Algoritmos de ordenamiento (2)

mat-151

Algoritmos de tipo *divide-and-conquer*

- Quicksort
- Mergesort

Quicksort

- Es probablemente el algoritmo de ordenamiento más utilizado.
- Diseñado por C.A.R. Hoare en 1960.
- Su complejidad varia dependiendo de la entrada, desde *nlog(n)* (mejor caso) a *cuadrática* (peor caso).
- Método del tipo *divide-and-conquer*.

Quicksort

- ***Idea***

  1. Dividir el conjunto de datos (arreglo, lista) en dos partes utilizando un ***pivote*** *a.*

  2. Ordenar cada una de las partes de forma independiente.

- La particularidad de este método es tratar el problema de ¿cómo partir los datos? o ¿qué elemento debe ser tomado como pivote?

Quicksort - dividir

- Para eso las tres condiciones siguientes deben cumplirse:
    - El elemento $a[i]$ está en su lugar final en el arreglo para alguna i .
    - Ninguno de los elementos en $a[1],\ldots,a[i-1]$ es mayor que $a[i]$.
    - Ninguno de los elementos en $a[i+1],\ldots,a[r]$ es menor que $a[i]$.

Quicksort - vencer

- Ordenar ambos sub-arreglos $a[1 \ldots i-1]$ y $a[i+1 \ldots r]$ con llamadas recursivas a quicksort.

Quicksort - combinar

- Dado que los sub-arreglos están arreglados *en-lugar* (in-place), no hay que trabajar para combinarlos.
- El arreglo entero $a[1 \ldots r]$ se encuentra ordenado.

Quicksort

- Si el conjunto de datos tiene uno o menos elementos, no entra al ciclo.
- En caso contrario, el conjunto de elementos se procesa con la función ***partition*** que pone ***a[i]*** en posición para alguna ***i*** entre ***l*** y ***r*** inclusive.
- los demás elementos son re-arreglados por las funciones recursivas.

```
void quickSort( Item a[], int l, int r )
{
    if ( r <= l ) return;
    int i = partition( a, l, r );
    quicksort(a, l, i-1);
    quicksort(a, i+1, r);
}
```

Quicksort

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Partición de arreglos con Quicksort

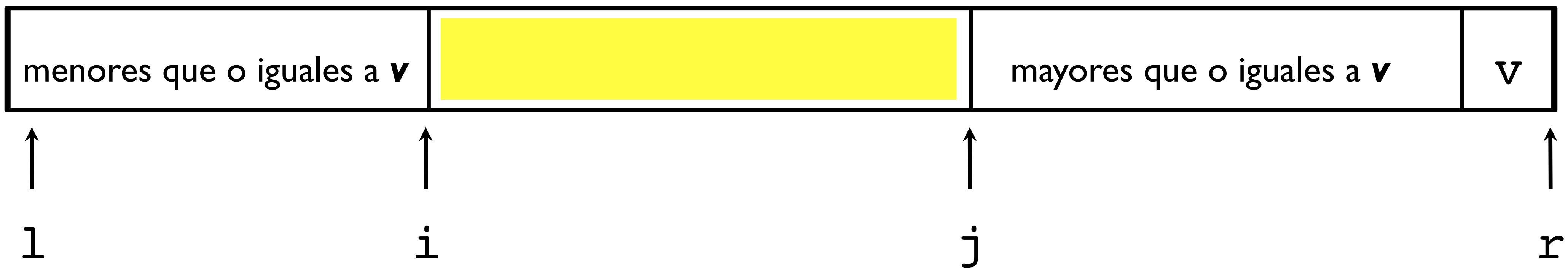
1) Se designa arbitrariamente ***a[r]*** como ***elemento de partición***.

2) Se **recorre** a partir del lado izquierdo hasta encontrar un elemento mayor al *elemento de partición.*

3) Se **recorre** a partir del lado derecho hasta encontrar un elemento menor al *elemento de partición*.

4) Se **intercambian** los elementos que detuvieron los recorridos.

Quicksort



v. valor del *elemento de partición.*

l. inicio del arreglo de elementos a ordenar.

i. apuntador izquierdo.

j. apuntador derecho.

r. fin del arreglo de elementos a ordenar.

Ejemplo - partición

2 8 7 1 3 5 6 4

Ejemplo - partición

2 8 7 1 3 5 6 4

2	8	7	1	3	5	6	4
2	8						
				3	5	6	
2	3			8	5	6	
		7					
			1				
		1	7				
			4				7
2	3	1	4	8	5	6	7

Partición de arreglos con Quicksort

```
void exch( Item &A, Item &B )
{
    Item t = A;
    A=B;
    B=t;
}
```

```
int partition( Item a[], int l, int r )
{
    int i=l-1;
    j = r;
    Item v = a[r];
    for(;;)
        {
            while ( a[++i] < v );
            while (v < a[--j] )
                if( j== i)
                    break;
            if ( i>=j )
                break;
            exch(a[i], a[j]);
        }
    exch( a[i], a[r] );
    return i;
}
```

Partición de arreglos con Quicksort

Partición de arreglos con Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	S													
										A	M	P	L	
A	A									S	M	P	L	E
		O												
								E	X					
A	A	E						O	X	S	M	P	L	E
			R											
			E	T	I	N	G							
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R

Partición de arreglos con Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	S													
										A	M	P	L	
A	A									S	M	P	L	E
		O												
								E	X					
A	A	E						O	X	S	M	P	L	E
			R											
			E	T	I	N	G							
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R

Partición de arreglos con Quicksort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	S													
										A	M	P	L	
A	A									S	M	P	L	E
		O												
								E	X					
A	A	E						O	X	S	M	P	L	E
			R											
			E	T	I	N	G							
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R

Friday, April 7, 17

Partición de arreglos con Quicksort

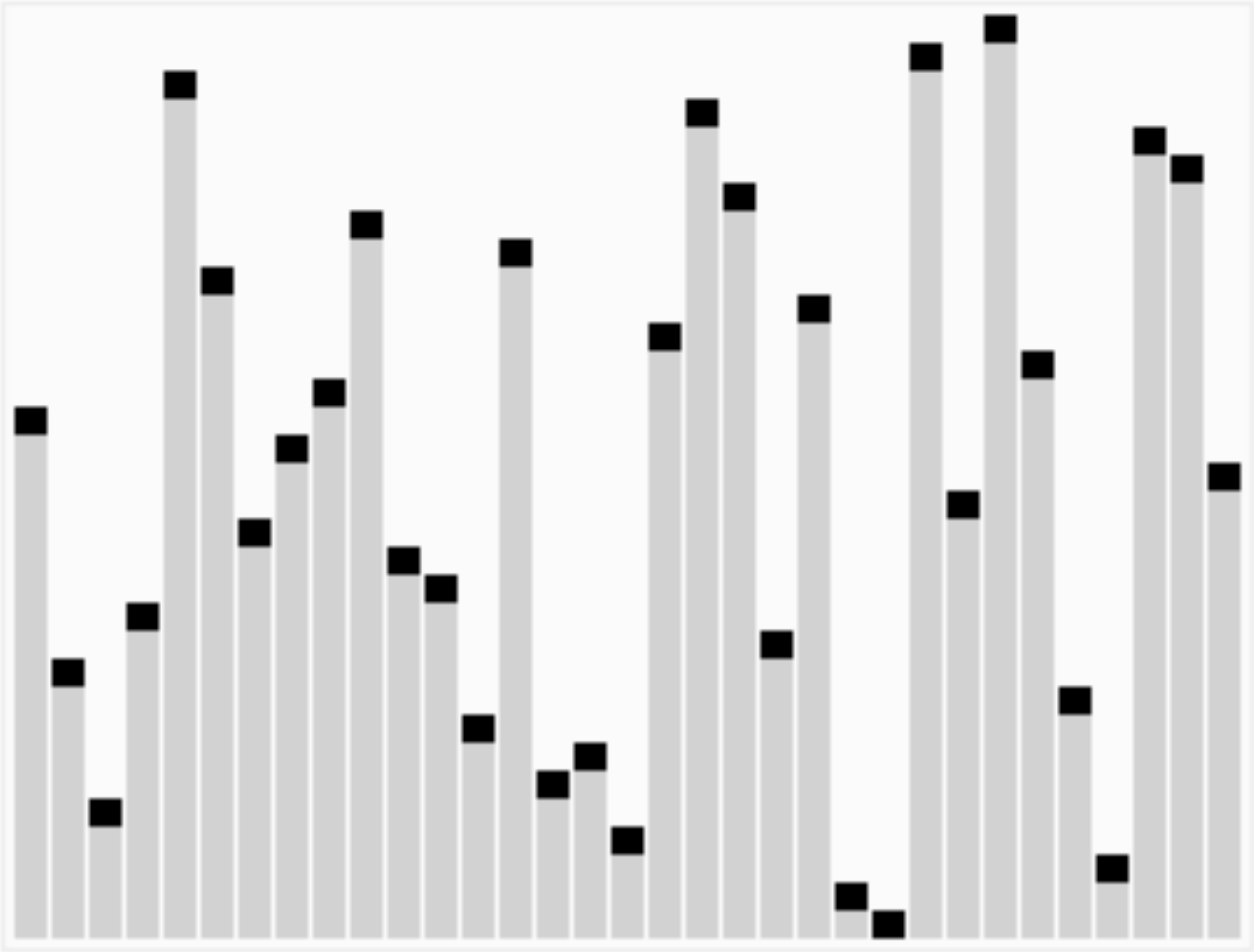
- El proceso de partición no es estable
    - cualquier llave puede moverse antes o después de un número de llaves iguales (que han podido o no ser examinadas).
- Condiciones para la terminación del proceso recursivo:
    - no llamarse a si mismo para arreglos de tamaño 1 o menores.
    - Optimalidad en partición, llamarse solamente si los arreglos son significativamente menores a la entrada.

Quicksort

- ***Ventajas:***
  - In-place (no necesita memoria adicional)
  - Complejidad *O(nlogn)* en promedio
  - Loop interno muy simple
- ***Desventajas:***
  - No es estable en el caso general
  - Complejidad $O(n^2)$ en el peor caso
  - Frágil (si hay errores en implementación pueden pasar desapercibidos pero dañar el desempeño del algoritmo)

Quicksort

Quicksort



Características de desempeño de Quicksort

- Para una entrada de tamaño N con sus elementos previamente ordenados, elimina solo un elemento en cada llamada: una lista con *n-1* elementos y otra con *0* elementos.

Propiedad 1

Quicksort usa cerca de $N^2/2$ comparaciones en el peor caso.

- El número de comparaciones utilizadas para una entrada en orden es entonces:

$$N + (N-1) + (N-2) + \ldots + 2 + 1 = (N+1)N/2.$$

- Lo mismo ocurre para entradas en orden reverso.

Quicksort en resumen (1)

- Algoritmo *divide-and-conquer*.
- Se basa en una operación de partición: elegir un elemento ***pivote***, mover los elementos más pequeños, ***adelante*** de éste y los elementos más grandes ***atrás*** de él.
- La operación de partición se puede hacer eficiente en tiempo lineal y en su lugar.
- Las listas de elementos mayores y menores se procesan recursivamente.
- Implementaciones de quicksort son típicamente inestables y algo complejas.
- Se encuentra entre los algoritmos de ordenamiento más rápidos en práctica.

Quicksort en resumen (2)

- El desempeño de quicksort depende en gran parte en la elección de un buen elemento de partición o ***pivote***.
- Una mala elección de pivote puede resultar en un desempeño mucho más lento ***$O(n^2)$***, pero si en cada paso elegimos la ***mediana*** como el pivote, entonces funciona en ***$O(n \log n)$***.

Mergesort

- Combinar **dos** conjuntos ordenados para hacer un conjunto ordenado más grande.
- Proceso complementario a ***quicksort*** (partir vs. combinar).
- Ejemplo clásico de un algoritmo de tipo ***divide-and-conquer***.

Mergesort - dividir

- Dividir la secuencia de *n*-elementos para ser ordenados en **dos** sub-secuencias de *n/2* elementos cada una.

Mergesort - vencer

- Ordenar las dos sub-secuencias recursivamente utilizando Mergesort hasta que las sub-listas sean de tamaño 1.

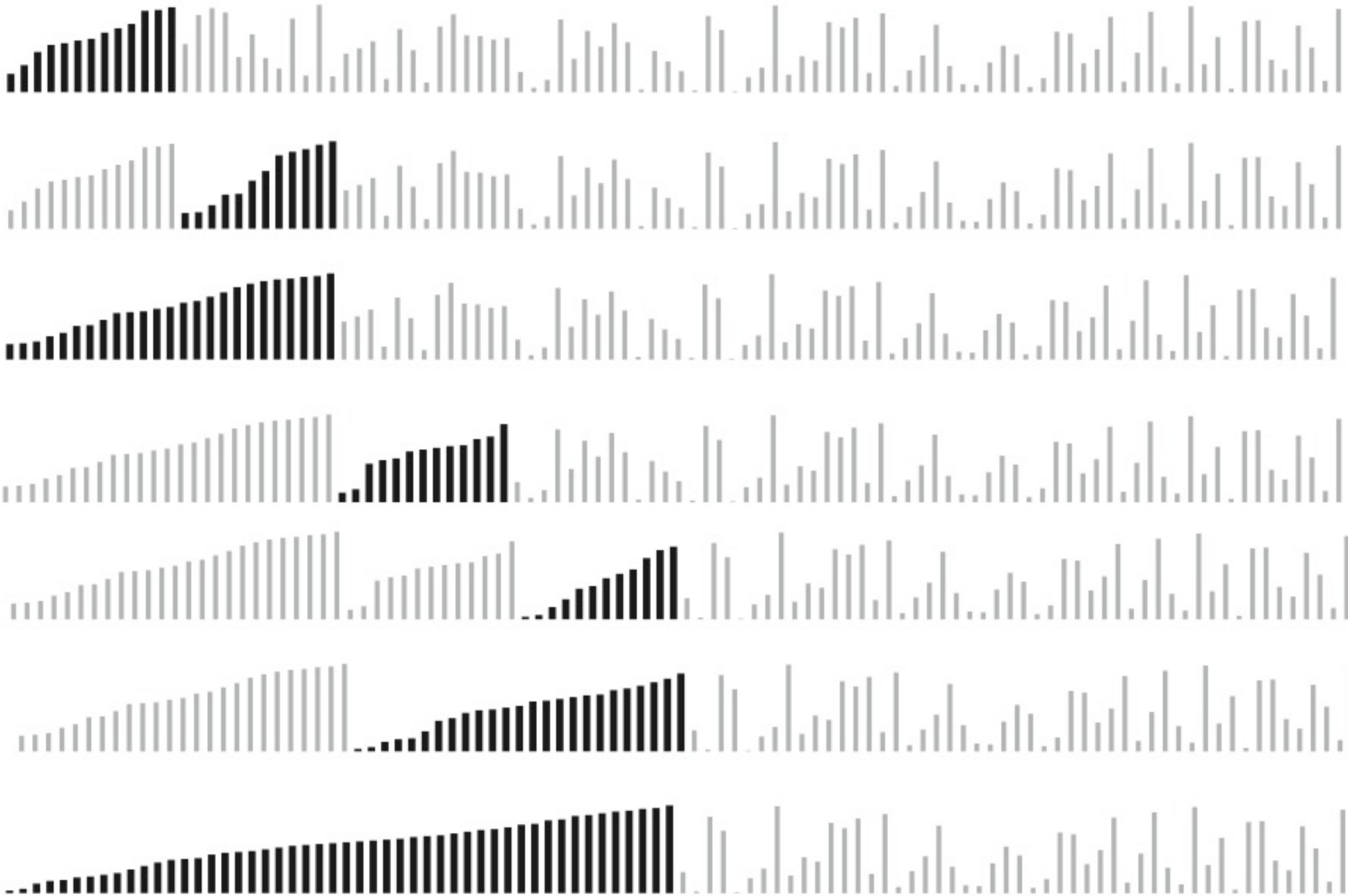
Mergesort - combinar

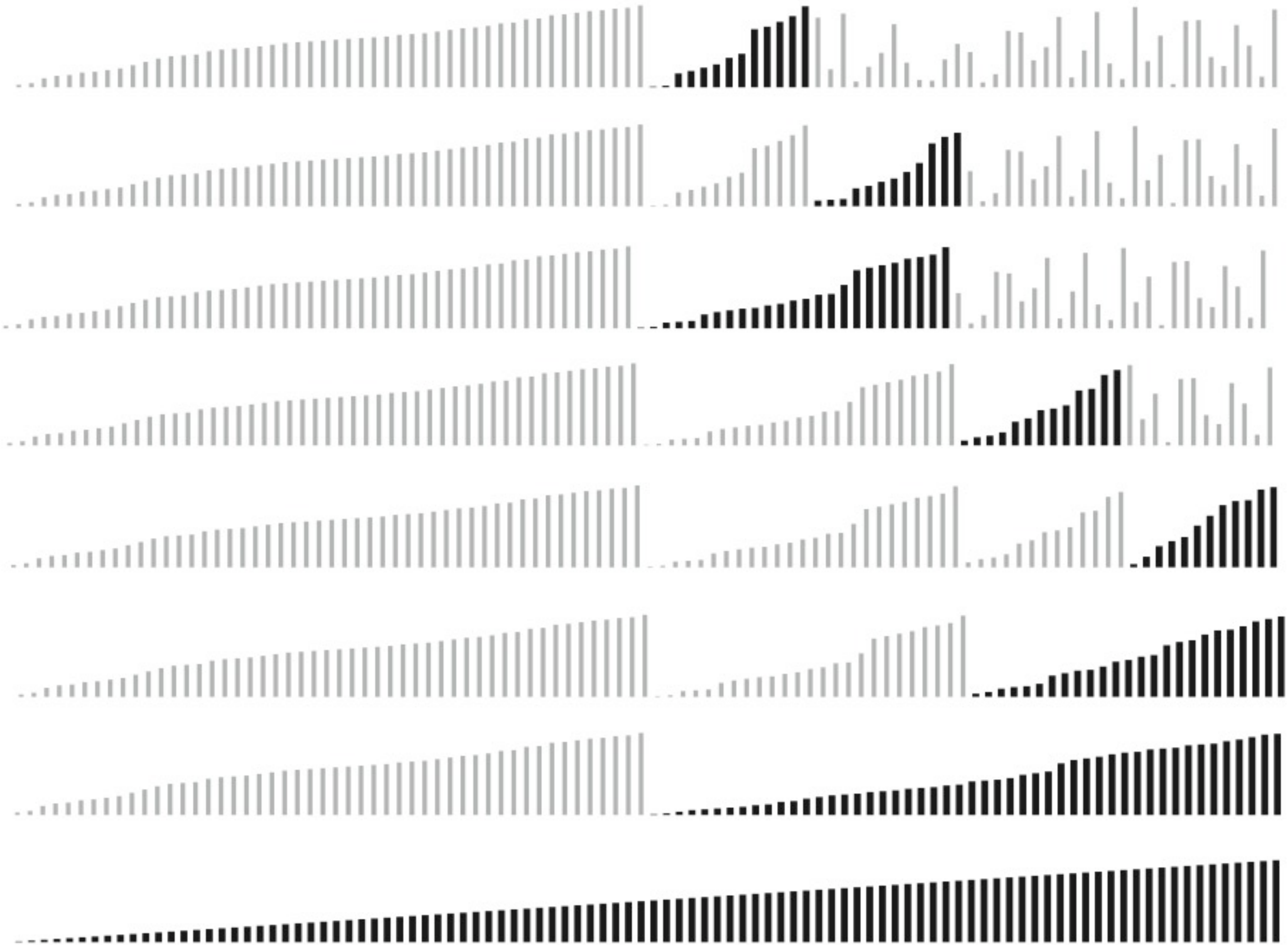
- Combinar las dos sub-secuencias ordenadas para producir una respuesta ordenada.
- Este paso es el más relevante del algoritmo.
- El procedimiento de MERGE tiene una complejidad $\Theta(n)$ donde n es el número de elementos a ser combinados.

Mergesort

entrada	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
ordenar mitad izquierda	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
ordenar mitad derecha	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
resultado de la combinación	**A**	**E**	**E**	**E**	**E**	**G**	**L**	**M**	**M**	**O**	**P**	**R**	**R**	**S**	**T**	**X**

Robert Sedgewick and Kevin Wayne - Princeton, 2008 - http://www.cs.princeton.edu/courses/archive/spring08/cos226/lectures.html





Friday, April 7, 17

Mergesort - combinar

- Supone que los sub-vectores están ordenadas.

```
void merge( Item c[], Item a[], int N, Item b[], int M )
{
    for ( int i=0, j=0, k=0; k < N+M; k++ ) {



    c[k] = ( a[i] < b[j] ) ? a[i++] : b[j++];
    }
}
```

- Lineal en el número de elementos en la salida: ***O(N+M)*** con encontrar al elemento mas pequeño en tiempo constante ***O(1).***

Mergesort - combinar

- Supone que los sub-vectores están ordenadas.

```
void merge( Item c[], Item a[], int N, Item b[], int M )
{
    for ( int i=0, j=0, k=0; k < N+M; k++ ) {
        if ( i==N ){
            c[k] = b[j++];
            continue;
        }
        if ( j==M ){
            c[k] = a[i++];
            continue;
        }

    c[k] = ( a[i] < b[j] ) ? a[i++] : b[j++];
    }
}
```

- Lineal en el número de elementos en la salida: ***O(N+M)*** con encontrar al elemento mas pequeño en tiempo constante ***O(1).***

Mergesort - No se puede combinar en su lugar (in-place)

- Pensemos ¿ Qué pasaría si usamos el programa anterior con la llamada:
    - merge(a, a, N/2, a+N/2, N-N/2)
    -

Mergesort - combinar en su lugar (in-place)

	k	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	i	j	aux[0]	aux[1]	aux[2]	aux[3]	aux[4]	aux[5]	aux[6]	aux[7]	aux[8]	aux[9]
entrada		E	E	G	M	R	A	C	E	R	T			-	-	-	-	-	-	-	-	-	-
copia izquierda		E	E	G	M	R	A	C	E	R	T			E	E	G	M	R	-	-	-	-	-
copia reversa derecha		E	E	G	M	R	A	C	E	R	T			E	E	G	M	R	T	R	E	C	A
												0	9										
	0	A										0	8	E	E	G	M	R	T	R	E	C	A
	1	A	C									0	7	E	E	G	M	R	T	R	E	C	
	2	A	C	E								1	7	E	E	G	M	R	T	R	E		
	3	A	C	E	E							2	7		E	G	M	R	T	R	E		
resultado	4	A	C	E	E	E						2	6			G	M	R	T	R	E		
	5	A	C	E	E	E	G					3	6			G	M	R	T	R			
	6	A	C	E	E	E	G	M				4	6				M	R	T	R			
	7	A	C	E	E	E	G	M	R			5	6					R	T	R			
	8	A	C	E	E	E	G	M	R	R		5	5						T	R			
	9	A	C	E	E	E	G	M	R	R	T	6	5						T				
resultado final		A	C	E	E	E	G	M	R	R	T												

Mergesort - combinar en su lugar (in-place)

marge(a, 0, 5, 11)

A E Q S U Y E I N O S T

```
void merge( Item a[], int l, int m, int r )
{
    int i,j;
    static Item aux[maxN];
    for ( i=m+1; i>l; i-- )
        aux[i-1] = a[i-1];
    for ( j=m; j<r; j++ )
        aux[r+m-j] = a[j+1];
    for ( int k=l; k<=r; k++ )
        if( aux[j] < aux[i] )
            a[k] = aux[j--];
        else
            a[k] = aux[i++];
}
```

Mergesort - combinar en su lugar (in-place)

- Esta última versión elimina la utilización de sentinelas (N y M).

- No se necesita copiar de ***c*** a ***a*** la salida.

Mergesort

- Merge Recursivo.

```
void mergeSort( Item a[], int l, int r)
{
    if ( r<=  l) return;
    int m = (r+1)/2;
    mergeSort(a,l,m);
    mergeSort(a, m+1, r);
    merge(a,l,m,r);
}
```

Mergesort

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 0, 0, 1)	**E**	**M**	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 2, 2, 3)	E	M	**G**	**R**	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 0, 1, 3)	**E**	**G**	**M**	**R**	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 4, 5)	E	G	M	R	**E**	**S**	O	R	T	E	X	A	M	P	L	E
merge(a, 6, 6, 7)	E	G	M	R	E	S	**O**	**R**	T	E	X	A	M	P	L	E
merge(a, 4, 5, 7)	E	G	M	R	**E**	**O**	**R**	**S**	T	E	X	A	M	P	L	E
merge(a, 0, 3, 7)	**E**	**E**	**G**	**M**	**O**	**R**	**R**	**S**	T	E	X	A	M	P	L	E
merge(a, 8, 10, 9)	E	E	G	M	O	R	R	S	**E**	**T**	X	A	M	P	L	E
merge(a, 10, 10, 11)	E	E	G	M	O	R	R	S	E	T	**A**	**X**	M	P	L	E
merge(a, 8, 9, 11)	E	E	G	M	O	R	R	S	**A**	**E**	**T**	**X**	M	P	L	E
merge(a, 12, 12, 13)	E	E	G	M	O	R	R	S	A	E	T	X	**M**	**P**	L	E
merge(a, 14, 14, 15)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	**E**	**L**
merge(a, 12, 13, 15)	E	E	G	M	O	R	R	S	A	E	T	X	**E**	**L**	**M**	**P**
merge(a, 8, 11, 15)	E	E	G	M	O	R	R	S	**A**	**E**	**E**	**L**	**M**	**P**	**T**	**X**
merge(a, 0, 7, 15)	**A**	**E**	**E**	**E**	**E**	**G**	**L**	**M**	**M**	**O**	**P**	**R**	**R**	**S**	**T**	**X**

Mergesort: análisis empirico

- Tiempo de ejecución estimado:
  - PC que ejecuta 10^8 comparaciones por segundo.
  - Supercomputadora que ejecuta 10^{12} comparaciones por segundo.

	insertion sort ***O(n²)***			merge sort ***O(n log n)***		
computadora	1000	millón	billón	1000	millón	billón
PC	instante	2.8 horas	317 años	instante	**1 segundo**	**18 minutos**
Super-computadora	instante	1 segundo	1 semana	instante	**instante**	**instante**

- mejores algoritmos son preferibles a mejores computadoras!

Mergesort

Propiedad 1

Mergesort necesita cerca de $N \log N$ comparaciones para ordenar cualquier secuencia de N elementos.

$T(n)$ = número de comparaciones en Mergesort para una entrada de tamaño N

$$\underbrace{T(N/2)}_{\text{mitad izquierda}} + \underbrace{T(N/2)}_{\text{mitad derecha}} + \underbrace{N}_{\text{combinación}}$$

Recurrencia:

$T(N) = 2T(N/2) + N$ para $N > 1$, con $T(1) = 0$

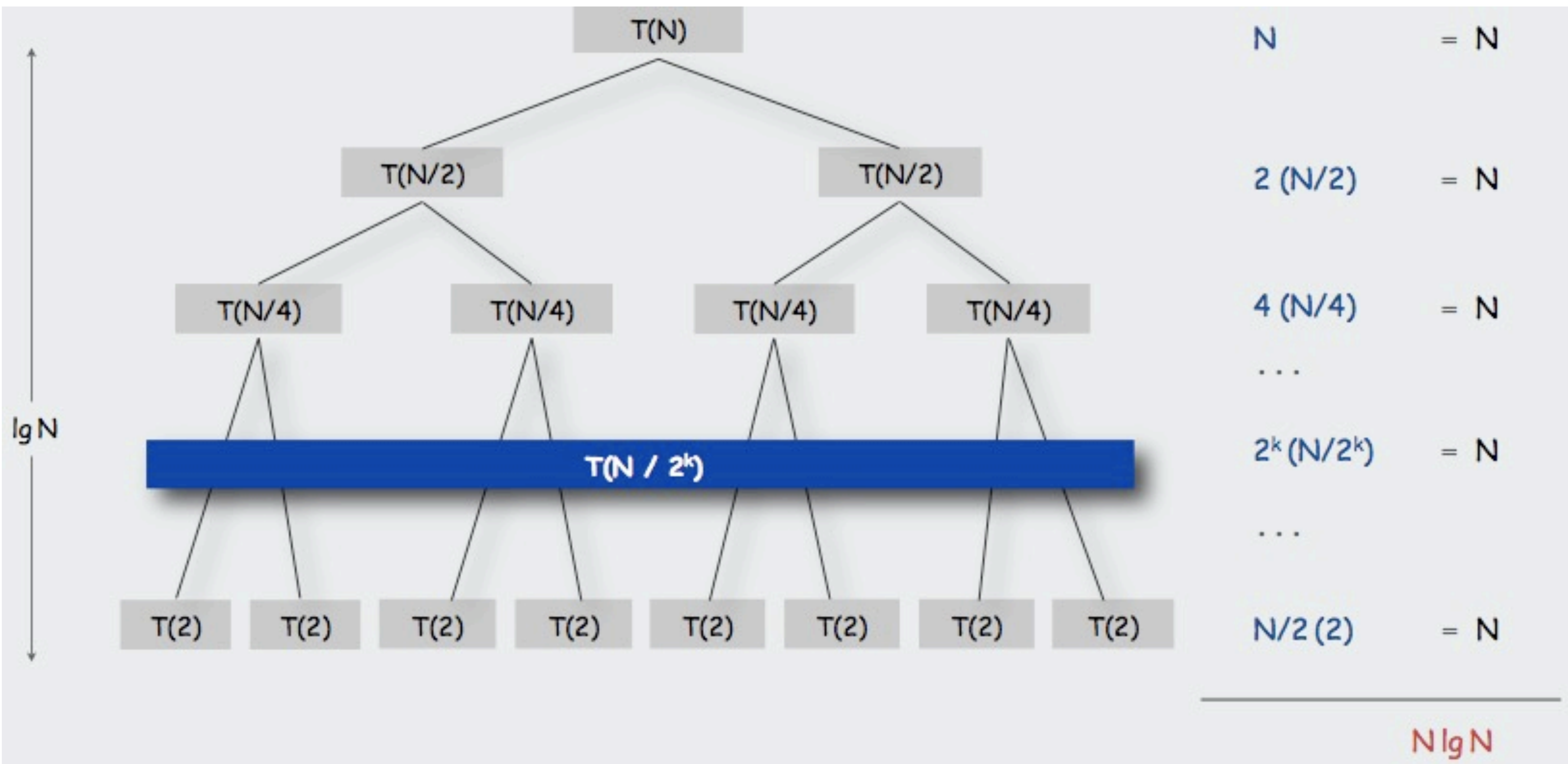
$$T(N) \sim N \log N$$

Comprobarlo de tarea

Mergesort

Recurrencia:

$T(N) = 2T(N/2) + N$ para $N > 1$, con $T(1) = 0$



Mergesort

Propiedad 2

Mergesort necesita memoria extra proporcional a N

- El arreglo **aux[ ]** tiene que ser del tamaño de N para el último *merge*.

E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Mergesort

Propiedad 3:

Mergesort es estable si las operaciones de combinación son estables

Propiedad 4:

Los requerimientos de recursos para mergesort son insensitivos al orden inicial de la entrada.

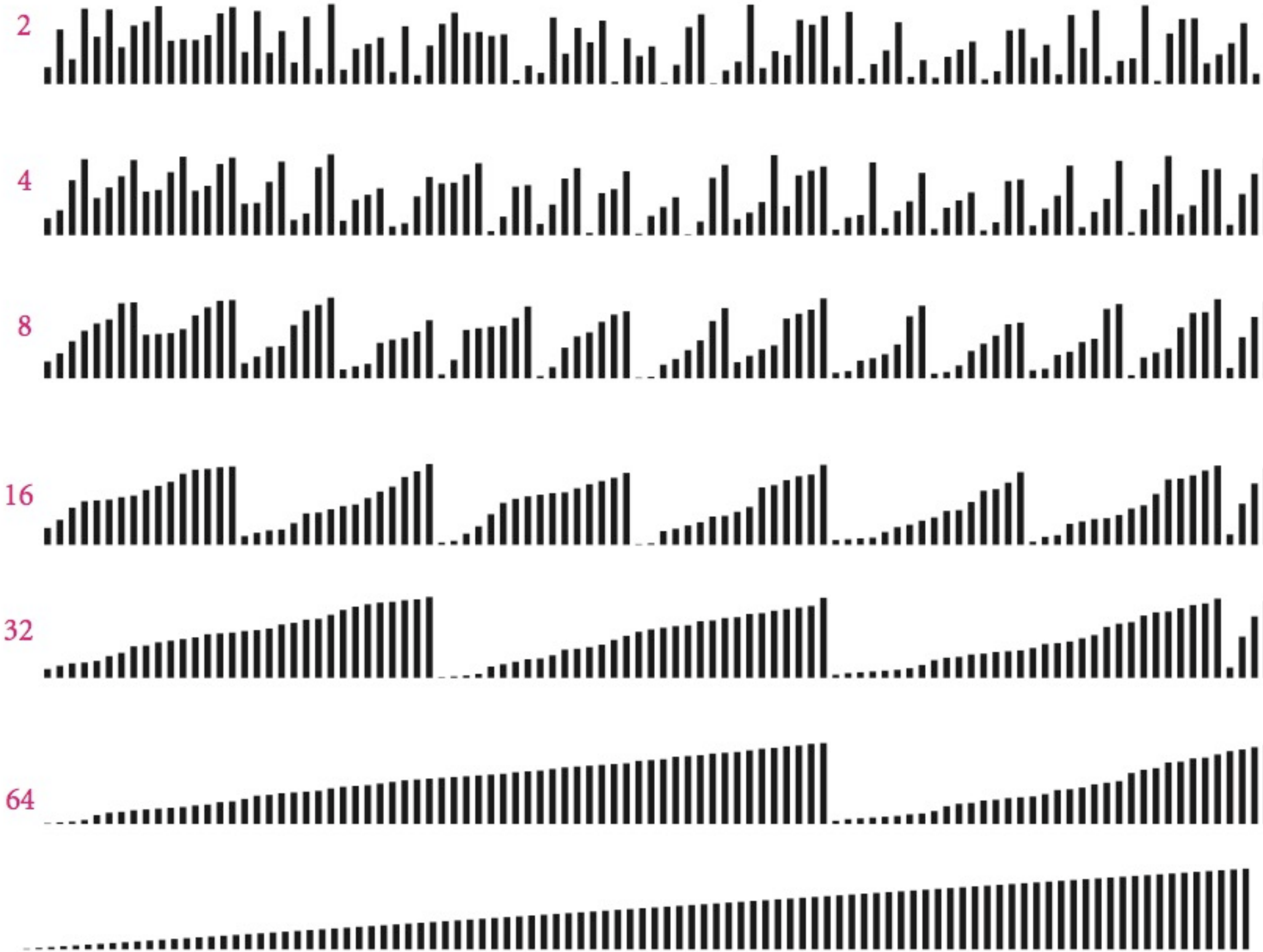
Mergesort - no recursivo

	lo	m	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	0,	0,	1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	2,	2,	3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	4,	4,	5)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	6,	6,	7)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	8,	8,	9)	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
merge(a,	10,	10,	11)	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
merge(a,	12,	12,	13)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a,	14,	14,	15)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
merge(a,	0,	1,	3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	4,	5,	7)	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
merge(a,	8,	9,	11)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a,	12,	13,	15)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a,	0,	3,	7)	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
merge(a,	8,	11,	15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge(a,	0,	7,	15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

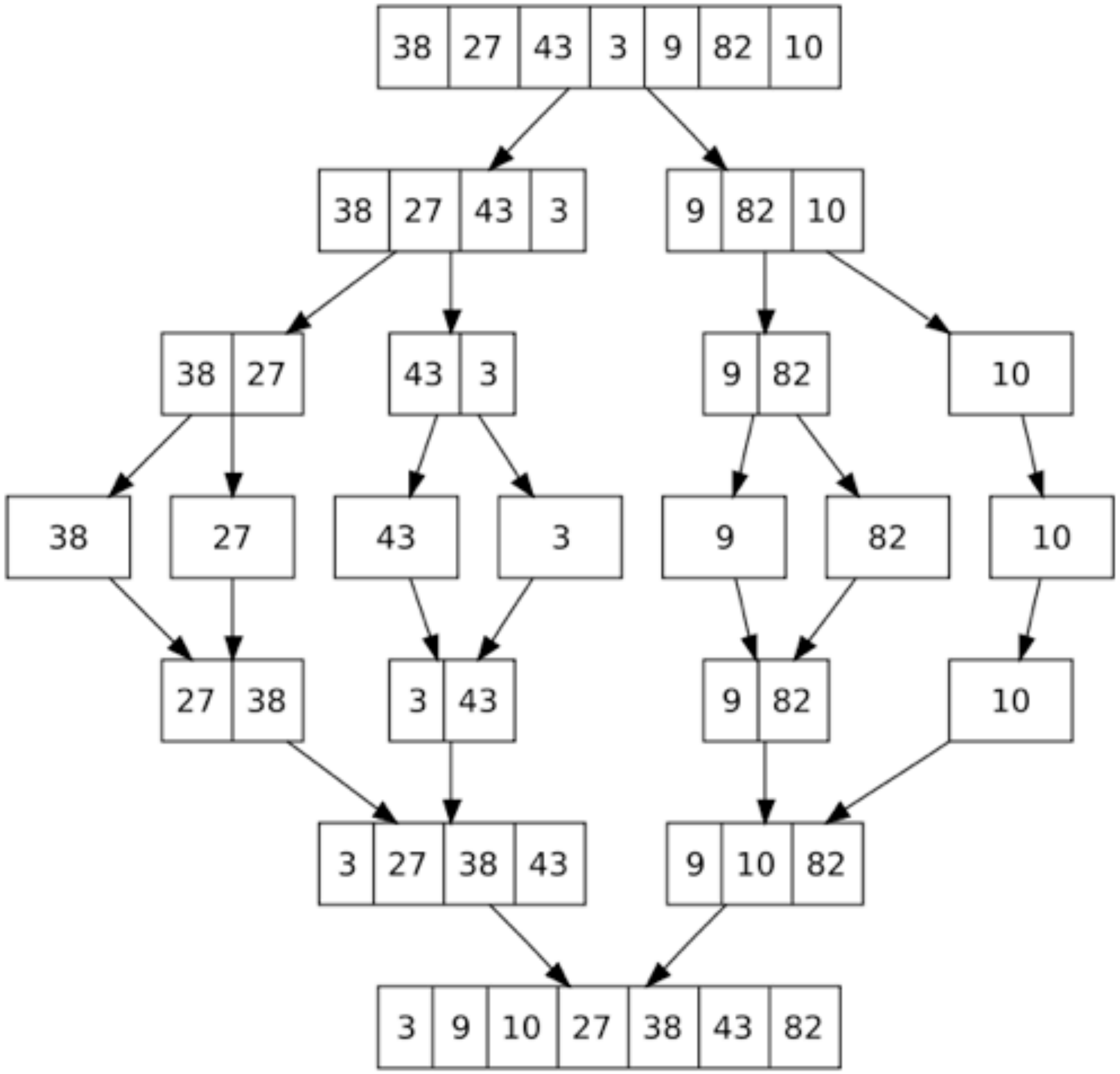
Mergesort - no recursivo

- Este algoritmo recursivo tiene su equivalente no-recursivo.

```
void mergesort( Item a[], int l, int r )
{
    for ( int m=1; m<=r-l; m=m+m)
        for( int i=l; i<r-m; i += m+m)
            merge(a, i, i+m-1, min(i+m+m-1, r));
}
```

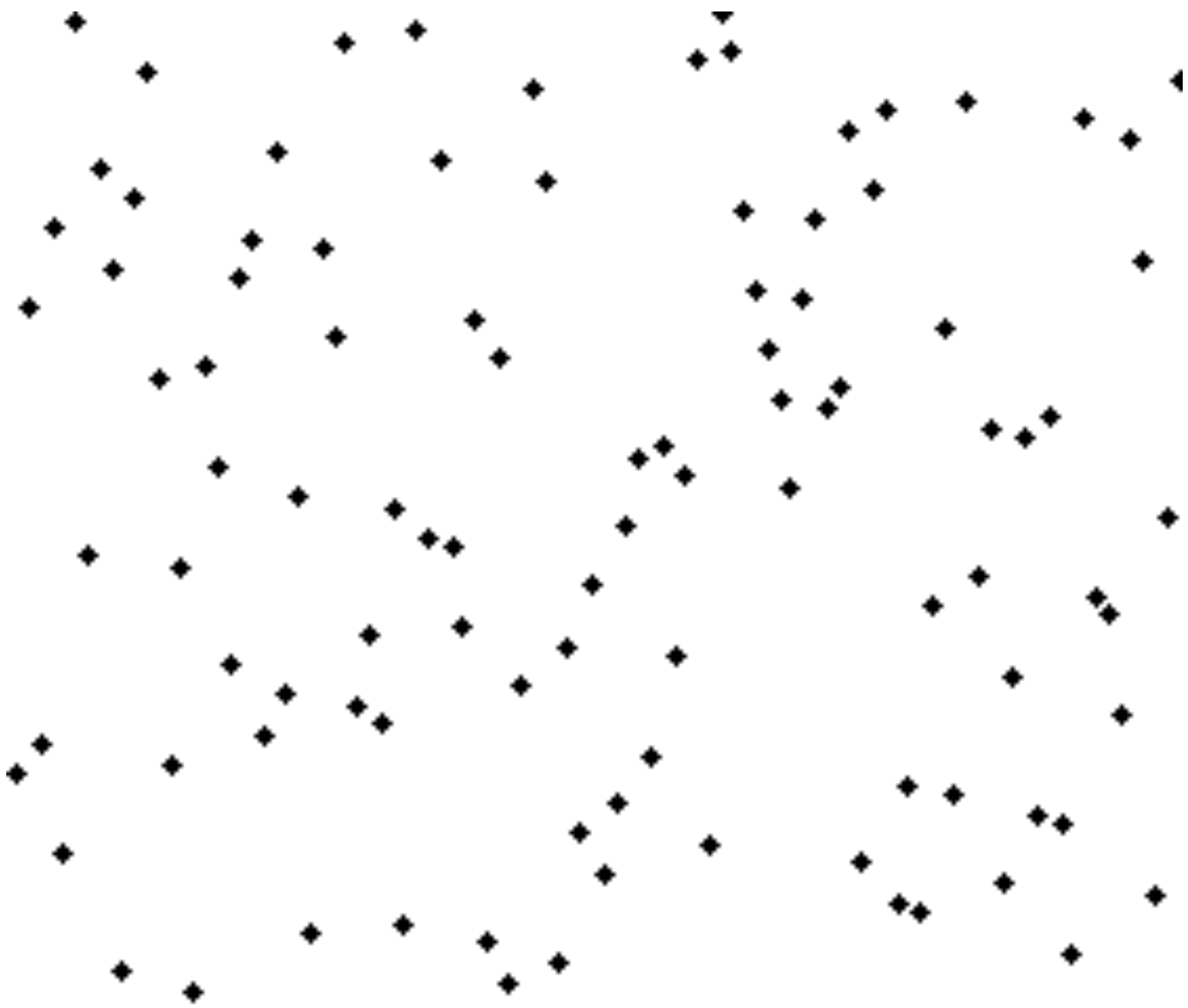

2
4
8
16
32
64

Mergesort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Mergesort

Mergesort



Mergesort

- ***Ventajas***
    - escala muy bien a listas muy grandes porque su complejidad en el peor caso es *O(n log n)*.
    - estable.
    - es muy adecuado para implementaciones en paralelo.
    - adecuado para ordenar elementos secuenciales: ***listas ligadas***.
- ***Desventajas***
    - utiliza memoria adicional.

Algoritmos de ordenamiento.

- ¿ Cuál usar?
  - estable
  - llaves múltiples
  - listas ligadas o arreglos
  - entradas grandes o chicas
  - necesidad de garantía en el desempeño

Algoritmos de ordenamiento

	en su lugar	estable	peor caso	caso promedio	mejor caso	comentarios
bubble	✗	✗	N^2	$N^2/2$	-	$N^2/2$ comparaciones, $N^2/2$ intercambios
selection	✗		$N^2/2$	$N^2/2$	$N^2/2$	$N^2/2$ comparaciones, N intercambios
insertion	✗	✗	$N^2/2$	$N^2/4$	N	utilizado para pequeños N o parcialmente ordenados.
shell	✗		?	?	N	código comprimido, sub-cuadrático
quick	✗		$N^2/2$	2 N log N	N lg N	garantía probabilística de N lg N, má́s rápido en la práctica.
merge		✗	N lg N	N lg N	N lg N	garantía N lg N, estable

http://math.hws.edu/TMCM/java/xSortLab/

Aplicaciones de algoritmos de ordenamiento

- Ordenar una lista de nombres
- organizar una librería de archivos mp3,
- desplegar resultados del Google PageRank
- listar elementos de noticias RSS en orden cronológico,
- encontrar la mediana,
- encontrar el punto más cercano,
- búsqueda binaria en una base de datos,
- encontrar duplicados en una lista de nombres,
- gráficos computacionales, biología computacional, paralelización...

Quick Sort Iterativo

```
class par{
public:
	int left,right;
	par(int l,int r){ left = l, right=r};
}
```

Quick Sort Iterativo

```
/* arr[] --> Array to be sorted, l  --> Starting index, r  --> Ending index */
void quickSortIterative (int arr[], int l, int r){
    // Create an auxiliary stack
    Stack<par> stack;
    // push initial values of l and r to stack
    par unPar(l,r)
    stack.push(unPar);

    // Keep popping from stack while is not empty
    while ( !stack.empty() ){
        // Pop r and l
        unPar = stack.pop();
        l = unPar.left; r = unPar.right;

        // Set pivot element at its correct position in sorted array
        int p = partition( arr, l, r );

        // If there are elements on left side of pivot, then push left side to stack
        if ( p-1 > l ){
            unPar.left = l; unPar.right = p-1;
            stack.push(unPar);
        }

        // If there are elements on right side of pivot then push right side to stack
        if ( p+1 < r ){
            unPar.left = p+1; unPar.right = r;
            stack.push(unPar);
        }
    }
}
```