ESTRATEGIAS DE PROGRAMACIÓN
(CONTINUACIÓN-2)

Programación 3

Javier Miranda

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Estrategias básicas de programación

- Fuerza bruta
- Vuelta atrás (backtracking)
- Greedy

- Divide y vencerás
  - Reduce y vencerás
  - Programación Dinámica

Estrategia Divide y Vencerás

- Probablemente la estrategia más conocida.

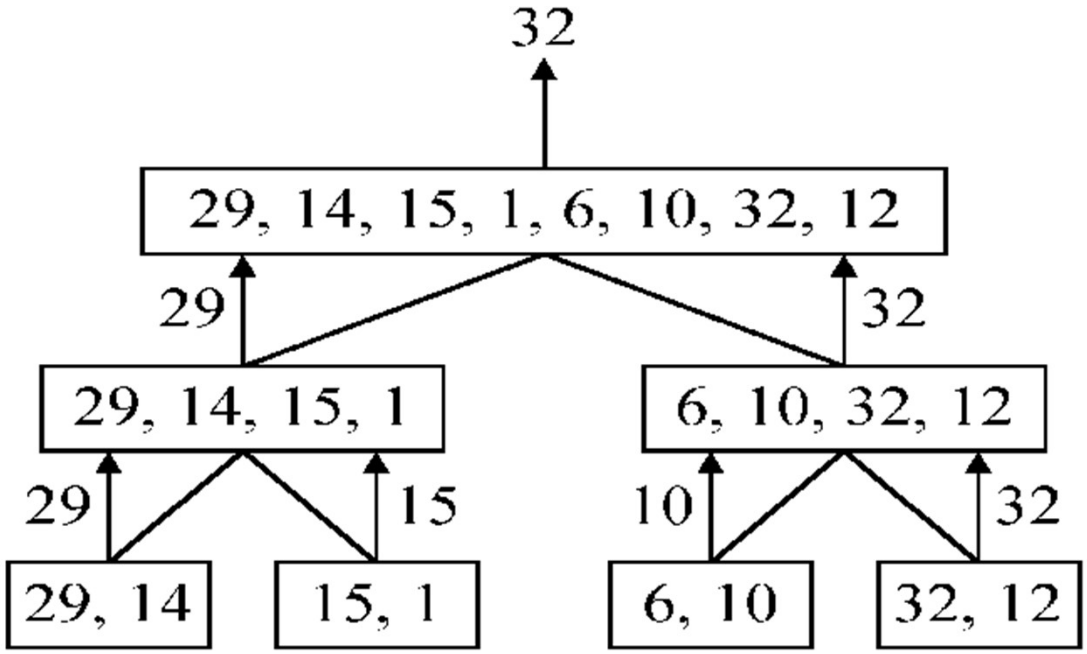
- Consta de 3 pasos:

1. **Dividir** el ejemplar en dos o más subproblemas
2. **Resolver** recursivamente los subproblemas
3. **Combinar** las soluciones para obtener la solución completa

https://en.wikipedia.org/wiki/Divide_and_conquer_algorithm

Ejemplo (1/3)

- Buscar el máximo de un conjunto de números

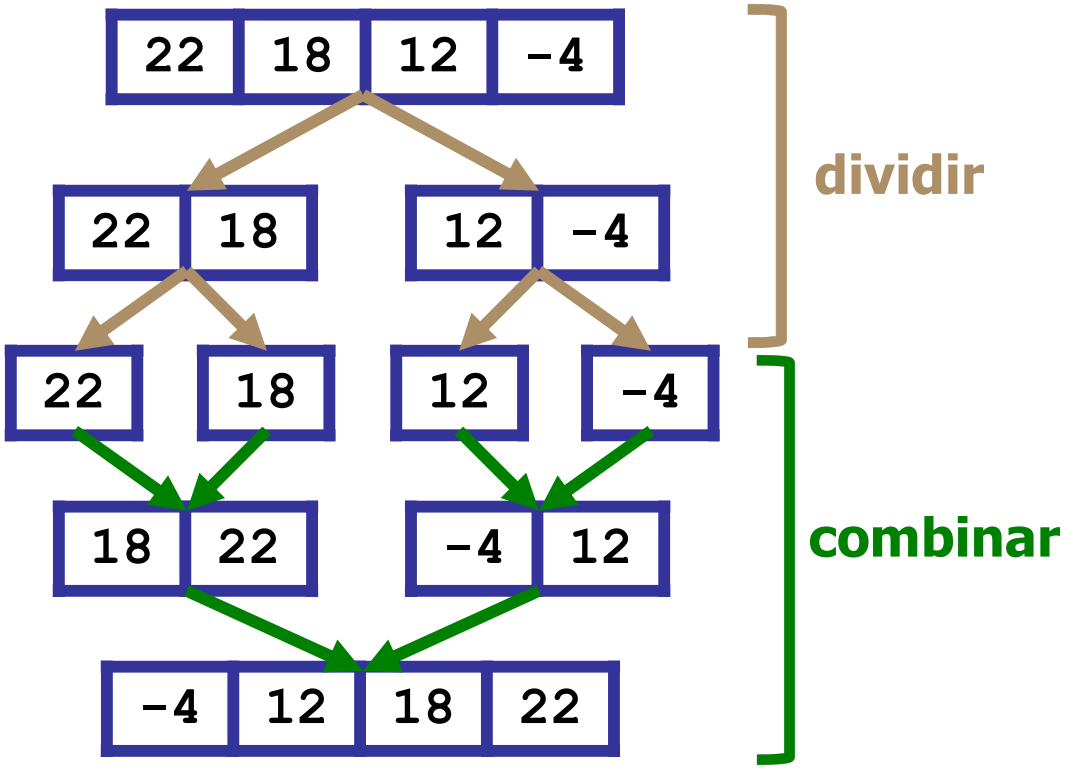


https://www.geeksforgeeks.org/divide-and-conquer/

1945

Ejemplo (2/3): Merge Sort

- Dividimos el array en dos partes
- Recursivamente ordenamos la primera mitad
- Recursivamente ordenamos la segunda mitad
- Unimos las dos partes ordenadas (*merge*)



John von Neumann

Pseudo-código

```
MergeSort (data, left, right) {
    if (left < right) {
       mid = divide (data, left, right)
       MergeSort (data, left, mid-1)
       MergeSort (data, mid, right)
       merge(data, left, mid, right)
    }
}
```

Llamadas recursivas

Pseudocode for Merge:

```
C = output [length = n]
A = 1st sorted array [n/2]
B = 2nd sorted array [n/2]
i = 1
j = 1

for k = 1 to n
        if A(i) < B(j)
                C(k) = A(i)
                i++
        else [B(j) < A(i)]
                C(k) = B(j)
                j++
end
```

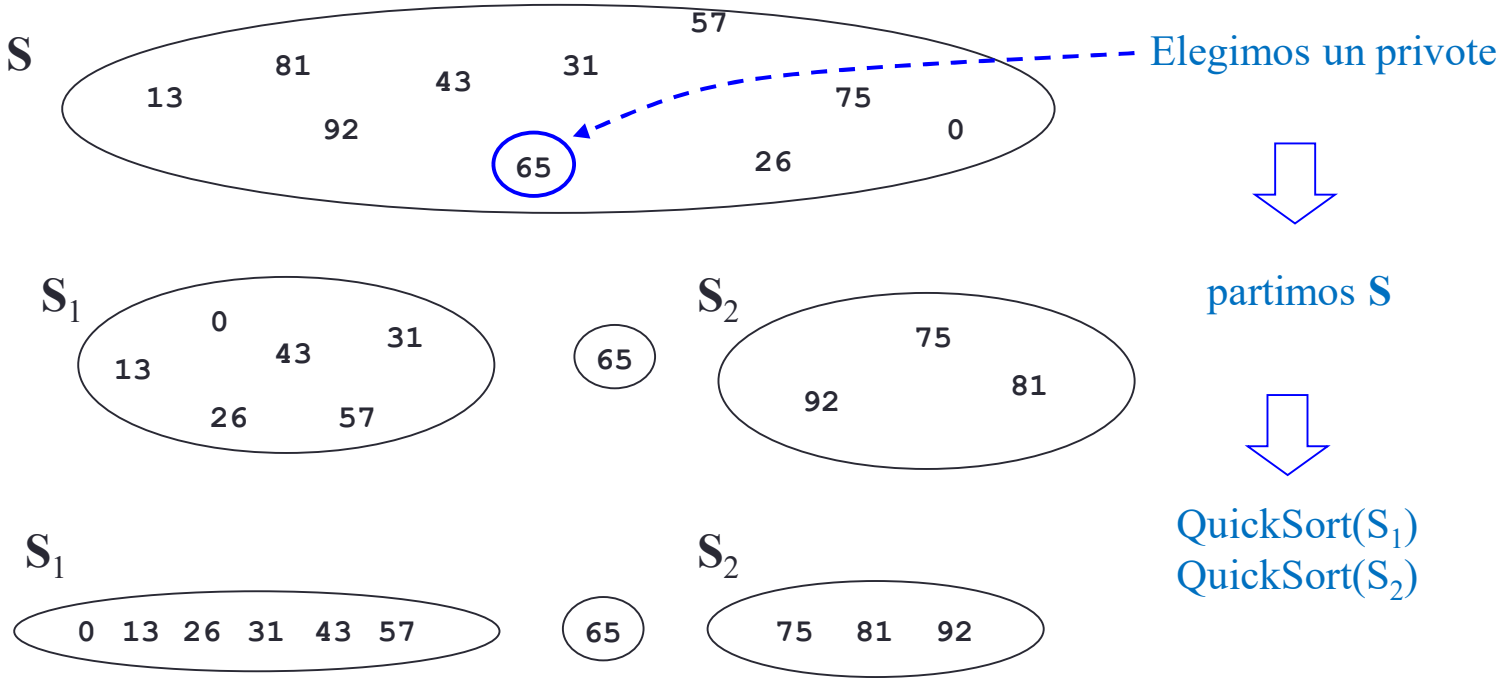
(ignores end cases)

https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/Mergesort.html

1959

Ejemplo (3/3): Quicksort

- Elegimos un elemento como ***pivote***, situando a su izquierda los elementos que sean más pequeños que él y los mayores a su derecha.
- Al final de este proceso el elemento que se ha usado como *pivote* queda en su posición definitiva, y ordenamos recursivamente las dos mitades.



Tony Hoare

Pseudo-código

```
QuickSort (data, left, right)
   if (left < right)
      pivot = Partition (data, left, right)
      Quicksort (data, left, pivot-1)
      Quicksort (data, pivot+1, right)
```

Compáralo con MergeSort:

```
MergeSort (data, left, right)
    if (left < right)
       mid = divide (data, left, right)
       MergeSort (data, left, mid-1)
       MergeSort (data, mid+1, right)
       merge(data, left, mid+1, right)
```

Estrategias de Elección del Pivote en QuickSort

1. Utilizar el primer (o el último) elemento
   - Funciona bien cuando los números están desordenados
   - Rendimiento malo (y potencial desbordamiento de pila) si los números están ordenados (todos los elementos van a S1 o a S2)

2. Elegir el pivote aleatoriamente (*Randomized Algorithm*)
   - Es generalmente más seguro
   - El coste de elegir un número aleatorio es importante

3. Desordenar el array y aplicar la primera estrategia

https://www.hackerearth.com/practice/algorithms/sorting/quick-sort/visualize/

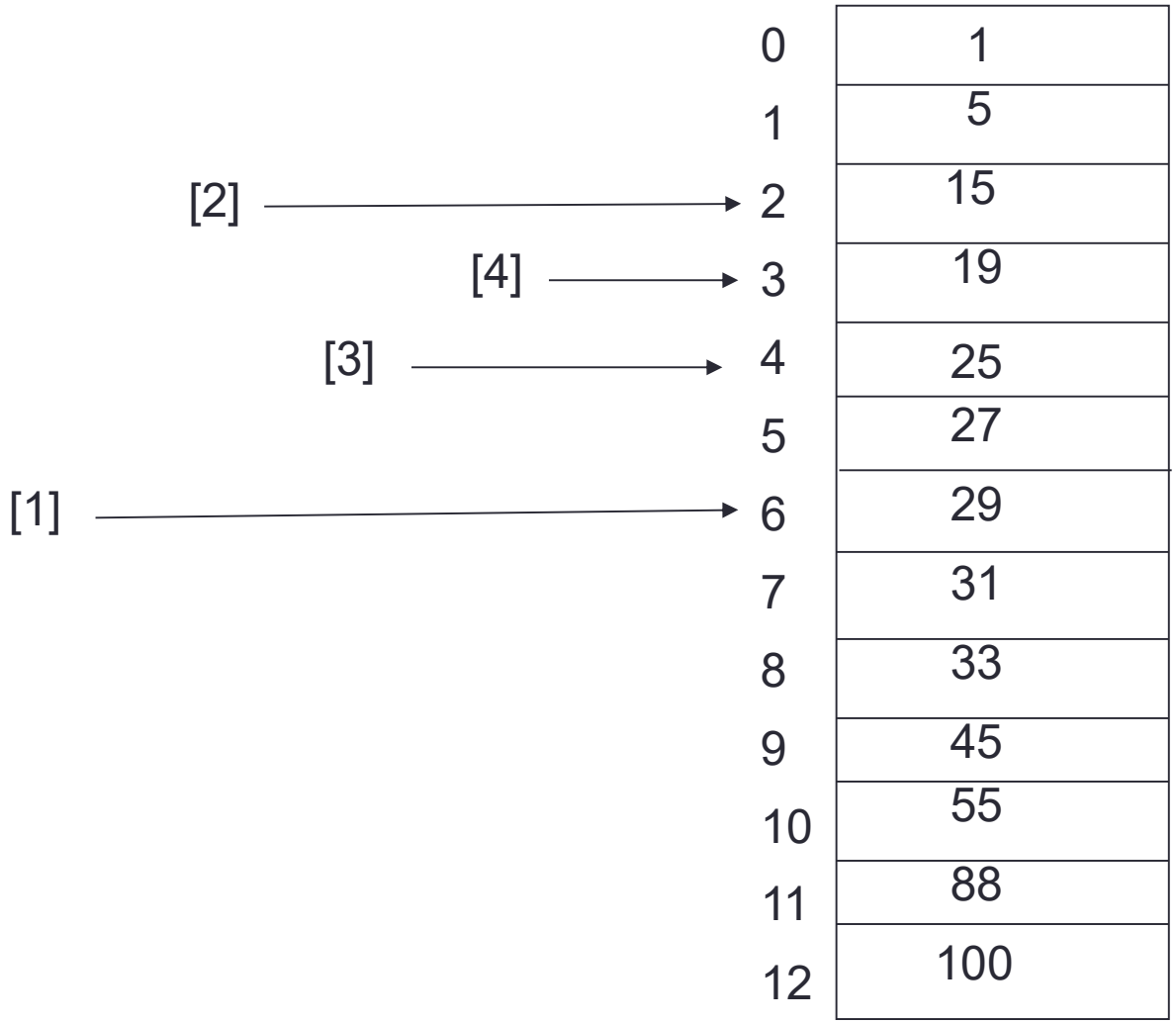
Decrease and Conquer

- Se utiliza este nombre en algoritmos que dividen el problema en varios subproblemas pero **sólamente necesitan resolver uno** de los subproblemas

- Ejemplos
  - Búsqueda binaria en un array ordenado
  - Busqueda de la mediana

Ejemplo (1/2): Búsqueda binaria

search key = 19



[1]: (0 + 12) / 2 = 6
[2]: (0 + 5) / 2 = 2
[3]: (3 + 5) / 2 = 4
[4]: (3 + 3) / 2 = 3

Ejemplo (2/2): Búsqueda de la Mediana

- Dado un vector no ordenado con un número impar de elementos distintos calcular el valor del elemento que va en la mitad del vector (posición $\lceil n/2 \rceil$) si el vector estuviese ordenado

- Vamos a ver una solución más general que nos permite calcular el valor del elemento que va en la posición k que necesitemos!
  - Se resuelve fácilmente reutilizando el método de partición que vimos en QuickSort

https://en.wikipedia.org/wiki/Quickselect

Ejemplo: Búsqueda de la Mediana

```
function QuickSelect(data, left, right, k)
    if left = right
       return data[left]

    pivot_index := partition(data, left, right)
    // El pivote está en su posición definitiva

    if k = pivot_index
        return data[k]
    else if k < pivot_index
        return QuickSelect(data, left, pivot_index - 1, k)
    else
        return QuickSelect(data, pivot_index + 1, right, k)
```

Solución Recursiva

https://en.wikipedia.org/wiki/Quickselect

Ejemplo: Búsqueda de la Mediana

```
function QuickSelect(data, left, right, k)
    loop
        if left = right
            return data[left]

        pivot_index := partition(data, left, right)
        // The pivot is in its final sorted position

        if k = pivot_index
            return data[k]
        else if k < pivot_index
            right := pivot_index - 1
        else
            left := pivot_index + 1
```

Solución Iterativa

Evita la recursividad de cola!