Algoritmos Paralelos

(11.09.15)

Prof. J.Fiestas

Ejercicio 1: Suma de vectores

- Programar la suma de vectores secuencialmente
- Utilizar nomenclatura de abstracción PRAM para programar la suma de dos vectores en paralelo

1 procesador:

$$\text{for } i=1..n \ \{ \ c[i]=a[i]+b[i]; \ \}$$

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$$

n procesadores: programa para procesador j

$$c[j]=a[j]+b[j]$$

Ejercicio 1: Suma de vectores

En serie:

```
#include<iostream>
using namespace std;

#define N 10
int main(){
int S=0, C[N];
int A[N]= {1,2,3,4,5,6,7,8,9,10};
int B[N]= {1,2,3,4,5,6,7,8,9,10};

for(int j=0;j<N;j++){
        C[j]=A[j]+B[j];
        cout<<"C[j]: "<<C[j]<<endl;
}

}
```

Ejercicio 1: Suma de vectores

En paralelo,
pseudo código:

```
for i:= 1 to n pardo
C[i] := A[i] + B[i]
endfor
```

Ejercicio 2: Adición de elementos

- Programar la adición de elementos de un array en C/C++ en forma secuencial
- Sea el número de procesadores **p** , tal que p < n
- Formular en PRAM un código que represente la ejecución en cada proceso p
- Considere repartir t=n/p tareas a cada proceso
- Diagramar un grafo para p=4, n=8

Ejercicio 2: Adición de elementos

Dados: n números A_1 .... A_n
Calcular: Suma de A_1 ... A_n

En serie:

```
#include<iostream>
using namespace std;

#define N 8
int main(){
int S=0;
int B[N]= {1,2,3,4,5,6,7,8};

for(int j=0;j<N;j++)
   S+=B[j];

cout<<"suma: "<<S<<endl;
}
```

Ejercicio 2: Adición de elementos

En paralelo:

pseudo código

```
1 for i := 1 to n pardo
2 |   B[i] := A[i]
3 endfor
4 for h := 1 to log(n) do
5 |   for i := 1 to n/2^h pardo
6 |   |   B[i] := B[2i − 1] + B[2i]
7 |   endfor
8 end
9 S := B[1];
```

Ejercicio 2: Adición de elementos

```
#include<iostream>
#include<cmath>
using namespace std;

int main(){
int N=8;
int B[N]={1,2,3,4,5,6,7,8};
int S;

for(int h=1;h<=log2(N);h++){
        for(int j=0;j<N/pow(2,h);j++)
                        B[j]=B[2*j]+B[2*j+1];
}

S=B[0];
cout<<"suma: "<<S<<endl;
}
```

Ejercicio 2: Adición de elementos

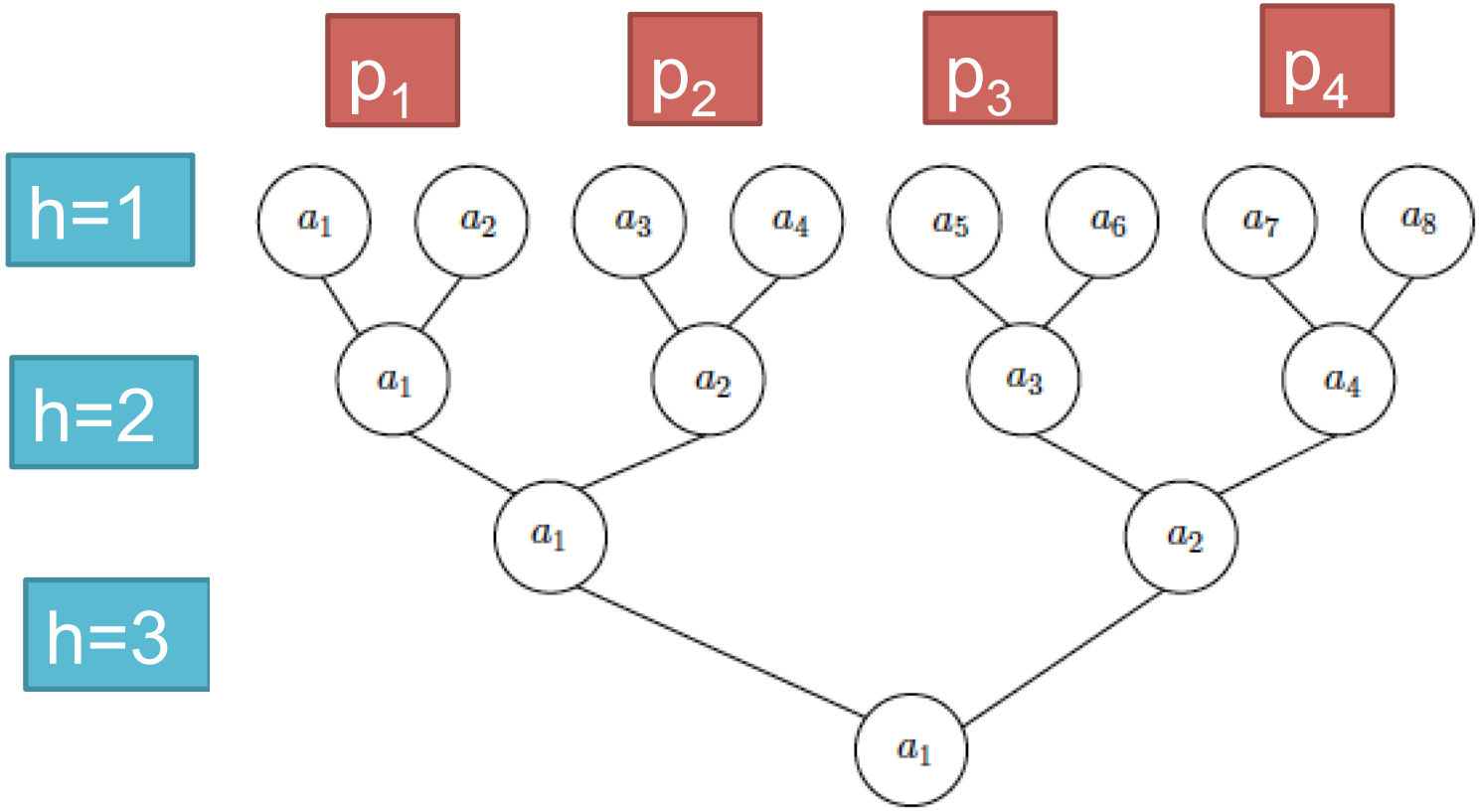
En paralelo:

pseudo código

```
1: for h := 0 to log n do
2:     for i := 0 step 2^(h+1) to n − 2^(h+1) par do
3:         A[i] := A[i] + A[i + 2^h];
4:     end for
5: end for
6: return A[0];
```

Ejercicio 2: Adición de elementos

Dados: n números a_1 .... a_n
Calcular: Suma de a_1 ... a_n



Ejercicio 3: Suma de prefijos

- Programar la suma de prefijos secuencialmente
- Utilizar nomenclatura de abstracción PRAM para programar la suma de prefijos en paralelo

Ejercicio 3: Suma de prefijos

Dados: n números a_1 .... a_n
Calcular: Suma de s_1 ... s_n , con $s_k = \sum_{i=1}^{k} a_i$

Secuencialmente:

```
s_0 := 0
for i:=1 to n do
    s_i := s_{i-1} + a_i;
end for
```

a :	2	5	6	1	2	3	5	4
s :	2	7	13	14	16	19	24	28

Cada bucle necesita tiempo de orden Θ(1), o un tiempo total de Θ(n)

Ejercicio 3: Suma de prefijos

```
#include<iostream>
using namespace std;

#define N 8
int main(){
int B[N]= {1,2,3,4,5,6,7,8};
int C[N];

C[0]=B[0];
for(int j=1;j<N;j++){
        C[j]=C[j-1]+B[j];
        cout<<"j: "<<j<<", C[j]: "<<C[j]<<endl;
        }

}
```

Ejercicio 3: Suma de prefijos

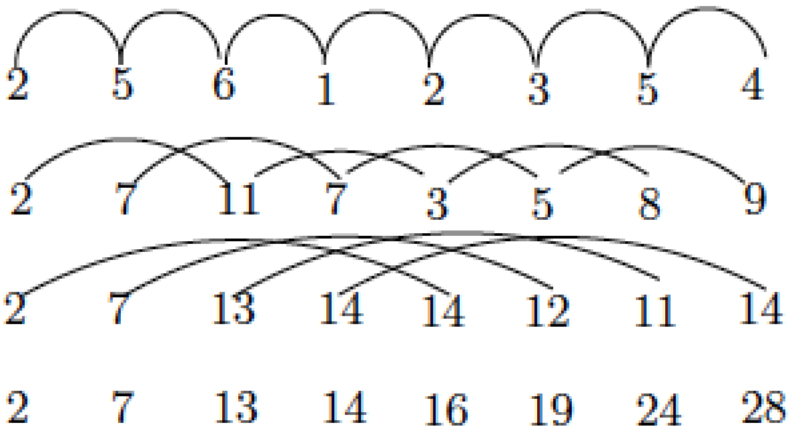
Pseudo código
En paralelo

$s_0 := 0$
for i:=1 **to** n **do** ***pardo***
  $s_i := s_{i-1} + a_i$;
end for

Ejercicio 3: Suma de prefijos

Dados: n números a_1 .... a_n
Calcular: Suma de s_1 ... s_n , con $s_k = \sum_{i=1}^{k} a_i$

Algoritmo paralelo para Suma acumulada:
Suma por pares y almacenamiento en segunda Posición. Luego suma por Pares de distancia 2,4,8, ... , n/2



El tiempo total para n números en n procesos será Θ(log n)

Ejercicio 3: Suma de prefijos

pseudo código en paralelo

```
h := 1;
for i := 1 to n par do
    s_i := a_i;
end for
while h ≤ n/2 do
    for i := h + 1 to n par do
        t_i := s_i + s_{i−h};
    end for
    h := h · 2;
    for i := 1 to n par do
        s_i := t_i;
    end for
end while
```

Ejercicio 3: Suma de prefijos

```
#include<iostream>
using namespace std;

int main(){
int N=8,h;
int B[N]={1,2,3,4,5,6,7,8};
int S[N],T[N];

for(int i=0;i<N;i++){
S[i]=B[i];
}

h=1;
T[0]=S[0];
while(h<=N/2){
for(int i=h;i<N;i++)
        T[i]=S[i]+S[i-h];

h=2*h;

for(int j=0;j<N;j++)
        S[j]=T[j];
}

for(int j=0;j<N;j++)
        cout<<"B[j]: "<<B[j]<<", S[j]: "<<S[j]<<endl;
}
```

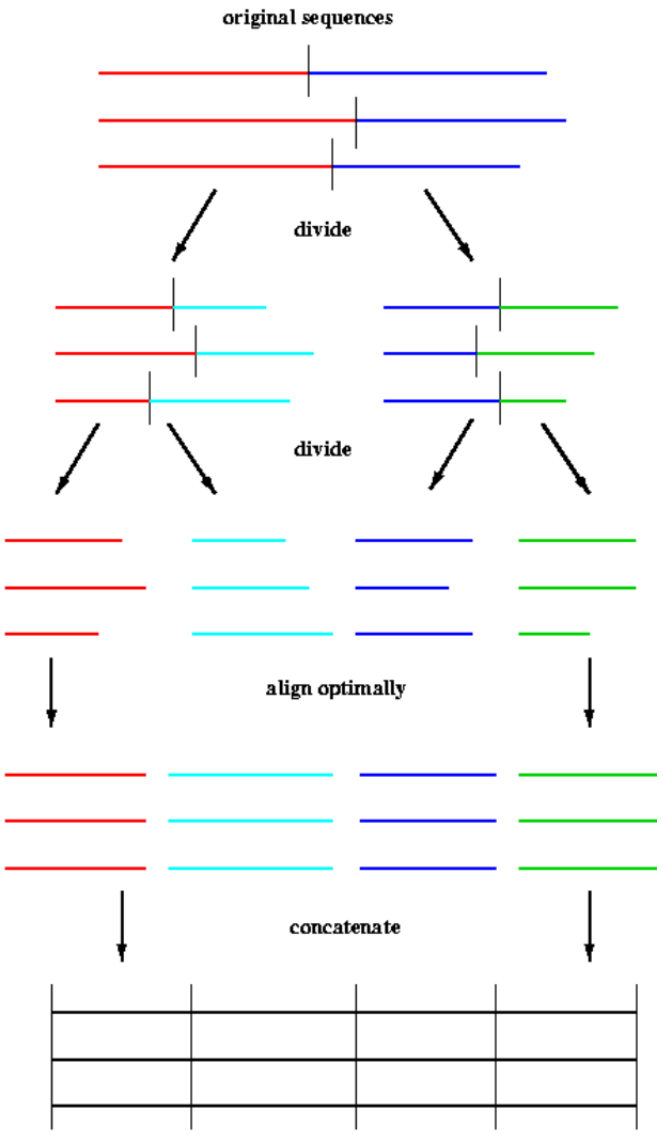
Paradigma Divide y Vencerás (divide and conquer)

Algoritmo heuristico acelerado para solución de secuencias múltiples y homologas. Funciona:

- Separando las secuencias en partes reduciendo su longitud. Esto es, dividiendo el problema en sub-problemas

Paradigma Divide y Vencerás (divide and conquer)

- Optimizar su distribución o resolver los sub-problemas (recursiva, directamente)
- Concatenar resultados o combinar soluciones de subproblemas en solución general

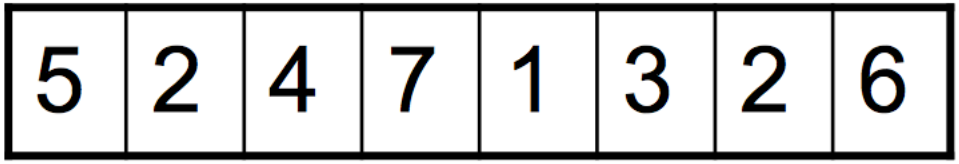


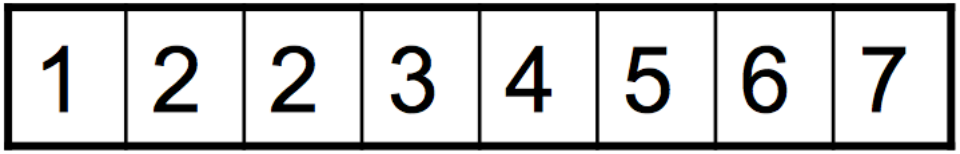
Divide and conquer

Ordenamiento:

Dado: un array

Objetivo: ordenar el array



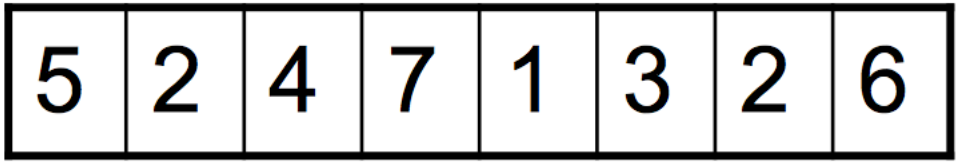


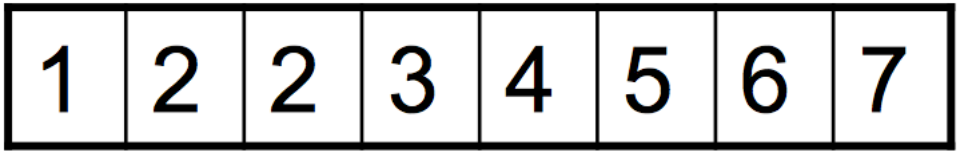
Divide and conquer

Ordenamiento:

Dado: un array

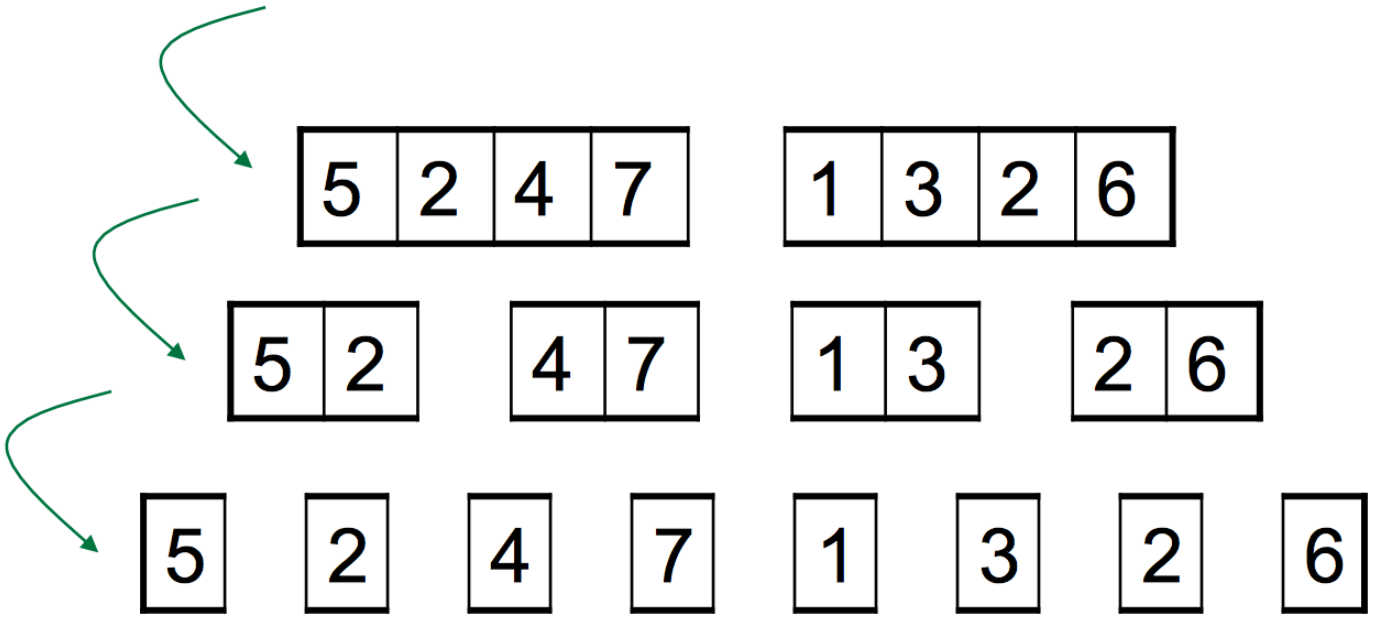
Objetivo: ordenar el array





Divide and conquer Ordenamiento:

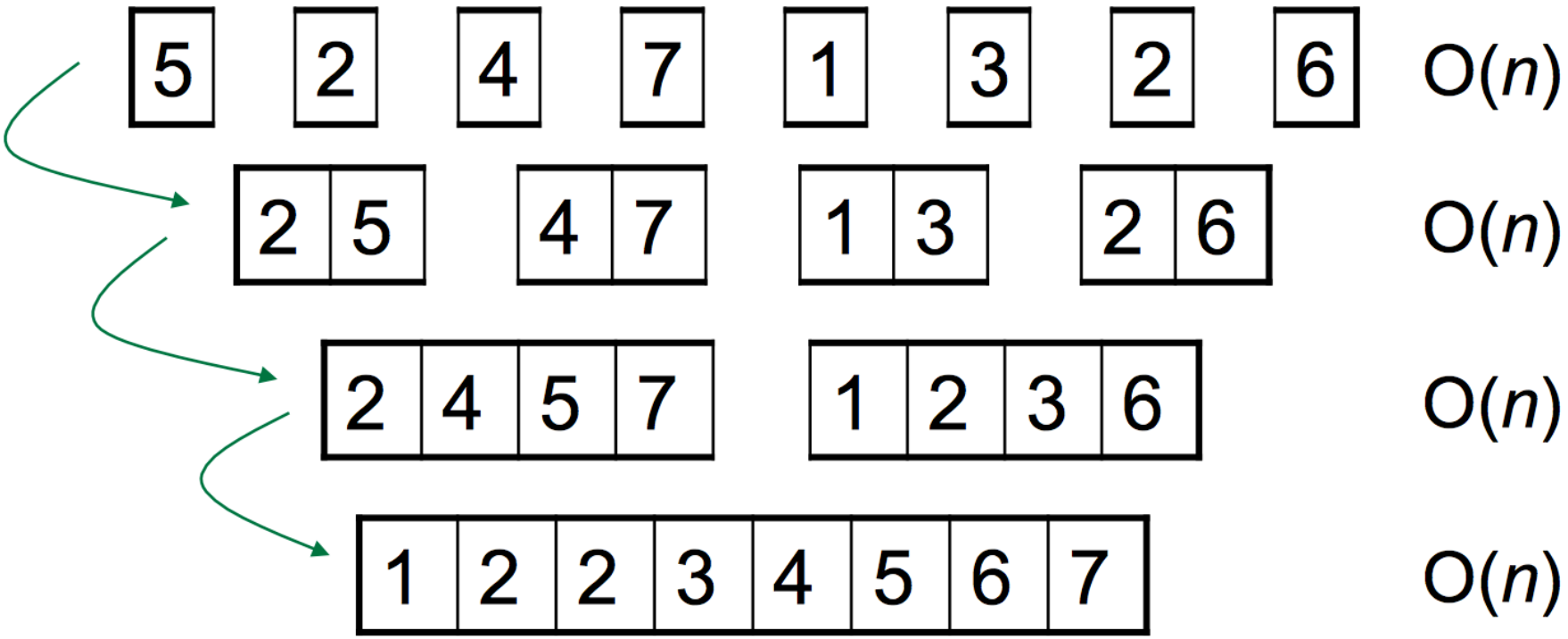
Paso 1: dividir el array. Necesita log(n) divisiones para lograr arrays de un elemento



Divide and conquer Ordenamiento:

Paso 2: Conquista, requiere log(n) iteraciones, cada una tomando tiempo O(n)

T(n) = O(n log n)



Divide and conquer Ordenamiento:

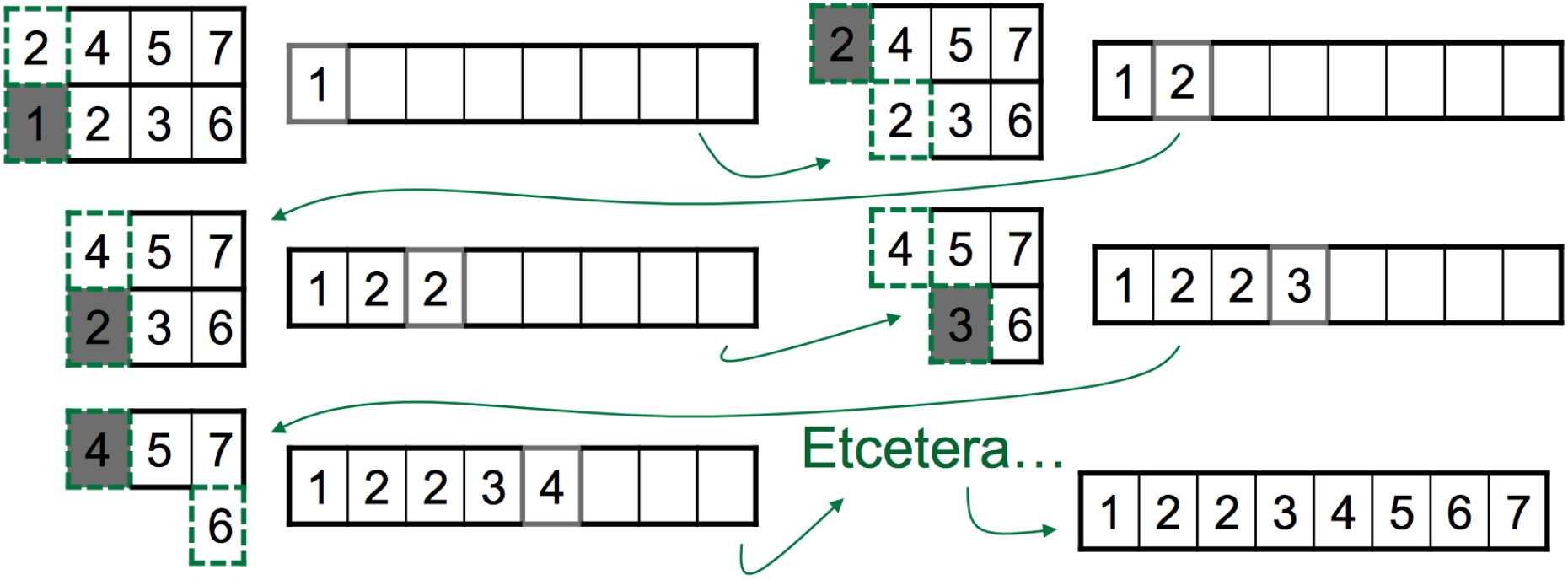
Paso 3: Combinar

Para 2 arrays de tamaño 1 la union es directa

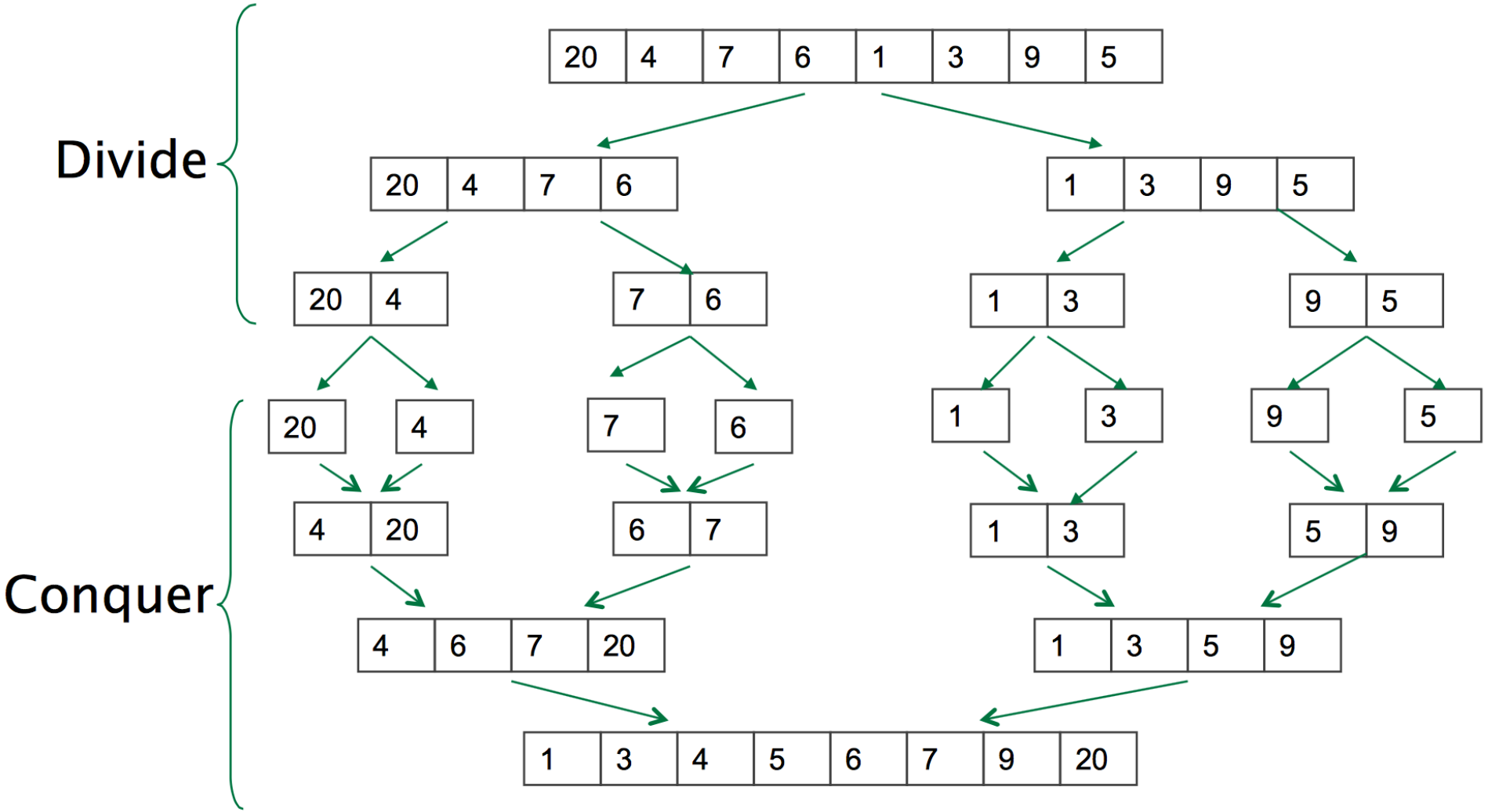
En general, 2 arrays ordenadas de tamaño n y m pueden ser combinadas en un tiempo O(n+m) para formar un array n+m ordenada.

Divide and conquer Ordenamiento: Ejemplo

Combinar dos arrays de tamaño 4



Divide and conquer Ordenamiento: Mergesort



Divide and conquer Ordenamiento:
Mergesort: algoritmo

mergesort(arr[],l,r**)**

If r>l

1. Encuentre el punto de división del array en dos mitades: middle m = (l+r)/2
2. Llamar mergeSort para la primera mitad:
   call mergeSort(arr, l, m)
3. Llamar mergeSort para la segunda mitad
   call mergeSort(arr, m+1, r)
4. Combinar las dos mitades ordenadas en 2 y 3
   call merge(arr, l, m, r)

Divide and conquer Ordenamiento:

Mergesort: complejidad

- Para la combinación i la complejidad es O(n)
- Numero de iteraciones es O(log n)
- Tiempo total es O(n log n)
- Se usa para ordenar listas concatenadas (linked lists) en tiempo O(n log n)

Observar algoritmo funcionando en:
http://visualgo.net/sorting.html

Ejercicio 7:

Paralelizar el método de **mergesort** Utiize la siguiente función **merge_sort()**

```
void merge_sort(int iArray[], int startIndex, int endIndex)
{
        int midIndex;

//Check for base case
if (startIndex >= endIndex)
{
   return;
  }

//First, divide in half
midIndex = (startIndex + endIndex)/2;

//First recursive call
merge_sort(iArray, startIndex, midIndex);

//Second recursive call
merge_sort(iArray, midIndex+1, endIndex);

 //The merge function
 merge(iArray, startIndex, endIndex);

  }
```

Ejercicio 7:

Paralelizar el método de **mergesort**
Y la función **merge**()

```
void merge(int a[], int startIndex, int endIndex)
{
int size = (endIndex - startIndex) + 1;
int *b = new int [size]();
int i = startIndex;
int mid = (startIndex + endIndex)/2;
int k = 0;
int j = mid + 1;

while (k < size)
{
    if((i<=mid) && (a[i] < a[j]))
        {
        b[k++] = a[i++];
    }
    else
    {
        b[k++] = a[j++];
     }
}

for(k=0; k < size; k++)
{
    a[startIndex+k] = b[k];
}
delete []b;
}
```

Ejercicio 7:

Paralelizar el método de **mergesort**

- Utilize un array de enteros aleatorios de 2^{18} elementos (valores ente 1 y 100)
- Distribuya el array en s=n/p partes iguales y ejecute merge_sort(). E.g.

```
MPI_Scatter(array,s,MPI_INT,subarray,s,MPI_INT,0,
MPI_COMM_WORLD);
merge_sort(subarray, 0, s-1);
```

Ejercicio 7:

Paralelizar el método de **mergesort**

- Recombine las partes en un array de dimension original

```
MPI_Gather(subarray, size, MPI_INT, sortedarray, size, MPI_INT, 0, MPI_COMM_WORLD);
```

- Ejecute un llamado final de merge_sort() desde el proceso principal (rank=0)
- Imprima el array ordenada
- Mida tiempos de ejecución con MPI_Wtime() para np=2,4,8

Ejercicio 8. Paralelizar la multiplicación de dos matrices en MPI.

```
for(i=0;i<NA;i++)
  for(j=0;j<NB;j++)
    for(k=0;k<NC;k++)
      C[i][j]+= A[i][k] * B[k][j];
```

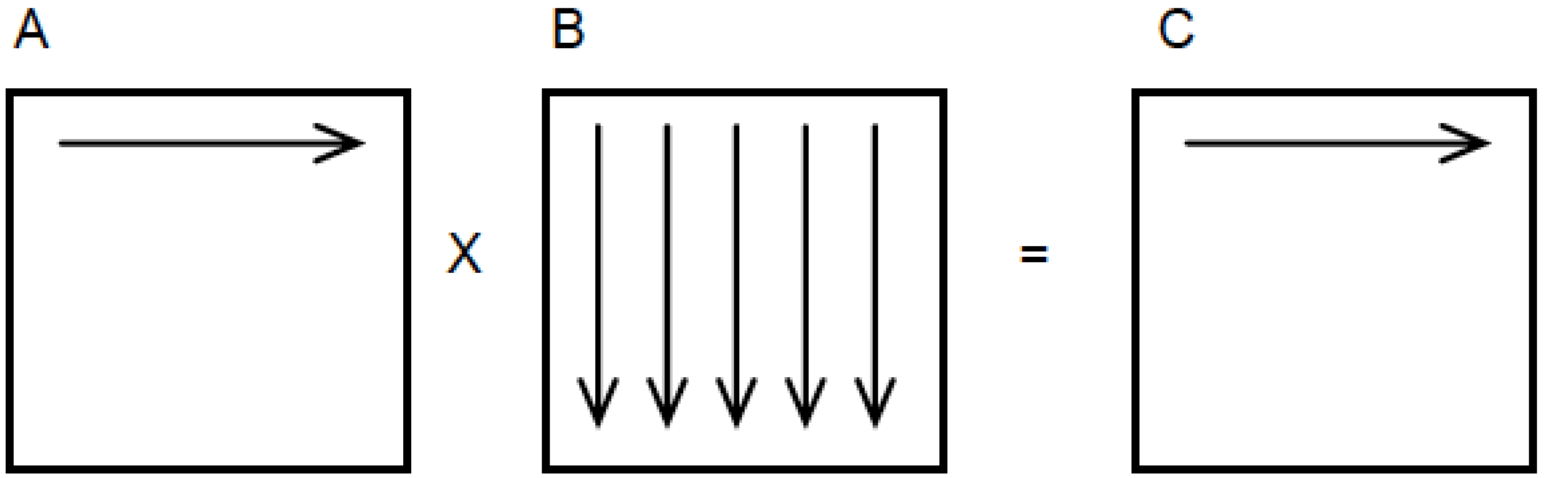
$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

(a)

$$\begin{aligned} \text{Task 1: } C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ \text{Task 2: } C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ \text{Task 3: } C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ \text{Task 4: } C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

(b)

Ejercicio 8. multiplicación de dos matrices



Algoritmo ejecuta lineas de la matriz C
Cada iteración, una linea de A y todas las columnas de B son procesadas
Complejidad del problema: O(ijk), donde las matrices son i x j

Ejercicio 9. multiplicación de dos matrices

La operación básica es calcular un elemento de C

$$c_{ij} = \left(a_i, b_j^T\right),\ a_i = \left(a_{i0,} a_{i1}, \ldots, a_{in-1}\right),\ b_j^T = \left(b_{0j}, b_{1j}, \ldots, b_{n-1j}\right)^T$$

Multiplicar dos matrices de 500x500 números enteros aleatorios con valores entre 1 y 100
Medir tiempos de cálculo para np=2,4,8

Ejercicio 9. multiplicación de dos matrices

```
if (rank == 0) {
      // initialize matrices
      // Send matrix data to the worker tasks
      for (dest=1; dest<=np-1; dest++)
      { .....
            rows = .....
MPI_Send(&a[offset][0], rows*NA, MPI_DOUBLE, dest, mtype,MPI_COMM_WORLD);
      }
       for (i=1; i<np; i++)
      { ......
MPI_Recv(&c[offset][0], rows*NB, MPI_DOUBLE, i, mtype,MPI_COMM_WORLD, &status);
      }
}
 if (rank > 0) {
 MPI_Recv(&a, rows*NA, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,&status);
      // calculate sub-matrix
MPI_Send(&c, rows*NB, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD);
....
}
```