}^{n-1}\frac{i(i+1)}{2} = \\
&= 1 + cn\cdot\frac{n(n+1)}{2} - \frac{c}{2}\sum_{i=0}^{n-1} i^2 - \frac{c}{2}\sum_{i=0}^{n-1} i = \\
&= 1 + cn\cdot\frac{n(n+1)}{2} - \frac{c}{2}\cdot\frac{(n-1)n(2(n-1)+1)}{6} - \frac{c}{2}\cdot\frac{(n-1)n}{2} \in O(n^3)
\end{aligned}$$

Por tanto, el tiempo de ejecución de la función `ejemplo2` es $O(n^3)$.

3. Código Fuente 3 (`ejemplo3`).

```
void ejemplo3 (int n)
{
    int i, j, x=0, y=0;

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

Código fuente 3: Función del Ejercicio 3.1.3 apartado 3.

Tenemos que:

$$\begin{aligned}
T(n) &= 1 + \sum_{i=1}^{n}\left(2 + \sum_{j=i}^{n} 1 + \sum_{j=0}^{i-1} 1\right) = 1 + \sum_{i=1}^{n}(2 + (n-i+1) + i) = \\
&= 1 + \sum_{i=1}^{n}(n+3) = 1 + n(n+3) \in O(n^2)
\end{aligned}$$

Por tanto, el tiempo de ejecución de la función `ejemplo3` es $O(n^2)$.

4. Código Fuente 4 (`ejemplo4`).

```
int ejemplo4 (int n)
{
    if (n <= 1)
        return 1;
    else
        return (ejemplo4(n - 1) + ejemplo4(n-1));
}
```

Código fuente 4: Función del Ejercicio 3.1.3 apartado 4.

En este caso, el tiempo de ejecución de la función `ejemplo4` es:

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 2T(n-1)+1 & \text{en otro caso} \end{cases}$$

Opción 1. Desarrolo en series. Desarrollando en series, tenemos que:

$$\begin{aligned} T(n) &= 2T(n-1)+1 = 2(2T(n-2)+1)+1 = 2^2T(n-2)+2+1 = \\ &= 2^3T(n-3)+2^2+2+1 = \ldots = 2^iT(n-i)+2^{i-1}+\ldots+2+1 = \\ &= 2^iT(n-i)+\sum_{j=0}^{i-1} 2^j \qquad \forall i \in \mathbb{N}, i \geqslant n-1 \end{aligned}$$

Para $i = n-1$, se tiene que:

$$\begin{aligned} T(n) = 2^{n-1}T(1)+\sum_{j=0}^{n-2} 2^j &= 2^{n-1}+\sum_{j=0}^{n} 2^j - 2^{n-1} - 2^n = \\ &= \cancel{2^{n-1}} + \frac{1-2^{n+1}}{1-2} - \cancel{2^{n-1}} - 2^n = \\ &= -1+2^{n+1}-2^n \end{aligned}$$

Comprobemos que $T(n) \in O(2^n)$:

$$\lim_{n\to\infty} \frac{T(n)}{2^n} = \lim_{n\to\infty} \frac{-1+2^{n+1}-2^n}{2^n} = \lim_{n\to\infty} \frac{-1}{2^n} + 2 - 1 = 2 - 1 = 1 \in \mathbb{R}^+$$

Por tanto, se tiene que $T(n) \in O(2^n)$.

Opción 2. Ecuación Característica.

Aplicando el cambio de variable $T(n) = x^n$, en la parte homogénea tenemos que:

$$x^n - 2x^{n-1} = 0 \Longrightarrow x^{n-1}(x-2) = 0$$

Como $x^{n-1} > 0$ para todo $n \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 2$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $1 = 1^n n^0$, por lo que el polinomio característico de la ecuación en diferencias es:

$$p(x) = (x-2)(x-1)$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(n) = x^n = c_1 \cdot 2^n + c_2 \cdot 1^n = c_1 \cdot 2^n + c_2 \in O(2^n)$$

De ambas formas, se tiene que el tiempo de ejecución de la función `ejemplo4` es $O(2^n)$.

5. Código Fuente 5 (`ejemplo5`).

```c
int ejemplo5 (int n)
{
    if (n == 1)
        return n;
    else
        return (ejemplo5(n/2) + 1);
}
```

Código fuente 5: Función del Ejercicio 3.1.3 apartado 5.

En este caso, el tiempo de ejecución de la función `ejemplo5` es:

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ T\left(\frac{n}{2}\right) + 1 & \text{en otro caso} \end{cases}$$

Opción 1. Desarrolo en series. Desarrollando en series, tenemos que:

$$T(n) = T\left(\frac{n}{2}\right) + 1 = T\left(\frac{n}{2^2}\right) + 1 + 1 = T\left(\frac{n}{2^3}\right) + 1 + 1 + 1 = \dots =$$
$$= T\left(\frac{n}{2^i}\right) + i \qquad \forall i \in \mathbb{N}, i \geqslant \log_2 n$$

Para $i = \log_2 n$, se tiene que:

$$T(n) = T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n = T(1) + \log_2 n = 1 + \log_2 n \in O(\log n)$$

Por tanto, el tiempo de ejecución de la función `ejemplo5` es $O(\log n)$.

Opción 2. Ecuación Característica.

Aplicando el cambio de variable $n = 2^m$, en la parte homogénea tenemos que:

$$T(2^m) - T(2^{m-1}) = 0$$

Por tanto, la ecuación característica de la ecuación en diferencias es:

$$x^m - x^{m-1} = 0 \Longrightarrow x^{m-1}(x-1) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 1$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $1 = 1^m$, por lo que el polinomio característico de la ecuación en diferencias es:

$$p(x) = (x-1)(x-1) = (x-1)^2$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(2^m) = c_1 \cdot 1^m + c_2 \cdot m \cdot 1^m = c_1 + c_2 \cdot m$$

Deshaciendo el cambio de variable $m = \log_2 n$, se tiene que:

$$T(n) = c_1 + c_2 \cdot \log_2 n \in O(\log n)$$

De ambas formas, se tiene que el tiempo de ejecución de la función `ejemplo5` es $O(\log n)$.

Ejercicio 3.1.4. Resolver las siguientes recurrencias:

a) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 2T(n-1)+1 & \text{en otro caso} \end{cases}$

Opción 1. Desarrolo en series.

Desarrollando en series, tenemos que:

$$\begin{aligned} T(n) &= 2T(n-1)+1 = 2(2T(n-2)+1)+1 = 2^2T(n-2)+2+1 = \\ &= 2^3T(n-3)+2^2+2+1 = \ldots = 2^iT(n-i)+2^{i-1}+\ldots+2+1 = \\ &= 2^iT(n-i)+\sum_{j=0}^{i-1} 2^j \qquad \forall i \in \mathbb{N}, i \geqslant n \end{aligned}$$

Para $i = n$, se tiene que:

$$\begin{aligned} T(n) &= 2^nT(0)+\sum_{j=0}^{n-1} 2^j = \sum_{j=0}^{n} 2^j - 2^n = \\ &= \frac{1-2^{n+1}}{1-2} - 2^n = -1+2^{n+1}-2^n \in O(2^n) \end{aligned}$$

Veamos que efectivamente $T(n) \in O(2^n)$:

$$\lim_{n\to\infty} \frac{T(n)}{2^n} = \lim_{n\to\infty} \frac{-1+2^{n+1}-2^n}{2^n} = \lim_{n\to\infty} \frac{-1}{2^n}+2-1 = 2-1 = 1 \in \mathbb{R}^+$$

Por tanto, se tiene que $T(n) \in O(2^n)$.

Opción 2. Ecuación Característica.

Aplicando el cambio de variable $T(n) = x^n$, en la parte homogénea tenemos que:

$$x^n - 2x^{n-1} = 0 \Longrightarrow x^{n-1}(x-2) = 0$$

Como $x^{n-1} > 0$ para todo $n \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 2$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $1 = 1^n n^0$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-2)(x-1)$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(n) = x^n = c_1 \cdot 2^n + c_2 \cdot 1^n = c_1 \cdot 2^n + c_2 \in O(2^n)$$

b) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 2T(n-1)+n & \text{en otro caso} \end{cases}$

Opción 1. Desarrolo en series.

Desarrollando en series, tenemos que:

$$\begin{aligned} T(n) &= 2T(n-1)+n = 2(2T(n-2)+n-1)+n = 2^2T(n-2)+2(n-1)+n = \\ &= 2^3T(n-3)+2^2(n-2)+2(n-1)+n = \ldots = \\ &= 2^iT(n-i)+2^{i-1}(n-i+1)+\ldots+2(n-1)+1\cdot n = \\ &= 2^iT(n-i)+\sum_{j=0}^{i-1}2^j(n-j) \qquad \forall i\in\mathbb{N}, i\geqslant n \end{aligned}$$

Para $i=n$, se tiene que:

$$\begin{aligned} T(n) &= 2^nT(0)+\sum_{j=0}^{n-1}2^j(n-j) = \sum_{j=0}^{n}2^j(n-j)-2^n(n-n) = \\ &= \sum_{j=0}^{n}2^jn-\sum_{j=0}^{n}2^jj = n\sum_{j=0}^{n}2^j-\sum_{j=0}^{n}j2^j \end{aligned}$$

Como vemos, la suma de la derecha no es de cálculo sencillo, por lo que vamos a intentar resolver la recurrencia por otro método.

Opción 2. Ecuación Característica.

Aplicando el cambio de variable $T(n)=x^n$, en la parte homogénea tenemos que:

$$x^n-2x^{n-1}=0 \Longrightarrow x^{n-1}(x-2)=0$$

Como $x^{n-1}>0$ para todo $n\in\mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x=2$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $n=1^n\cdot n^1$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x)=(x-2)(x-1)^2$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(n)=x^n=c_1\cdot 2^n+c_2\cdot 1^n+c_3\cdot n\cdot 1^n=c_1\cdot 2^n+c_2+c_3\cdot n\in O(2^n)$$

c) $T(n)=\begin{cases} 0 & \text{si } n=0 \\ 1 & \text{si } n=1 \\ T(n-1)+T(n-2) & \text{en otro caso} \end{cases}$

Aplicamos el cambio de variable $T(n)=x^n$, en la parte homogénea tenemos que:

$$x^n-x^{n-1}-x^{n-2}=0 \Longrightarrow x^{n-2}(x^2-x-1)=0$$

Resolviendo la ecuación de segundo grado, se tiene que:

$$x^2-x-1=0 \Longrightarrow x=\frac{1\pm\sqrt{5}}{2}$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(n)=x^n=c_1\left(\frac{1+\sqrt{5}}{2}\right)^n+c_2\left(\frac{1-\sqrt{5}}{2}\right)^n\in O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$

d) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ 3T(n-1) + 4T(n-2) & \text{en otro caso} \end{cases}$

Aplicamos el cambio de variable $T(n) = x^n$, en la parte homogénea tenemos que:

$$x^n - 3x^{n-1} - 4x^{n-2} = 0 \Longrightarrow x^{n-2}(x^2 - 3x - 4) = 0$$

Resolviendo la ecuación de segundo grado, se tiene que:

$$x^2 - 3x - 4 = 0 \Longrightarrow x = \frac{3 \pm \sqrt{25}}{2} = \frac{3 \pm 5}{2} = \begin{cases} x_1 = 4 \\ x_2 = -1 \end{cases}$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(n) = x^n = c_1 \cdot 4^n + c_2 \cdot (-1)^n \in O(4^n)$$

e) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ 5T(n-1) - 8T(n-2) + 4T(n-3) & \text{en otro caso} \end{cases}$

Aplicamos el cambio de variable $T(n) = x^n$, en la parte homogénea tenemos que:

$$x^n - 5x^{n-1} + 8x^{n-2} - 4x^{n-3} = 0 \Longrightarrow x^{n-3}(x^3 - 5x^2 + 8x - 4) = 0$$

Resolviendo la ecuación de tercer grado, se tiene que:

$$\begin{array}{r|rrrr} & 1 & -5 & 8 & -4 \\ 1 & & 1 & -4 & 4 \\ \hline & 1 & -4 & 4 & 0 \end{array}$$

Figura 3.1: División mediante Ruffini donde se ve que $x = 1$ es una solución.

Por tanto, tenemos que el polinomio característico queda:

$$(x-1)(x^2 - 4x + 4) = (x-1)(x-2)^2$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(n) = x^n = c_1 \cdot 1^n + c_2 \cdot 2^n + c_3 \cdot n \cdot 2^n \in O(n2^n)$$

f) $T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 36 & \text{si } n = 1 \\ 5T(n-1) + 6T(n-2) + 4 \cdot 3^n & \text{en otro caso} \end{cases}$

Aplicamos el cambio de variable $T(n) = x^n$, en la parte homogénea tenemos que:

$$x^n - 5x^{n-1} - 6x^{n-2} = 0 \Longrightarrow x^{n-2}(x^2 - 5x - 6) = 0$$

Resolviendo la ecuación de segundo grado, se tiene que:

$$x^2 - 5x - 6 = 0 \Longrightarrow x = \frac{5 \pm \sqrt{49}}{2} = \frac{5 \pm 7}{2} = \left\{ \begin{array}{l} x_1 = 6 \\ x_2 = -1 \end{array} \right.$$

Respecto a la parte no homogénea, se tiene que $4 \cdot 3^n = 3^n(4n^0)$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-6)(x+1)(x-3)$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(n) = x^n = c_1 \cdot 6^n + c_2 \cdot (-1)^n + c_3 \cdot 3^n \in O(6^n)$$

g) $T(n) = 2T(n-1) + 3^n$.

Aplicamos el cambio de variable $T(n) = x^n$, en la parte homogénea tenemos que:

$$x^n - 2x^{n-1} = 0 \Longrightarrow x^{n-1}(x-2) = 0$$

Como $x^{n-1} > 0$ para todo $n \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 2$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $3^n = 3^n n^0$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-2)(x-1)$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(n) = x^n = c_1 \cdot 2^n + c_2 \cdot 1^n = c_1 \cdot 2^n + c_2 \in O(2^n)$$

h) $T(n) = 2T(n-1) + n + 2^n$.

Aplicamos el cambio de variable $T(n) = x^n$, en la parte homogénea tenemos que:

$$x^n - 2x^{n-1} = 0 \Longrightarrow x^{n-1}(x-2) = 0$$

Como $x^{n-1} > 0$ para todo $n \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 2$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $n + 2^n = 1^n \cdot n + 2^n n^0$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-2)(x-1)^2(x-2) = (x-2)^2(x-1)^2$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(n) = x^n = c_1 \cdot 2^n + c_2 \cdot n \cdot 2^n + c_3 \cdot 1^n + c_4 \cdot n \cdot 1^n = c_1 \cdot 2^n + c_2 \cdot n2^n + c_3 + c_4 \cdot n \in O(n2^n)$$

i) $T(n) = 2T\left(n/2\right) + \log n$.

Aplicamos el cambio de variable $n = 2^m$, y tenemos que:

$$T(2^m) = 2T\left(2^{m-1}\right) + m\log 2$$

Aplicamos el cambio de variable $T(2^m) = x^m$, en la parte homogénea tenemos que:

$$x^m - 2x^{m-1} = 0 \Longrightarrow x^{m-1}(x-2) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 2$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $m\log 2 = 1^m \cdot (\log_2 m^1)$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-2)(x-1)^2$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(2^m) = x^m = c_1 \cdot 2^m + c_2 \cdot 1^m + c_3 \cdot m \cdot 1^m = c_1 \cdot 2^m + c_2 + c_3 \cdot m$$

Deshaciendo el cambio de variable $m = \log_2 n$, se tiene que:

$$T(n) = c_1 \cdot 2^{\log_2 n} + c_2 + c_3 \cdot \log_2 n = c_1 \cdot n + c_2 + c_3 \cdot \log_2 n \in O(n)$$

j) $T(n) = 4T\left(n/2\right) + n$.

Aplicamos el cambio de variable $n = 2^m$, y tenemos que:

$$T(2^m) = 4T\left(2^{m-1}\right) + 2^m$$

Aplicamos el cambio de variable $T(2^m) = x^m$, en la parte homogénea tenemos que:

$$x^m - 4x^{m-1} = 0 \Longrightarrow x^{m-1}(x-4) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 4$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $2^m = 2^m \cdot (m^0)$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-4)(x-2)$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(2^m) = x^m = c_1 \cdot 4^m + c_2 \cdot 2^m$$

Deshaciendo el cambio de variable $m = \log_2 n$, se tiene que:

$$T(n) = c_1 \cdot 4^{\log_2 n} + c_2 \cdot 2^{\log_2 n} = c_1 \cdot n^2 + c_2 \cdot n \in O(n^2)$$

k) $T(n) = 4T\left(n/2\right) + n^2$.

Aplicamos el cambio de variable $n = 2^m$, y tenemos que:

$$T(2^m) = 4T\left(2^{m-1}\right) + 2^{2m}$$

Aplicamos el cambio de variable $T(2^m) = x^m$, en la parte homogénea tenemos que:

$$x^m - 4x^{m-1} = 0 \Longrightarrow x^{m-1}(x-4) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 4$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $2^{2m} = 4^m \cdot (m^0)$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

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

Por tanto, la solución general de la ecuación en diferencias es:

$$T(2^m) = x^m = c_1 \cdot 4^m + c_2 \cdot m \cdot 4^m$$

Deshaciendo el cambio de variable $m = \log_2 n$, se tiene que:

$$T(n) = c_1 \cdot 4^{\log_2 n} + c_2 \cdot \log_2 n \cdot 4^{\log_2 n} = c_1 \cdot n^2 + c_2 \cdot n^2 \log_2 n \in O(n^2 \log n)$$

l) $T(n) = 2T\left(n/2\right) + n \log n$.

Aplicamos el cambio de variable $n = 2^m$, y tenemos que:

$$T(2^m) = 2T\left(2^{m-1}\right) + 2^m \cdot m \log 2$$

Aplicamos el cambio de variable $T(2^m) = x^m$, en la parte homogénea tenemos que:

$$x^m - 2x^{m-1} = 0 \Longrightarrow x^{m-1}(x-2) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 2$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $2^m \cdot m \log 2 = 2^m \cdot \log_2 m^1$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-2)(x-2)^2 = (x-2)^3$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(2^m) = x^m = c_1 \cdot 2^m + c_2 \cdot m \cdot 2^m + c_3 \cdot m^2 \cdot 2^m$$

Deshaciendo el cambio de variable $m = \log_2 n$, se tiene que:

$$\begin{aligned} T(n) &= c_1 \cdot 2^{\log_2 n} + c_2 \cdot \log_2 n \cdot 2^{\log_2 n} + c_3 \cdot \log_2^2 n \cdot 2^{\log_2 n} = \\ &= c_1 \cdot n + c_2 \cdot n \log_2 n + c_3 \cdot n(\log_2 n)^2 \in O(n \log^2 n) \end{aligned}$$

m) $T(n) = \begin{cases} 1 & \text{si } n = 2 \\ 2T\left(\sqrt{n}\right) + \log n & \text{si } n \geqslant 4 \end{cases}$

Aplicamos el cambio de variable $n = 2^{(2^m)}$, y tenemos que:

$$T\left(2^{(2^m)}\right) = 2T\left(2^{(2^{m-1})}\right) + 2^m$$

Aplicamos el cambio de variable $T\left(2^{(2^m)}\right) = x^m$, en la parte homogénea tenemos que:

$$x^m - 2x^{m-1} = 0 \Longrightarrow x^{m-1}(x-2) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 2$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $2^m = 2^m \cdot m^0$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-2)(x-2) = (x-2)^2$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T\left(2^{(2^m)}\right) = x^m = c_1 \cdot 2^m + c_2 \cdot m \cdot 2^m$$

Deshaciendo el cambio de variable $m = \log_2 \log_2 n$, se tiene que:

$$\begin{aligned} T(n) &= c_1 \cdot 2^{\log_2 \log_2 n} + c_2 \cdot \log_2 \log_2 n \cdot 2^{\log_2 \log_2 n} = \\ &= c_1 \cdot \log_2 n + c_2 \cdot \log_2 n \cdot \log_2 \log_2 n \in O(\log n \cdot \log \log n) \end{aligned}$$

n) $T(n) = \begin{cases} 1 & \text{si } n = 2 \\ 2T\left(\sqrt{n}\right) + \log \log n & \text{si } n \geqslant 4 \end{cases}$

Aplicamos el cambio de variable $n = 2^{(2^m)}$, y tenemos que:

$$T\left(2^{(2^m)}\right) = 2T\left(2^{(2^{m-1})}\right) + m$$

Aplicamos el cambio de variable $T\left(2^{(2^m)}\right) = x^m$, en la parte homogénea tenemos que:

$$x^m - 2x^{m-1} = 0 \Longrightarrow x^{m-1}(x-2) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 2$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $m = 1^m \cdot m^1$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-2)(x-1)^2$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T\left(2^{(2^m)}\right) = x^m = c_1 \cdot 2^m + c_2 \cdot 1^m + c_3 \cdot m \cdot 1^m = c_1 \cdot 2^m + c_2 + c_3 \cdot m$$

Deshaciendo el cambio de variable $m = \log_2 \log_2 n$, se tiene que:

$$\begin{aligned} T(n) &= c_1 \cdot 2^{\log_2 \log_2 n} + c_2 + c_3 \cdot \log_2 \log_2 n = \\ &= c_1 \cdot \log_2 n + c_2 + c_3 \cdot \log_2 \log_2 n \in O(\log \log n) \end{aligned}$$

o) $T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 5T\left(n/2\right) + (n\log n)^2 & \text{si } n \geqslant 2 \end{cases}$

Aplicamos el cambio de variable $n = 2^m$, y tenemos que:

$$T(2^m) = 5T\left(2^{m-1}\right) + (2^m \log 2^m)^2 = 5T\left(2^{m-1}\right) + (m2^m)^2 = 5T\left(2^{m-1}\right) + m^2 \cdot 4^m$$

Aplicamos el cambio de variable $T(2^m) = x^m$, en la parte homogénea tenemos que:

$$x^m - 5x^{m-1} = 0 \Longrightarrow x^{m-1}(x-5) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 5$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $m^2 \cdot 4^m = 4^m \cdot (m^2)$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-5)(x-4)^3$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(2^m) = x^m = c_1 \cdot 5^m + c_2 \cdot 4^m + c_3 \cdot m \cdot 4^m + c_4 \cdot m^2 \cdot 4^m$$

Deshaciendo el cambio de variable $m = \log_2 \log_2 n$, se tiene que:

$$\begin{aligned} T(n) &= c_1 \cdot 5^{\log_2 \log_2 n} + c_2 \cdot 4^{\log_2 \log_2 n} + c_3 \cdot \log_2 \log_2 n \cdot 4^{\log_2 \log_2 n} + \\ &\quad + c_4 \cdot (\log_2 \log_2 n)^2 \cdot 4^{\log_2 \log_2 n} = \\ &= c_1 \cdot (\log_2 n)^{\log_2 5} + c_2 \cdot (\log_2 n)^{\log_2 4} + c_3 \cdot \log_2 \log_2 n \cdot (\log_2 n)^{\log_2 4} + \\ &\quad + c_4 \cdot (\log_2 \log_2 n)^2 \cdot (\log_2 n)^{\log_2 4} = \\ &= c_1 \cdot (\log_2 n)^{\log_2 5} + c_2 \cdot (\log_2 n)^2 + c_3 \cdot \log_2 \log_2 n \cdot (\log_2 n)^2 + \\ &\quad + c_4 \cdot (\log_2 \log_2 n)^2 \cdot (\log_2 n)^2 \end{aligned}$$

Por tanto, tenemos que $T(n) \in O\left((\log(\log(n)) \cdot \log(n))^2\right)$.

p) $T(n) = \sqrt{n}T\left(\sqrt{n}\right) + n, \qquad n \geqslant 4.$

q) $T(n) = \begin{cases} 6 & \text{si } n = 1 \\ nT^2\left(n/2\right) & \text{si } n > 1 \end{cases}.$

r) $T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 4 & \text{si } n = 2 \\ T\left(n/2\right) \cdot T^2\left(n/2\right) & \text{si } n \geqslant 4 \end{cases}$

Ejercicio 3.1.5. El tiempo de ejecución de un Algotimo A viene descrito por la recurrencia

$$T(n) = 7T\left(n/2\right) + n^2$$

Otro algoritmo B tiene un tiempo de ejecución descrito por la recurrencia

$$T'(n) = aT'\left(n/4\right) + n^2$$

¿Cuál es el mayor valor de la constante $a \in \mathbb{R}$ que hace al algoritmo B asintóticamente más eficiente que A?

Estudiamos en primer lugar la eficiencia del algoritmo A. Aplicamos el cambio de variable $n = 2^m$, y tenemos que:

$$T(2^m) = 7T\left(2^{m-1}\right) + 4^m$$

Aplicamos el cambio de variable $T(2^m) = x^m$, en la parte homogénea tenemos que:

$$x^m - 7x^{m-1} = 0 \Longrightarrow x^{m-1}(x-7) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = 7$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $4^m = 4^m \cdot (m^0)$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-7)(x-4)$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(2^m) = x^m = c_1 \cdot 7^m + c_2 \cdot 4^m$$

Deshaciendo el cambio de variable $m = \log_2 n$, se tiene que:

$$T(n) = c_1 \cdot 7^{\log_2 n} + c_2 \cdot 4^{\log_2 n} = c_1 \cdot n^{\log_2 7} + c_2 \cdot n^{\log_2 4} = c_1 \cdot n^{\log_2 7} + c_2 \cdot n^2 \in O(n^{\log_2 7})$$

Estudiamos en segundo lugar la eficiencia del algoritmo B. Aplicamos el cambio de variable $n = 4^m$, y tenemos que:

$$T'(4^m) = aT'\left(4^{m-1}\right) + 16^m$$

Aplicamos el cambio de variable $T'(4^m) = x^m$, en la parte homogénea tenemos que:

$$x^m - ax^{m-1} = 0 \Longrightarrow x^{m-1}(x-a) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = a$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $16^m = 16^m \cdot (m^0)$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-a)(x-16)$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T'(4^m) = x^m = c_1 \cdot a^m + c_2 \cdot 16^m$$

Deshaciendo el cambio de variable $m = \log_4 n$, se tiene que:

$$T'(n) = c_1 \cdot a^{\log_4 n} + c_2 \cdot 16^{\log_4 n} = c_1 \cdot n^{\log_4 a} + c_2 \cdot n^2 \in O\left(n^{\text{máx}\{2, \log_4 a\}}\right)$$

Para que B sea asintóticamente más eficiente que A, es necesario que:

$$\text{máx}\{2, \log_4 a\} < \log_2 7 \Longleftrightarrow \log_4 a < \log_2 7 \Longleftrightarrow a < 7^{\log_2 4} = 7^2 = 49$$

Por tanto, los valores que hacen que el algoritmo B sea asintóticamente más eficiente que A son aquellos $a \in \mathbb{R}$ tales que $a < 49$.

Ejercicio 3.1.6. Resuelva la siguiente recurrencia:

$$T(n) = aT\left(\frac{n}{b}\right) + n^k$$

con $a, b, k \in \mathbb{R}$, $a \geqslant 1,\ b \geqslant 2,\ k \geqslant 0$.

Aplicamos el cambio de variable $n = b^m$, y tenemos que:

$$T(b^m) = aT\left(b^{m-1}\right) + b^{km} = aT\left(b^{m-1}\right) + (b^k)^m$$

Aplicamos el cambio de variable $T(b^m) = x^m$, en la parte homogénea tenemos que:

$$x^m - ax^{m-1} = 0 \Longrightarrow x^{m-1}(x-a) = 0$$

Como $x^{m-1} > 0$ para todo $m \in \mathbb{N}$, se tiene que la única solución del polinomio caractertístico es $x = a$, con multiplicidad simple. Añadiendo la parte no homogénea, se tiene que $(b^k)^m = (b^k)^m \cdot (m^0)$, por lo que el polinomio caractertístico de la ecuación en diferencias es:

$$p(x) = (x-a)(x-b^k)$$

Por tanto, la solución general de la ecuación en diferencias es:

$$T(b^m) = x^m = c_1 \cdot a^m + c_2 \cdot b^{km}$$

Deshaciendo el cambio de variable $m = \log_b n$, se tiene que:

$$T(n) = c_1 \cdot a^{\log_b n} + c_2 \cdot b^{k \log_b n} = c_1 \cdot n^{\log_b a} + c_2 \cdot n^k \in O\left(n^{\text{máx}\{k, \log_b a\}}\right)$$