Estructuras de Datos

Algoritmos Tipo Divide y Vencerás

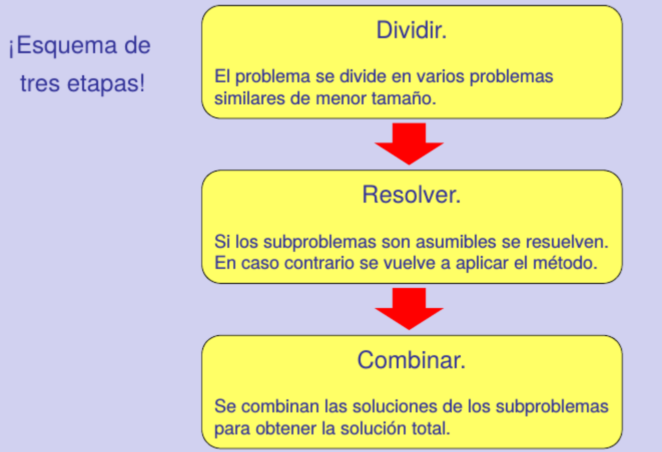
Luis Garreta

Ingeniería de Sistemas y Computación
Pontificia Universidad Javeriana – Cali

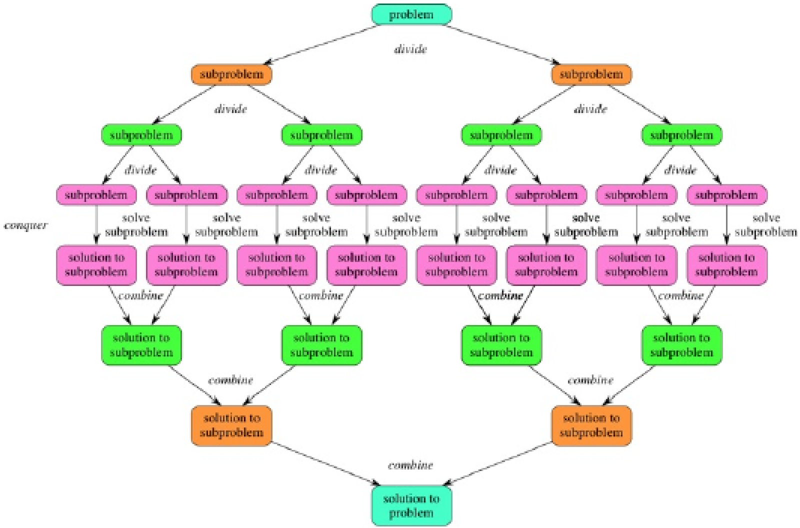
22 de agosto de 2017

Divide y Vencerás

Una de las estrategias más conocidas para el diseño de algoritmos:



Esquema Divide y Vencerás



Características

Características que deben cumplir los problemas para que se pueda aplicar esta técnica:

- El problema se debe poder descomponer en otros similares pero más pequeños.
- Los nuevos problemas deben ser disjuntos.
- Debe ser posible combinar las soluciones individuales para obtener la global.

Ejemplo Sencillo: *maxArreglo*

Encontrar el valor máximo en un arreglo de enteros

A : [11, 21, 22, ..., .. 9, 8, 55]

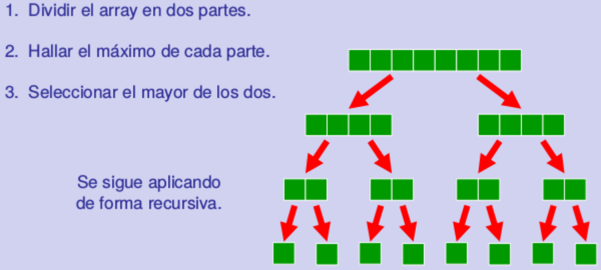
- Algoritmo Fuerza bruta

```
int maxArreglo (int A[], int n) {
  int maximo = A [0];
  for (int i=1; i<n; i++) {
    if (A[i] > maximo)
      maximo = A[i]

  return maximo;
}
```

- Pertence a O (n):

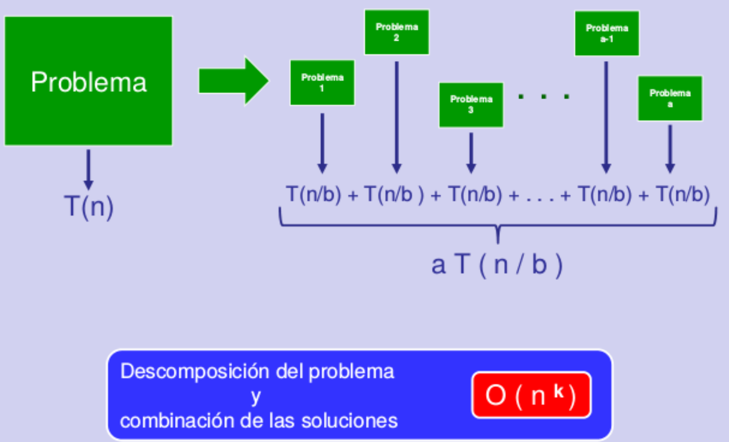
Enfoque Divide y Venceras *maxArreglo*



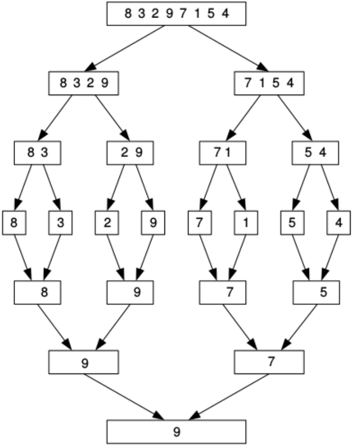
Algoritmo DyV *maxArreglo*

```
int maxArreglo (int A[], int ini, int fin) {
  int mitad, maxIzquierda, maxDerecha;
  if (ini == fin)
    return A [ini];
  else {
    mitad = (ini+fin)/2;
    maxIzquierda = maxArreglo (A, ini, mitad);
    maxDerecha   = maxArreglo (A, mitad+1, fin);
    if (maxIzquierda > maxDerecha)
      return maxIzquierda;
    else
      return maxDerecha;
  }
  return -1;    // Buena prActica por si no retorna nada
}
```

Complejidad del Método



Analisis de Complejidad



En general responderá a esta ecuación:

$$T(n) = a\,T(n/b) + O(n^k)$$

a : Número de subproblemas en que se descompone.

n/b : Tamaño de cada nuevo subproblema.

$$a \geq 1, \quad b \geq 2, \quad k \geq 0$$

Cuya solución es:

$$T(n) = \begin{cases} O(n^k), & a < b^k \\ O(n^k \log n), & a = b^k \\ O(n^{\log_b a}), & a > b^k \end{cases}$$

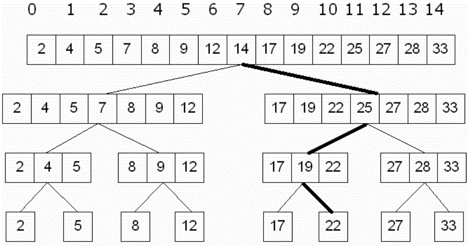
¡Teorema maestro!

Búsqueda Binaria

```
int busquedaBinaria (int A[], int ini, int fin, int k) {
  int mitad;
  if (fin >= ini) {
    mitad = ini + (fin-ini)/2;
    if (k == A [mitad])
      return mitad;
    if (k < A [mitad])
      return busquedaBinaria (A, ini, mitad-1, k);
    else
      return busquedaBinaria (A, mitad+1, fin, k);
  }
  return -1;
}
```

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

Analisis Búsqueda Binaria



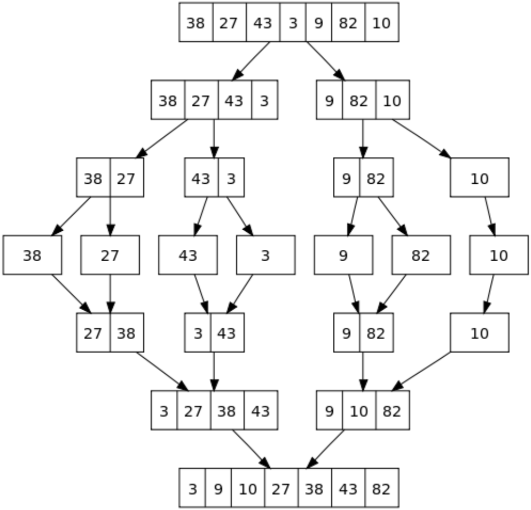
$$T(n) = \begin{cases} T(n/2) + \Theta(1) & \text{for } n > 1 \\ \Theta(1) & \text{for } n = 1 \end{cases}$$

$a = 1, b = 2, f(n) = \Theta(1),$

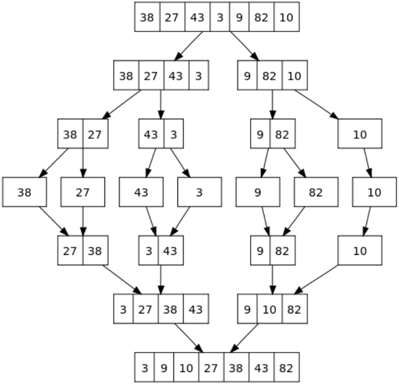
$n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

Case 2 $\Rightarrow T(n) = \Theta(\log n)$

Ejemplo 2: Ordenamiento Merge Sort



Dividir y Conquistar Mergesort



DIVIDIR: Dividir el arreglo a ordenar de n elementos en dos arreglos de tamaño n/2
CONQUISTAR: Ordenar recursivamente los dos subarreglos usiando Mergesort
COMBINAR: Mezclar los dos subarreglos ordenados para producir un nuevo arreglo ordenado

Función Merge de Mergesort

- Input: Sorted arrays K[1..n_1], L[1..n_2]
- Output: Merged sorted array M[1.. n_1+n_2]

$$
\begin{array}{l}
i = 1,\ \ j = 1 \\
\textbf{for}\ t = 1\ \text{to}\ n_1 + n_2 \\
\quad \{\ \textbf{if}\ (i \le n_1\ \text{and}\ (\ j > n_2\ \ \text{or}\ K[i] < L[j])\ ) \\
\qquad\qquad \textbf{then}\ \{\ M[t] = K[i],\ \ i = i + 1\ \} \\
\qquad\qquad \textbf{else}\ \{\ M[t] = L[j],\ \ j = j + 1\ \} \\
\quad \}
\end{array}
$$

Linear Time Complexity: $\Theta(n_1 + n_2)$

Algoritmo MergeSort

Merge-Sort A[1...n]

If $n > 1$ then

1. Recursively merge-sort $A[1 \cdots \lfloor n/2 \rfloor]$ and $A[\lfloor n/2 \rfloor + 1 \cdots n]$
2. Merge the two sorted subsequences

Análisis de Mergesort

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{for } n > 1 \\ \Theta(1) & \text{for } n = 1 \end{cases}$$

Assume for simplicity that n is a power of 2

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

$$a = 2, b = 2, f(n) = \Theta(n), n^{\log_b a} = n^{\log_2 2} = n$$

$$\text{Case 2} \;\; \Rightarrow T(n) = \Theta(n \log n)$$

Tarea: Problema del subarreglo máximo

- Input: Array A[1…n] of integers (positive and negative)
- Problem: Compute a subarray A[i*…j*] with maximum sum
  i.e., if s(i,j) denotes the sum of the elements of a subarray A[i…j], $s(i,j) = \sum_{k=i}^{j} A[k]$

We want to compute indices i*≤j* such that

$$s(i^*, j^*) = \max\{s(i,j) \mid 1 \le i \le j \le n\}$$

Example: $3\ \ -4\ \ \underbrace{5\ \ -2\ \ -2\ \ 6\ \ -3\ \ 5}_{\text{max sum} = 9}\ \ -3\ \ 2$

Ejercicios

Obtenga los límites asintóticos (O(n)) para T(n) en las siguientes recurrencias:

a. $T(n) = 2T(n/2) + n^4$.

b. $T(n) = T(7n/10) + n$.

c. $T(n) = 16T(n/4) + n^2$.

d. $T(n) = 7T(n/3) + n^2$.

e. $T(n) = 7T(n/2) + n^2$.

f. $T(n) = 2T(n/4) + \sqrt{n}$.

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