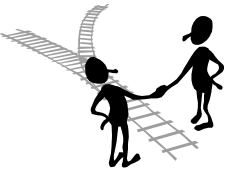
Algorítmica
Capítulo 2: Algoritmos Divide y Vencerás
Tema 6: Aplicaciones

- **Otras aplicaciones**
  - Multiplicación de enteros
  - Multiplicación de matrices.
  - Metodo de Strassen.
  - Multiplicación de polinomios
  - El problema del “skyline”

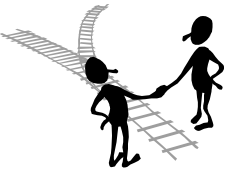


Multiplicación de enteros

- El problema que consideramos es el de la multiplicación de enteros muy grandes.
- Sean x e y dos números de n bits.
- Los métodos tradicionales de multiplicación requieren O(n^2) operaciones sobre los bits:

```
x = 1 0 1 1 0 1 0 1
y = 1 0 1 0 0 1 1 0
-------------------
      1 0 1 1 0 1 0 1
    1 0 1 1 0 1 0 1
  1 0 1 1 0 1 0 1
1 0 1 1 0 1 0 1
-------------------
1 1 1 0 1 0 1 0 1 0 1 1 1 1 0
```

```
x = 1 0 1 1
y = 0 1 1 0
-----------
    1 0 1 1
  1 0 1 1
-----------
  1 0 0 0 0 1 0
```



Multiplicación de enteros

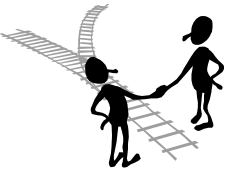
- Supongamos que $n = 2^m$,
- Tratamos a x e y como dos strings de n bits y los partimos en dos mitades como x = a * b, y= c * d, de manera que las concatenaciones de a, b y de c, d reproducen x e y.
- Como son números binarios, podemos expresarlos como

$$xy = (a\,2^{n/2} + b)(c\,2^{n/2} + d)$$

$$= ac\,2^{n} + (ad + bc)\,2^{n/2} + bd \ldots\ldots\ldots\ldots (*)$$

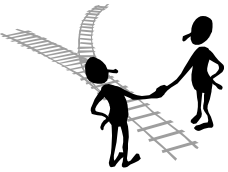
Ejemplo: Si x = 13 → x = 1101 (n = 4) → A = 3 y B = 1 → $13 = A \cdot 2^{n/2} + B = 3 \cdot 2^{n/2} + 1 = (11)_{(2)} \cdot 2^{n/2} + (01)_{(2)}$.

- De esta manera el cálculo de xy se simplifica a la realización de cuatro multiplicaciones de números de (n/2)-bits y algunas adiciones y desplazamientos de 0 y 1 (las multiplicaciones son por potencias d e 2).



Multiplicación de enteros

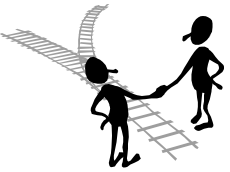
- Sea T(n) el número de operaciones requeridas para multiplicar los dos números de n bits.



Multiplicación de enteros

- Sea T(n) el número de operaciones requeridas para multiplicar los dos números de n bits.
- Está claro que,

$$
\begin{aligned}
T(n) &= 4T(n/2) + cn \\
&= \\
&= \\
&= \\
&= \\
&= \\
&=
\end{aligned}
$$

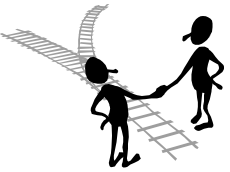


Multiplicación de enteros

- Sea T(n) el número de operaciones requeridas para multiplicar los dos números de n bits.
- Esta claro que,

$$
\begin{aligned}
T(n) &= 4T(n/2) + cn \\
&= 4\,(4T(n/2^2) + cn/2) + cn \\
&= 4^2T(n/2^2) + 2cn + cn \\
&= 4^3T(n/2^3) + 2^2cn + 2cn + cn \\
&\quad \text{M} \\
&= 4^mT(1) + (2^{m-1} + \ldots + 1)\,cn \\
&= 2^m2^m + (2^m - 1)\,cn \\
&= O(n^2)
\end{aligned}
$$

- Con lo que se demuestra que no hemos adelantado nada



Multiplicación de enteros

Pero, consideremos la siguiente recurrencia

$$T(n) = \begin{cases} \text{si } n = 1 \\ \lceil (n/c) + bn \quad \text{si } n > 1 \end{cases}$$

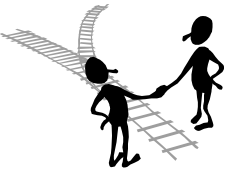
tomando $n = c^m$ podemos resolverla y obtener ($r = a/c$)

$$a < c\ ;\ r < 1,\ r^m \to 0,\ T(n) \to \frac{bc}{c-a}\, n = O(n)$$

$$a = c\ ;\ r = 1,\ T(n) = bn(m+1) = O(n\log n)$$

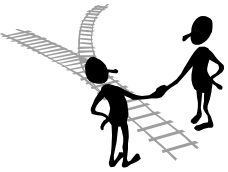
when $$a > c\ ;\ r > 1,\ T(n) \to bnr^m = bn\left(\frac{a}{c}\right)^{\log_c n} = ba^{\log_c n} = O(n^{\log_c a})$$

- T(n) depende del numero de subproblemas y de su tamaño.
- Si el numero de subproblemas fuera 1, 3, 4, 8, entonces los algoritmos serian de ordenes n, $n^{\log 3}$, n^2, n^3 respectivamente.



Multiplicación de enteros

- Si observamos la situación, en nuestro caso, la multiplicación de enteros va a producir algoritmos O(n^2) si el número de subproblem as de tamaño mitad es cuatro, ya que el 4 que acompaña al t(n/2) es el que nos hace el orden cuadrático,
- Por tanto intentamos disminuir el número de subproblemas DV (concretamente con tres productos en vez de 4).
- Para reducir la complejidad en tiempo intentaremos reducir el número de subproblemas (y por tanto de multiplicaciones) a costa de hacer mas adiciones y multiplicaciones por potencias de dos, y haremos así una multiplicación menos
- El truco es especialmente significativo conforme mas grande es n, es decir, asintóticamente



Multiplicación de enteros

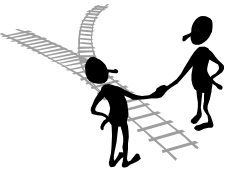
- Teníamos

$$xy = (a2^{n/2} + b)(c2^{n/2} + d)$$

- Ahora

$$x \cdot y = ac \cdot 2^n + [(a - b)(d - c) + ac + bd] \cdot 2^{n/2} + bd$$

- Es evidente que con esta forma de multiplicar x e y solo tenemos que hacer
  - tres multiplicaciones de dos números de (n/2)-bits
  - seis adiciones y
  - multiplicaciones por potencias de 2
- Por tanto el correspondiente algoritmo DV debe ser mas eficiente

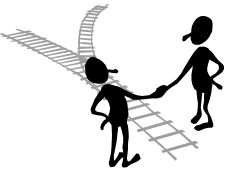


Multiplicación de enteros

- Podemos usar la rutina de multiplicación recursivamente para calcular los tres productos. Las adiciones y desplazamientos requie ren un tiempo O(n). Asi, la complejidad en tiempo de la multiplica ción de dos enteros de n bits esta acotada superiormente por

$$T(n) = \begin{cases} pn^2 & \text{si } n \le n_o \\ T(n/2) + kn & \text{si } n \ge n_o \end{cases}$$

- Donde k es una constante que incorpora las adiciones y las transferencias de bits.
- Es evidente que el tiempo de este algoritmo es $O(n^{\log 3}) = O(n^{1.59})$.

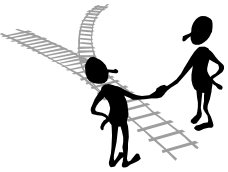


Ejemplo

$$x = 1\ 0\ 1\ 1 \qquad a = 1\ 0 \qquad b = 1\ 1$$
$$y = 0\ 1\ 1\ 0 \qquad c = 0\ 1 \qquad d = 1\ 0$$

$$u = (a+b)(c+d) = (1\ 0\ 1)(1\ 1) = 1\ 1\ 1\ 1$$
$$v = ac = (1\ 0)(0\ 1) = 1\ 0$$
$$w = bd = (1\ 1)(1\ 0) = 1\ 1\ 0$$
$$z = u - v - w = 1\ 1\ 1$$
$$xy = v2^4 + z2^2 + w = 1\ 0\ 0\ 0\ 0\ 0 + 1\ 1\ 1\ 0\ 0 + 1\ 1\ 0 = 1\ 0\ 0\ 0\ 0\ 1\ 0$$

Este mísmo enfoque puede emplearse para diseñar un algoritmo que multiplique dos polinomios de grado n

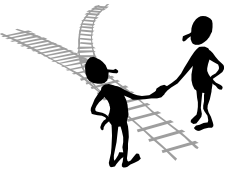


Multiplicación de matrices

- Si tenemos dos matrices A y B cuadradas en donde A tiene el mismo número de filas que columnas de B, se trata de multiplicar A y B para obtener una nueva matriz C.
- La multiplicación de matrices se realiza conforme a

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$$

- Esta fórmula corresponde a la multiplicación normal de matrices, que consiste en tres bucles anidados, por lo que es $O(n^3)$.
- Para aplicar la técnica DV, vamos a proceder como con la multiplicación de enteros, con la intención de obtener un algoritmo mas (?) eficiente para multiplicar matrices.



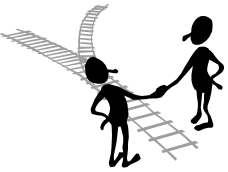
Multiplicación de matrices

La multiplicación puede hacerse como sigue:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix} = \begin{pmatrix} ae+bf & ag+bh \\ ce+df & cg+dh \end{pmatrix}$$

C A B

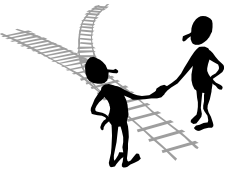
- Esta formulación divide una matriz *n* x *n en matrices de tamaños* n/2 x n/2, con lo que divide el problema en 8 subproblemas de tamaños n/2.
- Notese que n se usa como tamaño del caso aunque la dimension de la matriz es n^2
- Este enfoque da la siguiente recurrencia,



Multiplicación de matrices

$$T(n) = \begin{cases} & \text{si } n = 1 \\ 3T(n/2) + bn^2 & \text{si } n > 1 \end{cases}$$

- A partir de la cual, es evidente que T(n) sigue siendo O(n^3).
- Pero, basándonos en el enfoque DV que empleamos para multiplicar enteros, la multiplicación de las matrices también pued e calcularse como sigue.



El método de Strassen

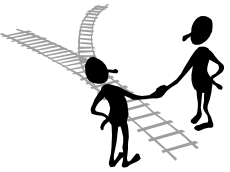
$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix} = \begin{pmatrix} ae+bf & ag+bh \\ ce+df & cg+dh \end{pmatrix}$$

$C \quad A \quad B$

$$\begin{aligned} P &= (a+d)(e+h) \\ Q &= (c+d)e \\ R &= a(g-h) \\ S &= d(f-e) \\ T &= (a+b)h \\ U &= (c-a)(e+g) \\ V &= (b-d)(f+h) \end{aligned}$$

$$\begin{aligned} r &= P+S-T+V \\ s &= R+T \\ t &= Q+S \\ u &= P+R-Q+U \end{aligned}$$

- Es evidente que solo se necesitan 7 multiplicaciones y 18 adiciones/su bstracciones, en lugar de las anteriores 8.



El método de Strassen

$$A \times B = C$$

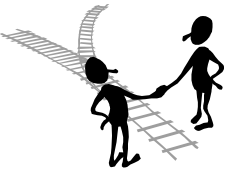
$$\begin{bmatrix} A_0 & A_1 \\ A_2 & A_3 \end{bmatrix} \times \begin{bmatrix} B_0 & B_1 \\ B_2 & B_3 \end{bmatrix} = \begin{bmatrix} A_0 \times B_0 + A_1 \times B_2 & A_0 \times B_1 + A_1 \times B_3 \\ A_2 \times B_0 + A_3 \times B_2 & A_2 \times B_1 + A_3 \times B_3 \end{bmatrix}$$

Divide las matrices en submatrices A_0, A_1, A_2 etc.

Recursivamente sigue dividiendo las submatrices en otras submatrices

Usa ecuaciones de multiplicación de matrices en bloques

Multiplica recursivamente las submatrices



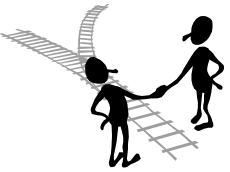
El método de Strassen

$$T(n) = 7T(n/2) + bn^2$$

$$= O(7^m) = O(n^{\log 7}) = O(n^{2.81})$$

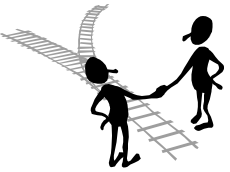
Se conocen mejoras del algoritmo, pero en la práctica las rebajas que consiguen son a costa de grandes aumentos en los valores de las correspondientes constantes ocultas

¿Y si las matrices no fueran cuadradas ?



El método de Strassen: Comentarios

- Algoritmo de Strassen: $O(n^{2.80736})$
- Algoritmo de Coppersmith–Winograd (1990): $O(n^{2.376})$
  - Frecuentemente usado como subprocedimiento de otros algoritmos para calcular mejoras teóricas en las cotas de eficiencia.
  - No se usa en la práctica ya que solo proporciona mejoras efectivas cuando se trata de multiplicar matrices extremadamente grandes
- Mejor cota alcanzada hasta la fecha: $O(n^{2.3727})$ en 2011
- ¿Cual será la eficiencia del mejor caso?
  - $\Omega(n^2)$
  - Al menos tendrémos que rellenar la matriz de la respuesta



Multiplicación de enteros

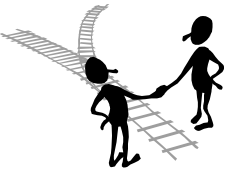
- Al método de multiplicación de enteros se le conoce con el nombre de **Algoritmo de Karatsuba**
- Sean a, b dos números con dos dígitos cada uno. Los partimos en dos

  $a = p \times 10 + q$ y $b = r \times 10 + s$.
- Si a = 78, b = 21, entonces

  p = 7, q = 8, and r = 2, s = 1.

  $a \times b = (p \times 10 + q) \times (r \times 10 + s)$

  $= (p \times r) \times 100 + (p \times s + q \times r) \times 10 + q \times s.$
- $78 \cdot 21 = (7 \cdot 2) \cdot 100 + (7 \cdot 1 + 8 \cdot 2) \cdot 10 + 8 \cdot 1 = 1638.$
- 4 multiplicaciones de números de un digito + sumas



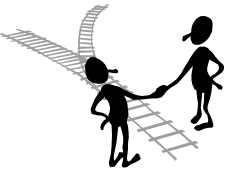
Algoritmo de Karatsuba (n =2)

- **Karatsuba:** multiplicar dos números de 2 dígitos utilizando tres multiplicaciones de números de 1 dígito
- $u = p \times r$, $v = (q - p) \times (s - r)$, $w = q \times s$.
- $a \times b = u \times 10^2 + (u + w - v) \times 10 + w$.
- $u + w - v = p \times r + q \times s - (q - p) \times (s - r)$
- $= p \times s + q \times r$.
- En nuestro ejemplo
- $u = 7 \times 2 = 14$, $v = (8 - 7) \times (1 - 2) = -1$, $w = 8 \times 1 = 8$.
- $78 \times 21 = 14 \times 100 + (14 + 8 - (-1)) \times 10 + 8$

  $= 1400 + 230 + 8$

  $= 1638$.

$a = p \times 10 + q$
$b = r \times 10 + s$



Algoritmo de Karatsuba (n =4)

- Cada mitad es un número de dos dígitos
- $a = p \times 10^2 + q$ y $b = r \times 10^2 + s$.
- $u = p \times r$, $v = (q - p) \times (s - r)$, $w = q \times s$.
- Ahora $a \times b = u \times 10^4 + (u + w - v) \times 10^2 + w$.
- Ejemplo a = 5678 and b = 4321.

  a -> p = 56, q = 78; b -> r = 43 y s = 21.

  u = 56 × 43 = 2408,

  v = (78 − 56) × (21 − 43) = −484,

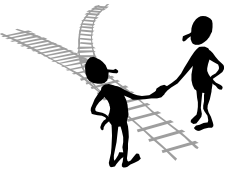
  w = 78 × 21 = 1638.

  (producto de dos números de 2 digitos)

- 5678 × 4321 = 2408 × 10000 + (2408 + 1638 − (−484)) × 100 + 1638

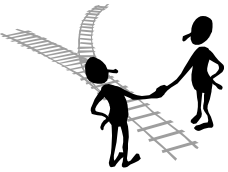
  = 24080000+ 453000+ 1638

  = 24534638.



Algoritmo de Karatsuba

digits	Karatsuba	long multiplication
$1 = 2^0$	1	1
$2 = 2^1$	3	4
$4 = 2^2$	9	16
$8 = 2^3$	27	64
$16 = 2^4$	81	256
$32 = 2^5$	243	1024
$64 = 2^6$	729	4096
$128 = 2^7$	2187	16 384
$256 = 2^8$	6561	65 536
$512 = 2^9$	19 638	262 144
$1024 = 2^{10}$	59 049	1 048 576
$1\,048\,576 = 2^{20}$	3 486 784 401	1 099 511 627 776
$\dots$	$\dots$	$\dots$
$n = 2^k$	3^k	4^k



Multiplicación de polinomios

- Supongamos dos polinomios:

$$P(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{n-1} x^{n-1}$$

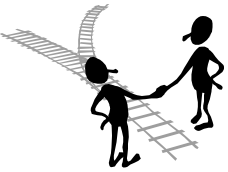
$$Q(x) = q_0 + q_1 x + q_2 x^2 + \ldots + q_{n-1} x^{n-1}$$

- Con el exponente n par: hay n coeficientes y el grado de cada polinomio es n – 1.
- DV sugiere dividir los polinomios por la mitad, quedando de la siguiente manera:
- P(x):

$$P_I(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{n/2-1} x^{n/2-1}$$

$$P_D(x) = p_{n/2} + p_{n/2+1} x + p_{n/2+2} x^2 + \ldots + p_{n-1} x^{n/2-1}$$

- Q(x) (análogo a P (x)): $Q_I(x)$ y $Q_D(x)$



Multiplicacion de polinomios

- Entonces:

$$P(x) = P_I(x) + P_D(x)\, x^{n/2}$$

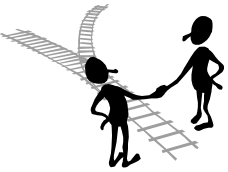
$$Q(x) = Q_I(x) + Q_D(x)\, x^{n/2}$$

- El algoritmo DV nos llevaría a:

$$P(x)\cdot Q(x) =$$

$$P_I(x)Q_I(x) + \left(P_I(x)Q_D(x) + P_D(x)Q_I(x)\right) x^{n/2} + P_D(x)Q_D(x)\, x^{n}$$

- El algoritmo subyacente es de orden cuadrado, como el convencional que se desprende de la multiplicación de polinomios



Multiplicación de polinomios

- Con el mismo método que en los problemas anteriores, la multiplicación de los polinomios puede hacerse teniendo en cuenta que

$$R_I(x) = P_I(x)\, Q_I(x)$$

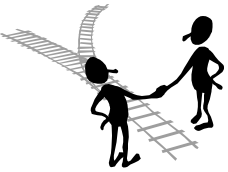
$$R_D(x) = P_D(x)\, Q_D(x)$$

$$R_H(x) = (P_I(x) + P_D(x))\,(Q_I(x) + Q_D(x))$$

- Entonces la multiplicación de P por Q queda:

- $P(x)\cdot Q(x) = R_I(x) + (R_H(x) - R_I(x) - R_D(x))\, x^{n/2} + R_D(x)\, x^n$

- Que conlleva “solo” tres multiplicaciones de polinomios de grado mitad que los originales



Ejemplo

- $P(x) = 1 + x + 3x^2 - 4x^3$
- $Q(x) = 1 + 2x - 5x^2 - 3x^3$

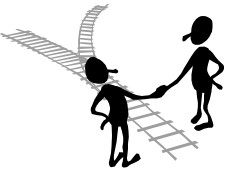
- Entonces,

$$R_I(x) = (1 + x)(1 + 2x) = 1 + 3x + 2x^2$$

$$R_D(x) = (3 - 4x)(-5 - 3x) = -15 + 11x + 12x^2$$

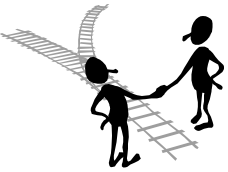
$$R_H(x) = (4 - 3x)(-4 - x) = -16 + 8x + 3x^2$$

- y solo queda hacer las multiplicaciones



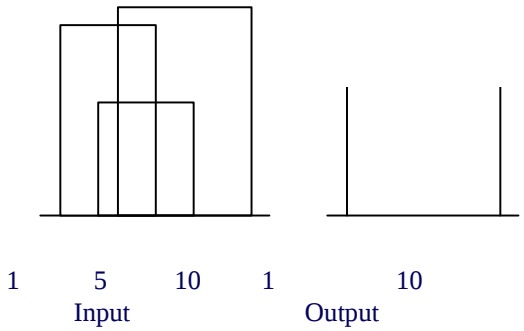
El problema de la linea del horizonte

- Con la comercialización y popularización de las estaciones de trabajo gráficas de alta velocidad, el CAD (“computer-aided design”) y otras areas (CAM, diseño VLSI) hacen un uso masivo y efectivo de los computadores.
- Un importante problema a la hora de dibujar imágenes con computadores es la eliminación de líneas ocultas, es decir, la eliminación de líneas que quedan ocultadas por otras partes del dibujo.
- Este interesante problema recibe el nombre de **Problema de la Linea del Horizonte ("Skyline Problem").**

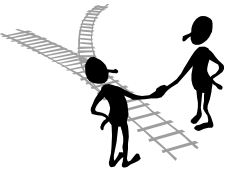


El problema de la linea del horizonte

- El problema se establece en los siguientes sencillos términos
- Dadas las situaciones exactas (coordenadas)
  - Por ejemplo (1,11,5), (2.5,6,7), (2.8,13,10), donde cada valor es la coordenada izquierda, la altura y la coordenada derecha de un edificio
- y las formas de n edificios rectangulares en una ciudad bi-dimensional,

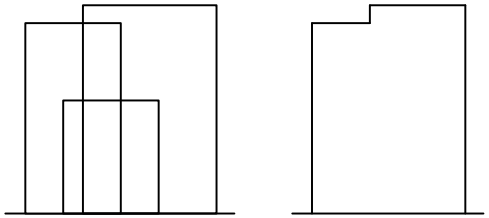


- construir un algoritmo Divide y Vencerás que calcule eficientemente el "skyline " (en dos dimensiones) de esos edificios, eliminando las líneas ocultas

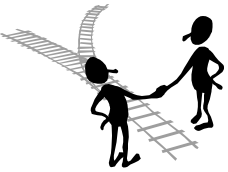


El problema de la linea del horizonte

- El problema se establece en los siguientes sencillos términos
- Dadas las situaciones exactas (coordenadas)
  - Por ejemplo (1,11,5), (2.5,6,7), (2.8,13,10), donde cada valor es la coordenada izquierda, la altura y la coordenada derecha de un edificio
- y las formas de n edificios rectangulares en una ciudad bi-dimensional,

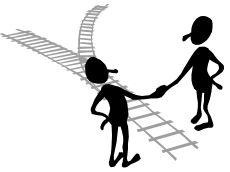


- construir un algoritmo Divide y Vencerás que calcule eficientemente el "skyline" (en dos dimensiones) de esos edificios, eliminando las líneas ocultas



Particularización del método DV

- **1. Tamaño:** el problema es una colección de edificios, por tanto el número de edificios en una colección es una elección natural para la forma de medir el tamaño del problema, así
  - Tamaño(Edificios) = número de edificios en el input.
- **2. Caso base del problema:** Como el input está restringido a la colección de edificios, entonces el caso base del problema es una colección constituida por un solo edificio, puesto que el "skyline" de un único edificio es el mismo edificio. Por tanto
  - TamañoCasoBase = 1



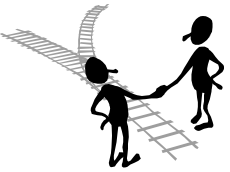
Particularización del método DV

- **3. Solución del Caso Base**: Como la solución para un único edificio es el edificio en si mismo, el cuerpo del procedimiento

  EncuentraSolucionCasoBase(Edificios; Skyline)

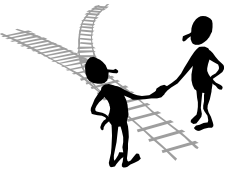
  puede definirse como sigue:

```
Procedimiento EncuentraSolucionCasoBase(Edificios; Skyline(
   Begin
    Skyline = Edificios[1[
   end;
```



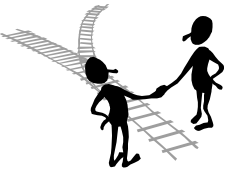
Particularización del método DV

- **4. División del problema**: El unico requerimiento es que el tamaño de los subproblemas obtenidos, de la colección de edificios, debe ser estrictamente menor que el tamaño del problema original.
- Teniendo en cuenta la complejidad en tiempo del algoritmo, funcionará mejor si los tamaños de los subproblemas son parecidos.
- Diseñaremos un procedimiento de division de manera que los dos subproblemas que obtendremos, EdificiosIzquierda y EdificiosDerecha, tengan tamaños aproximadamente iguales
- **5. Combinación**: Hay que combinar dos "skylines" subsoluciones en un único "skyline" :
- En esencia la parte de combinación del algoritmo lo que hace es tomar dos "skylines ", los analiza punto a punto desde la izquierda a la derecha, y en cada punto toma el mayor valor de los dos " skylines ".
- Ese punto es el que toma como valor del "skyline" combinado



Algoritmo DV del Skyline

```
Procedimiento Skyline(Edificios,Skyline)
  Begin
    If CasoBase(Edificios)
      Then
        EncuentraSolucionCasoBase(Edificios,Skyline)
      Else
        Begin
          Dividir(Edificios, EdificiosIzquierda, EdificiosDerecha)
          EncuentraSkyline(EdificiosIzquierda, SkylineIzquierda)
          EncuentraSkyline(EdificiosDerecha, SkylineDerecha)
          Combina(SkylineIzquierda, SkylineDerecha, Skyline)
        End
End;
```



Algoritmo DV del "Skyline"

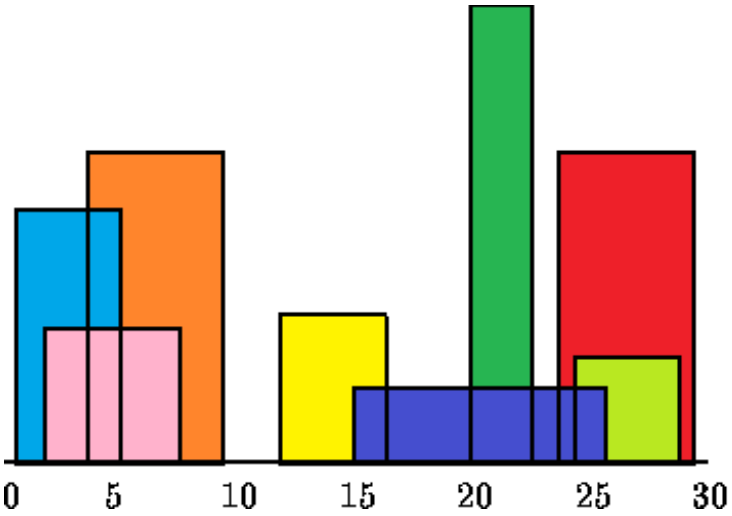
- Por tanto siguiendo la estrategia del DV, en ese algoritmo primero comprobamos si el array de edificios que nos dan contiene un solo edificio.
- Si es asi, la solución es ese mismo edificio
- En caso contrario, dividimos el conjunto de edificios, resolvemos recursivamente cada mitad, y finalmente combinamos los dos "skylines" obtenidos
- El tiempo del algoritmo se encuentra a partir de la recurrencia

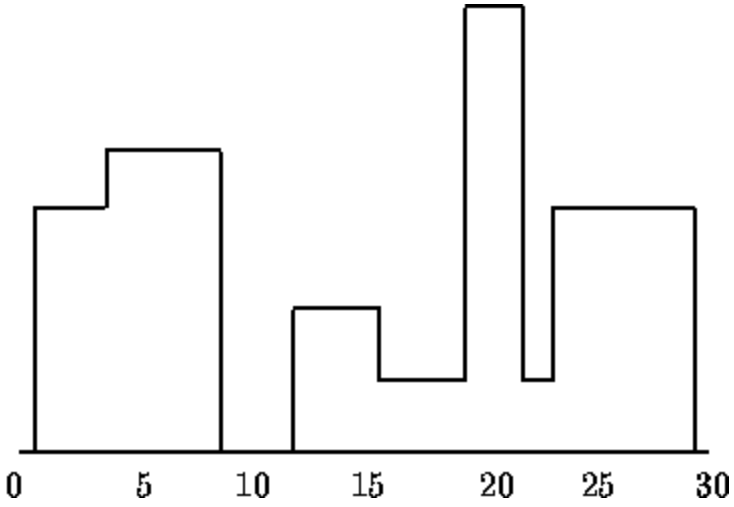
$$T(n) = 2T(n/2)+O(n)$$

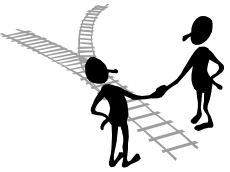
- Por tanto el algoritmo tiene orden O(nlogn)

Ejemplo del Algoritmo DV del Skyline

- Considerar el diagrama de la izquierda, en el que :
  - (1,11,5), (2,6,7), (3,13,9), (12,7,16), (14,3,25), (19,18,22), (23,13,29), (24,4,28)
- Obtener entonces el correspondiente Skyline
  - (1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 23, 13, 29, 0)







Otras aplicaciones de los Algoritmos DV

- Compresión de datos
  - Códigos de Huffman
- Informática gráfica
  - Multiplicación de matrices
- Biología computacional
- Gestión de cadenas de aprovisionamiento
- Interpolación
- Organización de la carga de trabajo en supercomputadores
- Teoría de la Señal
  - Transformada Rápida de Fourier
  - Algoritmo de Horner para evaluar repetidamente polinomios