uc3m | Universidad Carlos III de Madrid

Tema 7. Estrategias Algorítmicas: Divide y Vencerás.

Estructura de datos y algoritmos (EDA)

CC BY NC SA

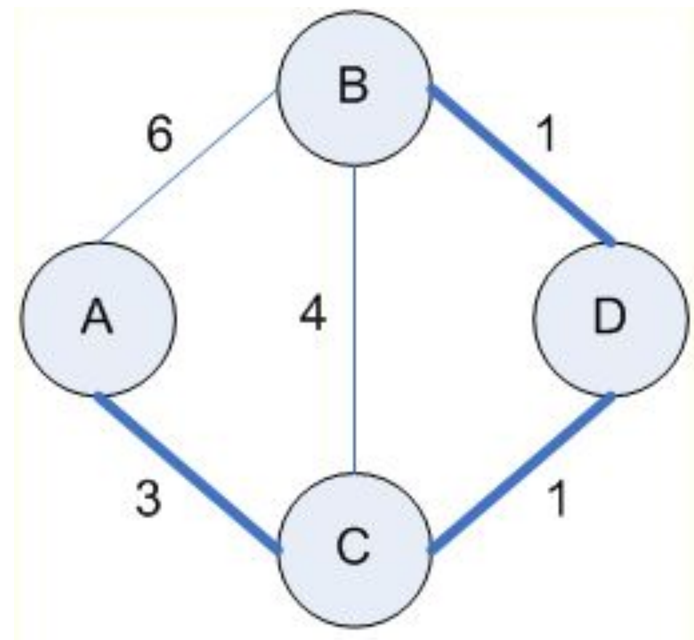
Algunos conceptos

- Un **algoritmo** es una secuencia finita y bien definida de pasos utilizada para resolver un problema bien definido
- **Estrategia del algoritmo**
  - Enfoque para resolver un problema
  - Se pueden combinar varios enfoques
- **Estructura de algoritmos:**
  - **Iterativo:** se utiliza un bucle para encontrar la solución
  - **Recursivo:** una función que se llama a sí misma

cc BY NC SA

Problema tipo

- **Se satisface**
  - Encontrar cualquier solución que satisfaga
  - Ej: Encontrar un camino de A a B
- **Optimización**
  - Encontrar la mejor solución
  - Ej: Encontrar el camino más corto de A a B



This example was taken from http://cs.smu.ca/~porter/csc/common_341_342/notes/graphs_shortest_path.html

uc3m | Universidad Carlos III de Madrid
cc BY NC SA

Principales estrategias de algoritmos

- Algoritmos recursivos
- Algoritmos de divide y vencerás
- Algoritmos de Backtracking
- Algoritmos de programación dinámica
- Algoritmos '*greedy*'
- Algoritmos de fuerza de bruta
- Algoritmos de ramificación y poda
- Algoritmos heurísticos

Heurística y optimización, Curso 3ª, Semestre 1º

These slides are based on the course CMSC 132 's materials, University of Maryland

cc BY NC SA

Divide y vencerás

- Basado en dividir el problema en sub-problemas
- Aproximación en tres pasos:

1) ***Dividir***: dividir el problema en sub-problemas más pequeños del mismo tipo
2) ***Vencer***: resolver recursivamente cada sub-problema, resolver cada sub-problema independientemente.
3) ***Combinar***: combinar soluciones para resolver el problema original

- Por lo general, contiene dos o más llamadas recursivas

uc3m | Universidad Carlos III de Madrid
CC BY NC SA

Divide y vencerás

- Esquema General

divide_venceras (p: problema)

dividir (p, p_1, p_2, ..., p_k)

para i = 1, 2, ..., k

s_i = *resolver* (p_i)

solucion = *combinar* (s_1, s_2, ..., s_k)

resolver (pi) puede ser recursivo siendo una nueva llamada a divide_venceras

uc3m | Universidad Carlos III de Madrid

CC BY NC SA

Divide y vencerás

- Algunos ejemplos:
  - Búsqueda binaria
  - Encontrar el elemento máximo en un array
  - Merge-sort
  - Quick-sort

uc3m | Universidad Carlos III de Madrid
cc BY NC SA

Divide y vencerás. Búsqueda binaria

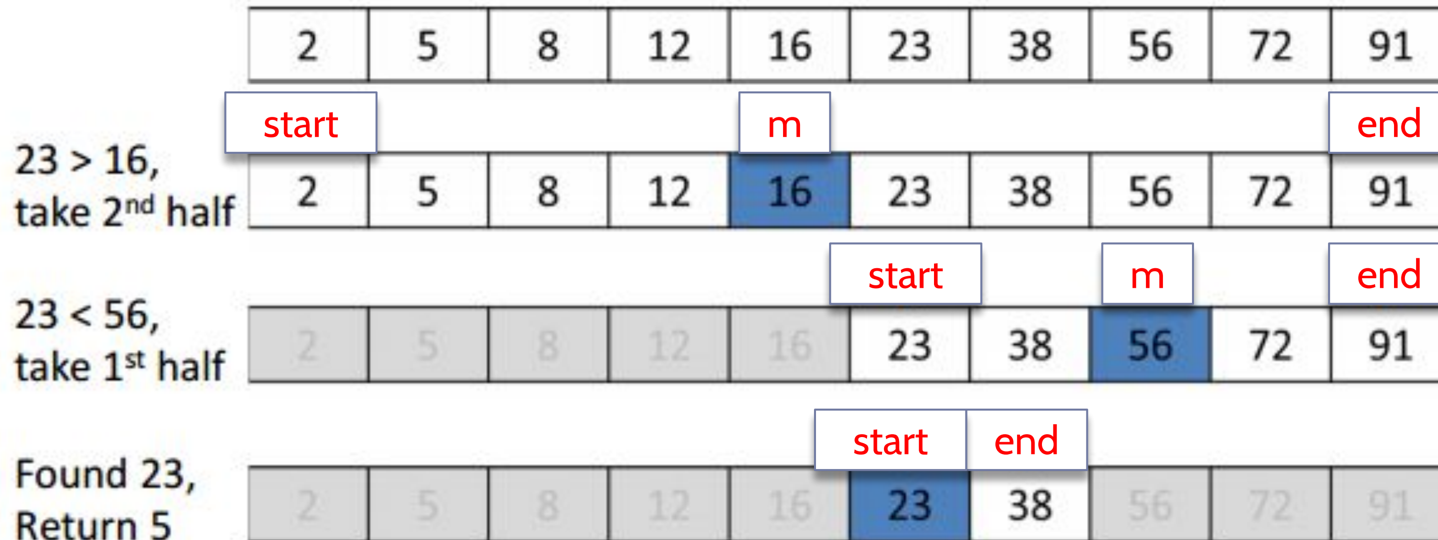
- Dado un array A[] ordenado de enteros, escribir un método que busque un número entero x en A[]
- El método tiene que devolver la (primera) posición de x en A[]. Si no existe, entonces devuelve -1

CC BY NC SA

Divide y vencerás – Búsqueda binaria Ejemplo

Pre-requisitos:

- El array debe tomar valores únicos
- El array debe de estar ordenado de manera ascendente



cc BY NC SA

Divide y vencerás – Búsqueda binaria

```java
public static int searchBinary(int A[], int x) {
    if (A==null ||A.length==0) {
        System.out.println("Error: array is empty");
        return -1;
    }
    return searchBinary(A,0,A.length-1,x);
}
```


cc BY NC SA

Divide y vencerás – Búsqueda binaria

```
public static int searchBinary(int A[], int start, int end, int x) {
    if (start>end || start<0 ||end>=A.length) {
        System.out.println("Error: indexed out of range");
        return -1;
    }
    if (start==end) {
        if (A[start]==x) return start;
        else return -1;
    } else {
        int m = start + (end - start)/2;
        if (x==A[m]) return m;
        else if (x<A[m]) return searchBinary(A,start,m-1,x);
        else  return searchBinary(A,m+1,end,x);
    }
}
```

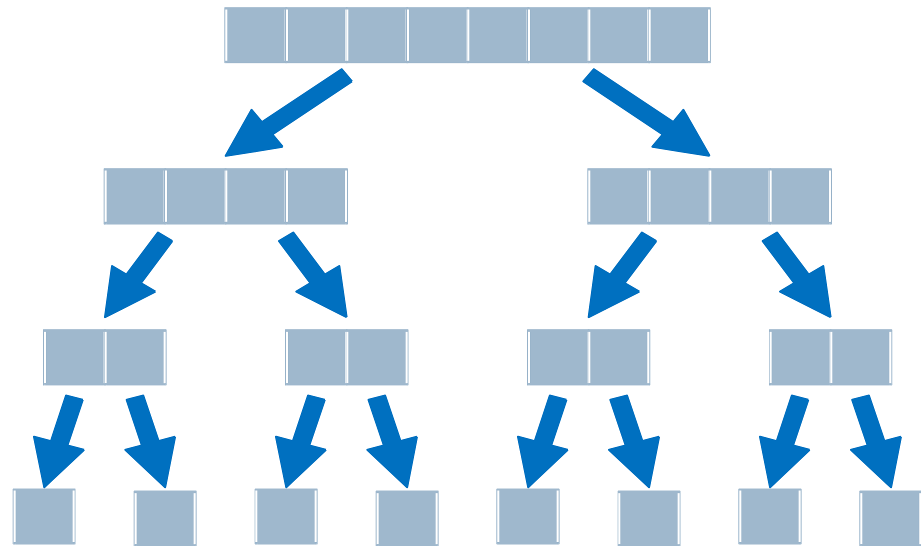

cc BY NC SA

Divide y vencerás – Encontrar máximo

A:	-3	24	11	-13	…	33	-7	12	1

1. Divide el array en dos partes
2. Encontrar el máximo en cada parte
3. Comparar ambos números y devolver del mayor.

- Es aplicado mediante llamadas recursivas

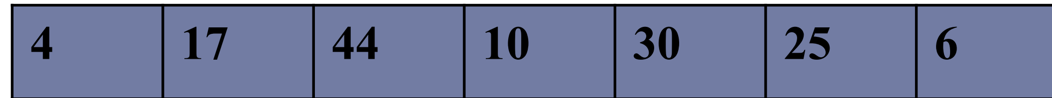


uc3m | Universidad Carlos III de Madrid

CC BY NC SA

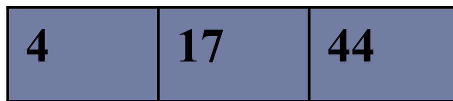
Divide y vencerás – Encontrar máximo Ejemplo

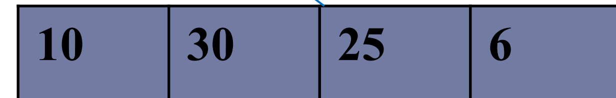
Max(44, 30)=44



Max(17, 44)=44

Max(30, 25)=30

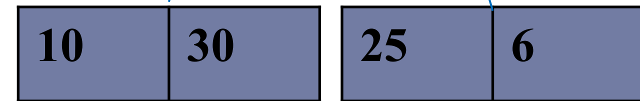


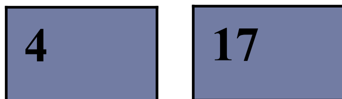


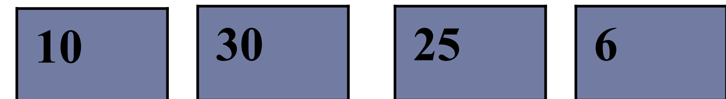
Max(4,17)=17

Max(10,30)=30

Max(25,6)=25







cc BY NC SA

Divide y vencerás – Encontrar máximo

```
public static int findMax (int[] vector, int i, int j){
  int med, max_left, max_right;
  if (i==j)
      return vector[i];
  else
      jmed = (i + j) / 2;
      max_left = findMax (vector, i, med);
      max_right = findMax (vector, med+1, );
  if (max_left > max_right)
       return max_left;
  else
       return max_right;
}
```

Si hay un único elemento, es el máximo

Divide a la mitad
Máximo de cada mitad

El máximo del array es el máximo de los máximos de cada mitad

uc3m | Universidad Carlos III de Madrid

CC BY NC SA

Divide y vencerás - Mergesort

- Merge-sort algoritmo: ordenar un array

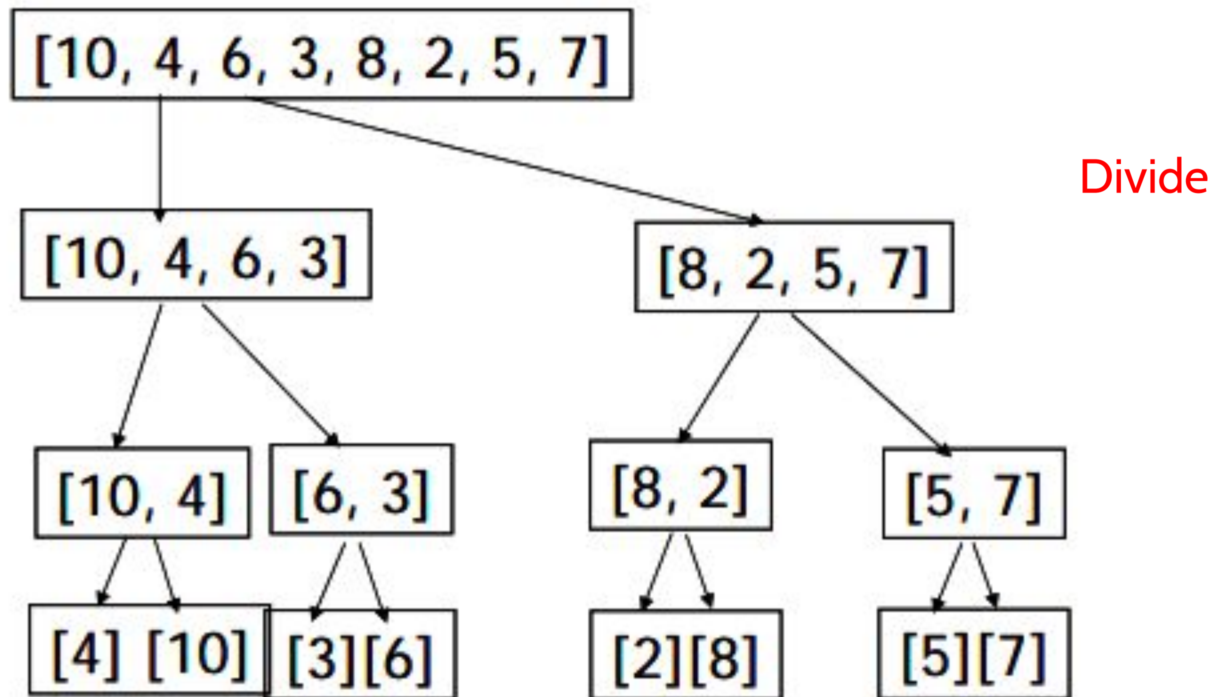
1) ***Dividir***:
   - Dividir el array en dos (aproximadamente la mitad)
   - Seguir dividiendo los arrays resultantes hasta que ya no pueda dividir más (arrays de longitud uno)

2) ***Vencer***:
   - Ordenar cada sub-array recursivamente
   - Los arrays de longitud uno están ordenados

4) ***Combinar***:
   - A continuación, combinar los arrays adyacentes para formar un solo array ordenado.
   - Repetir el proceso hasta que tengamos un único array ordenado

cc BY NC SA

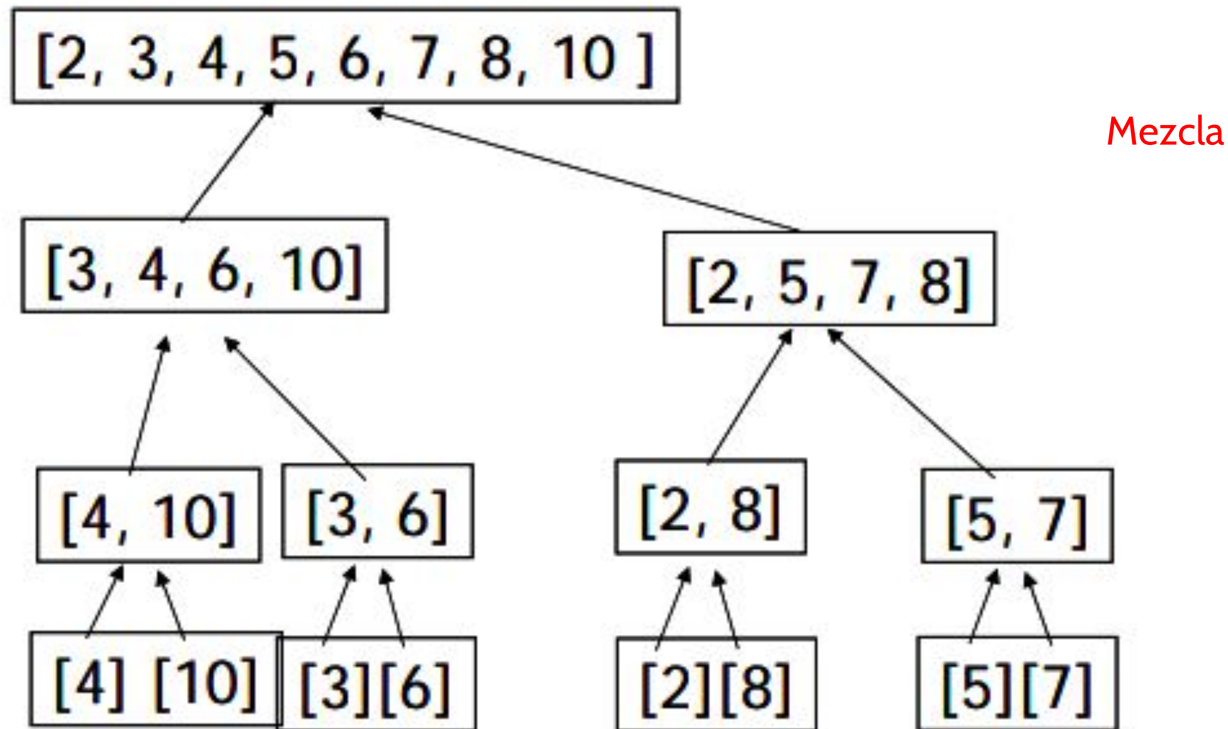
Divide y vencerás – Mergesort Ejemplo



Divide

cc BY NC SA

Divide y vencerás – Mergesort Ejemplo



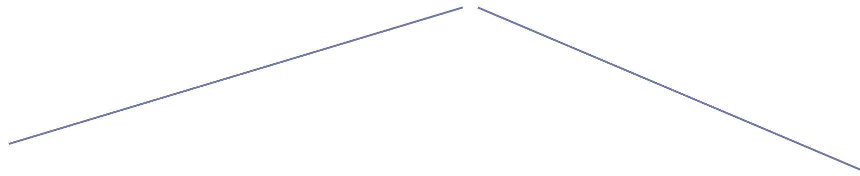
Mezcla

uc3m | Universidad Carlos III de Madrid

cc BY NC SA

Divide y vencerás – Mergesort
Ejemplo

3	4	6	10	2	5	7	8

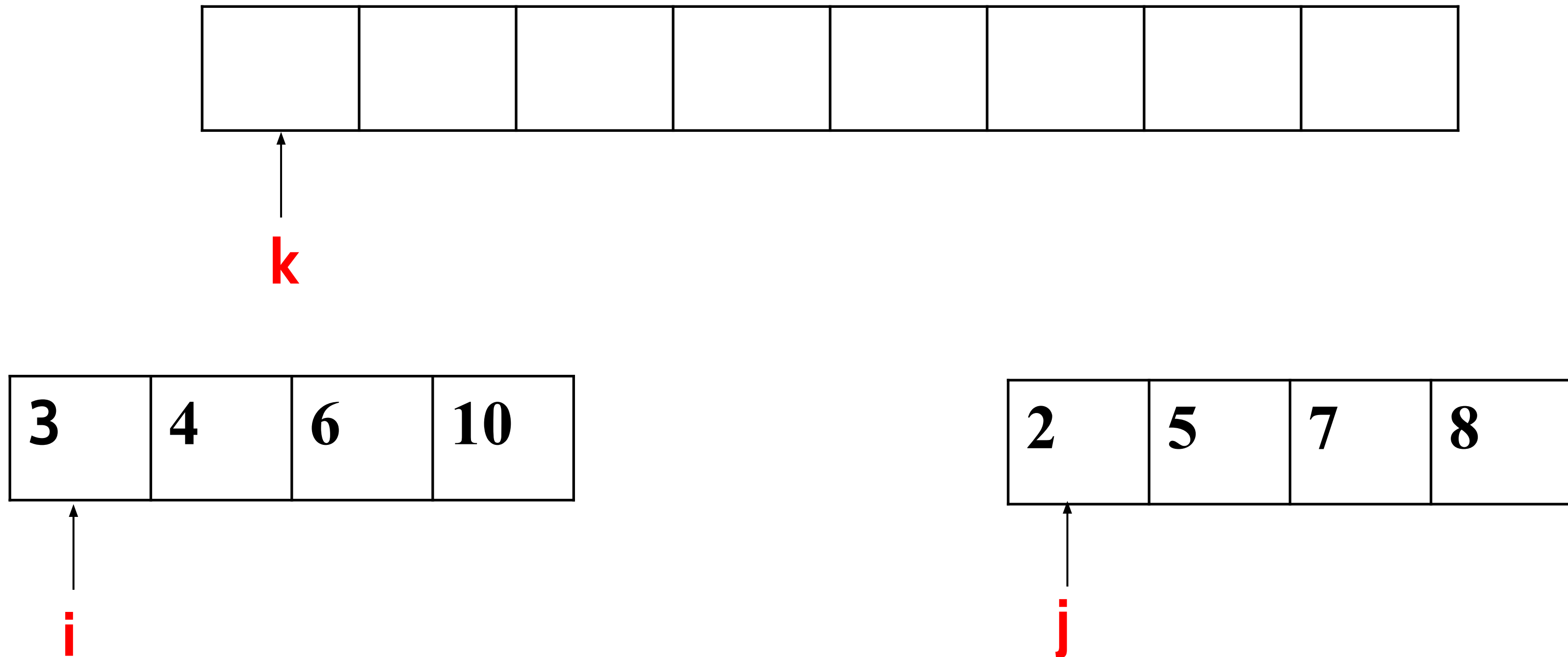


3	4	6	10

2	5	7	8

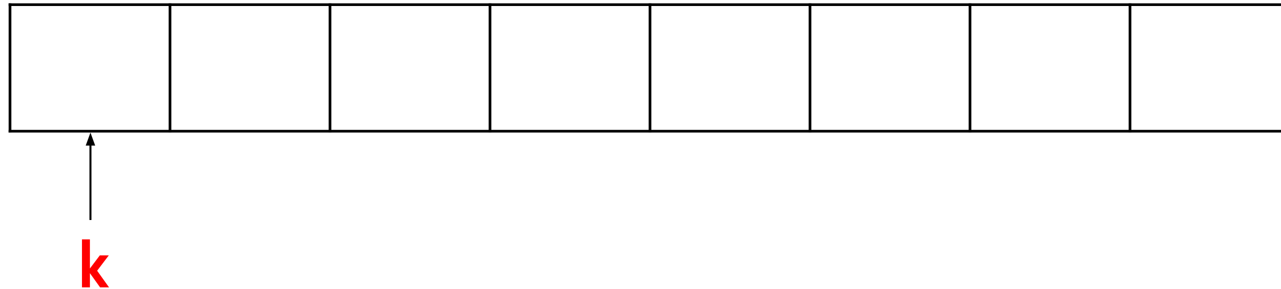
uc3m | Universidad Carlos III de Madrid
cc BY NC SA

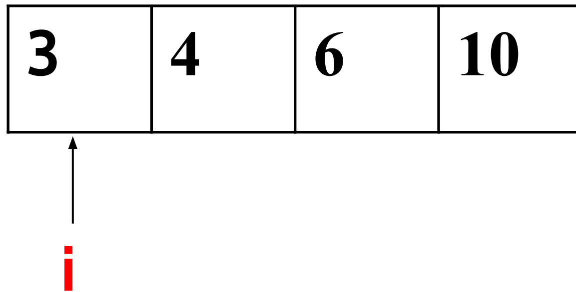
Divide y vencerás – Mergesort
Ejemplo

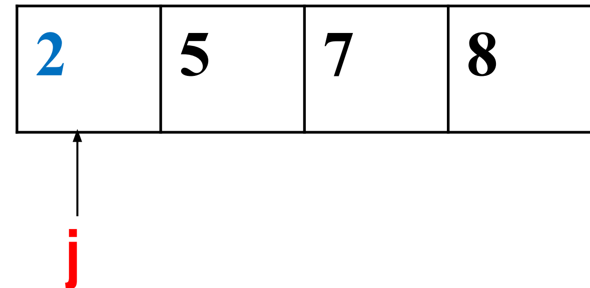


CC BY NC SA

Divide y vencerás – Mergesort
Ejemplo



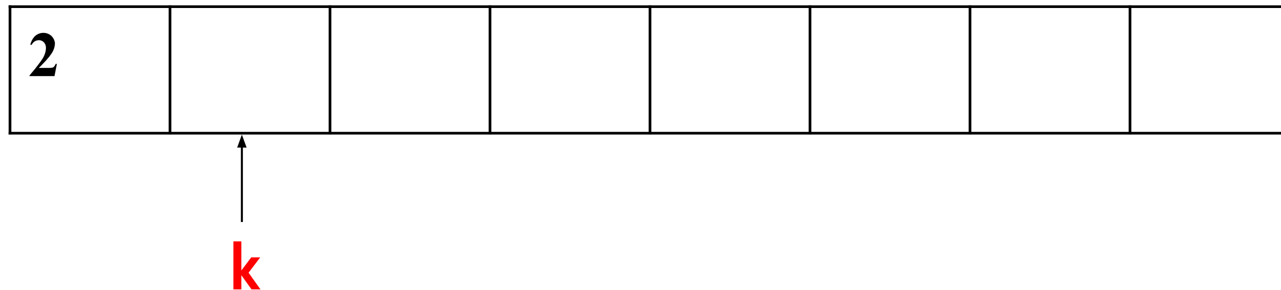


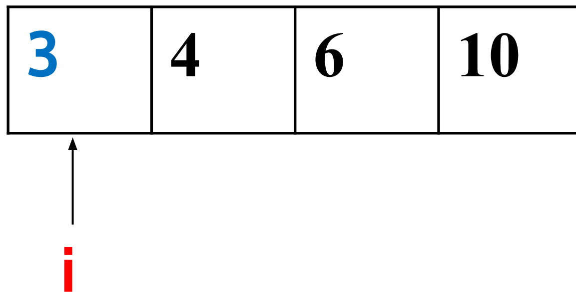


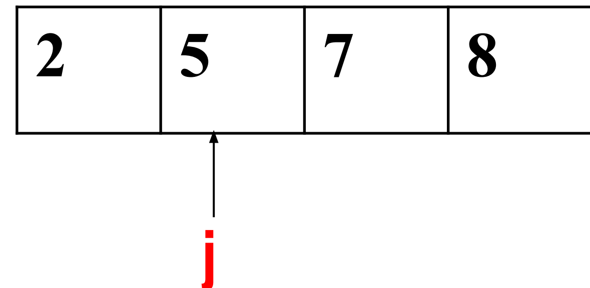
2< 3=> subo el 2 al array m[] y aumento índices j, k

cc BY NC SA

Divide y vencerás – Mergesort
Ejemplo





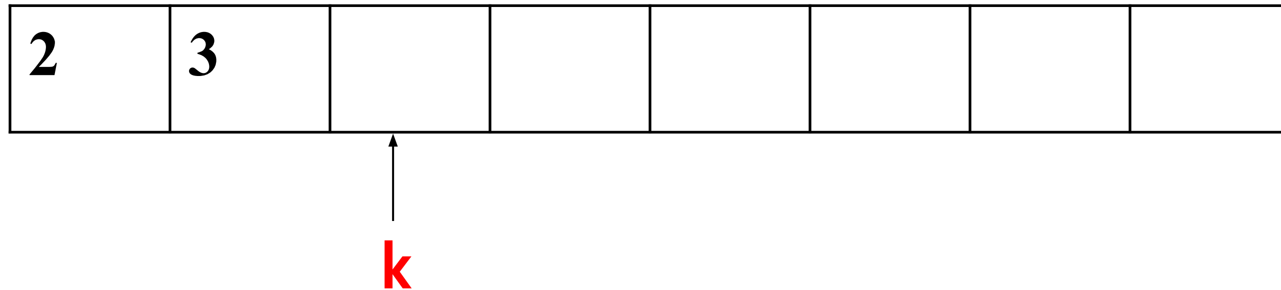


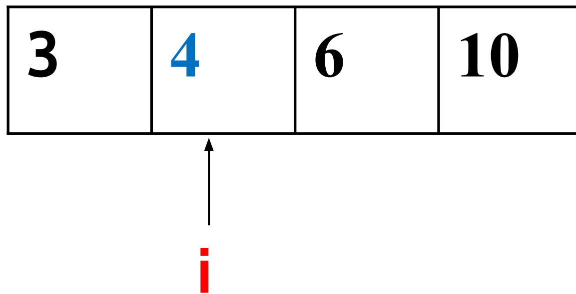
3<5 => subo el 3 al array m[] y aumento índices i, k

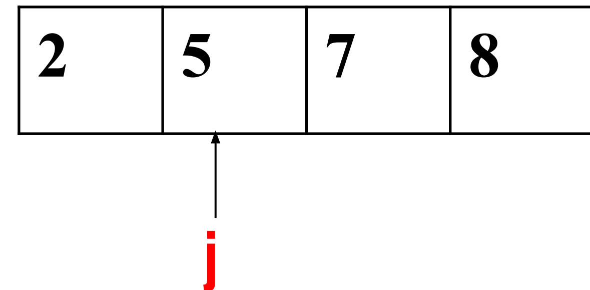
uc3m | Universidad Carlos III de Madrid

BY NC SA

Divide y vencerás – Mergesort
Ejemplo



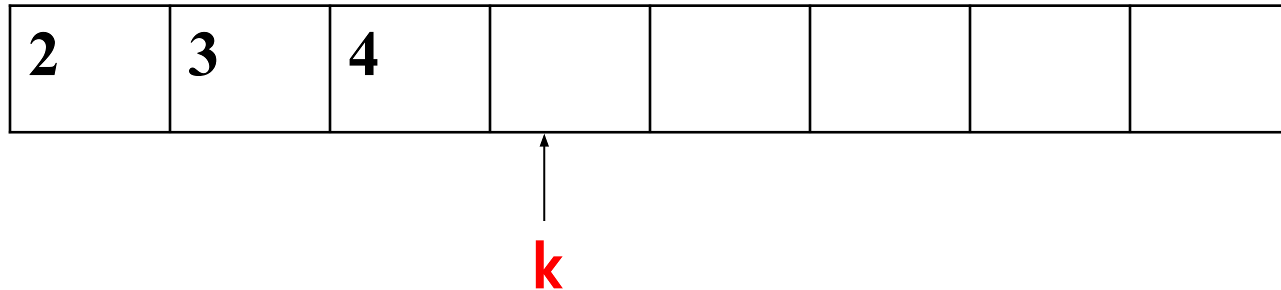


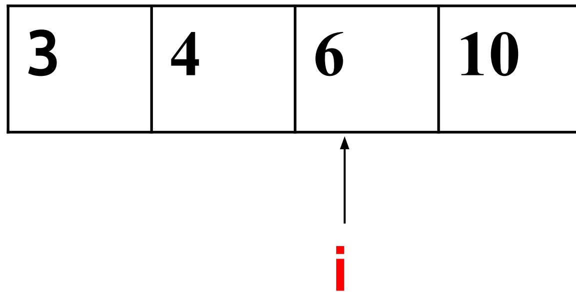


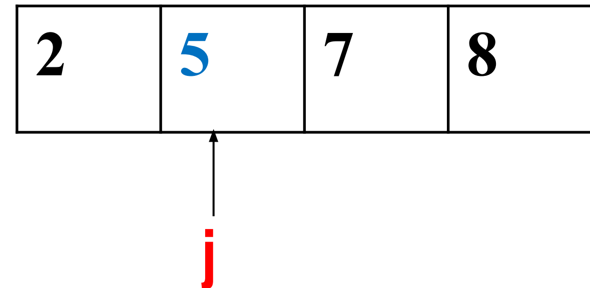
4<5 => subo el 4 al array m[] y aumento índices i, k

CC BY NC SA

Divide y vencerás – Mergesort
Ejemplo



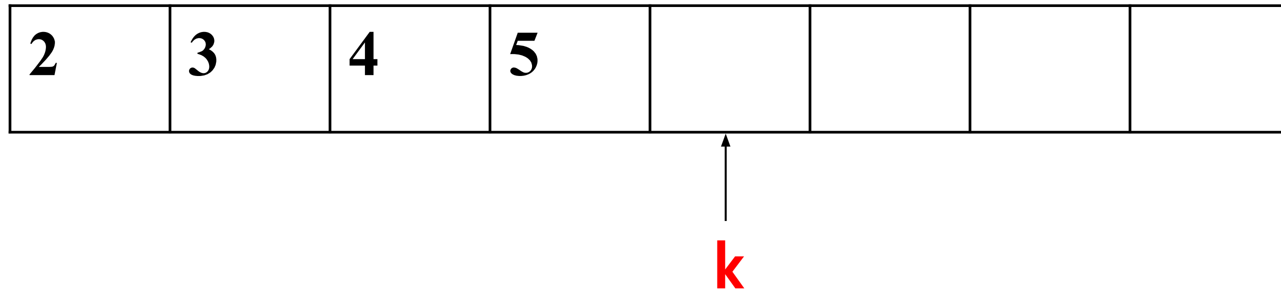


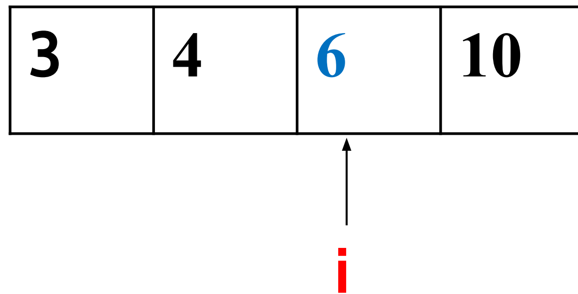


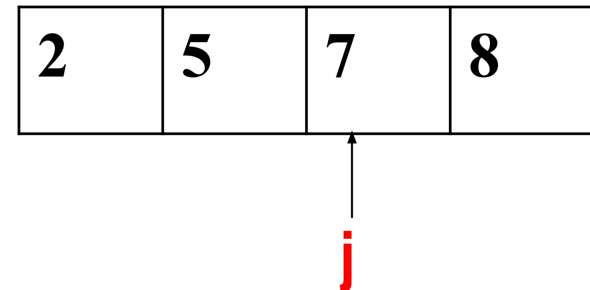
5<6 => subo el 5 al array m[] y aumento índices j, k

cc BY NC SA

Divide y vencerás – Mergesort
Ejemplo





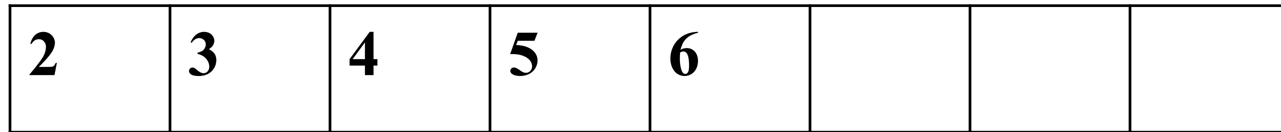


6<7 => subo el 6 al array m[] y aumento índices i, k

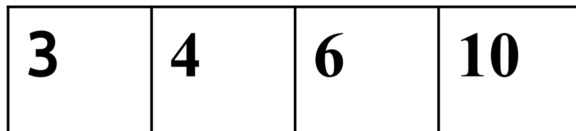
uc3m | Universidad Carlos III de Madrid

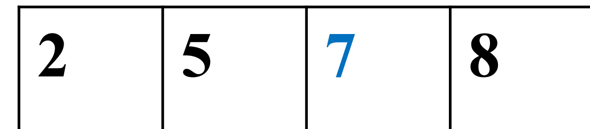
cc BY NC SA

Divide y vencerás – Mergesort
Ejemplo



k





i

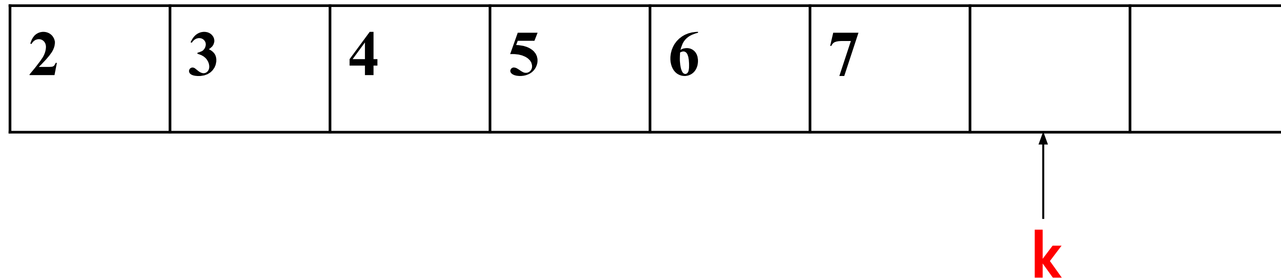
j

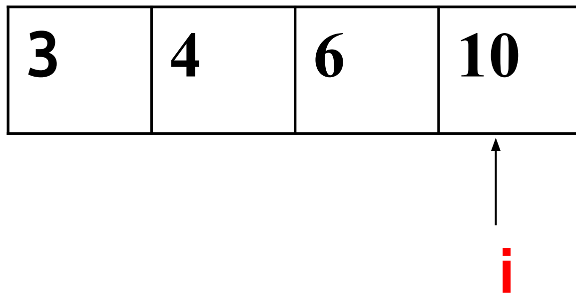
7<10 => subo el 7 al array m[] y aumento índices j, k

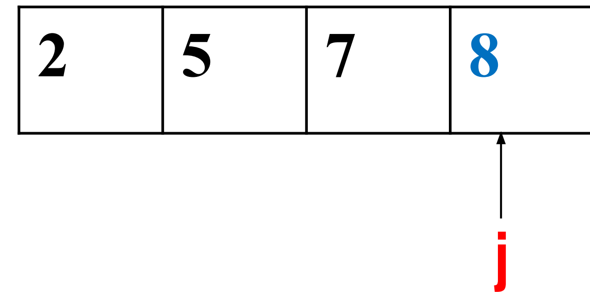
uc3m | Universidad Carlos III de Madrid

CC BY NC SA

Divide y vencerás – Mergesort
Ejemplo





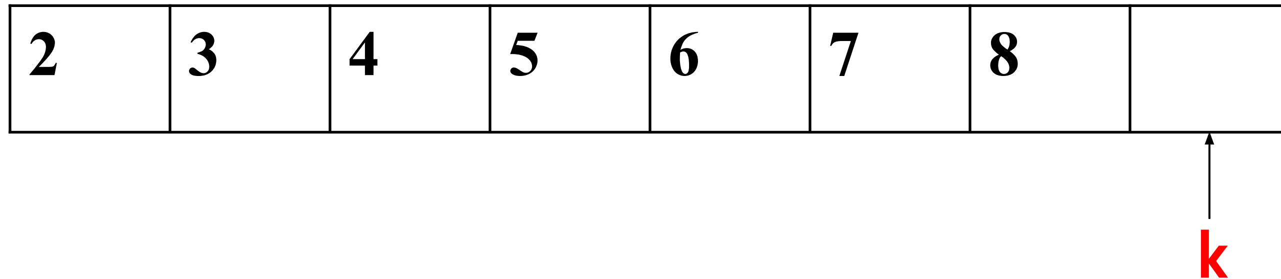


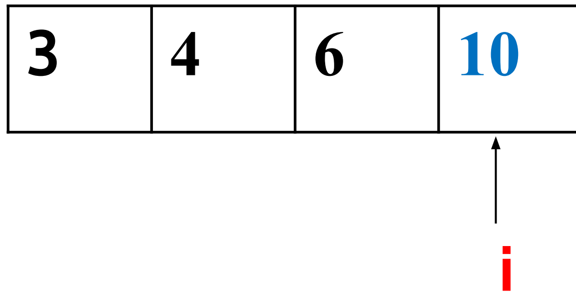
8<10 => subo el 8 al array m[] y aumento índices j, k

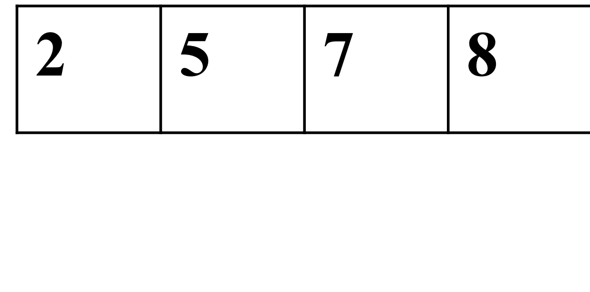
uc3m | Universidad Carlos III de Madrid

cc BY NC SA

Divide y vencerás – Mergesort Ejemplo



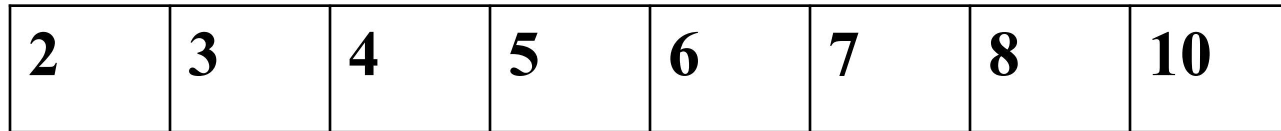


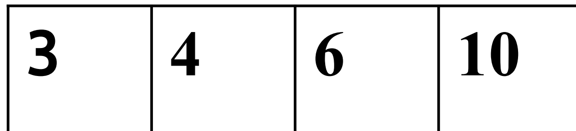


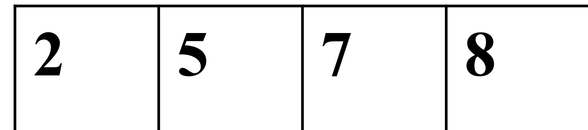
subo el 10 al array m[] y aumento índices i, k

cc BY NC SA

Divide y vencerás – Mergesort
Ejemplo







uc3m | Universidad Carlos III de Madrid
cc BY NC SA

Divide y vencerás – MergeSort Ejemplo

```
public static int[] mergesort(int a[]) {
    if (a==null) {
        System.out.println("array cannot be null!!!");
        return null;
    }
    return mergesort(a,0,a.length-1);
}

public static int[] mergesort(int a[], int start, int end) {

    if (start==end) return new int[]{a[start]};

        int middle=(start+end)/2;
        int[] m1=mergesort(a,start,middle);
        int[] m2=mergesort(a,middle+1,end);
        int m[]=merge(m1,m2);
        return m;

}
```

Divide al vector en dos partes iguales, y se realiza la llamada recursiva a un método merge que seguirá dividiendo y mezcla

cc BY NC SA

Divide y vencerás - Mergesort

Obtiene un array m[] ordenado a partir de dos sub-array ordenados a[] y b[]

```java
//merges the arrays
public static int[] merge(int a[],int b[]) {
    if (a==null || b==null) {
        System.out.println("arrays cannot be null!!!");
        return null;
    }
    int m[]=new int[a.length+b.length];
    int k=0;
    int i=0;
    int j=0;
    while (i<a.length && j<b.length) {
        if (a[i]<b[j]) {
            m[k]=a[i];
            i++;
        } else {
            m[k]=b[j];
            j++;
        }
        k++;
    }
    while (i<a.length) {
        m[k]=a[i];
        i++;
        k++;
    }
    while (j<b.length) {
        m[k]=b[j];
        j++;
        k++;
    }
    return m;
}
```

mezcla

Ya se ha recorrido b,[] añadimos a m[], los elementos de a[] que están ordenados

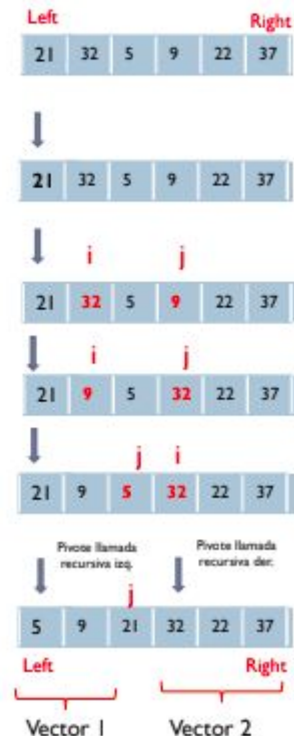
Ya se ha recorrido a[] añadimos a m[] los elementos de b[] que están ordenados

Divide y vencerás – Quicksort

- Es el algoritmo de ordenación más rápido.
- Se basa en la técnica **divide y vencerás.** Consiste en dividir el array en arrays más pequeños, y ordenar éstos.
  - Se toma un valor del array como **pivote**, y se mueven todos los elementos menores que este pivote a su izquierda, y los mayores a su derecha.
  - A continuación se aplica el mismo método a cada una de las dos partes en las que queda dividido el array.
- Después de elegir el pivote se realizan dos búsquedas:
  - Izquierda a derecha, buscando un elemento mayor que el pivote
  - Derecha a izquierda, buscando un elemento menor que el pivote.
- Cuando se han encontrado los dos elementos anteriores, se intercambian, y se sigue realizando la búsqueda hasta que las dos búsquedas se encuentran.

uc3m | Universidad Carlos III de Madrid
cc BY NC SA

Divide y vencerás – Quicksort
Ejemplo 1



En este caso, hemos decidido tomar como pivote el primer elemento (es posible probar con otros pivotes)

La búsqueda de izquierda a derecha encuentra el valor 32, mayor que el pivote y la búsqueda de derecha a izquierda encuentra el valor 9, menor que el pivote

Se intercambian

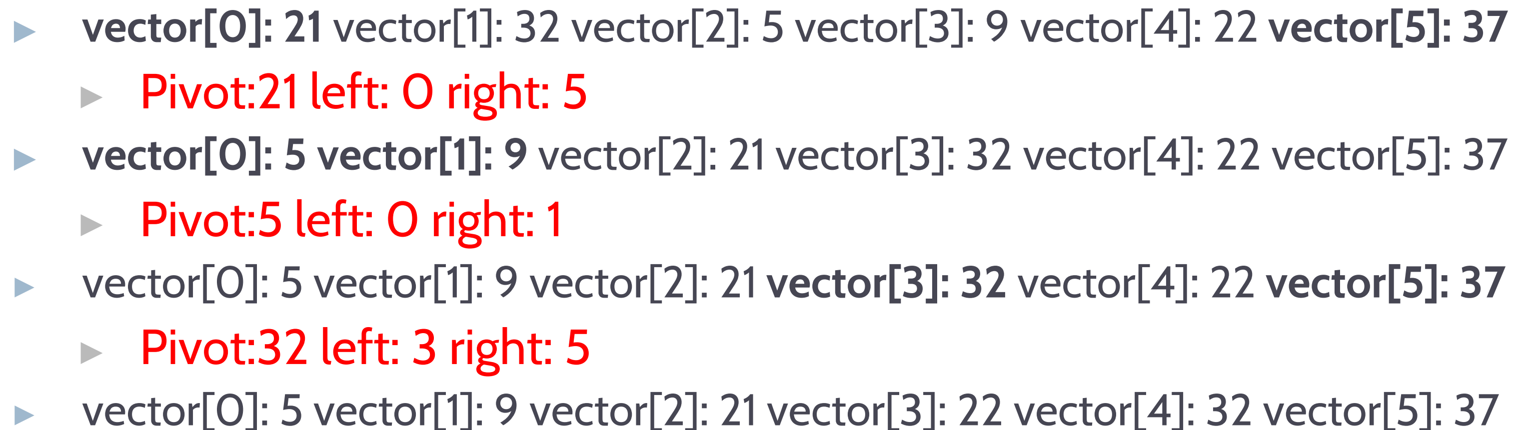
Continúa la búsqueda, se encuentra el valor 32 mayor que el pivote y el valor 5 menor que el pivote, pero ya se han cruzado. Terminar.

Por último, se intercambia el valor de la posición pivote con la posición j. Se continúa aplicando el mismo proceso de forma recursiva dividiendo en 2 el Vector (Left hasta j-1 y j+1 hasta Right).

cc BY NC SA

Divide y vencerás – Quicksort
Ejemplo 1

ARRAY DESORDENADO

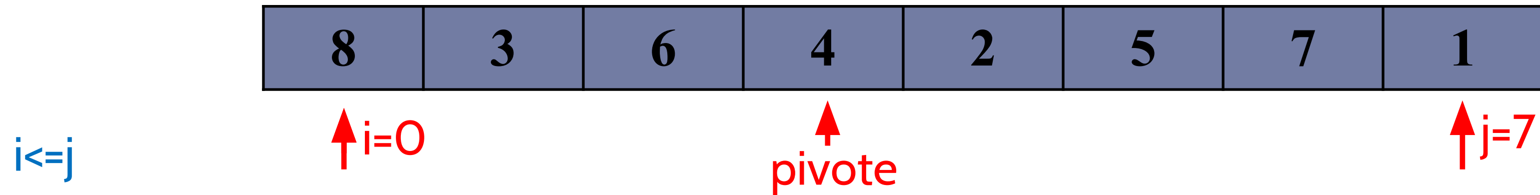


- **vector[0]: 21** vector[1]: 32 vector[2]: 5 vector[3]: 9 vector[4]: 22 **vector[5]: 37**
  - Pivot:21 left: 0 right: 5
- **vector[0]: 5 vector[1]: 9** vector[2]: 21 vector[3]: 32 vector[4]: 22 vector[5]: 37
  - Pivot:5 left: 0 right: 1
- vector[0]: 5 vector[1]: 9 vector[2]: 21 **vector[3]: 32** vector[4]: 22 **vector[5]: 37**
  - Pivot:32 left: 3 right: 5
- vector[0]: 5 vector[1]: 9 vector[2]: 21 vector[3]: 22 vector[4]: 32 vector[5]: 37

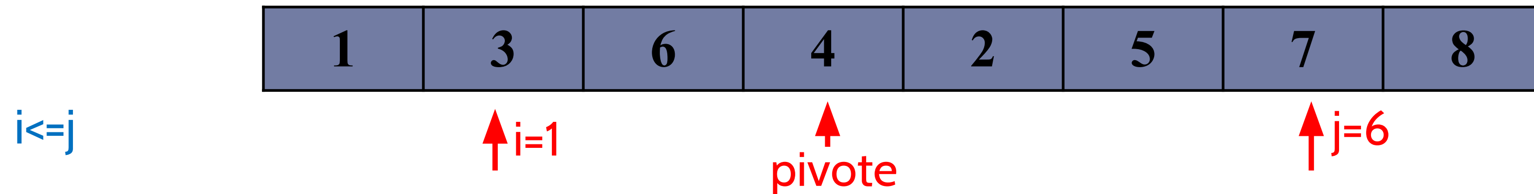
ARRAY ORDENADO

cc BY NC SA

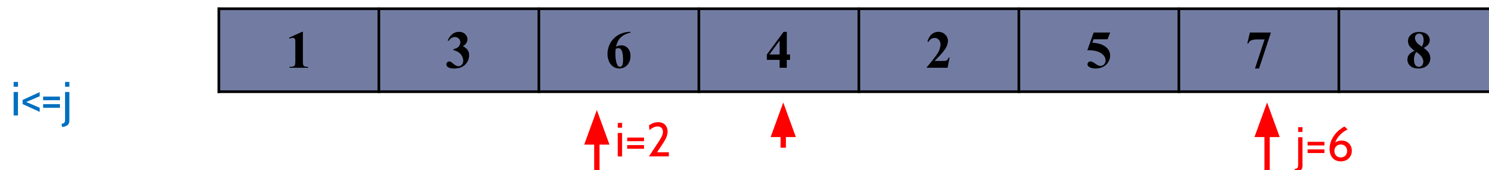
Divide y vencerás – Quicksort
Ejemplo 2



8 no es menor 4, y 1 no es mayor que 4 => intercambio e incremento índices i,j



3 es menor 4,=> incremento i



6 no es menor 4 (no incremento i)
7 es mayor que 4=>decremento j

uc3m | Universidad Carlos III de Madrid
cc BY NC SA

Divide y vencerás – Quicksort
Ejemplo 2

1	3	6	4	2	5	7	8

i<=j

i=2 pivote j=5

5 es mayor que 4=>decremento j

1	3	6	4	2	5	7	8

i<=j

i=2 pivote j=4

2 no es mayor que 4=>intercambio 6 y 2, e incremento i y decremento j

1	3	2	4	6	5	7	8

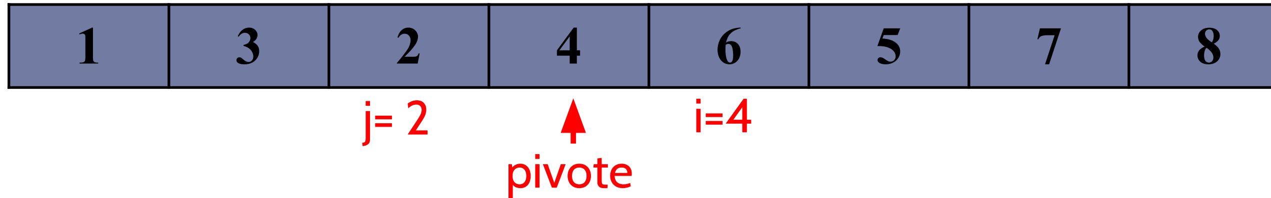
i<=j

i= 3 pivote j=3

4 no es menor ni mayor que 4=>intercambio 4 y 4, e incremento i y decremento j

CC BY NC SA

Divide y vencerás – Quicksort
Ejemplo 2



i>j => llamada recursiva a array [0,j] y [i, 7]

cc BY NC SA

Divide y vencerás – Quicksort

```java
public static void quicksort(int a[]) {
    doPartition(a,0,a.length-1);


}
```

```java
    public static void doPartition(int a[],int start, int end) {
        if (a==null) {
            System.out.println("array cannot be null!!!");
            return;
        }
        if (start>=end) return;
        int middle = start + (end- start) / 2;
        int pivote = a[middle];
        System.out.println("pivote:"+pivote);
        int i=start;
        int j=end;

        while (i<=j) {

            while (a[i]<pivote) i++;
            while (a[j]>pivote) j--;

            if (i<=j) {
                int x=a[i];
                a[i]=a[j];
                a[j]=x;
                i++;
                j--;
            }
        }
        if (start<j) doPartition(a,start,j);
        if (i<end) doPartition(a,i,end);
    }
}
```


uc3m | Universidad Carlos III de Madrid
CC BY NC SA

Resumen

- Se divide el problema original en sub-problemas
  - Recursivamente, cada sub-problema se divide de nuevo
- Cuando el caso a resolver es lo suficientemente sencillo se resuelve utilizando un algoritmo directo (no recursivo)
  - El algoritmo directo debe ser eficiente para problemas sencillos
  - No importa que no lo sea para problemas grandes
- Cada sub-problema se resuelve de forma independiente
- Finalmente se combinan las soluciones de todos los sub-problemas para formar la solución del problema original

cc BY NC SA

uc3m | Universidad Carlos III de Madrid