UNIVERSIDADE DA CORUÑA

Algoritmos:
Análisis de algoritmos

Alberto Valderruten

LFCIA - Departamento de Computación

Facultad de Informática

Universidad de A Coruña, España

www.lfcia.org/alg

www.fi.udc.es

Contenido

- Análisis de la eficiencia de los algoritmos
- Notaciones asintóticas
- Cálculo de los tiempos de ejecución
- Resolución de recurrencias

1.1 Análisis de la eficiencia de los algoritmos (1)

- **Objetivo:** *Predecir* el *comportamiento* del algoritmo
  $\Rightarrow$ aspectos cuantitativos:
  - tiempo de ejecución
  - cantidad de memoria

- Disponer de una *medida* de su eficiencia:
  - "teórica"
  - no exacta: *aproximación* suficiente para *comparar*, *clasificar*
  $\Rightarrow$ acotar $T(n)$: tiempo de ejecución, $n =$ *tamaño* de la entrada
  $n \to \infty$: *comportamiento asintótico*
  $\Rightarrow T(n) = O(f(n))$
  $f(n)$: una **cota superior** de $T(n)$ suficientemente ajustada
  $f(n)$ crece más deprisa que $T(n)$

- Aproximación?
  1. Ignorar factores constantes:
     20 multiplicaciones por iteración $\to$ 1 operación por iteración
     ¿cuántas iteraciones? $\to$ iteraciones en función de n
  2. Ignorar términos de orden inferior: $n + cte \to n$

1.1 Análisis de la eficiencia de los algoritmos (2)

- **Ejemplo 1**: 2 algoritmos (A1 y A2) para un mismo problema A:
  - algoritmo A1: $100n$ pasos $\rightarrow$ un recorrido de la entrada
    $T(n) = O(n)$: algoritmo *lineal*
  - algoritmo A2: $2n^2 + 50$ pasos $\rightarrow$ n recorridos de la entrada
    $T(n) = O(n^2)$: algoritmo *cuadrático*
  - *Comparar*: A2 "más lento" que A1, aunque con $n \leq 49$ sea más rápido
    $\Rightarrow$ **A1 es mejor**
  - *Clasificar*: lineales, cuadráticos...
    Tasas de crecimiento características:
    $O(1), O(logn), O(n), O(nlogn), O(n^2), O(n^3), ...O(2^n), ...$

- **Ejemplo 2**: aproximación $\Rightarrow$ limitaciones:
  2 algoritmos (B1 y B2) para un mismo problema B:
  - algoritmo B1: $2n^2 + 50$ pasos $\rightarrow O(n^2)$
  - algoritmo B2: $100n^{1,8}$ pasos $\rightarrow O(n^{1,8})$
    $\Rightarrow$ B2 es "mejor"...
    pero a partir de algún valor de n entre 310 y $320 * 10^6$

1.2 Notaciones asintóticas (1)

- **Objetivo:** Establecer un orden relativo entre las funciones, comparando sus tasas de crecimiento

- **La notación O:**
  $T(n), f(n) : Z^+ \rightarrow R^+$

  Definición:
  $T(n) = O(f(n))$ si $\exists$ constantes $c > 0$ y $n_0 > 0$: $T(n) \leq c * f(n)$ $\forall n \geq n_0$
  $\square$

  n_0: umbral
  $T(n)$ es $O(f(n))$, $T(n) \in O(f(n))$
  "la tasa de crecimiento de $T(n) \leq$ que la de $f(n)$"
  $\rightarrow$ $f(n)$ es *una* cota superior de $T(n)$

- **Ejemplo:** ¿$5n^2 + 15 = O(n^2)$?
  $5n^2 + 15 \leq 6n^2$ $\forall n \geq 4$ ($< c, n_0 >=< 6, 4 >$ en la definición)
  pero además $\exists$ infinitos $< c, n_0 >$ que satisfacen la desigualdad

1.2 Notaciones asintóticas (2) - La notación O

- **Observación 1**: Según la definición, $T(n)$ podría estar muy por debajo:
  ¿$5n^2 + 15 = O(n^3)$?
  $5n^2 + 15 \leq 1n^3 \ \forall n \geq 6$ ($< c, n_0 >=< 1, 6 >$ en la definición)
  pero es más preciso decir $= O(n^2) \equiv$ ajustar cotas
  $\Rightarrow$ **Para el análisis de algoritmos, usar las aproximaciones vistas:**
  $5n^2 + 4n \rightarrow O(n^2)$
  $log_2 n \rightarrow O(logn)$
  $13 \rightarrow O(1)$
  ...

- **Observación 2**: La notación O también se usa en expresiones como
  $3n^2 + O(n)$

- **Ejemplo 1**:
  ¿Cómo se consigue una mejora más drástica,
  - mejorando la eficiencia del algoritmo, o
  - mejorando el ordenador?

1.2 Notaciones asintóticas (3) - La notación O

- **Ejemplo 1** (cont.):

$T(n)$	tiempo$_1$ 1000 pasos/s	tiempo$_2$ 2000 pasos/s	tiempo$_3$ 4000 pasos/s	tiempo$_4$ 8000 pasos/s
$log_2 n$	0,010	0,005	0,003	0,001
n	1	0,5	0,25	0,125
$nlog_2 n$	10	5	2,5	1,25
$n^{1,5}$	32	16	8	4
n^2	1.000	500	250	125
n^3	1.000.000	500.000	250.000	125.000
$1,1^n$	10^{39}	10^{39}	10^{38}	10^{38}

Tabla: Tiempos de ejecución (en s) para 7 algoritmos de distinta complejidad (n=1000).

- **Ejemplo 2**: Ordenar 100.000 enteros aleatorios:
  - 17 s en un 386 utilizando Quicksort
  - 17 min en un procesador 100 veces más rápido utilizando Burbuja

1.2 Notaciones asintóticas (4) - La notación O

- **Verificación empírica del análisis**:

  "Método empírico":

  medir tiempos de ejecución (experimentos sistemáticos)

  $\Rightarrow$ tabla de tiempos para distintos valores de n

  $\Rightarrow$ ¿O?

  Método empírico: Renacimiento, s. XVII

  "Mide lo que se pueda medir, lo que no se pueda... hazlo medible!"

- Verificación empírica: se parte de una función $f(n)$ candidata.

- **Ref:** Trabajo en prácticas

1.2 Notaciones asintóticas (5) - La notación O

- **Reglas prácticas** para trabajar con la O:

Definición:
$f(n)$ es **monótona creciente** si $n_1 \geq n_2 \Rightarrow f(n_1) \geq f(n_2)$
□

Teorema: $\forall c > 0, a > 1, f(n)$ monótona creciente:

$$(f(n))^c = O(a^{f(n)})$$

$\equiv$ *"Una función exponencial (ej: 2^n) crece más rápido que una función polinómica (ej: n^2)"*

$$\rightarrow \begin{cases} n^c = O(a^n) \\ (log_a n)^c = O(a^{log_a n}) = O(n) \end{cases}$$

$\rightarrow (logn)^k = O(n)\ \forall k$ constante.

$\equiv$ *"n crece más rápido que cualquier potencia de logaritmo"*

$\equiv$ *"los logaritmos crecen muy lentamente"*

1.2 Notaciones asintóticas (6) - La notación O

- **Reglas prácticas** para trabajar con la O (Cont.):

  Suma y multiplicación:
  $T_1(n) = O(f(n)) \wedge T_2(n) = O(g(n)) \Rightarrow$

  $$\begin{cases} (1) & T_1(n) + T_2(n) = O(f(n) + g(n)) = max(O(f(n)), O(g(n))) \\ (2) & T_1(n) * T_2(n) = O(f(n) * g(n)) \end{cases}$$

  Aplicación: $\begin{cases} (1) \text{ Secuencia:} & 2n^2 = O(n^2) \wedge 10n = O(n) \Rightarrow 2n^2 + 10n = O(n^2) \\ (2) \text{ Bucles} \end{cases}$

  Observación: **No** extender la regla para la resta ni para la división
  $\leftarrow$ relación $\leq$ en la definición de la O

- ... suficientes para ordenar la mayoría de las funciones.

1.2 Notaciones asintóticas (7) - Otras notaciones asintóticas

1. O

2. **Definición**:
   $T(n) = \Omega(f(n))$ ssi $\exists$ constantes c y n_0: $T(n) \geq cf(n)$ $\forall n \geq n_0$,
   $T(n), f(n) : Z^+ \rightarrow R^+$
   $\square$
   $\rightarrow f(n)$: **cota inferior** de $T(n) \equiv$ trabajo mínimo que realiza un algoritmo
   Ejemplo: $3n^2 = \Omega(n^2)$: es la cota inferior más ajustada;
   pero también $3n^2 = O(n^2)$...

3. **Definición**:
   $T(n) = \Theta(f(n))$ ssi $\exists$ constantes c_1, c_2 y n_0: $c_1 f(n) \leq T(n) \leq c_2 f(n)$
   $\forall n \geq n_0$, $T(n), f(n) : Z^+ \rightarrow R^+$
   $\square$
   $\equiv O \wedge \Omega$
   $\rightarrow f(n)$: **cota exacta** de $T(n)$, del orden exacto
   Ejemplo: $5nlog_2n - 10 = \Theta(nlogn)$: $\begin{cases} (1) \text{ demostrar } O \rightarrow < c, n_0 > \\ (2) \text{ demostrar } \Omega \rightarrow < c', n_0' > \end{cases}$

1.2 Notaciones asintóticas (8) - Otras notaciones asintóticas

4. **Definición**:
   $T(n) = o(f(n))$ ssi $\forall$ constante $C > 0$, $\exists n_0 > 0$: $T(n) < Cf(n)$ $\forall n \geq n_0$,
   $T(n), f(n) : Z^+ \rightarrow R^+$
   $\square$
   $\equiv O \wedge \neg\Theta \equiv O \wedge \neg\Omega$
   $\rightarrow f(n)$: **cota estrictamente superior** de $T(n) \equiv \lim_{n\rightarrow\infty} \frac{T(n)}{f(n)} = 0$
   Ejemplos: $\frac{n}{log_2 n} = o(n)$ $\quad$ $\frac{n}{10} \neq o(n)$

5. **Definición**:
   $T(n) = \omega(f(n))$ ssi $\forall$ constante $C > 0$, $\exists n_0 > 0$: $T(n) > Cf(n)$ $\forall n \geq n_0$,
   $T(n), f(n) : Z^+ \rightarrow R^+$
   $\square$
   $\leftrightarrow f(n) = o(T(n))$
   $\rightarrow f(n)$: **cota estrictamente inferior** de $T(n)$

6. **Notación OO** [Manber]:
   $T(n) = OO(f(n))$ si es $O(f(n))$ pero con constantes demasiado grandes para casos prácticos
   Ref: ejemplo 2 (p. 4): $B1 = O(n^2)$, $B2 = OO(n^{1,8})$

1.2 Notaciones asintóticas (9) - Otras notaciones asintóticas

Reglas prácticas (Cont.):

1. $T(n) = a_0 + a_1 n + a_2 n^2 + ... + a_k n^k \Rightarrow T(n) = \Theta(n^k)$
   (polinomio de grado k)

2. **Teorema**: $\forall c > 0, a > 1, f(n)$ monótona creciente:

$$(f(n))^c = o(a^{f(n)})$$

   $\equiv$ *"Una función exponencial* ***crece más rápido*** *que una función polinómica"* $\rightarrow$ no llegan a igualarse

1.3 Cálculo de los tiempos de ejecución (1) - Modelo de computación

- Calcular O para $T(n) \equiv$ contar número de "pasos" $\rightarrow f(n)$? *¿paso?*

- **Modelo de computación**:
  - ordenador secuencial
  - instrucción $\leftrightarrow$ paso (no hay instrucciones complejas: matrices...)
  - entradas: tipo único ("entero") $\rightarrow$ sec(n)
  - memoria infinita + *"todo está en memoria"*

- Alternativas: Un paso es...

1. **Operación elemental**:
   Operación cuyo tiempo de ejecución está acotado superiormente por una constante que sólo depende de la implementación $\rightarrow = O(1)$
2. **Operación principal** [Manber]:
   Operación *representativa* del trabajo del algoritmo
   Ejemplo: la comparación en un algoritmo de ordenación
   El número de operaciones principales que se ejecutan debe ser *proporcional* al número total de operaciones (verificarlo!)

1.3 Cálculo de los tiempos de ejecución (2) - Modelo de computación

- **Observación**: La hipótesis de la operación principal supone una aproximación mayor.

- En general, se trabaja usando la hipótesis de la operación elemental.

- En cualquier caso, se ignora: lenguaje de programación, procesador, sistema operativo, carga...
  $\Rightarrow$ Sólo se considera el algoritmo y el tamaño de la entrada

- **Debilidades**:
  - operaciones de coste diferente
    ("todo en memoria" $\Rightarrow$ lectura en disco = asignación)
    $\rightarrow$ contar separadamente según tipo de instrucción y luego ponderar?
    factores $\equiv$ dependiente de la implementación
    $\Rightarrow$ costoso y generalmente inútil
  - faltas de página ignoradas
  - etc.

  $\rightarrow$ *Aproximación*

1.3 Cálculo de los tiempos de ejecución (3) - Análisis de casos

- **Análisis de casos**:

  Consideramos distintas funciones para $T(n)$:

  $$\begin{cases} T_{mejor}(n) & \\ T_{medio}(n) & \leftarrow \text{representativa, puede más complicada de obtener} \\ T_{peor}(n) & \leftarrow \text{en general, la más utilizada} \end{cases}$$

  $$T_{mejor}(n) \leq T_{medio}(n) \leq T_{peor}(n)$$

- ¿El tiempo de respuesta es crítico?
  $\rightarrow$ *Sistemas de Tiempo Real*

Ordenación por Inserción

```
procedimiento Ordenación por Inserción (var T[1..n])
    para i:=2 hasta n hacer
        x:=T[i];
        j:=i-1;
        mientras j>0 y T[j]>x hacer
            T[j+1]:=T[j];
            j:=j-1
        fin mientras;
        T[j+1]:=x
    fin para
fin procedimiento
```

3	1	4	1	2	9	5	6	5	3
1	3	4	1	2	9	5	6	5	3
1	3	**4**	1	2	9	5	6	5	3
1	**1**	3	4	2	9	5	6	5	3
1	1	**2**	3	4	9	5	6	5	3
1	1	2	3	4	**9**	5	6	5	3
1	1	2	3	4	**5**	9	6	5	3
1	1	2	3	4	5	**6**	9	5	3
1	1	2	3	4	5	**5**	6	9	3
1	1	2	3	**3**	4	5	5	6	9

Ordenación por Selección

```
procedimiento Ordenación por Selección (var T[1..n])
    para i:=1 hasta n-1 hacer
        minj:=i;
        minx:=T[i];
        para j:=i+1 hasta n hacer
            si T[j]<minx entonces
                minj:=j;
                minx:=T[j]
            fin si
        fin para;
        T[minj]:=T[i];
        T[i]:=minx
    fin para
fin procedimiento
```

3	1	4	1	2	9	5	6	5	3
1	3	4	1	2	9	5	6	5	3
1	**1**	4	3	2	9	5	6	5	3
1	1	**2**	3	4	9	5	6	5	3
1	1	2	**3**	4	9	5	6	5	3
1	1	2	3	**3**	9	5	6	5	4
1	1	2	3	3	**4**	5	6	5	9
1	1	2	3	3	4	**5**	6	5	9
1	1	2	3	3	4	5	**5**	6	9
1	1	2	3	3	4	5	5	**6**	9

1.3 Cálculo de los tiempos de ejecución (4) - Análisis de casos

- **Ordenación por Inserción**
  - Peor caso → "insertar siempre en la primera posición"
    ≡ entrada en orden inverso
    ⇒ el bucle interno se ejecuta 1 vez en la primera iteración, 2 veces en la segunda, . . . , $n-1$ veces en la última iteración del bucle principal:
    ⇒ $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ iteraciones del bucle interno

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

    ⇒ $T(n) = \frac{n(n-1)}{2}c_1 + (n-1)c_2 + c_3$: polinomio de grado 2
    ⇒ $\boxed{T(n) = \Theta(n^2)}$
  - Mejor caso → "no insertar nunca" ≡ entrada ordenada
    ⇒ el bucle interno no se ejecuta
    ⇒ $T(n) = (n-1)c_1 + c_2$: polinomio de grado 1
    ⇒ $\boxed{T(n) = \Theta(n)}$

⇒ $T(n)$ depende *también* del estado inicial de la entrada

1.3 Cálculo de los tiempos de ejecución (5) - Análisis de casos

- **Ordenación por Selección**

  $\boxed{T(n) = \Theta(n^2)}$ sea cual sea el orden inicial (ejercicio)

  $\leftrightarrow$ la comparación `T[j]<minx` se ejecuta el mismo número de veces

  Empíricamente: los tiempos de ejecución no fluctúan más de 15 %

algoritmo	mínimo	máximo
Inserción	0,004	5,461
Selección	4,717	5,174

Tabla: Tiempos (en segundos) obtenidos para $n = 4000$

- **Comparación**:

algoritmo	peor caso	caso medio	mejor caso
Inserción	$\Theta(n^2)$	$\Theta(n^2)$	$\boxed{\Theta(n)}$
Selección	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$O(n^2)$	$\boxed{O(nlogn)}$	$O(nlogn)$

1.3 Cálculo de los tiempos de ejecución (6) - Análisis de casos

- **Ejemplo: exponenciación**
  - Potencia1: $x^n = x * x * ... * x$ (bucle, n veces x)
    $\Rightarrow$ **Operación principal**: multiplicación
    $\Rightarrow$ ¿Número de multiplicaciones? : $f_1(n) = n - 1 \Rightarrow T(n) = \Theta(n)$
  - Potencia2 (recursivo):

$$x^n = \begin{cases} x^{\lfloor n/2 \rfloor} * x^{\lfloor n/2 \rfloor} & \text{si n par} \\ x^{\lfloor n/2 \rfloor} * x^{\lfloor n/2 \rfloor} * x & \text{si n impar} \end{cases}$$

    $\Rightarrow$ ¿Número de multiplicaciones? : ¿$f_2(n)$?

$$\begin{cases} \text{mín: n par en cada llamada} & \rightarrow n = 2^k, k \in Z^+ \leftrightarrow \text{mejor caso} \\ \text{máx: n impar en cada llamada} & \rightarrow n = 2^k - 1, k \in Z^+ \leftrightarrow \text{peor caso} \end{cases}$$

    - *Mejor caso*: $f_2(2^k) = \begin{cases} 0 & \text{si } k = 0 \\ f_2(2^{k-1}) + 1 & \text{si } k > 0 \ (1) \end{cases}$
    - *Peor caso*: $f_2(2^k - 1) = \begin{cases} 0 & \text{si } k = 1 \\ f_2(2^{k-1} - 1) + 2 & \text{si } k > 1 \ (2) \end{cases}$

$\rightarrow$ *relaciones de recurrencia*

1.3 Cálculo de los tiempos de ejecución (7) - Análisis de casos

- Potencia2 (Cont.)

  - *Mejor caso*: $f_2(2^k) = \begin{cases} 0 & \text{si } k = 0 \\ f_2(2^{k-1}) + 1 & \text{si } k > 0 \ (1) \end{cases}$

$$\begin{array}{rccc} k = & 0 & \rightarrow f_2(& 1 \quad) = \quad 0 \\ & 1 & & 2 \qquad 1 \\ & 2 & & 4 \qquad 2 \\ & 3 & & 8 \qquad 3 \\ & \ldots & & \end{array}$$

$\Rightarrow$ Hipótesis de inducción: $\boxed{f_2(2^\alpha) = \alpha}$: $0 \leq \alpha \leq k-1$

Paso inductivo: $(1) \rightarrow f_2(2^k) = f_2(2^{k-1}) + 1 = (k-1) + 1 = k$

forma explícita correcta de la relación de rec.

1.3 Cálculo de los tiempos de ejecución (8) - Análisis de casos

- Potencia2 (Cont.)
  - *Peor caso*: $f_2(2^k-1) = \begin{cases} 0 & \text{si } k = 1 \\ f_2(2^{k-1}-1)+2 & \text{si } k > 1 \ (2) \end{cases}$

$$\begin{array}{rcccl} k = & 1 & \rightarrow f_2(& 1 &) = \quad 0 \\ & 2 & & 3 & \quad\ 2 \\ & 3 & & 7 & \quad\ 4 \\ & 4 & & 15 & \quad\ 6 \\ & 5 & & 31 & \quad\ 8 \\ & 6 & & 63 & \quad 10 \\ & \ldots & & & \end{array}$$

$\Rightarrow$ Hipótesis de inducción: $\boxed{f_2(2^\alpha - 1) = 2(\alpha - 1)}$: $1 \leq \alpha \leq k-1$

Paso inductivo: $\begin{aligned} (2) \rightarrow f_2(2^k-1) &= f_2(2^{k-1}-1)+2 \\ &= 2(k-1-1)+2 \\ &= 2(k-1) \end{aligned}$

1.3 Cálculo de los tiempos de ejecución (9) - Análisis de casos

- Potencia2 (Cont.)
  - $n = 2^k$ (mejor caso):
    $f_2(2^k) = k$ para $k \geq 0$
    $\rightarrow f_2(n) = log_2 n$ para $n = 2^k$ y $k \geq 0$ (ya que $log_2 2^k = k$)
    $\Rightarrow \boxed{f_2(n) = \Omega(logn)}$
  - $n = 2^k - 1$ (peor caso):
    $f_2(2^k - 1) = 2(k-1)$ para $k \geq 1$
    $\rightarrow f_2(n) = 2[log_2(n+1) - 1]$ para $n = 2^k - 1$ y $k \geq 1$
    $\Rightarrow \boxed{f_2(n) = O(logn)}$
  - $\Rightarrow f_2(n) = \Theta(logn)$
    Modelo de computación con operación principal = multiplicación
    $\Rightarrow \boxed{T(n) = \Theta(logn)}$

mejor caso $\leftrightarrow \Omega$
peor caso $\leftrightarrow O$

1.3 Cálculo de los tiempos de ejecución (10) - Reglas para calcular O

1. operación elemental $= 1 \leftrightarrow$ Modelo de Computación

2. **secuencia**: $S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
   $\Rightarrow \boxed{S_1; S_2} = O(f_1(n) + f_2(n)) = O(max(f_1(n), f_2(n)))$
   Observación: también con Θ

3. **condición**: $B = O(f_B(n)) \wedge S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
   $\Rightarrow \boxed{\textbf{si } B \textbf{ entonces } S_1 \textbf{ sino } S_2} = O(max(f_B(n), f_1(n), f_2(n)))$
   Observación: si $f_1(n) \neq f_2(n)$ y $max(f_1(n), f_2(n)) > f_B(n) \leftrightarrow$ **Peor caso**
   ¿Caso medio? $\rightarrow f(n)$: promedio de f_1 y f_2 ponderado con frecuencias de cada rama $\rightarrow O(max(f_B(n), f(n)))$

4. **iteración**: $B; S = O(f_{B,S}(n)) \wedge$ nº iter$= O(f_{iter}(n))$
   $\Rightarrow \boxed{\textbf{mientras } B \textbf{ hacer } S} = O(f_{B,S}(n) * f_{iter}(n))$
   ssi el coste de las iteraciones no varía, sino: $\sum$ costes individuales
   Caso particular: $\boxed{\textbf{para } i \leftarrow x \textbf{ hasta } y \textbf{ hacer } S} = O(f_S(n) *$ nº iter$)$
   B es comparar 2 enteros $= O(1)$; nº iter $= y - x + 1$

1.3 Cálculo de los tiempos de ejecución (11) - Reglas para calcular O

- Uso de las reglas:
  - análisis "de adentro hacia afuera"
  - analizar primero los subprogramas
  - recursividad: intentar tratarla como un ciclo, sino resolver relación de recurrencia

- **Ejemplo**: $\sum_{i=1}^{n} i^3$: ¿$T(n)$? ¿n?

```
    función suma (n:entero) : entero
{1}   s:=0;
{2}   para i:=1 hasta n hacer
{3}     s:=s+i*i*i;
{4}   devolver s
    fin función
```

$\Theta(1)$ en $\{3\}$ y no hay variaciones
$\Rightarrow \Theta(n)$ en $\{2\}$ (regla 4)
$\Rightarrow T(n) = \Theta(n)$ (regla 2)

- *Observación*: El razonamiento ya incluye las aproximaciones

1.3 Cálculo de los tiempos de ejecución (12) - Reglas para calcular O

- **Ejemplo: Ordenación por Selección**

  $\Theta(1)$ en {5} (regla 2)

  $\Rightarrow O(max(\Theta(1), \Theta(1), 0)) = \Theta(1)$ en {4}
  (regla 3: no estamos en peor caso)

  $\rightarrow S = \Theta(1)$; nº iter$=n - i \Rightarrow \Theta(n - i)$ en {3} (regla 4)

  $\Theta(1)$ en {2} y en {6} (regla 2)

  $\Rightarrow \Theta(n - i)$ en {2-6} (regla 2)

  $\rightarrow S = \Theta(n - i)$ **varía**: $\begin{cases} i = 1 & \rightarrow \Theta(n) \\ i = n - 1 & \rightarrow \Theta(1) \end{cases}$

  $\Rightarrow \sum_{i=1}^{n-1}(n - i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i$ en {1} (regla 4)

  $= (n - 1)n - \frac{n(n-1)}{2}$: polinomio de grado 2

  $\Rightarrow T(n) = \Theta(n^2)$ *en cualquier caso*

Ordenación por Selección

```
    procedimiento Ordenación por Selección (var T[1..n])
{1}     para i:=1 hasta n-1 hacer
{2}         minj:=i; minx:=T[i];
{3}         para j:=i+1 hasta n hacer
{4}             si T[j]<minx entonces
{5}                 minj:=j; minx:=T[j]
                fin si
            fin para;
{6}         T[minj]:=T[i]; T[i]:=minx
        fin para
    fin procedimiento
```

1.3 Cálculo de los tiempos de ejecución (13) - Reglas para calcular O

- **Ejemplo: Suma de la Subsecuencia Máxima**

  $a_1..a_n \rightarrow \sum_{k=i}^{j} a_k$ máxima?

  Ejemplo: $SSM(-2, 11, -4, 13, -5, -2) = 20$ [2..4]

1. SSM **recursiva**: estrategia Divide y Vencerás

   Divide la entrada en 2 *mitades* $\rightarrow$ 2 soluciones recursivas

   Vence usando las 2 soluciones $\rightarrow$ solución para la entrada original

   La SSM puede estar:
   - en la 1ª mitad
   - en la 2ª mitad
   - entre las 2 mitades

   Las dos primeras soluciones son las obtenidas recursivamente.

   La 3ª solución se obtiene sumando:

   - la SSM de la 1ª mitad *que incluye el extremo derecho*, y
   - la SSM de la 2ª mitad *que incluye el extremo izquierdo*.

2. SSM **en línea**

Suma de la Subsecuencia Máxima: SSM recursiva (1)

```
     función SSM ( a[1..n] ) : valor              {función interfaz}
       devolver SSM recursiva (a, 1, n)
     fin función

     función SSM recursiva (var a[1..n], izq, der) : valor
{1}    si izq = der entonces
{2}       si a[izq] > 0 entonces
{3}          devolver a[izq]                     {caso base: si >0, es SSM}
          sino
{4}          devolver 0
          fin si
       sino
{5}       Centro := (izq + der) div 2 ;
{6}       Primera solución := SSM recursiva (a, izq, Centro) ;
{7}       Segunda solución := SSM recursiva (a, Centro + 1, der) ;
```

Suma de la Subsecuencia Máxima: SSM recursiva (2)

```
{8}       Suma máxima izquierda := 0 ; Suma izquierda := 0 ;
{9}       para i := Centro hasta izq paso -1 hacer
{10}        Suma izquierda := Suma izquierda + a[i] ;
{11}        si Suma izquierda > Suma máxima izquierda entonces
{12}           Suma máxima izquierda := Suma izquierda
          fin para ;

{13}      Suma máxima derecha := 0 ; Suma derecha := 0 ;
{14}      para i := Centro + 1 hasta der hacer
{15}        Suma derecha := Suma derecha + a[i] ;
{16}        si Suma derecha > Suma máxima derecha entonces
{17}           Suma máxima derecha := Suma derecha
          fin para ;

{18}      devolver max (Primera solución, Segunda solución,
                        Suma máxima izquierda + Suma máxima derecha)
      fin si
    fin función
```

1.3 Cálculo de los tiempos de ejecución (14) - Reglas para calcular O

- **Complejidad temporal de la función SSM recursiva**:
  Caso base: $\{1\text{-}4\} \Rightarrow T(1) = \Theta(1)$
  Ciclos $\{9\text{-}12\}$ y $\{14\text{-}17\}$: $O(n)$ *en conjunto*: $a_1..a_n$
  Llamadas recursivas $\{6\}$ y $\{7\}$: $T(n/2)$ cada una (aproximación)
  Resto: $O(1)$: se puede ignorar frente a $O(n)$...

  Relación de recurrencia: $\begin{cases} T(1) = 1 \\ T(n) = 2T(n/2) + n, n > 1 (*) \end{cases}$

- "Intuitivamente":

$$\begin{array}{lllll} T(2) = 4 = 2 * 2 \\ 4 \quad 12 \quad 4 * 3 \\ 8 \quad 32 \quad 8 * 4 \\ 16 \quad 80 \quad 16 * 5 \\ \ldots \end{array}$$

$\Rightarrow T(n) = n(k+1)$ para $n = 2^k$
$\Rightarrow T(n) = n(log_2 n + 1) = O(nlogn)$

1.3 Cálculo de los tiempos de ejecución (15) - Reglas para calcular O

- **Complejidad temporal de la función SSM recursiva** (Cont.):
  - Manejando proyecciones:

a) dividir (*) por n:

b) proyectar ($n = 2^k$):

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1$$
$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + 1$$
$$\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + 1$$
$$\ldots$$
$$\frac{T(2)}{2} = \frac{T(1)}{1} + 1$$

c) sumar:
$$\frac{T(n)}{n} = T(1) + log_2 n \Rightarrow T(n) = O(nlogn)$$

  - Con teoremas de resolución de recurrencias: **Teorema Divide y Vencerás**

$T(n) = lT(n/b) + cn^k, n > n_0,$
con $l \geq 1$, $b \geq 2$, $c > 0 \in R$, $l \geq 0 \in N$, $n_0 \geq 1 \in N$
Caso $l = b^k \Rightarrow T(n) = \Theta(n^k logn)$ $(l = 2, b = 2, k = 1)$
$\Rightarrow T(n) = \Theta(nlogn)$

$\rightarrow$ Usar teoremas

1.3 Cálculo de los tiempos de ejecución (15) - Reglas para calcular O

- **Complejidad temporal de la función SSM recursiva** (Cont.):
  Observación: pasar el vector **por referencia** (var), sino:

  sea R(n): nº de copias del vector: $\begin{cases} R(1) = 0 \\ R(n) = 2R(n/2) + 2, n > 1 \end{cases}$

  $\Rightarrow R(n) = 2n - 2$ copias $*\Theta(n)$ cada una $\Rightarrow T(n) = \Theta(n^2)$

  También la complejidad espacial sería cuadrática!

- **Complejidad temporal de la función SSM en línea**:
  - Acceso secuencial
  - Respuesta para la subsecuencia parcial

  $\Rightarrow$ Algoritmo en línea

  Además: *espacio constante* (no es necesario memorizar la entrada)
  y tiempo lineal
  $\rightarrow$ *mejor imposible*

  Ejercicio: escribir el algoritmo para tener espacio constante

Suma de la Subsecuencia Máxima: SSM en línea

```
     función SSM en línea ( a[1..n] ) : <i, j, valor>
{1}    i := 1 ; EstaSuma := 0 ; SumaMax := 0 ; MejorI := 0 ; MejorJ := 0 ;
{2}    para j := 1 hasta n hacer
{3}       EstaSuma := EstaSuma + a[j] ;
{4}       si EstaSuma > SumaMax entonces
{5}          SumaMax := EstaSuma ;
{6}          MejorI := i ;
{7}          MejorJ := j
{8}       sino si EstaSuma < 0 entonces
{9}          i := j+1 ;
{10}         EstaSuma := 0
          fin si
       fin para ;
{11}   devolver < MejorI, MejorJ, SumaMax>
     fin función
```

1.3 Cálculo de los tiempos de ejecución (16)

- **Ejemplo: Búsqueda binaria**

Ejemplo de algoritmo logarítmico

Dados x y un vector *ordenado* a_1, a_2, ... a_n de enteros,

devolver: $\begin{cases} i \text{ si } \exists a_i = x \\ \text{“elemento no encontrado”} \end{cases}$

$\rightarrow$ Comparar x y a_{medio}, con $medio = (i + j)div2$,
siendo $a_i..a_j$ el *espacio de búsqueda*:

1. $x = a_{medio}$: terminar (interrupción)
2. $x > a_{medio}$: seguir buscando en $a_{medio+1}..a_j$
3. $x < a_{medio}$: seguir buscando en $a_i..a_{medio-1}$

nº iter $\leftrightarrow$ evolución del tamaño d del espacio de búsqueda

Invariante: $d = j - i + 1$

¿Cómo decrece d? $\begin{cases} i \leftarrow medio + 1 \\ j \leftarrow medio - 1 \end{cases}$

Peor caso: se alcanza la terminación “normal” del bucle $\equiv i > j$

Búsqueda Binaria

```
    función Búsqueda Binaria (x, a[1..n]) : posición
    {a: vector ordenado de modo no decreciente}

{1}    i := 1 ; j := n ;                          {espacio de búsqueda: i..j}
{2}    mientras i <= j hacer
{3}       medio := (i + j) div 2 ;
{4}       si a[medio] < x entonces
{5}          i := medio + 1
{6}       sino si a[medio] > x entonces
{7}          j := medio - 1
{8}       sino devolver medio                     {se interrumpe el bucle}
       fin mientras;
{9}    devolver "elemento no encontrado"          {fin normal del bucle}
    fin función
```

1.3 Cálculo de los tiempos de ejecución (16)

- **Búsqueda binaria** (Cont.): análisis del peor caso

  Sea $< d, i, j >$ iteración $< d', i', j' >$

1. $i \leftarrow medio + 1$:

$$i' = (i + j)div2 + 1$$
$$j' = j$$
$$d' = j' - i' + 1 \quad = j - (i + j)div2 - 1 + 1$$
$$\leq j - (i + j - 1)/2$$
$$= (j - i + 1)/2$$
$$= d/2$$

$$\boxed{\rightarrow d' \leq d/2}$$

2. $j \leftarrow medio - 1$:

$$i' = i$$
$$j' = (i + j)div2 - 1$$
$$d' = j' - i' + 1 \quad = (i + j)div2 - i - 1 + 1$$
$$\leq (i + j)/2 - i$$
$$< (j - i + 1)/2$$
$$= d/2$$

$$\boxed{\rightarrow d' < d/2}$$ (decrece más rápido)

1.3 Cálculo de los tiempos de ejecución (17)

- **Búsqueda binaria**: análisis del peor caso (Cont.)

  ¿$T(n)$? Sea d_l: d *después* de la l-ésima iteración

  $$\begin{cases} d_0 = n \\ d_l \leq d_{l-1}/2 \ \forall l \geq 1 \end{cases} \text{ (inducción)} \rightarrow d_l \leq n/2^l$$

  hasta $d < 1 \rightarrow l = \lceil log_2 n \rceil + 1 = O(logn)$ iteraciones

  Cada iteración es $\Theta(1)$ (reglas) $\Rightarrow T(n) = O(logn)$

  Razonamiento alternativo: *pensar en versión recursiva*

  $$T(n) = \begin{cases} 1 & \text{si } n = 0, 1 \\ T(n/2) + 1 & \text{si } n > 1 \end{cases}$$

  Teorema Divide y Vencerás: $l = 1, b = 2, c = 1, k = 0, n_0 = 1$

  Caso $l = b^k \Rightarrow T(n) = \Theta(n^k logn) \rightarrow T(n) = \Theta(logn)$

Observaciones:

- Pensar en versión recursiva puede ser otro recurso útil
- El algoritmo es Divide y Vencerás? $\rightarrow$ Algoritmos de reducción $(l = 1)$
- $T(n) = \Theta(logn)$

  $\leftrightarrow$ los datos ya están en memoria (Modelo de Computación)

1.4 Resolución de recurrencias (1) [Brassard & Bratley]

- **Recurrencias homogéneas lineales con coeficientes constantes**

  Ecuaciones de la forma:

$$a_0 t_n + a_1 t_{n-1} + \cdots + a_k t_{n-k} = 0 \qquad (1)$$

  t_i son los valores de la recurrencia.
  Además, k valores de t_i: *condiciones iniciales*
  $\longrightarrow$ permiten pasar de infinitas soluciones a una solución

- **Ejemplo: fibonacci**

$$f_n - f_{n-1} - f_{n-2} = 0$$

$$\begin{cases} k = 2 \\ a_0 = 1 \\ a_1 = a_2 = -1 \end{cases} \qquad \begin{cases} f_0 = 0 \\ f_1 = 1 \end{cases}$$

1.4 Resolución de recurrencias (2)

- **Regla**: *toda combinación lineal de soluciones es solución*
  Si $\sum_{i=0}^{k} a_i f_{n-i} = 0$
  i.e. si f_n satisface (1), y g_n también satisface (1), y
  $t_n = cf_n + dg_n$ (donde c y d son constantes arbitrarias)
  entonces t_n también es solución de (1).
  $\rightarrow$ Se generaliza para cualquier número de soluciones.

- **¿Buscar una solución de la forma $t_n = x^n$?**
  (1): $a_0x^n + a_1x^{n-1} + \cdots + a_kx^{n-k} = 0$ (Solución trivial: x=0)
  $\Leftrightarrow \underbrace{a_0x^k + a_1x^{k-1} + \cdots + a_k}_{p(x)} = 0$
  es la *ecuación característica* de (1);
  $p(x)$ es el *polinomio característico* de (1).
  $\rightarrow$ Técnica de la ecuación característica

1.4 Resolución de recurrencias (3)

- **Técnica de la ecuación característica:**
  todo polinomio de grado k posee k raíces: $p(x) = \prod_{i=1}^{k}(x - r_i)$
  r_i: pueden ser complejos, únicas soluciones de $p(x) = 0$, i.e. $p(r_i) = 0$.
  $x = r_i$ es solución de la ecuación característica;
  $\rightarrow r_i^n$ es solución de la recurrencia (1).

$$t_n = \sum_{i=1}^{k} c_i r_i^n \qquad (2)$$

(2) *satisface la recurrencia* (1) con c_i: constantes adecuadas;
es la *solución más general*.
(1) sólo posee soluciones de esta forma, siempre y cuando todas las r_i sean diferentes.
Las k constantes se determinan a partir de las k condiciones iniciales, resolviendo un sistema de k ecuaciones lineales con k incógnitas.

1.4 Resolución de recurrencias (4)

- **Ejemplo: fibonacci** (cont.)

$$f_n - f_{n-1} - f_{n-2} = 0$$
$$p(x) = x^2 - x - 1 \Rightarrow r_1 = \frac{1+\sqrt{5}}{2}, r_2 = \frac{1-\sqrt{5}}{2}$$
$$\Rightarrow f_n = c_1 r_1^n + c_2 r_2^n$$

Condiciones iniciales:

$$n = 0, f_0 = 0 \wedge f_0 = c_1 + c_2$$
$$n = 1, f_1 = 1 \wedge f_1 = c_1 r_1 + c_2 r_2$$

Sistema de ecuaciones a resolver:

$$\begin{cases} c_1 + c_2 = 0 \\ c_1 r_1 + c_2 r_2 = 1 \end{cases}$$
$$\Rightarrow c_1 = \frac{1}{\sqrt{5}}, c_2 = -\frac{1}{\sqrt{5}}$$
$$\Rightarrow f_n = \frac{1}{\sqrt{5}}[(\underbrace{\frac{1+\sqrt{5}}{2}}_{\phi})^n - (\frac{1-\sqrt{5}}{2})^n]$$

(Más exacto que $f_n = O(\phi^n)$)

1.4 Resolución de recurrencias (5)

- **Problema**: ¿las k raices no son todas diferentes entre sí?
  (2) sigue siendo solución, pero ya no es la más general: ¿Otras soluciones?
  Sea r una raíz múltiple (de momento, de multiplicidad 2):
  por definición, existe un polinomio q(x), de grado k-2, tal que:

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

Para todo $n \geq k$, consideremos:

$u_n(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_k x^{n-k}$

$v_n(x) = a_0 n x^n + a_1 (n-1) x^{n-1} + \cdots + a_k (n-k) x^{n-k}$

$\rightarrow v_n(x) = x u_n'(x) \quad (*)$

$u_n(x) = x^{n-k} p(x) = x^{n-k}(x-r)^2 q(x) = (x-r)^2 [x^{n-k} q(x)]$

$u_n'(x) = 2(x-r) x^{n-k} q(x) + (x-r)^2 [x^{n-k} q(x)]'$

$\Rightarrow u_n'(r) = 0$

$(*) \Rightarrow v_n(r) = x u_n'(r) = 0, \qquad \forall n \geq k$

$\Leftrightarrow a_0 n r^n + a_1 (n-1) r^{n-1} + \cdots + a_k (n-k) r^{n-k} = 0$

$\Rightarrow t_n = n r^n$ también es solución de (1) (nueva)

1.4 Resolución de recurrencias (6)

- **Raíces de multiplicidad** m:
  Si r tiene multiplicidad m:

$$t_n = r^n, t_n = nr^n, t_n = n^2 r^n, \ldots, t_n = n^{m-1} r^n$$

son soluciones de (1).

Solución general: combinación lineal $\rightarrow$ $r_1, r_2, \ldots, r_l$: raices distintas de $p(x)$, de multiplicidades $m_1, m_2, \ldots, m_l \Rightarrow$

$$t_n = \sum_{i=1}^{l} \sum_{j=0}^{m_i - 1} c_{ij} n^j r_i^n \qquad (3)$$

(3) es la solución general de (1).
Los c_{ij} se determinan a partir de las condiciones iniciales:
$\rightarrow$ son k constantes, $c_1, c_2, \ldots, c_k$.
$\sum_{i=1}^{l} m_i = k$: suma de las multiplicidades, i.e. número total de raices.

1.4 Resolución de recurrencias (7)

- **Ejemplo:**

$$t_n = \begin{cases} n & n = 0, 1, 2 \\ 5t_{n-1} - 8t_{n-2} + 4t_{n-3} & sino \end{cases}$$

Con la forma de la ecuación (1): $t_n - 5t_{n-1} + 8t_{n-2} - 4t_{n-3} = 0$

$$p(x) = x^3 - 5x^2 + 8x - 4 = (x-1)(x-2)^2 \qquad \begin{cases} r_1 = 1 & m_1 = 1 \\ r_2 = 2 & m_2 = 2 \end{cases}$$

$$(3) \Rightarrow t_n = c_1 1^n + c_2 2^n + c_3 n 2^n$$

$$\left.\begin{array}{lll} n = 0: & c_1 + c_2 & = 0 \\ n = 1: & c_1 + 2c_2 + 2c_3 & = 1 \\ n = 2: & c_1 + 4c_2 + 8c_3 & = 2 \end{array}\right\} \begin{array}{l} c_1 = -2 \\ c_2 = 2 \\ c_3 = -\frac{1}{2} \end{array}$$

$$\Rightarrow t_n = 2^{n+1} - n2^{n-1} - 2$$

1.4 Resolución de recurrencias (8)

- **Recurrencias no homogéneas**
  La combinación lineal ya no es $= 0$
  $\rightarrow$ la combinación lineal de soluciones ya no es solución.
  Consideramos inicialmente recurrencias de la forma:

$$a_0 t_n + a_1 t_{n-1} + \cdots + a_k t_{n-k} = b^n p(n) \qquad (4)$$

  Donde b es una constante y $p(n)$ un polinomio de grado d.

- **Ejemplo 1**: $t_n - 2t_{n-1} = 3^n$ (*1: polinomio de grado 0)
  Reducir al caso homogéneo:
  Multiplicar por 3: $\rightarrow 3t_n - 6t_{n-1} = 3^{n+1}$
  Sustituir n por n-1: $\rightarrow 3t_{n-1} - 6t_{n-2} = 3^n$
  Diferencia entre 2 ecuaciones:

$$\begin{array}{rrrcl} t_n & -2t_{n-1} & & = & 3^n \\ - & 3t_{n-1} & -6t_{n-2} & = & 3^n \\ \hline t_n & -5t_{n-1} & +6t_{n-2} & = & 0 \end{array}$$

$$\Rightarrow \underbrace{x^2 - 5x + 6}_{p(x)} = (x-2)(x-3)$$

1.4 Resolución de recurrencias (9)

- **Ejemplo 1**: (Cont.)
  $(x-2)(x-3) \Rightarrow$ Las soluciones son de la forma: $t_n = c_1 2^n + c_2 3^n$
  Y como $t_n \geq 0 \quad \forall n \geq 0$, se deduce que $t_n = O(3^n)$
  Pero c_1 y c_2 ya no se determinan a partir de las condiciones iniciales:
  $t_n = 2^n$ y $t_n = 3^n$ **no** son soluciones de la recurrencia original.

$$\rightarrow t_n - 2t_{n-1} = 3^n \Rightarrow t_1 = 2t_0 + 3 \Rightarrow \left\{ \begin{array}{lcll} c_1 + c_2 & = & t_0 & n = 0 \\ 2c_1 + 3c_2 & = & 2t_0 + 3 & n = 1 \end{array} \right.$$

$$\Rightarrow \left\{ \begin{array}{l} c_1 = t_0 - 3 \\ c_2 = 3 \end{array} \right. \Rightarrow t_n = (t_0 - 3)2^n + 3^{n+1} = \theta(3^n)$$

$\equiv$ Solución general, independientemente de las condiciones iniciales

Otro razonamiento:

$$\begin{array}{rcl} 3^n & = & t_n - 2t_{n-1} \\ & = & (c_1 2^n + c_2 3^n) - 2(c_1 2^{n-1} + c_2 3^{n-1}) \\ & = & c_2 3^{n-1} \end{array}$$

$\Rightarrow c_2 = 3$ independientemente de t_0, i.e. queda descartado que $c_2 = 0$
$\Rightarrow t_n = \theta(3^n)$

1.4 Resolución de recurrencias (10)

- **Ejemplo 2**:

$$\begin{array}{lccccc} & t_n & -2t_{n-1} & & & = & (\overbrace{n+5}^{d=1})3^n \\ \star(-6), n-1: & & -6t_{n-1} & +12t_{n-2} & & = & -6(n+4)3^{n-1} \\ \star 9, n-2: & & & 9t_{n-2} & -18t_{n-3} & = & 9(n+3)3^{n-2} \\ \hline & t_n & -8t_{n-1} & +21t_{n-2} & -18t_{n-3} & = & 0 \end{array}$$

$$p(x): x^3 - 8x^2 + 21x - 18 = (x-2)(x-3)^2$$

$\Rightarrow t_n = c_1 2^n + c_2 3^n + c_3 n 3^n$: Solución general

La recurrencia original impone las siguientes restricciones:

$$t_1 = 2t_0 + 18$$

$$t_2 = 2t_1 + 63 = 4t_0 + 99$$

$$\Rightarrow \left\{ \begin{array}{lccccc} n=0: & c_1 & +c_2 & & = & t_0 \\ n=1: & 2c_1 & +3c_2 & +3c_3 & = & 2t_0+18 \\ n=2: & 4c_1 & +9c_2 & +18c_3 & = & 4t_0+99 \end{array} \right| \Rightarrow \left\{ \begin{array}{l} c_1 = t_0 - 9 \\ c_2 = 9 \\ c_3 = 3 \end{array} \right.$$

$\Rightarrow$ Solución general: $t_n = (t_0 - 9)2^n + (n+3)3^{n+1} = \theta(n3^n)$, independientemente de t_0.

Alternativamente, sustituyendo la solución general en la recurrencia original, se llega a idéntico resultado (ejercicio).

1.4 Resolución de recurrencias (11)

- **Generalización**:

$$
\begin{array}{lll}
ejemplo1: & t_n - 2t_{n-1} = 3^n & \rightarrow \quad p(x) = (x-2)(x-3)^{0+1} \\
ejemplo2: & \underbrace{t_n - 2t_{n-1}}_{(x-2)} = (n+5)3^n & \rightarrow \quad p(x) = (x-2)(x-3)^{1+1}
\end{array}
$$

$\rightarrow$ para resolver (4), utilizar:

$$(a_0 x^k + a_1 x^{k-1} + \cdots + a_k)(x-b)^{d+1} \tag{5}$$

Se procede igual que en el caso homogéneo, salvo que algunas ecuaciones para determinar las constantes se obtienen a partir de la recurrencia.

1.4 Resolución de recurrencias (12)

- **Ejemplo 3: Las torres de Hanoi**

```
procedimiento Hanoi (m,i,j)
   si m>0 entonces Hanoi (m-1,i,6-i-j);
                   mover_anillo (i,j);    {instr. elemental}
                   Hanoi (m-1,6-i-j,j)
```

Problema de los monjes: Hanoi (64,1,2)

$t(m)$: número de ejecuciones de la instrucción elemental en Hanoi$(m, -, -)$.

$$t(m) = \begin{cases} 0 & m = 0 \\ 2t(m-1)+1 & sino \end{cases}$$

$\rightarrow t(m) - 2t(m-1) = 1$ (4) con $b = 1$ y $p(n) = 1$, $d = 0$.

(5) $\Rightarrow p(x) = (x-2)(x-1)$

Todas las soluciones son de la forma $t(m) = c_1 1^m + c_2 2^m$

$$\begin{cases} t(0) = 0 \\ t(1) = 2t(0)+1 = 1 \end{cases} \Rightarrow \begin{cases} c_1 + c_2 = 0 & m = 0 \\ c_1 + 2c_2 = 1 & m = 1 \end{cases} \Rightarrow \begin{cases} c_1 = -1 \\ c_2 = 1 \end{cases}$$

$\Rightarrow t(m) = 2^m - 1$

¿Determinar θ sin calcular c_1 y c_2?

$$\begin{array}{l} t(m) = c_1 + c_2 2^m \\ t(m) \geq m \end{array} \Rightarrow c_2 > 0 \Rightarrow t(m) = \theta(2^m)$$

1.4 Resolución de recurrencias (13)

- **Ejemplo 4**: $t_n = 2t_{n-1}+n \rightarrow t_n - 2t_{n-1} = n$ (4): $b = 1, p(n) = n, d = 1$

$(5) \Rightarrow p(x) = (x-2)(x-1)^2$
$\Rightarrow$ Solución de la forma $t_n = c_1 2^n + c_2 1^n + c_3 n 1^n$
$(t_0 \geq 0 \Rightarrow t_n \geq 0 \; \forall n) \Rightarrow t_n = O(2^n)$

¿$t_n = \theta(2^n)$? $\Leftrightarrow$ ¿$c_1 > 0$?

$$\begin{aligned} n &= t_n - 2t_{n-1} \\ &= (c_1 2^n + c_2 + c_3 n) - 2(c_1 2^{n-1} + c_2 + c_3(n-1)) \\ &= \underbrace{2c_3 - c_2}_{=0} \underbrace{-c_3}_{=1} n \end{aligned} \quad \Rightarrow \begin{cases} c_3 = -1 \\ c_2 = -2 \\ c_1 = ? \end{cases}$$

$$\rightarrow \begin{array}{l} t_n = c_1 2^n - n - 2 \\ (t_0 \geq 0 \Rightarrow t_n \geq 0 \; \forall n) \end{array} \Rightarrow c_1 > 0 \Rightarrow t_n = \theta(2^n)$$

c_1 también se puede calcular, pero no es necesario.

Problema: no siempre se puede usar este método para determinar θ.

1.4 Resolución de recurrencias (14)

- **Ejemplo 5**: $t_n = \begin{cases} 1 & n = 0 \\ 4t_{n-1} - 2^n & sino \end{cases}$

$$t_n - 4t_{n-1} = -2^n \qquad (4)\text{: } b = 2, p(n) = -1, d = 0$$

$$(5) \Rightarrow p(x) = (x-4)(x-2)^{0+1} \qquad \Rightarrow t_n = c_1 4^n + c_2 2^n$$

¿$t_n = \theta(4^n)$?

$$\begin{aligned} -2^n &= t_n - 4t_{n-1} \\ &= (c_1 4^n + c_2 2^n) - 4(c_1 4^{n-1} + c_2 2^{n-1}) \\ &= -\underbrace{c_2}_{=1} 2^n \end{aligned}$$

Cálculo de c_1:

i sistema de ecuaciones

ii $\begin{cases} t_n = c_1 4^n + 2^n \\ t_0 = 1 \end{cases} \Rightarrow 1 = c_1 + 1 \Rightarrow c_1 = 0$!

$\Rightarrow t_n = 2^n \neq \theta(4^n)$... Pero con $t_0 > 1, t_n = \theta(4^n)$

Conclusión: para algunas recurrencias las condiciones iniciales son determinantes, mientras que en otras sólo importa que $t_0 \geq 0$.

1.4 Resolución de recurrencias (15)

- **Generalización**:

  Las recurrencias de la forma:

$$a_0 t_n + a_1 t_{n-1} + \cdots + a_k t_{n-k} = b_1^n p_1(n) + b_2^n p_2(n) + \ldots \qquad (6)$$

  donde las b_i son constantes y los $p_i(n)$ son polinomios de grado d_i, se resuelven con:

$$(a_0 x^k + a_1 x^{k-1} + \cdots + a_k)(x - b_1)^{d_1+1}(x - b_2)^{d_2+1} \ldots \qquad (7)$$

  Un factor corresponde al lado izquierdo, un factor por cada término al lado derecho.

1.4 Resolución de recurrencias (16)

- **Ejemplo 6**: $t_n = \begin{cases} 0 & n = 0 \\ 2t_{n-1} + n + 2^n & sino \end{cases}$

$t_n - 2t_{n-1} = n + 2^n$ (6): $\begin{array}{l} b_1 = 1, p_1(n) = n, d_1 = 1 \\ b_2 = 2, p_2(n) = 1, d_2 = 0 \end{array}$

$(7) \Rightarrow (x-2)(x-1)^2(x-2) = (x-1)^2(x-2)^2$
$\Rightarrow t_n = c_1 1^n + c_2 n 1^n + c_3 2^n + c_4 n 2^n$
$t_n = O(n2^n)$
¿$t_n = \theta(n2^n)$? $\Leftrightarrow$ ¿$c_4 > 0$?

i (en recurrencia original) $\rightarrow n + 2^n = (2c_2 - c_1) - c_2 n + \underbrace{c_4}_{=1} 2^n$

$\Rightarrow t_n = \theta(n2^n)$

ii (sistema de ecuaciones) $\rightarrow t_n = n2^n + 2^{n+1} - n - 2$

$\Rightarrow t_n = \theta(n2^n)$

1.4 Resolución de recurrencias (17) - Cambios de variable

- **Ejemplo 1**: $T(n) = \begin{cases} 1 & n = 1 \\ 3T(n/2) + n & n = 2^i, n > 1 \end{cases}$

Sustituir n por 2^i $\rightarrow$ nueva recurrencia $t_i = T(2^i)$

$n/2$ $\rightarrow$ $2^i/2 = 2^{i-1}$

$T(n)$ función de $T(n/2)$ $\rightarrow$ t_i función de t_{i-1}: lo que sabemos resolver

$\Rightarrow t_i = T(2^i) = 3T(2^{i-1}) + 2^i = 3t_{i-1} + 2^i$

$\Rightarrow t_i - 3t_{i-1} = 2^i$ (4) $\Rightarrow$ (5) $(x-3)(x-2)$

$$\Rightarrow \left.\begin{array}{l} t_i = c_1 3^i + c_2 2^i \\ T(2^i) = t_i \Leftrightarrow T(n) = t_{log_2 n}, n = 2^i \end{array}\right\} \Rightarrow T(n) = c_1 3^{log_2 n} + c_2 2^{log_2 n}$$

$\Rightarrow T(n) = c_1 n^{log_2 3} + c_2 n = O(n^{log 3})$

¿$T(n) = \theta(n^{log 3})$? $\Leftrightarrow$ ¿$c_1 > 0$?

$$\begin{aligned} n &= T(n) - 3T(n/2) \\ &= (c_1 n^{log_2 3} + c_2 n) - 3(c_1 (n/2)^{log_2 3} + c_2 (n/2)) \quad \leftarrow (1/2)^{log_2 3} = 1/3 \\ &= -c_2 (n/2) \end{aligned}$$

$$\rightarrow \left.\begin{array}{l} c_2 = -2 \\ T(n) > 0 \end{array}\right\} \Rightarrow c_1 > 0 \Rightarrow T(n) = \theta(n^{log 3})$$ si n es potencia de 2

1.4 Resolución de recurrencias (18) - Cambios de variable

- **Ejemplo 2: Recurrencias Divide y Vencerás**

$$T(n) = \ell T(n/b) + cn^k, n > n_0 \qquad (8)$$

Con $\ell \geq 1, b \geq 2, k \geq 0, n_0 \geq 1 \in N$ y $c > 0 \in R$,
cuando n/n_0 es una potencia exacta de b ($n \in \{bn_0, b^2n_0, b^3n_0 \dots\}$).

Cambio de variable: $n = b^i n_0$

$$\Rightarrow t_i = T(b^i n_0) = \ell T(b^{i-1} n_0) + c(b^i n_0)^k$$
$$= \ell t_{i-1} + cn_0^k b^{ik}$$

$\Rightarrow t_i - \ell t_{i-1} = cn_0^k (b^k)^i$ (4): $p(i) = cn_0^k, d = 0, b = b^k$

(5) $\Rightarrow (x - \ell)(x - b^k)$

$\Rightarrow t_i = c_1 \ell^i + c_2 (b^k)^i \quad (\star)$

$i = log_b(n/n_0)$ cuando n/n_0 es una potencia exacta de b

$\Rightarrow d^i = (n/n_0)^{log_b d}$ para $d > 0$

$$\Rightarrow T(n) = (c_1/n_0^{log_b \ell}) n^{log_b \ell} + (c_2/n_0^k) n^k$$
$$= c_3 n^{log_b \ell} + c_4 n^k \quad (\star\star)$$

(en recurrencia original) $\rightarrow cn^k = T(n) - \ell T(n/b) = \dots$

$$= (1 - \ell/b^k) c_4 n^k \Rightarrow c_4 = c/(1 - \ell/b^k)$$

1.4 Resolución de recurrencias (19) - Cambios de variable

- **Ejemplo 2: Recurrencias Divide y Vencerás**: (Cont.)

  ¿Notación asintótica para $T(n)$? $\leftrightarrow$ ¿término dominante en $(\star\star)$?

1. $\ell < b^k \Rightarrow c_4 > 0 \wedge k > log_b \ell \Rightarrow c_4 n^k$ domina
   $\Rightarrow T(n) = \theta(n^k)$
2. $\ell > b^k \Rightarrow c_4 < 0 \wedge log_b \ell > k \Rightarrow c_3 > 0, c_3 n^{log_b \ell}$ domina
   $\Rightarrow T(n) = \theta(n^{log_b \ell})$
3. $\ell = b^k \Rightarrow c_4 = c/0$!
   Problema: $(\star)$ no proporciona la solución general de la recurrencia
   $(x - \ell)(x - b^k) \quad \rightarrow (x - b^k)^2$
   $\Rightarrow t_i = c_5 (b^k)^i + c_6 i (b^k)^i$
   $\Leftrightarrow T(n) = c_7 n^k + c_8 n^k log_b(n/n_0)$
   En recurrencia original: $\rightarrow c_8 = c > 0 \Rightarrow c n^k log_b(n/n_0)$ domina
   $\Rightarrow T(n) = \theta(n^k logn)$

1.4 Resolución de recurrencias (20) - Teorema Divide y Vencerás

Si una recurrencia es de la forma (8), se aplica el *Teorema Divide y Vencerás*:

$$T(n) = \begin{cases} \theta(n^k) & si \quad \ell < b^k \\ \theta(n^k log n) & si \quad \ell = b^k \\ \theta(n^{log_b \ell}) & si \quad \ell > b^k \end{cases} \tag{9}$$

Para el análisis de algoritmos, se suelen manejar desigualdades:

$T(n) \leq \ell T(n/b) + cn^k, n > n_0$ con n/n_0 potencia exacta de b

$$\Rightarrow T(n) = \begin{cases} O(n^k) & si \quad \ell < b^k \\ O(n^k log n) & si \quad \ell = b^k \\ O(n^{log_b \ell}) & si \quad \ell > b^k \end{cases}$$

UNIVERSIDADE DA CORUÑA

Algoritmos:
Análisis de algoritmos

Alberto Valderruten

LFCIA - Departamento de Computación

Facultad de Informática

Universidad de A Coruña, España

www.lfcia.org/alg

www.fi.udc.es