



1INF27

Algoritmia y Estructura de Datos

2024

Profesores:

Cueva, R. | Allasi, D. | Roncal, A. | Huamán, F.

0581

0582

0583

0584

Capítulo 1

DIVIDE Y VENCERÁS



DIVIDE Y VENCERAS

- Estrategia introducida por los romanos en el siglo IV antes de Cristo.
 - Divide et impera
- Como estrategia algorítmica es una de las más exitosas.
 - Dividir un problema en instancias más pequeñas
 - Resolver cada instancia de manera independiente
 - Combinar las soluciones para solucionar problema original.



DIVIDE Y VENCERAS

- Problemas clásicos como ordenación: soluciones más eficientes son del tipo divide y vencerás.
- Es más fácil aprender esta estrategia con ejemplos!



Búsqueda Binaria

BÚSQUEDA BINARIA

- Buscar una clave K en un arreglo ordenado A

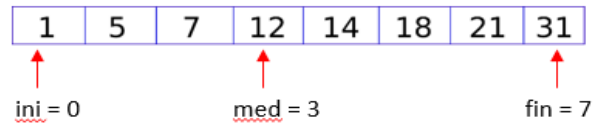
1	5	7	12	14	18	21	31	$K = 7$
---	---	---	----	----	----	----	----	---------

- Proceso
 - Como arreglo está ordenado, si verificamos a la mitad del arreglo, podemos descartar la mitad en donde sabemos que no encontraremos el elemento

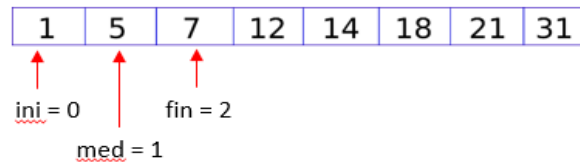
1	5	7	12	14	18	21	31
\uparrow <u>ini</u> = 0			\uparrow <u>med</u> = 3				\uparrow <u>fin</u> = 7

 $A[\underline{\text{med}}] > K ?$  K se encuentra en la primera mitad $A[\underline{\text{med}}] \leq K ?$  K se encuentra en la segunda mitad

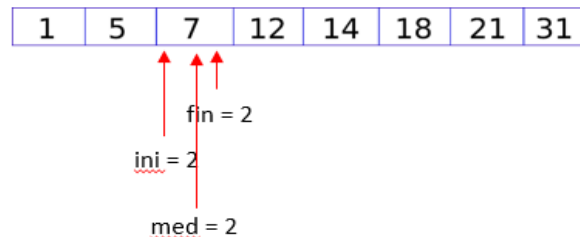
BÚSQUEDA BINARIA

 $K = 7$  $A[\text{med}] > K ?$

K se encuentra en la primera mitad

 $\text{fin} = \text{med} - 1$  $A[\text{med}] \leq K ?$

K se encuentra en la segunda mitad

 $\text{ini} = \text{med} + 1$  $A[\text{med}] == K ?$ K se encuentra en la posición med

BÚSQUEDA BINARIA

- Si el elemento se encuentra, las dos posiciones extremos siempre convergerán al elemento buscado.
- Si el elemento no se encuentra, la posición inicial sobrepasará a la posición final cerca del elemento buscado.
- NOTAS
 - El problema es recursivo
 - Usamos las observaciones anteriores para definir nuestro caso base.



BÚSQUEDA BINARIA - Algoritmo

- Pre-condición: Arreglo A [0..N-1] ordenado, clave K, posición inicial, posición final
- Post-condición: V o F
 - Si $ini > fin$
 - Retornar F
 - $med = (fin + ini) / 2$
 - Si $A[med] = K$
 - Retornar V
 - Caso contrario Si $A[med] < K$
 - Retornar $BusquedaBinaria(A, K, med+1, fin)$
 - Caso contrario
 - Retornar $BusquedaBinaria(A, K, ini, med - 1)$



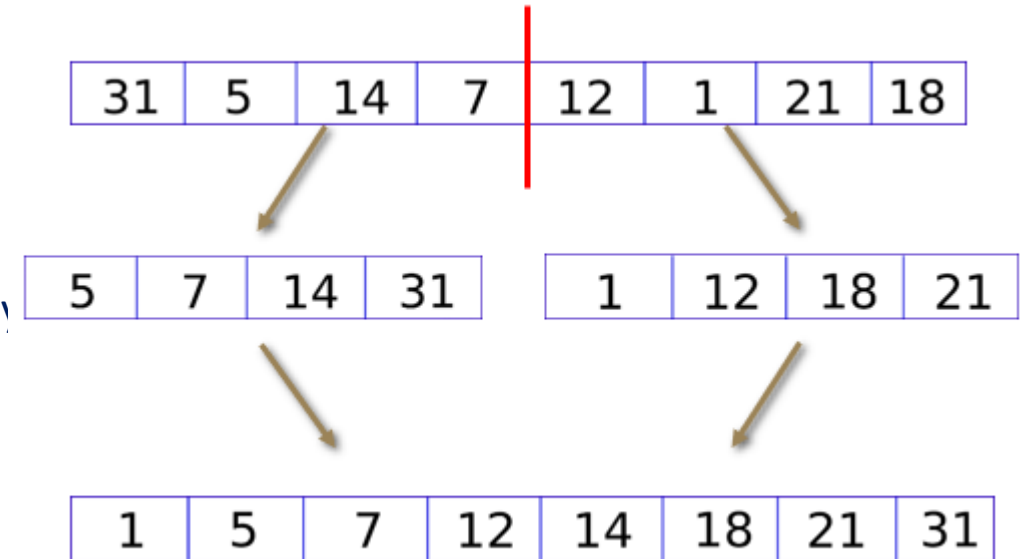
Aplicación de la Estrategia

Ordenación por Mezcla



ORDENACIÓN POR MEZCLA

- Algoritmo eficiente para ordenar una secuencia de números
- Idea
 - Dividir el arreglo en 2
 - Ordenar recursivamente las dos mitades
 - Mezclar las dos listas ordenadas en una sola lista ordenada
- El caso base de la recursión es cuando hay un solo elemento, en cuyo caso ya está ordenado.



MERGE SORT - Pseudocódigo

- Pre-condición: Arreglo $A[0..N-1]$, posición inicial, posición final
- Post-condición: Arreglo $A[0..N-1]$ ordenado
- Si $ini = fin$
 - Retornar
- $med = (fin + ini)/2$
- $Mergesort(A, ini, med)$
- $Mergesort(A, med+1, fin)$
- $Merge(A, ini, med, fin)$

Merge Sort Algorithm

Data Structure



MERGE SORT - Pseudocódigo

- El procedimiento que hace todo el trabajo es el algoritmo de mezcla
- Si se implementa eficientemente, toda la ordenación será eficiente
- Idea
 - Crear copias de ambos subarreglos e ir comparando los elementos, insertándolos en el orden correcto



MERGE SORT - Ejecución

A =

5	7	14	31
---	---	----	----

p = 0

B =


1	12	18	21
---	----	----	----

q = 0

C =

1							
---	--	--	--	--	--	--	--

m = 0

$B[q] < A[p]$  $C[m] = B[q]$ $q++$ $m++$



MERGE SORT - Ejecución

A =

5	7	14	31
---	---	----	----

p = 0

B =


1	12	18	21
---	----	----	----

q = 1

C =

1	5						
---	---	--	--	--	--	--	--

m = 1

$B[q] < A[p]$  $C[m] = B[q]$ $q++$ $m++$

$B[q] > A[p]$  $C[m] = A[p]$ $p++$ $m++$



MERGE SORT - Ejecución

A =

5	7	14	31
---	---	----	----

↑
 $p = 1$

B =


1	12	18	21
---	----	----	----


↑
 $q = 1$


C =

1	5	7					
---	---	---	--	--	--	--	--

↑
 $m = 2$

$B[q] < A[p]$  $C[m] = B[q]$ $q++$ $m++$

$B[q] > A[p]$  $C[m] = A[p]$ $p++$ $m++$

$B[q] > A[p]$  $C[m] = A[p]$ $p++$ $m++$



MERGE SORT - Ejecución

A =

5	7	14	31
---	---	----	----

$p = 2$

B =


1	12	18	21
---	----	----	----


$q = 1$


C =


1	5	7	12				
---	---	---	----	--	--	--	--

$m = 3$

$B[q] < A[p]$  $C[m] = B[q]$ $q++$ $m++$

$B[q] > A[p]$  $C[m] = A[p]$ $p++$ $m++$

$B[q] > A[p]$  $C[m] = A[p]$ $p++$ $m++$

$B[q] < A[p]$  $C[m] = B[q]$ $q++$ $m++$



MERGE SORT - Ejecución

A =

5	7	14	31
---	---	----	----

↑
 $p = 2$

B =


1	12	18	21
---	----	----	----


↑
 $q = 2$


C =


1	5	7	12				
---	---	---	----	--	--	--	--

↑
 $m = 4$

$B[q] < A[p]$  $C[m] = B[q]$ $q++$ $m++$

$B[q] > A[p]$  $C[m] = A[p]$ $p++$ $m++$

$B[q] > A[p]$  $C[m] = A[p]$ $p++$ $m++$

$B[q] < A[p]$  $C[m] = B[q]$ $q++$ $m++$



MERGE SORT

A =

5	7	14	31
---	---	----	----

↑
p = 5

B =

1	12	18	21
---	----	----	----

↑
q = 5

C =

1	5	7	12	14	18	21	31
---	---	---	----	----	----	----	----

↑
m = 9

- Proceso termina cuando C contiene todos los elementos mezclados
- Problema: tener cuidado cuando se termina de analizar un arreglo y aún quedan elementos en el otro arreglo
 - Con quién comparamos?
- Truco: aumentar ambos sub-arreglos con números muy grandes y ejecutar proceso hasta que C contenga todos los elementos.

A =

5	7	14	31	∞
---	---	----	----	----------

B =

1	12	18	21	∞
---	----	----	----	----------



DIVIDE Y VENCERAS - Resumen

- Estrategia muy útil para resolver eficientemente problemas
 - Idea: resolver problemas independientemente (clave para reducir complejidad) y luego fusionar soluciones
- Generalmente la solución se puede expresar recursivamente
 - No necesariamente la implementación tiene que ser recursiva
 - Se puede implementar iterativamente, aunque eso suponga algún esfuerzo extra del programador.

