BISOFT-28
Estructuras de Datos 2

Diseño de Algoritmos I:
Voracidad & Divide y Vencerás

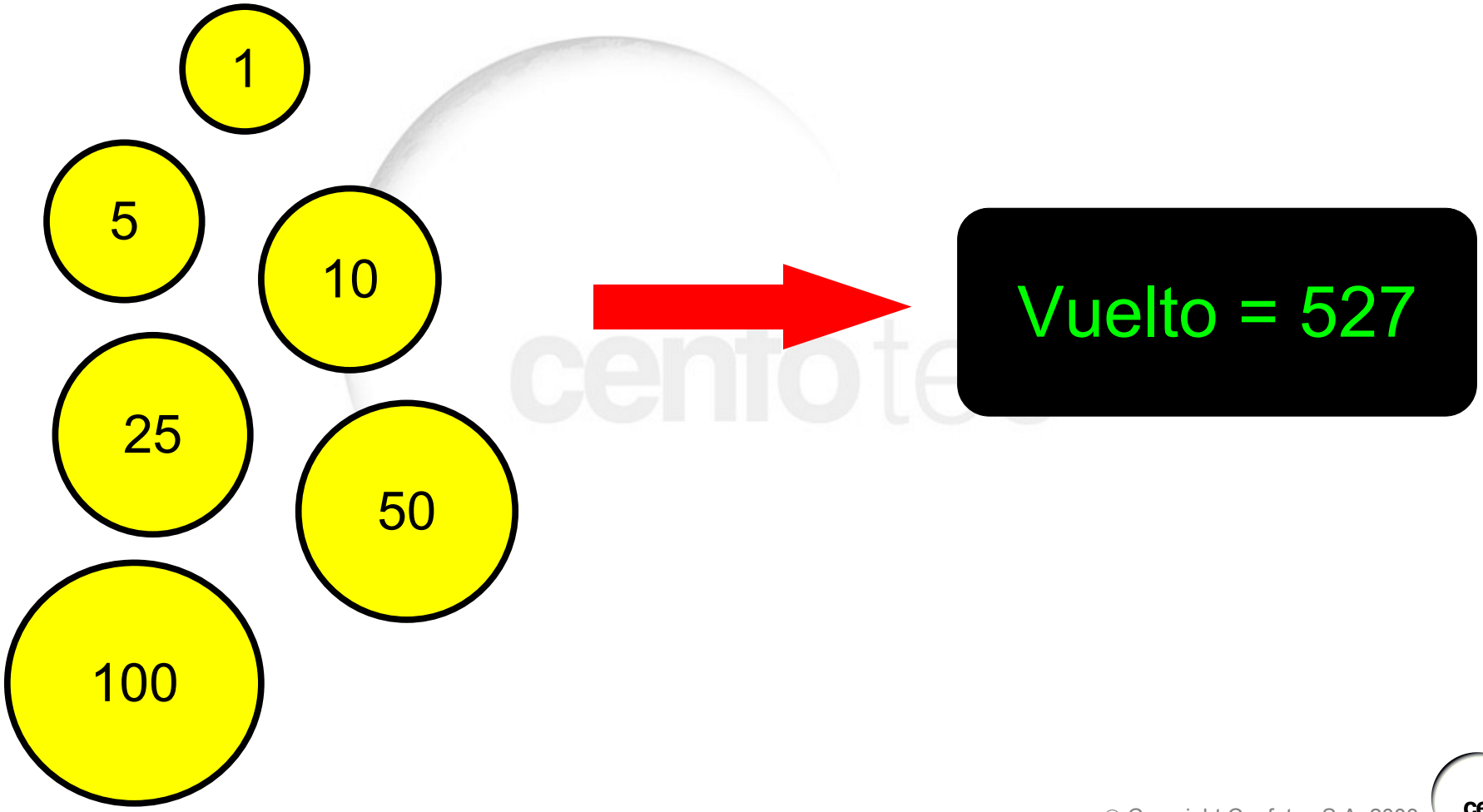
Instructor Esteban Meneses

Introducción

- “Existen muchas formas de matar pulgas”.
- Los algoritmos **voraces** son muy sencillos de crear, pero típicamente no consiguen la solución óptima.
- **Divide y conquista** es una técnica empleada para encontrar la solución a un problema a partir de subcasos.
- La **programación dinámica** pretende minimizar el cálculo en los subcasos.
- El **retroceso** supone la búsqueda de la solución a partir de reconsiderar decisiones tomadas.

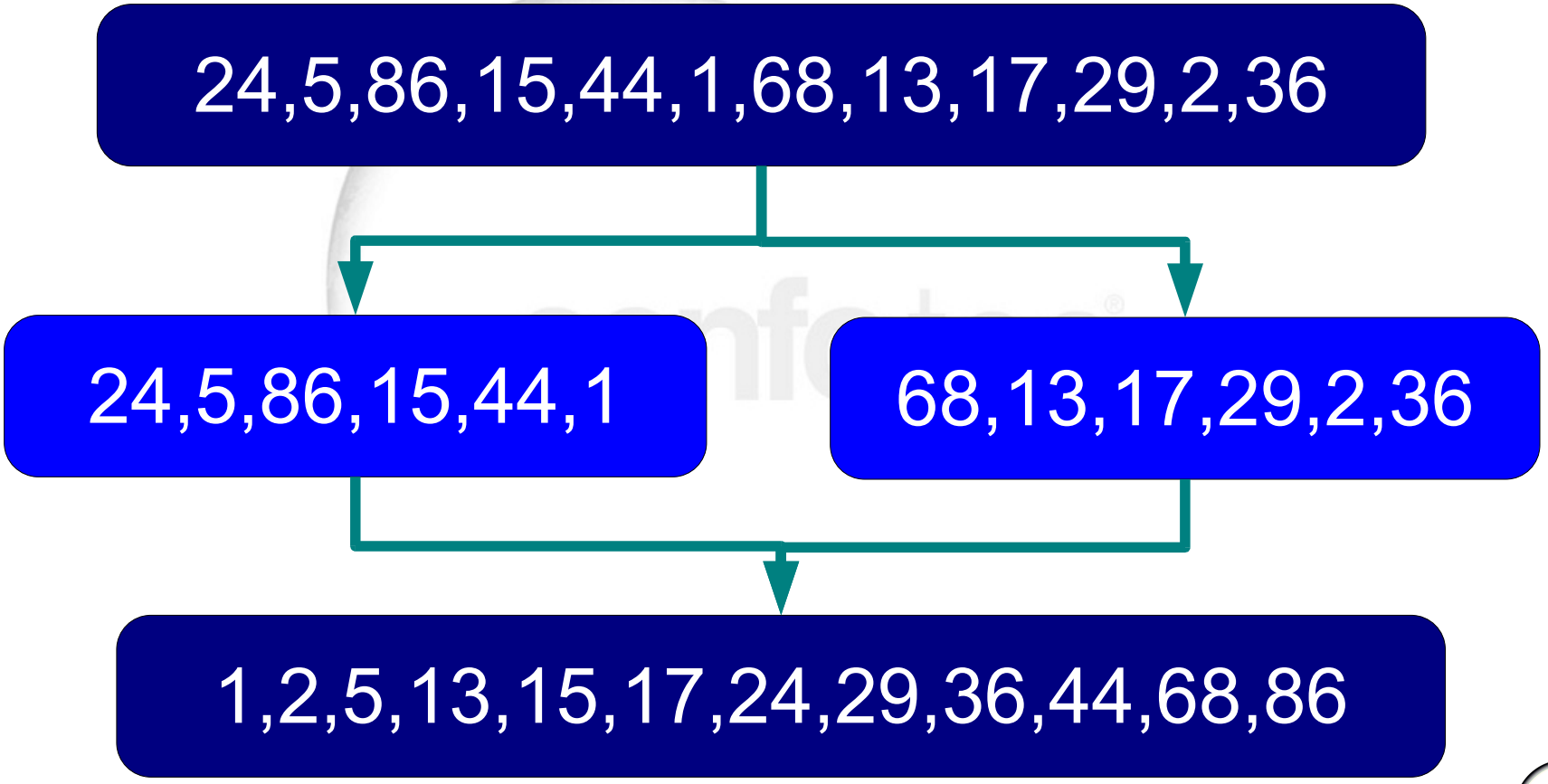
Estrategias de Diseño de Algoritmos

- Técnicas ávidas: no **reconsideran** decisiones una vez hechas.



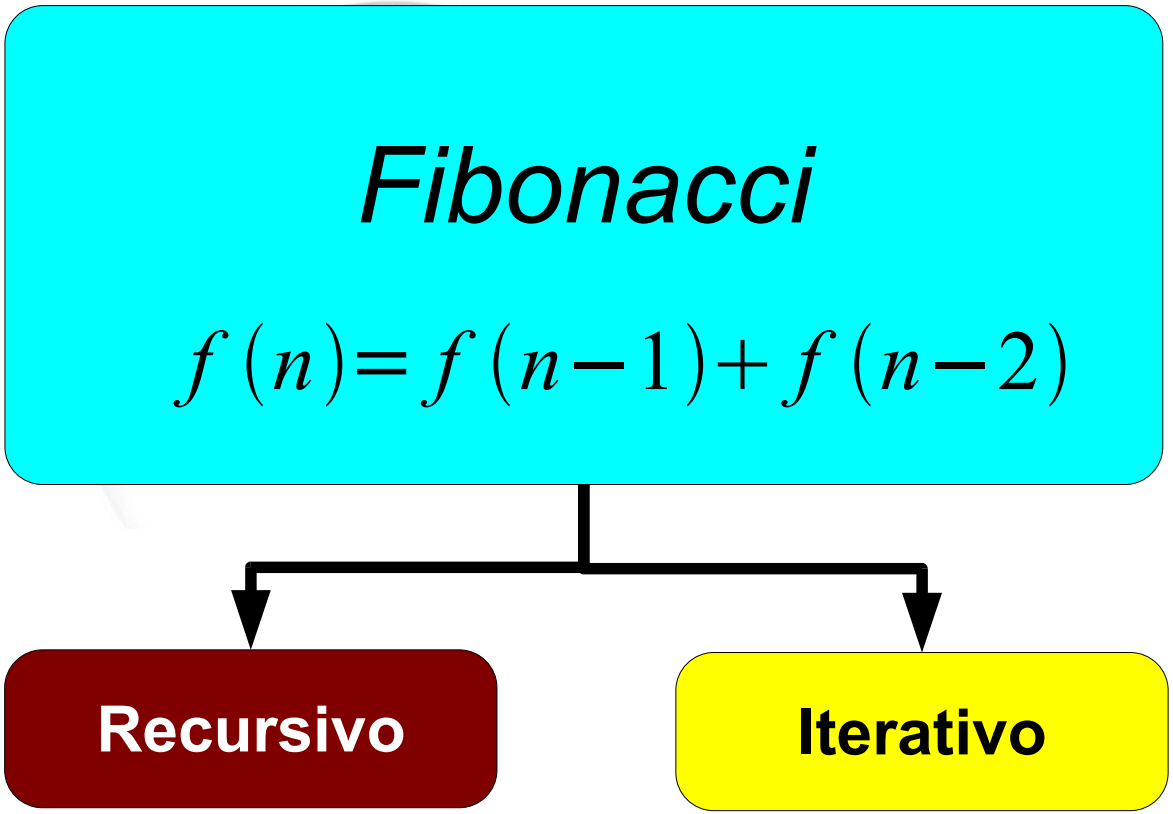
Estrategias de Diseño de Algoritmos

- Divide y conquista: dividen el problema en **subproblemas** y resuelven recursivamente los mismos.



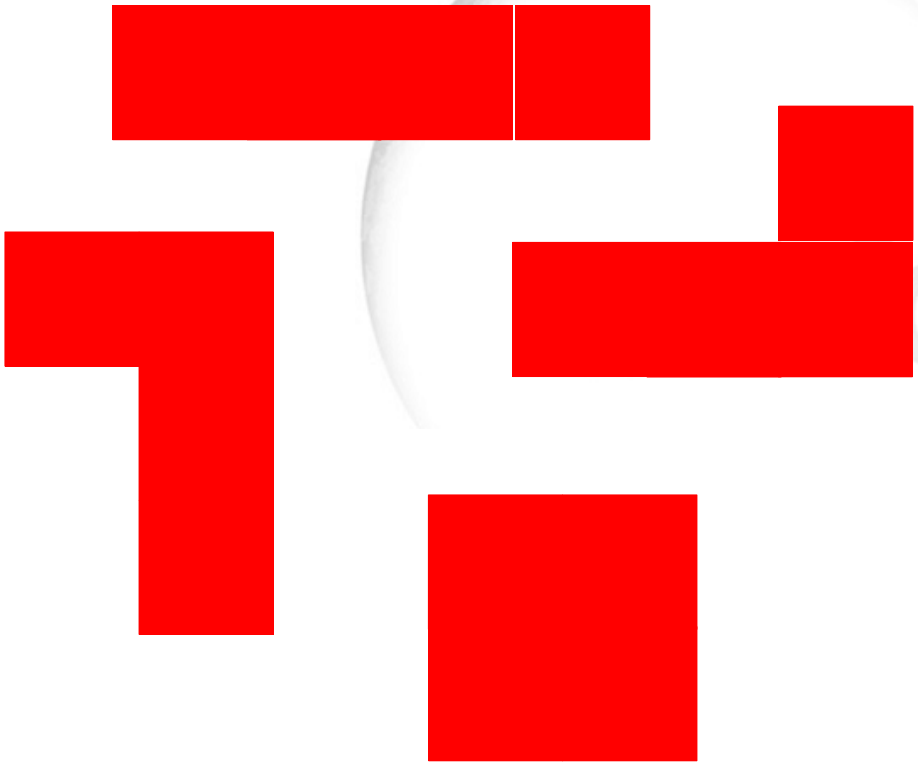
Estrategias de Diseño de Algoritmos

- Programación dinámica: utilizan una **tabla** para resolver problemas más pequeños y luego construyen la solución.



Estrategias de Diseño de Algoritmos

- Retroceso: algunas decisiones pueden ser **revertidas**.



Voracidad

- **Características**:
  - Sencillos.
  - Eficientes.
  - No necesariamente encuentran la solución óptima.
  - Una vez tomada una decisión es imposible revertirla.
- **Ejemplos**:
  - Algoritmo de *Dijkstra*.
  - Algoritmo de *Prim*.

Caso 1: Un problema sencillo de calendarización

- Se tiene un único **procesador**.
- Se tienen *N* **trabajos** $j_1, j_2, \ldots, j_N$ con **tiempos** de corrida $t_1, t_2, \ldots, t_N$
- ¿Cuál es la mejor forma de **calendarizar** los trabajos de manera que se minimice el tiempo de espera?
- ¿Qué sucede cuando se tienen **varios** procesadores?

Ejemplo

Trabajo	Tiempo
j1	15
j2	8
j3	3
j4	10

Calendarización 1: **j1 – j2 – j3 - j4**

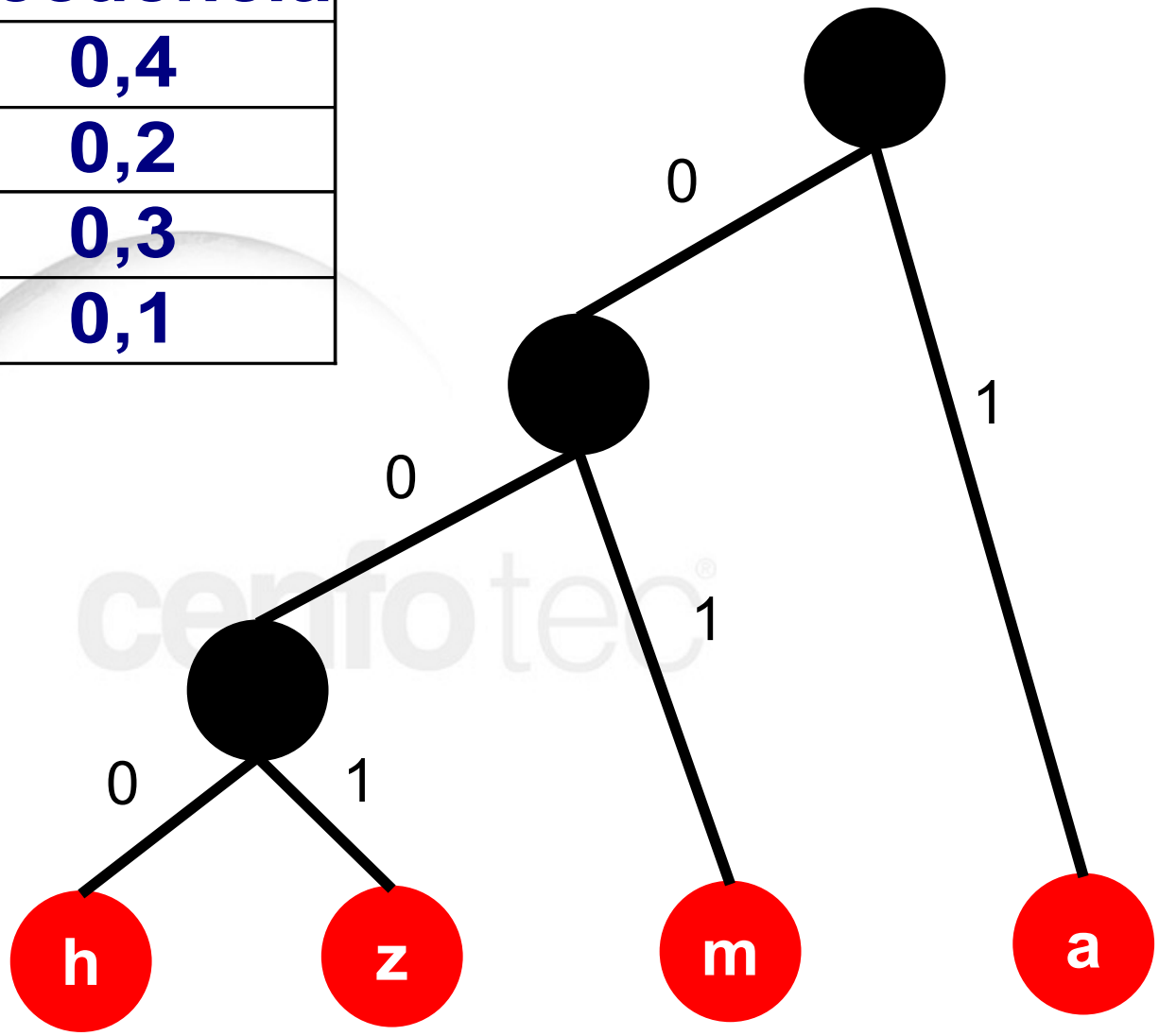
Calendarización 2: **j3 – j2 – j4 - j1**

Caso 2: Códigos de Huffman

- Se tiene un texto que se quiere **comprimir**.
- Se debe determinar la **frecuencia** de cada símbolo.
- Asociar el código más **pequeño** al símbolo más frecuente.
- Codificación **ASCII** = 8 bits.

Ejemplo

Símbolo	Frecuencia
a	0,4
h	0,2
m	0,3
z	0,1

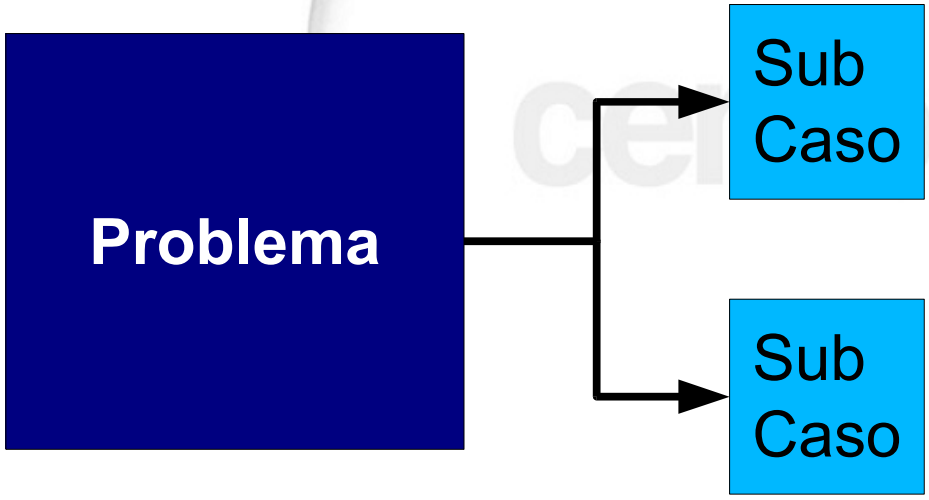


Caso 3: Empaque aproximado

- Se tienen *N* ítemes: $s_1, s_2, \ldots, s_N$ y una cantidad infinita de **recipientes** donde se deben empacar los ítemes.
- ¿Cómo se deben empacar de manera que se utilice la **menor** cantidad posible de recipientes?
- Algoritmos ***on-line***:
  - *Next-fit*
  - *First-fit*
  - *Best-fit*
- Algoritmo ***off-line***:
  - *First-fit non-increasing*

Divide y Vencerás

- **Descompone** el problema en casos más pequeños y luego combina las soluciones.
- **Ejemplos**:
  - *MergeSort*.
  - *QuickSort.*



Caso 4: Encontrar el par de puntos más cercanos

- Se tiene un **conjunto** *P* de *N* puntos y se requiere encontrar el par de puntos que esté más cercano.
- Sea $P = \{p_1, p_2, ..., p_N\}$, donde $p_i=(x_i,y_i)$, la **distancia** utilizada *d* es la distancia euclídea:

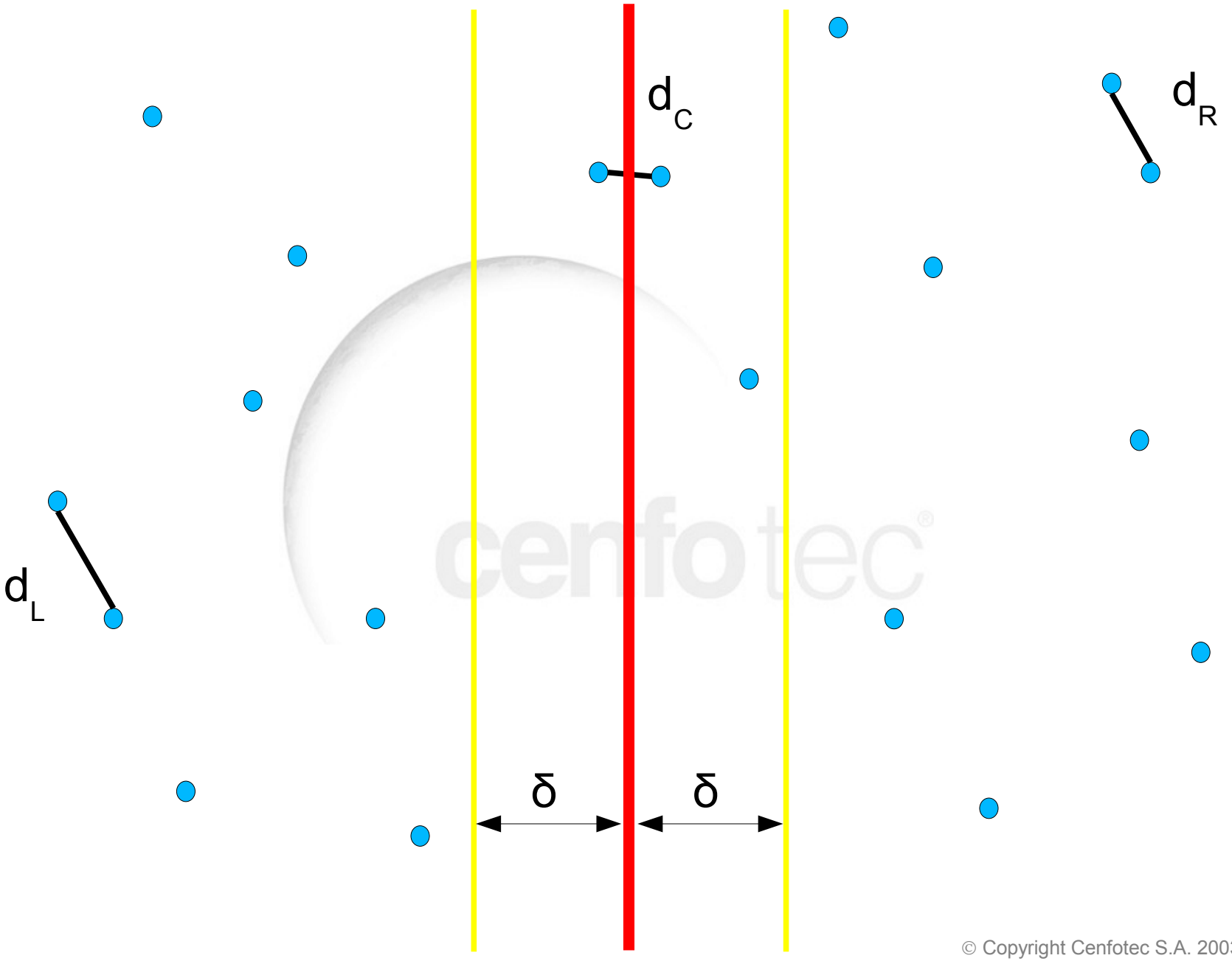
$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- Método **exhaustivo**: comparar todas las posibles parejas: $N(N-1)/2 = O(N^2)$
- Usando **divide y vencerás**: $O(N \log(N))$

Caso 4: Encontrar el par de puntos más cercanos

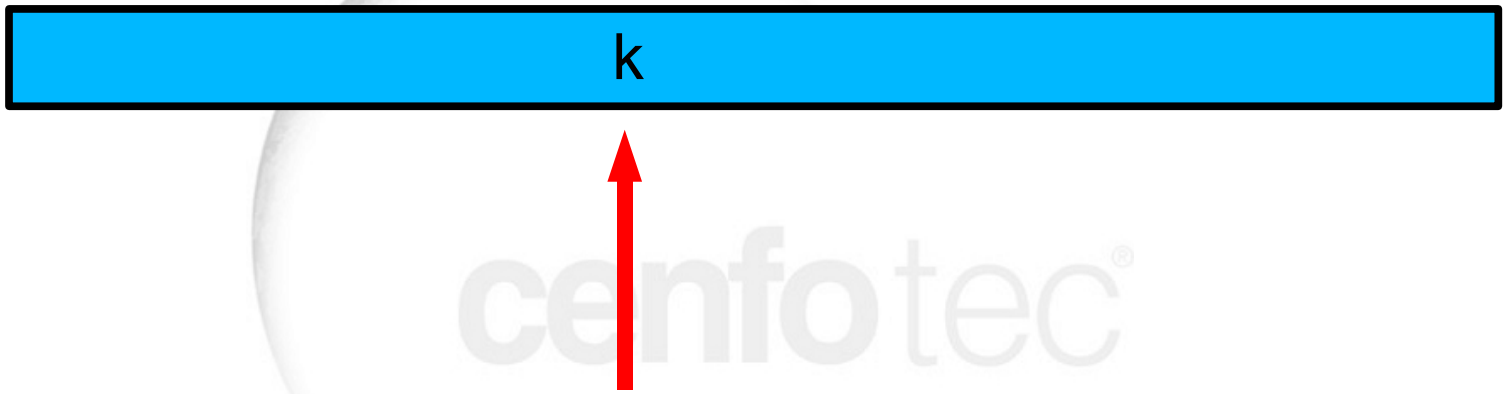
- Algoritmo:
  - **Dividir** el espacio de búsqueda. Encontrar la distancia mínima en la izquierda d_L y la distancia mínima en la derecha d_R.
  - Sea $\delta=\min\{d_L,d_R\}$. **Comparar** las distancias en la franja de amplitud δ a partir del centro.

Caso 4: Encontrar el par de puntos más cercanos



Caso 5: Encontrar la k-mediana

- La **k-mediana** es el elemento que estaría en la posición k-ésima si el arreglo estuviera ordenado.



Caso 5: Encontrar la k-mediana

- Algoritmo:
  - **Dividir** la lista en grupos de 5.
  - **Ordenar** cada grupo de 5 elementos y en cada grupo hallar el centro.
  - **Ordenar** el vector de todos los centros, el elemento del medio será la mediana, *m*.
  - **Separar** la lista original en 3 grupos:
    - $S_1(< m)$, $S_2(= m)$ y $S_3(> m)$
  - Si $|S_1| > k$, buscar la k-mediana en S_1.
  - Si $|S_1 + S_2| > k$, *m* es la k-mediana.
  - Si $|S_1 + S_2| < k$, buscar la $(k-|S_1 + S_2|)$mediana en S_3.

FIN