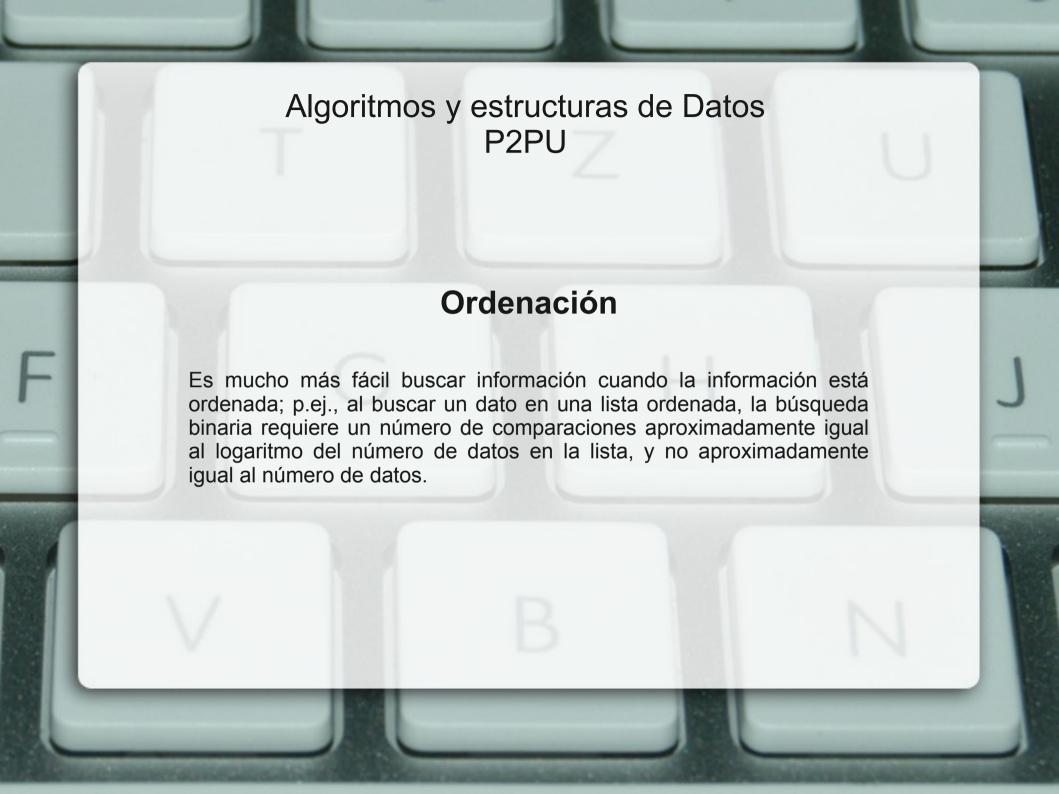
Algoritmos y estructuras de Datos P2PU Algoritmos de ordenación Semana 2 Dictado por Marco González Núñez 07 de Febrero de 2011



El problema

Entrada: Una secuencia de números a1,a2,a3,...,an, usualmente es un

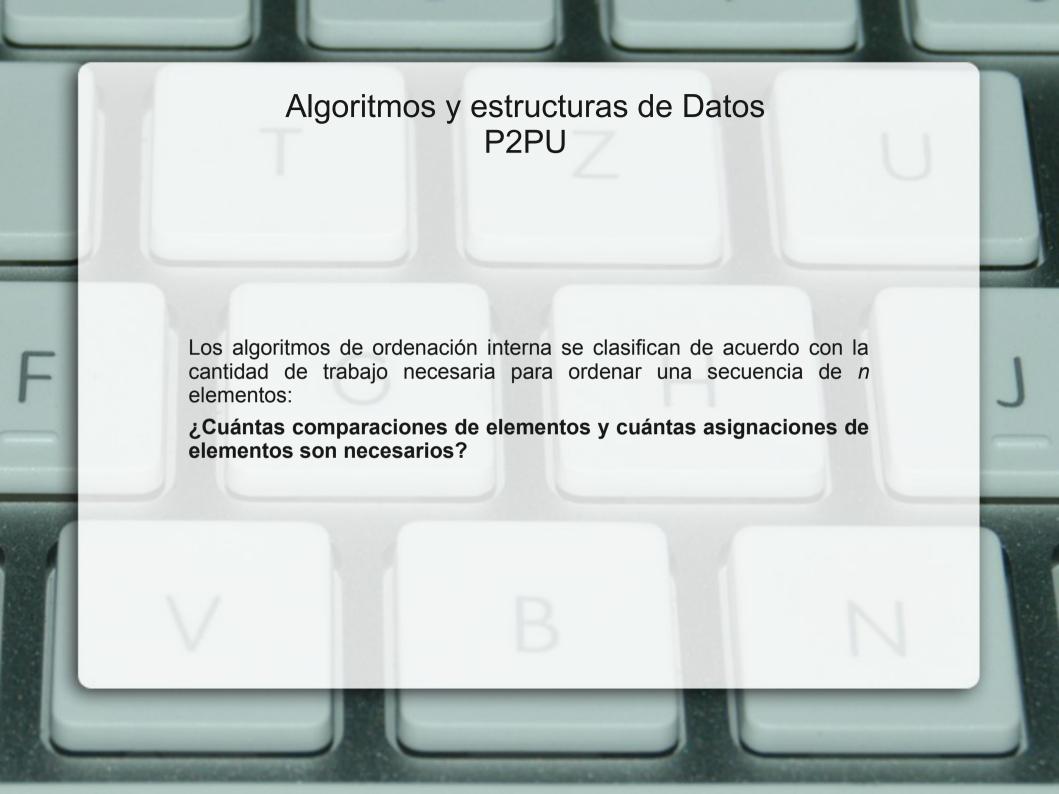
arreglo de tamaño n.

Salida: Una permutación del arreglo tal que a1'<a2'<a3'<...<an'.

Existen dos categorias disjuntas de algoritmos de ordenación:

Ordenación interna: La cantidad de registros es pequeña y el proceso puede llevarse a cabo en memoria.

Ordenación externa: Hay demasiados registros para usar ordenación interna; deben almacenarse en disco.



Algoritmos de ordenación simple

Sea *n* el número de elementos en el arreglo *A*:

Insert-Sort(A, n):

Para cada valor de j, inserta A[j] en el puesto que le corresponde en la secuencia ordenada $A[0] \dots A[j-1]$

for
$$(j = 1; n-1; j++)$$

 $k = A[j]$
 $i = j-1$
while $(i \ge 0 \&\& A[i] > k)$
 $A[i + 1] = A[i]$
 $i = i-1$

$$A[i+1]=k$$

Select-Sort(A, n):

{Para cada valor de j, selecciona el menor elemento de la secuencia no ordenada $A[j] \dots A[n-1]$ y lo intercambia con A[j]}

for
$$(j = 0; n-2; j++)$$

 $k = j$
for $(i = j+1; n-1; i++)$
if $(A[i] < A[k]) k = i$
intercamble $(A[j], A[k])$

Bubble-Sort(A, n):

{Para cada valor de i, compara con todo los elementos de A[j] ... A[n-1] si es menor, lo intercambia con A[i]}

for
$$(i = 0; i < n - 1; i++)$$

for $(j = i + 1; j < n; j++)$
if $(A[i] > A[j])$
intercamble $(A[i], A[j])$

Planteamiento y Resolución de Recurrencias

Si un algoritmo contiene una llamada recursiva, su tiempo de ejecución $\sigma(n)$ puede describirse mediante una recurrencia

una ecuación que describe una función recursivamente, es decir, en términos de su valor para argumentos más pequeños.

P.ej., el tiempo de ejecución en el peor caso del algoritmo de ordenación por mezcla puede describirse por la recurrencia

cuya solución es $\sigma(n) = \Theta(n \log n)$.

En la práctica, omitimos algunos detalles, p.ej.,

el $\sigma(n)$ de un algoritmo normalmente sólo está definido cuando n es un número entero,

normalmente $\sigma(n) = \Theta(1)$ — una constante — para valores pequeños de n.

Quick-Sort(A,n)

Descripción: Basado en el paradigma dividir-y-conquistar, estos son los tres pasos para ordenar un subarreglo $A[p] \dots A[r]$:

Dividir: El arreglo $A[p] \dots A[r]$ es particionado en dos subarre-glos no vacíos $A[p] \dots A[q]$ y $A[q+1] \dots A[r]$:

cada dato de A[p] ... A[q] es menor o igual que cada dato de A[q+1] ... A[r];

q se calcula como parte de este proceso de partición.

Conquistar. Ordenamos los subarreglos $A[p] \dots A[q]$ y $A[q+1] \dots A[r]$ mediante sendas llamadas recursivas a Quicksort.

Combinar. Ya que los subarreglos son ordenados in situ, no es necesario hacer ningún trabajo extra para combinarlos; todo el arreglo $A[p] \dots A[r]$ está ahora ordenado.

```
Quicksort (A, p, r):

if (p < r)

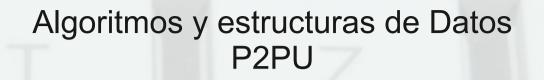
q = Partition(A, p, r)

Quicksort(A, p, q)

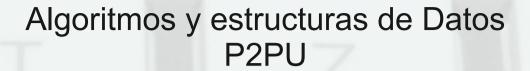
Quicksort(A, q+1, r)
```

```
Partition (A, p, r):
Quicksort (A, p, r):

if (p < r)
q = \text{Partition}(A, p, r)
Quicksort (A, p, q)
Quicksort (A, p, q)
Quicksort (A, q+1, r) x = A[p]; i = p-1; j = r+1
while (1)
\text{do } j = j-1 \text{ while } (A[j] > x)
\text{do } i = i+1 \text{ while } (A[i] < x)
if (i < j)
\text{intercamble } (A[i], A[j])
else return j
```



Tarea: Investigue acerca de cuales son los mejores y peores casos para los algoritmos de ordenación mencionados anteriormente (InsertSort, SelectSort, BubbleSort y QuickSort) y escriba un código (en su lenguaje preferido) de un algoritmo adaptativo donde esten mezclados los algoritmos de ordenación mencionados. El código debe tener como entrada una matriz A y el número de elementos n, y debe estimar que algoritmo de ordenación ocupar según su peor y mejor caso.



La tarea debe ser enviada a mi correo electrónico (markogonzalez84@gmail.com), a más tardar a las 23:59 del día Viernes 11 de Febrero de 2011, hora de Chile, en formato pdf.

Debe tener una portada con el nombre y nick de cada uno(P2PU), letra Arial 16, para títulos y 12 para texto, interlineado 1.5 y justificado.

Tarea opcional:

Investigue acerca del algoritmo de ordenación de la verrsión aleatorizada de QuickSort y explique porque es más rapido que Quicksort normal.

Prguntas para debatir en los foros:

- 1.- ¿Cuál es el mejor algoritmo de ordenación?
- 2.- ¿Dónde podemos aplicar (caso real) estos algoritmos?

Frase de la semana: "Si no puedes con ellos... DIVIDELOS!!"