Teoria de Algoritmos

Capitulo 2: Algoritmos Divide y Venceras

Tema 5: Busqueda y ordenación

- Algoritmos de busqueda
- Algoritmos de ordenacion
  - Ordenacion por mezcla
  - Quicksort

Algoritmos de busqueda

- Busqueda lineal (secuencial)
  - Figo
  - Ayala
  - Tamudo
  - Raul
  - Djalminha
  - Cañizares
  - Zidane
  - Saviola
  - Ronaldo
  - Hierro
  - Valeron
  - Iker
  - Joaquin
  - Tristan

- 14 items
- Numero minimo de comparaciones = 1
- Numero maximo de comparaciones = 13
- En promedio: 13/2 = 6 o 7 comparaciones

No es un metodo muy eficiente

Por tanto no se usará ¡nunca!

- Busqueda lineal sobre una lista ordenada
- Busqueda binaria

Busqueda lineal en una lista ordenada

Ayala
Cañizares
Djalminha
Figo
Hierro
Iker
Joaquin
Raul
Ronaldo
Saviola
Tamudo
Tristan
Valeron
Zidane

- 14 items
- Numero minimo de comparaciones = 1
- Numero maximo de comparaciones = 13
- Numero promedio: 13/2 = 6 o 7

¿Por que entonces es mas eficiente buscar sobre una lista ordenada que en una no ordenada?

¿Deberiamos ordenar siempre la lista antes de buscar?

Busqueda lineal en una lista ordenada

```
Funcion BuscaBin (T[1, 2, ..., n])
     If n = 0 or x < T[1]
     then return 0
Return Binrec (T, x)
```

```
Funcion Binrec (T[i, ...,j], x)
   if i = j then return i
   k = i + j + 1/ div 2
   if x < t[k]
   then return binrec (t[i, ..., k – 1], x)
   else return binrec (t[k, ..., j]
```

- $T(n) = O(1)$ si $n \leq 0$

  $= T(n/2) + a$ si $n \geq 0$
- $T(n) = c_1 \log n$, entonces $T(n)$ es $O(\log n)$.
- La búsqueda binaria mas que ser una técnica DV pura, es un caso de simplificación.

Algoritmos de Ordenación

- El esquema general de ordenación Divide y Vencerás es el siguiente

```
Algoritmo General de Ordenacion con Divide y Vencerás
Begin Algoritmo
    Iniciar Ordenar(L)
      Si L tiene longitud mayor de 1 Entonces
      Begin
            Partir la lista en dos listas, izquierda y derecha
              Iniciar Ordenar(izquierda)
              Iniciar Ordenar(derecha)
              Combinar izquierda y derecha
      End
End Algoritmo
```

Ordenacion por mezcla

- Aplicamos el metodo DV a la resolucion del siguiente problema de ordenacion
- **Problema**: Dados n elementos de la misma naturaleza, ordenarlos en orden no decreciente
- Divide y Venceras:
  - Si n=1 terminar (toda lista de 1 elemento esta ordenada)
  - Si n>1, partir la lista de elementos en dos o mas subcolecciones; ordenar cada una de ellas; combinar en una sola lista.

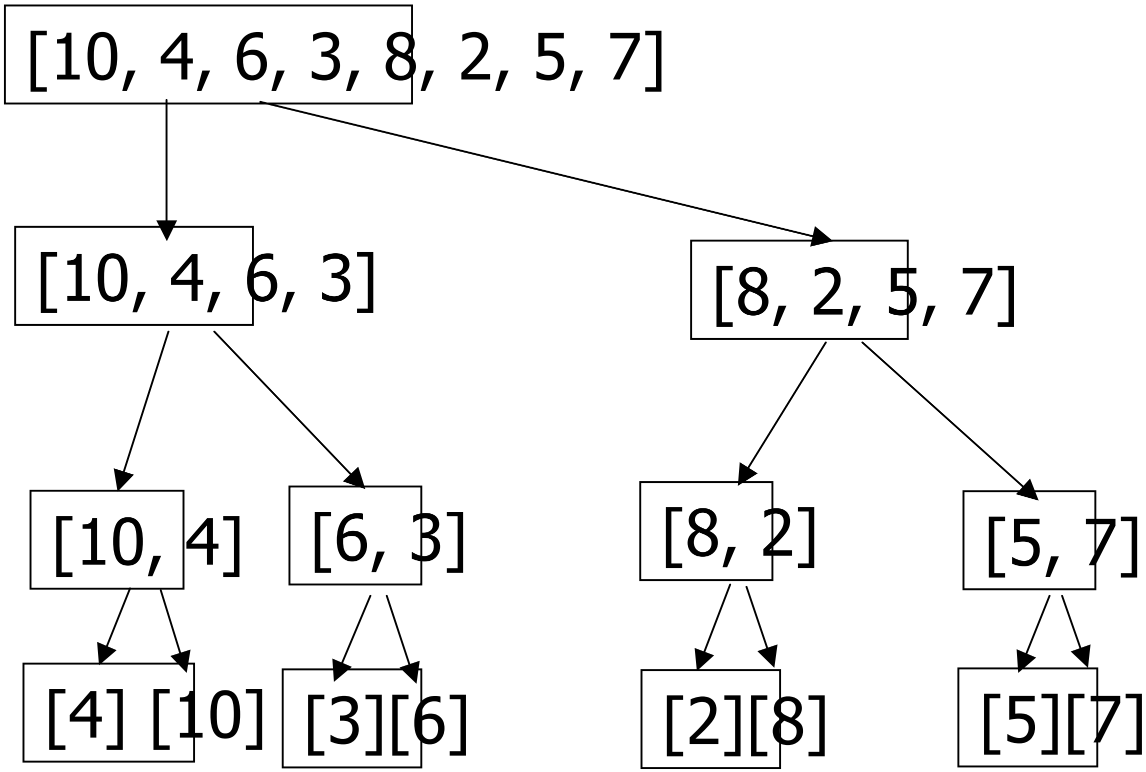
Pero, **¿Como hacer la particion?**

Ordenacion por mezcla

- Buscamos hacer una particion equilibrada de la lista en dos partes A y B
- En A habra *n/k* elementos, y en B el resto
- Ordenamos entonces A y B recursivamente
- Combinamos las listas ordenadas A y B usando un procedimiento llamado ***mezcla***, que combina las dos listas en una sola
- El problema queda resuelto
- Las diferentes posibilidades nos las va a dar el valor k que escojamos

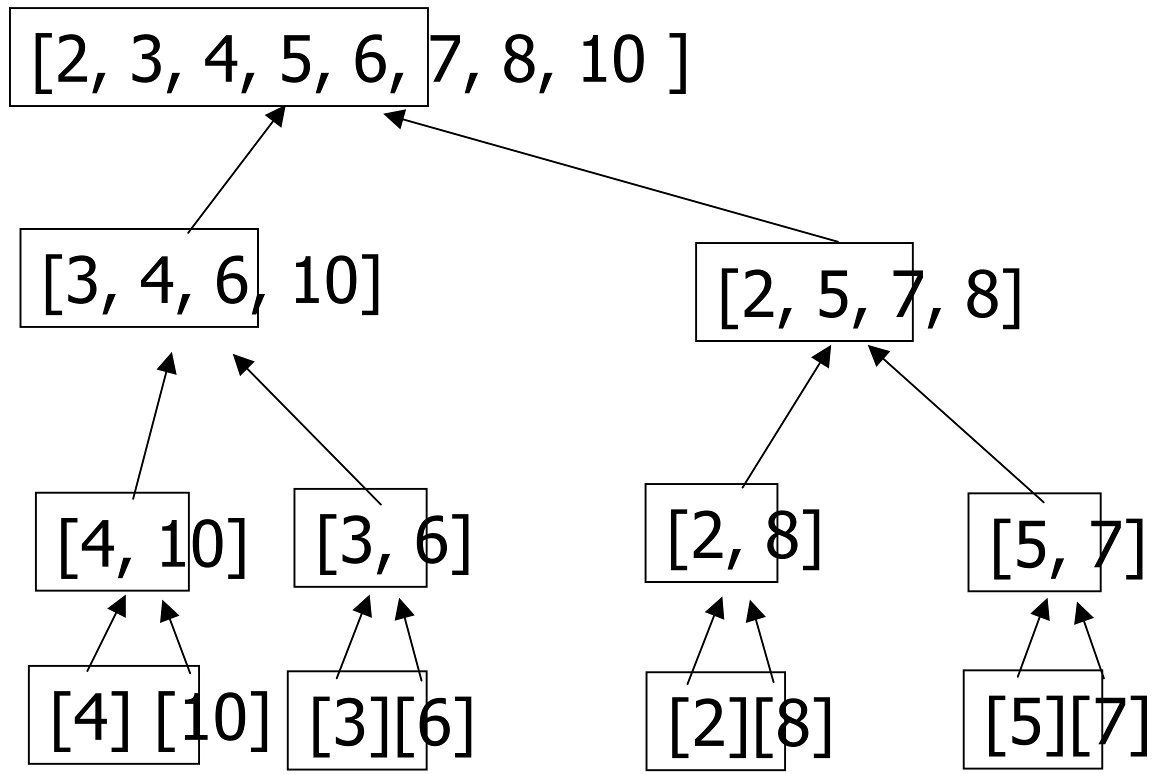
Ejemplo

- Sea k=2
- Partimos la lista en otras dos de tamaños n/2



Ejemplo

- La operación de mezcla para k=2 produciria



Codigo de ordenacion por mezcla

```
void mergeSort(Comparable []a, int left, int right)
{
    // sort a[left:right]
    if (left < right)
    {// at least two elements
        int mid = (left+right)/2;   //midpoint
        mergeSort(a, left, mid);
        mergeSort(a, mid + 1, right);
        merge(a, b, left, mid, right); // merge from a to b
        copy(b, a, left, right);  //copy result back to a
    }
}
```

Calculo de la eficiencia

- **Ecuacion recurrente:**
- Suponemos que n es potencia de 2

$$T(n) = \begin{cases} c_1 & \text{si } n=1 \\ 2T(n/2) + c_2 n & \text{si } n>1,\ n=2^k \end{cases}$$

- Podemos intentar la solucion por expansión

$T(n) = 2T(n/2) + c_2 n;$ $T(n/2) = 2T(n/4) + c_2 n/2$

$T(n) = 4T(n/4) + 2\ c_2 n;$ $T(n) = 8T(n/8) + 3\ c_2 n$

- En general,

$$T(n) = 2^i T(n/2^i) + i c_2 n$$

Solucion

- Tomando $n = 2^k$, la expansion termina cuando llegamos a $T(1)$ en el lado de la derecha, lo que ocurre cuando $i=k$

$$T(n) = 2^k T(1) + kc_2 n$$

- Como $2^k = n$, entonces $k=\log n$;
- Como ademas

$$T(1) = c_1$$

- Tenemos

$$T(n) = c_1 n + c_2 n\log n$$

- Por tanto el tiempo para el algoritmo de ordenacion por mezcla es $O(n\log n)$

Quick Sort

- Tambien se le conoce con el nombre de Algoritmo de Hoare
- Es el algoritmo (general) de ordenacion mas eficiente
- En sintesis
  - Ordena el array A eligiendo un valor clave v entre sus elementos, que actua como pivote
  - Organiza tres secciones: izquierda, pivote, derecha
  - todos los elementos en la izquierda son menores que el pivote, todos los elementos en la derecha son mayores o iguales que el pivote
  - ordena los elementos en la izquierda y en la derecha, sin requerir ninguna mezcla para combinarlos.
  - Lo ideal seria que el pivote se colocara en la mediana para que la parte izquierda y la derecha tuvieran el mismo tamaño

Pseudo Codigo para quicksort

Algoritmo QUICKSORT(S,T)

// Precondicion: Existe el conjunto S y es finito.

// Postcondicion: los elementos de S son de la misma naturaleza, estan dispuestos en una estructura lineal y son ordenables en una estructura T.

Begin Algoritmo

IF TAMAÑO(S) $\leq$ q (umbral) THEN INSERCION(S,T)

ELSE

Elegir cualquier elemento p del array como pivote

Partir S en (S1,S2,S3) de modo que

1. $\forall$ x$\in$S1, y$\in$S2, z$\in$S3 se verifique x<p<z and y=p
2. TAMAÑO(S1) < TAMAÑO(S) y TAMAÑO(S3) < TAMAÑO(S)

QUICKSORT(S1,T1) // ordena recursivamente particion izquierda

QUICKSORT(S3,T3) // ordena recursivamente particion derecha

Combinacion: T = T1 || S2 || T3 //S2 es el elemento intermedio entre cada mitad ordenada

End Algoritmo

La eleccion del pivote

- Cada uno podemos diseñar hoy mismo nuestro propio algoritmo Quicksort (otra cosa es que funcione mejor que los que ya hay...): **La eleccion condiciona el tiempo de ejecucion**
- El pivote puede ser cualquier elemento en el dominio, pero no necesariamente tiene que estar en S
  - Podria ser la media de los elementos seleccionados en S
  - Podria elegirse aleatoriamente, pero la funcion RAND() consume tiempo, que habria que añadirselo al tiempo total del algoritmo
- Pivotes usuales son la mediana de un minimo de tres elementos, o el elemento medio de S.

La eleccion del pivote

- El empleo de la mediana de tres elementos no tiene justificacion teorica.
- Si queremos usar el concepto de mediana, deberiamos escoger como pivote la mediana del array porque lo divide en dos sub-arrays de igual tamaño
  - mediana = (n/2)º mayor elemento
  - elegir tres elementos al azar y escoger su mediana; esto suele reducir el tiempo de ejecucion aproximadamente en un 5%
- Escoger como pivote el elemento en la posicion central del array, dividiendo este en dos mitades

Algoritmo Quicksort (Hoare)

Procedimiento quicksort (T[i..j])

{ordena un array T[i..j] en orden creciente}

Si j-i es pequeño Entonces Insercion (T[i..j])

Caso contrario pivot (T[i..j], l)

{tras el pivoteo, $i \leq k < l \Rightarrow T[k] \leq T[l]$ y, $l < k \leq j \Rightarrow T[k] > T[l]$}

quicksort (T[i..l-1])

quicksort (T[l+1..j])

- No es dificil diseñar un algoritmo en tiempo lineal para el pivoteo. Sin embargo, es crucial en la practica que la constante oculta sea pequeña.
- La mejor forma de pivotear es escoger como pivote, entre los dos primeros elementos del array, el mayor de ellos

Pivoteo lineal

- Sea p = T[i] el pivote.
- Una buena forma de pivotear consiste en explorar el array T[i..j] solo una vez, pero comenzando desde ambos extremos.
- Los punteros k y l se inicializan en i y j+1 respectivamente.
- El puntero k se incrementa entonces hasta que T[k] > p, y el puntero l se disminuye hasta que T[l] $\leq$ p. Ahora T[k] y T[l] estan intercambiados. Este proceso continua mientras que k < l.
- Finalmente, T[i] y T[l] se intercambian para poner el pivote en su posicion correcta.

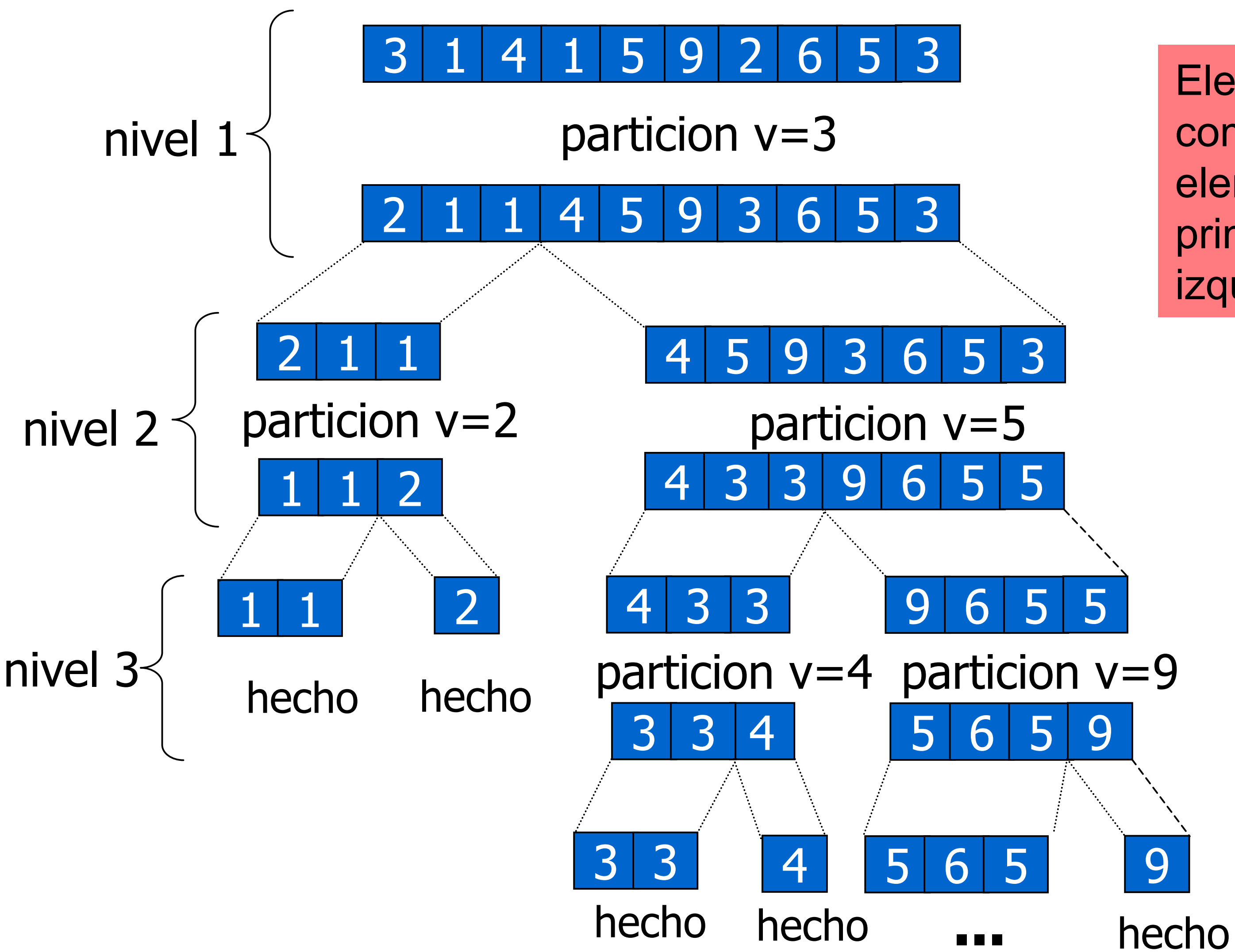
Algoritmo de pivoteo

Procedimiento pivot (T[i..j])

{permuta los elementos en el array T[i..j] de tal forma que al final $i \leq l \leq j$, los elementos de T[i..l-1] no son mayores que p, T[l] = p, y los elementos de T[l+1..j] son mayores que p, donde p es el valor inicial de T[i]}

```
p = T[i]
k = i; l = j+1;
repetir k = k+1 hasta T[k] > p o k ≥ j
repetir  l = l -1  hasta T[l] ≤ p
Mientras k < l hacer
    intercambiar T[k] y T[l]
    repetir k = k+1 hasta T[k] > p
    repetir  l = l -1  hasta T[l] ≤ p
intercambiar T[i] y T[l]
```

Ejemplo



Elegimos el pivote como el mayor elemento de los dos primeros por la izquierda

Eficiencia de quicksort

- Si admitimos que
  - El procedimiento de pivoteo es lineal,
  - Quicksort lo llamamos para T[1..n], y
  - Elegimos como peor caso que el pivote es el primer elemento del array,
- Entonces el tiempo del anterior algoritmo es

$$T(n) = T(1) + T(n-1) + an$$

- Que evidentemente proporciona un tiempo cuadratico

Analisis de Quicksort

- Recordemos que el algoritmo de ordenacion por Insercion hacia aproximadamente1/2 n^2 - 1/n comparaciones, es decir es O(n^2) en el peor caso.
- En el peor caso quicksort es tan malo como el peor caso del metodo de insercion (y tambien de seleccion).
- Es que el numero de intercambios que hace quicksort es unas 3 veces el numero de intercambios que hace el de insercion.
- Sin embargo, en la practica quicksort es el mejor algoritmo de ordencion que se conoce...
- ¿Que pasara con el tiempo del caso promedio?

Analisis del caso promedio

- Suponemos que la lista esta dada en orden aleatorio
- Suponemos que todos los posibles ordenes del array son igualmente probables
- El pivote puede ser cualquier elemento
- Puede demostrarse que en el caso promedio quicksort tiene un tiempo $T(n) = 2n \ln n + O(n)$, que se debe al numero de comparaciones que hace en promedio en una lista de n elementos
- En definitiva, quicksort, tiene un tiempo promedio O(n log n)