Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

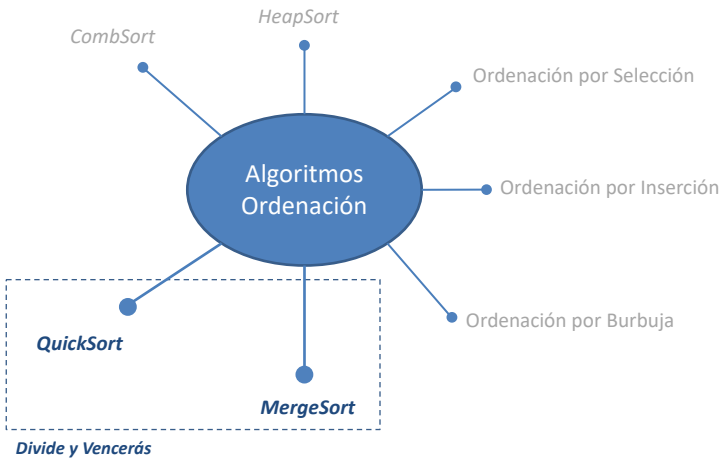
Tema 6. Algoritmos de Ordenación basados en el Esquema Divide y Vencerás

Algorítmica y Complejidad

1

QuickSort

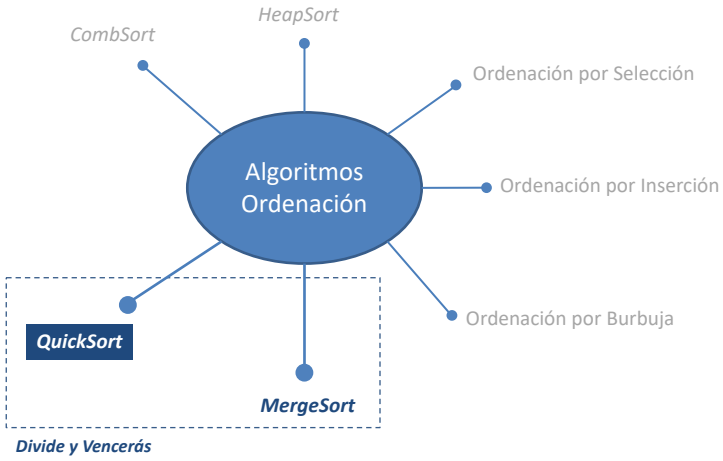
Algoritmos de Ordenación basados en Divide y Vencerás



2

QuickSort

Algoritmos de Ordenación basados en Divide y Vencerás



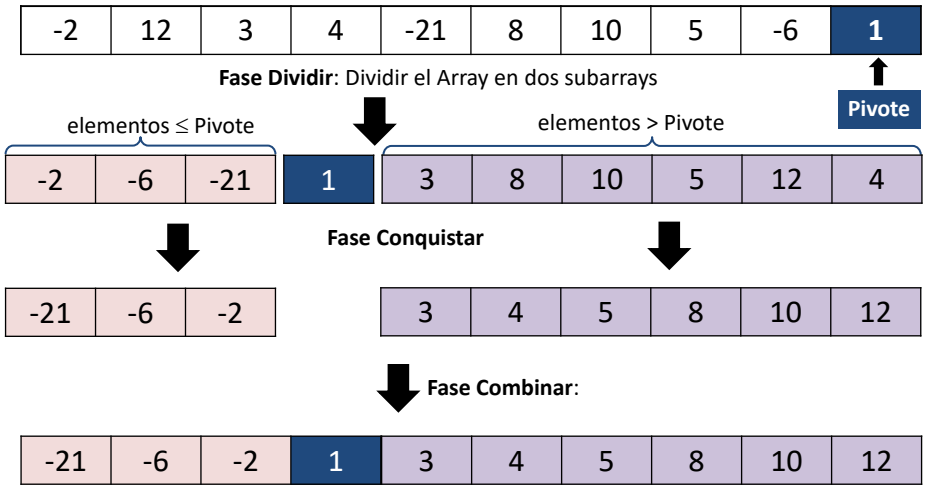
1. QuickSort

1 2

3

QuickSort

Algoritmo basado en Divide y Vencerás



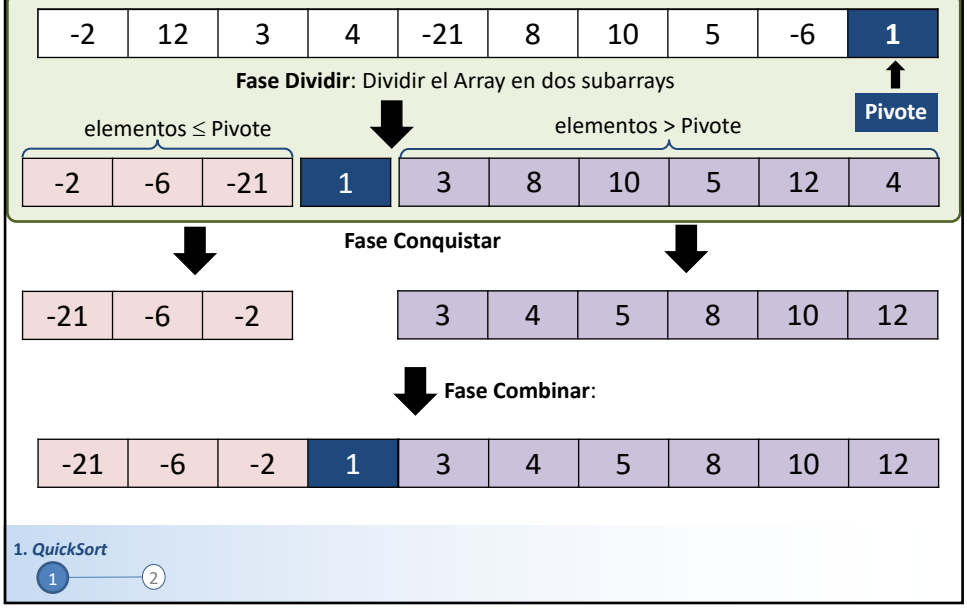
1. QuickSort

1 2

4

QuickSort

Algoritmo basado en Divide y Vencerás



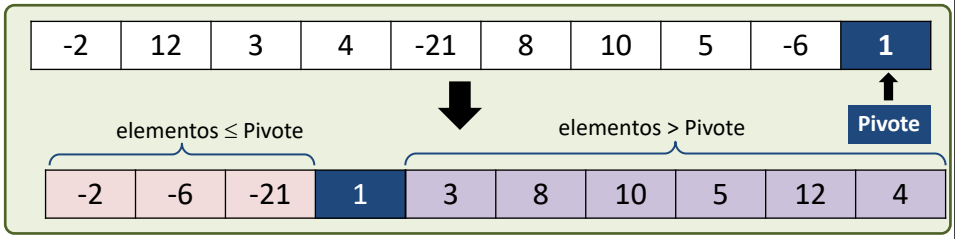
1. *QuickSort*

1 2

5

QuickSort

Algoritmo basado en Divide y Vencerás



Actividad 6.1. Diseña e implementa un algoritmo tal que dado un vector, lo ordene de manera que todos los elementos menores del pivote queden a la izquierda de éste, y los elementos mayores del pivote queden a la derecha de éste.

Considera que el pivote es el elemento más a la derecha del *array*

1. *QuickSort*

1 2

6

QuickSort

ordenarPorPivote

-2	12	3	4	-21	8	10	5	-6	1

```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

1. QuickSort

1 — 2

7

QuickSort

ordenarPorPivote

i:0 j:8

-2	12	3	4	-21	8	10	5	-6	1

```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
➡     int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

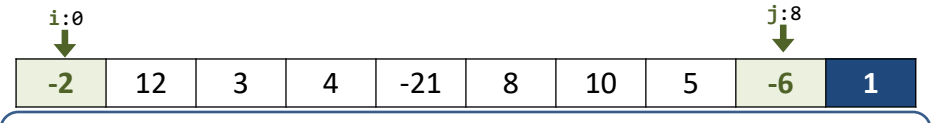
1. QuickSort

1 — 2

8

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

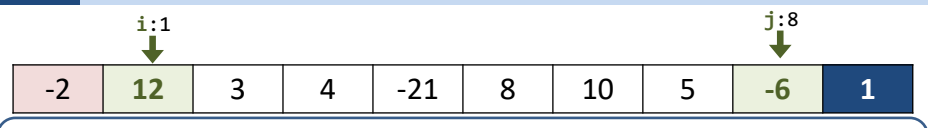
1. QuickSort

1 — 2

9

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

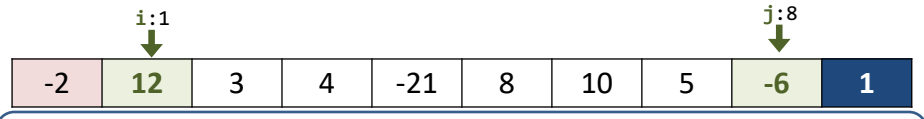
1. QuickSort

1 — 2

10

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
➡        int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

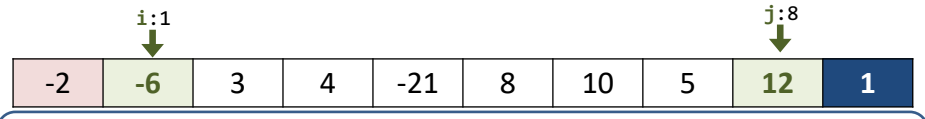
1. QuickSort

1 — 2

11

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
➡        while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

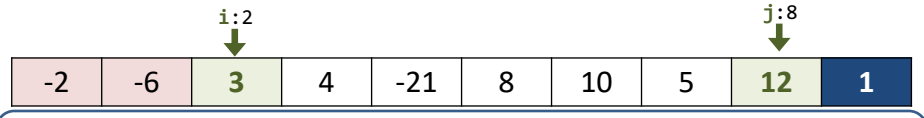
1. QuickSort

1 — 2

12

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
➡        while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

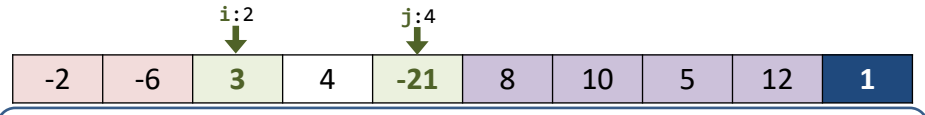
1. QuickSort

1 — 2

13

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
➡        int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

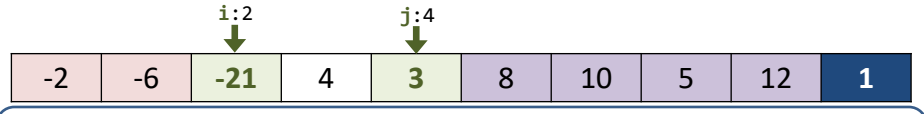
1. QuickSort

1 — 2

14

QuickSort

ordenarPorPivote



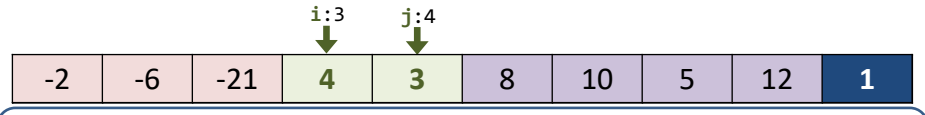

```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

1. QuickSort

15

QuickSort

ordenarPorPivote



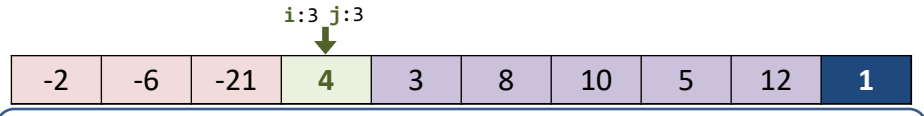

```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

1. QuickSort

16

QuickSort

ordenarPorPivote



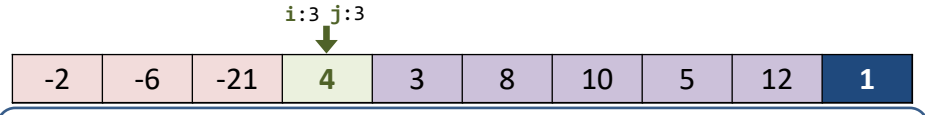

```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
➡        int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

1. QuickSort

17

QuickSort

ordenarPorPivote



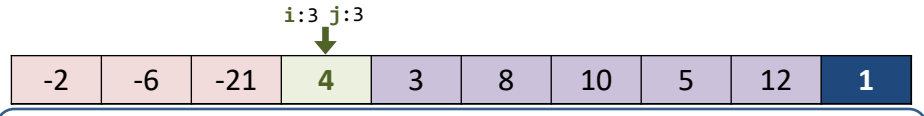

```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
➡     while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

1. QuickSort

18

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
➡     if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

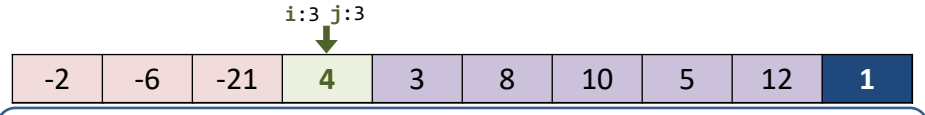
1. QuickSort

1 — 2

19

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
➡        vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

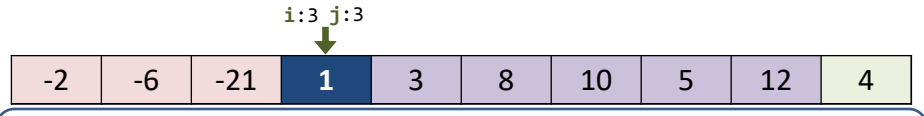
1. QuickSort

1 — 2

20

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

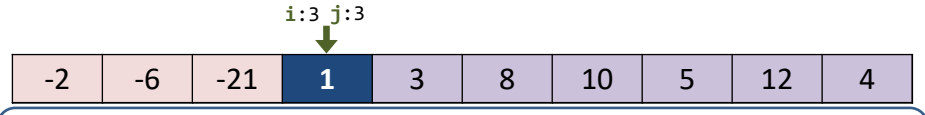
1. QuickSort

1 — 2

21

QuickSort

ordenarPorPivote




```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

1. QuickSort

1 — 2

22

QuickSort

Complejidad de ordenarPorPivote

Actividad 6.2. Calcula la complejidad del algoritmo en función del tamaño del vector

```
int ordenarPorPivote (int[] vector, int i0, int iN){
      int pivote = vector[iN];
      int i=i0;
      int j=iN-1;
      while (i<j) {
         while (vector[i]<= pivote && i<j) {i++;}
         while (vector[j]> pivote && i<j) {j--;}
         int aux = vector[i]; vector[i] = vector[j]; vector[j] = aux;
      }
      if (vector[i] > pivote) {
         vector[iN] = vector[i]; vector[i] = pivote;
         return i;
      }
      else
         return iN;
}
```

Complejidad: $\Theta(N)$

N-1 Iteraciones

1. *QuickSort*

1 — 2

23

QuickSort

ordenarPorPivoteCentral

-2	12	1	4	**3**	-21	10	5	-6	8

Pivote

elementos ≤ Pivote | elementos > Pivote

-2	-6	1	-21	**3**	4	10	5	12	8

Actividad 6.3. Implementar el método anterior considerando ahora que el pivote es el elemento situado en el centro del *array*

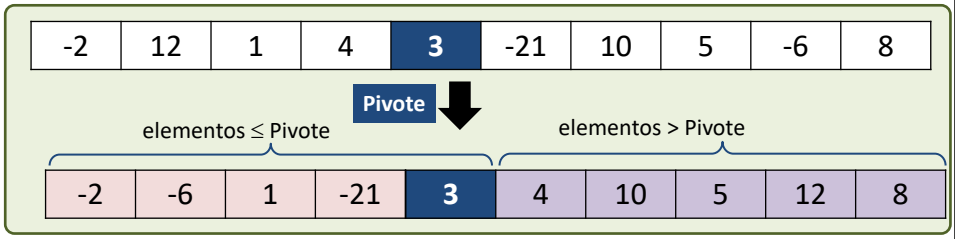
1. *QuickSort*

1 — 2

24

QuickSort

ordenarPorPivoteCentral



Actividad 6.3. **Implementar el método anterior considerando ahora que el pivote es el elemento situado en el centro del *array***

```
int ordenarPorPivoteCentral (int[] vector, int i0, int iN){
    int i = (i0 + iN) / 2;
    int aux = vector[iN]; vector[iN] = vector[i]; vector[i] = aux;
    return ordenarPorPivote(vector, i0, iN);
}
```

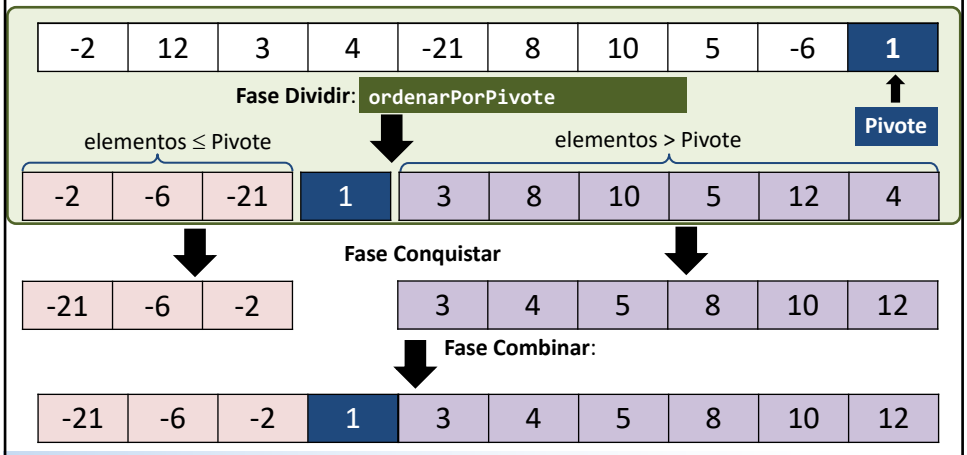
1. *QuickSort*

1 2

25

QuickSort

Implementación



1. *QuickSort*

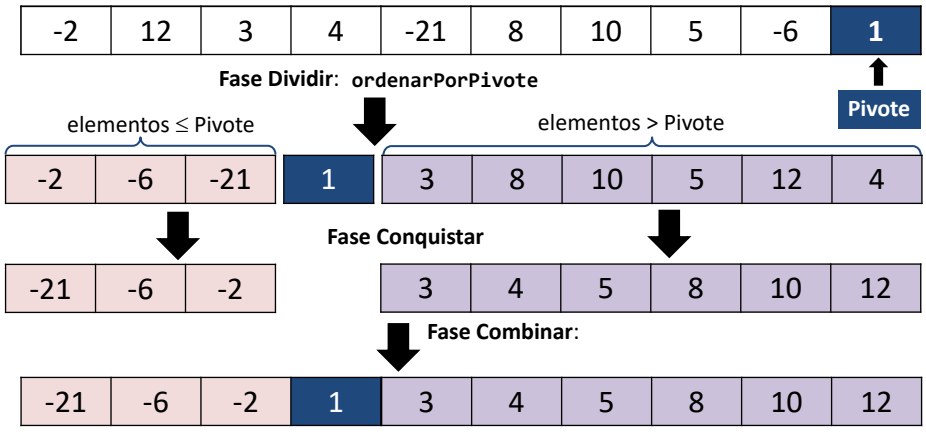
1 2

26

QuickSort

Implementación

Actividad 6.4. Implementa el algoritmo QuickSort



1. QuickSort

QuickSort

Implementación

Actividad 6.4. Implementa el algoritmo QuickSort

x_0	...	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	...	x_{50}

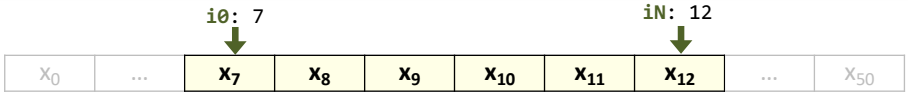
```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
void quickSort(int[] vector) {
      quickSortAux(vector,0, vector.length-1);
}
```

1. QuickSort

QuickSort

Implementación

Actividad 6.4. Implementa el algoritmo QuickSort




```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
void quickSort(int[] vector) {
      quickSortAux(vector,0, vector.length-1);
}
```

1. QuickSort

QuickSort

Implementación

Actividad 6.4. Implementa el algoritmo QuickSort

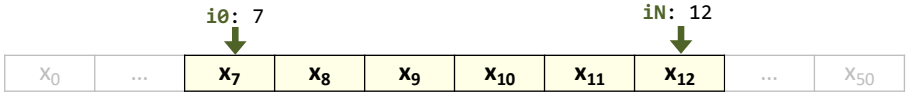

```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0,iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
void quickSort(int[] vector) {
      quickSortAux(vector,0, vector.length-1);
}
```

1. QuickSort

QuickSort

Implementación

Actividad 6.4. Implementa el algoritmo QuickSort




```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
void quickSort(int[] vector) {
      quickSortAux(vector,0, vector.length-1);
}
```

1. QuickSort

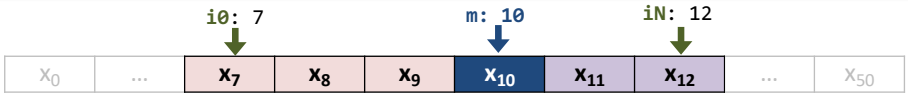
1 — 2

31

QuickSort

Implementación

Actividad 6.4. Implementa el algoritmo QuickSort




```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
void quickSort(int[] vector) {
      quickSortAux(vector,0, vector.length-1);
}
```

Dividir

1. QuickSort

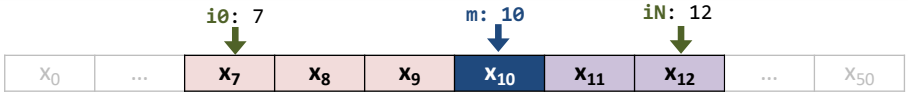
1 — 2

32

QuickSort

Implementación

Actividad 6.4. Implementa el algoritmo QuickSort




```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
void quickSort(int[] vector) {
      return quickSortAux(vector,0, vector.length-1);
}
```

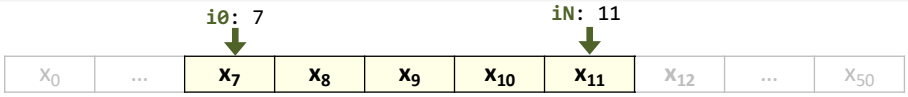
Conquistar

33

QuickSort

Complejidad

Actividad 6.5. Calcula la complejidad en el **mejor** caso




```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
```


34

QuickSort

Complejidad

Actividad 6.5. Calcula la complejidad en el **mejor** caso m=(iN+i0)/2


```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
```

1. QuickSort

35

QuickSort

Complejidad

Actividad 6.5. Calcula la complejidad en el **mejor** caso m=(iN+i0)/2


```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
```

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 2 \cdot T\left(\frac{N}{2}\right) + \Theta(N) & N > 1 \end{cases}$$

1. QuickSort

36

QuickSort

Complejidad

Actividad 6.5. Calcula la complejidad en el **mejor** caso — m=(iN+i0)/2


```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
          int m = ordenarPorPivote(vector, i0, iN);
          quickSortAux(vector, i0, m-1);
          quickSortAux(vector, m+1, iN);
      }
}
```

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 2 \cdot T\left(\frac{N}{2}\right) + \Theta(N^1) & N > 1 \end{cases}$$

➡ **Teorema Maestro**

(2º Caso)
$\text{Log}_2(2)=1$

1. QuickSort

1 — 2

37

QuickSort

Complejidad

Actividad 6.5. Calcula la complejidad en el **mejor** caso — m=(iN+i0)/2


```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
          int m = ordenarPorPivote(vector, i0, iN);
          quickSortAux(vector, i0, m-1);
          quickSortAux(vector, m+1, iN);
      }
}
```

Complejidad $\Theta(N \cdot \log N)$

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 2 \cdot T\left(\frac{N}{2}\right) + \Theta(N) & N > 1 \end{cases}$$

➡ **Teorema Maestro**

(2º Caso)
$\text{Log}_2(2)=1$

1. QuickSort

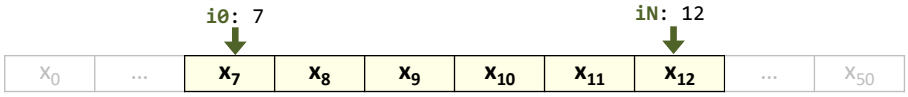
1 — 2

38

QuickSort

Complejidad

Actividad 6.6. Calcula la complejidad en el peor caso



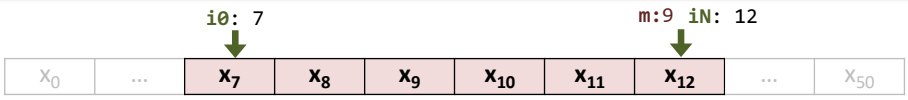

```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
void quickSort(int[] vector) {
      quickSortAux(vector,0, vector.length-1);
}
```


39

QuickSort

Complejidad

Actividad 6.6. Calcula la complejidad en el peor caso **m=iN**




```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
void quickSort(int[] vector) {
      quickSortAux(vector,0, vector.length-1);
}
```

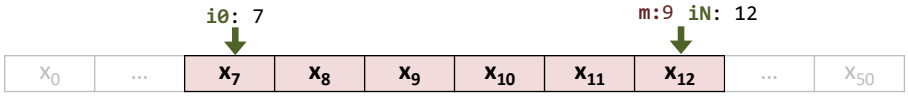

40

QuickSort

Complejidad

Actividad 6.6. Calcula la complejidad en el peor caso

m=i0




```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
```

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ T(N-1) + \Theta(N) & N > 1 \end{cases}$$

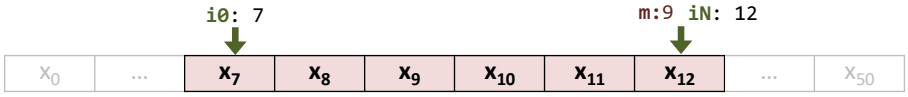
41

QuickSort

Complejidad

Actividad 6.6. Calcula la complejidad en el peor caso

m=i0




```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0, iN);
         quickSortAux(vector, i0, m-1);
         quickSortAux(vector, m+1, iN);
      }
}
```

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ T(N-1) + \Theta(N) & N > 1 \end{cases}$$

Complejidad $\Theta(N^2)$

42

QuickSort

Complejidad

Caso peor	Caso medio	Caso mejor
$\Theta(N^2)$	$\Theta(N \cdot \log N)$	$\Theta(N \cdot \log N)$

```
void quickSortAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return;
      else {
         int m = ordenarPorPivote(vector, i0,iN);
         quickSortAux(vector, m+1, iN, k);
         quickSortAux(vector, i0, m, k-(iN-m+1));
      }
}
void quickSort(int[] vector) {
      quickSortAux(vector,0, vector.length-1);
}
```

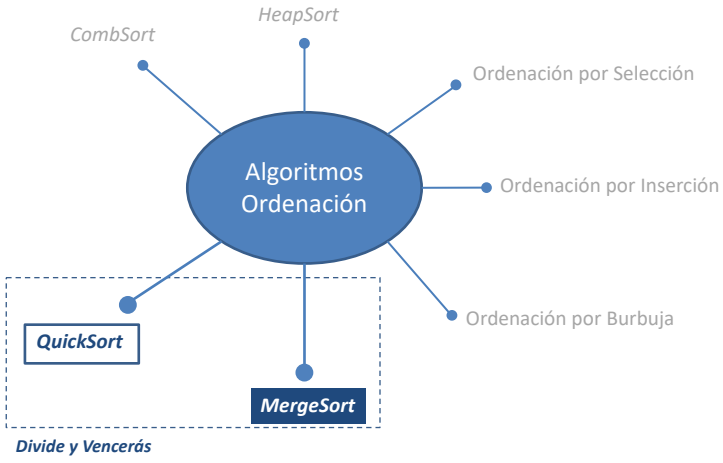
1. *QuickSort*

1 — 2

43

MergeSort

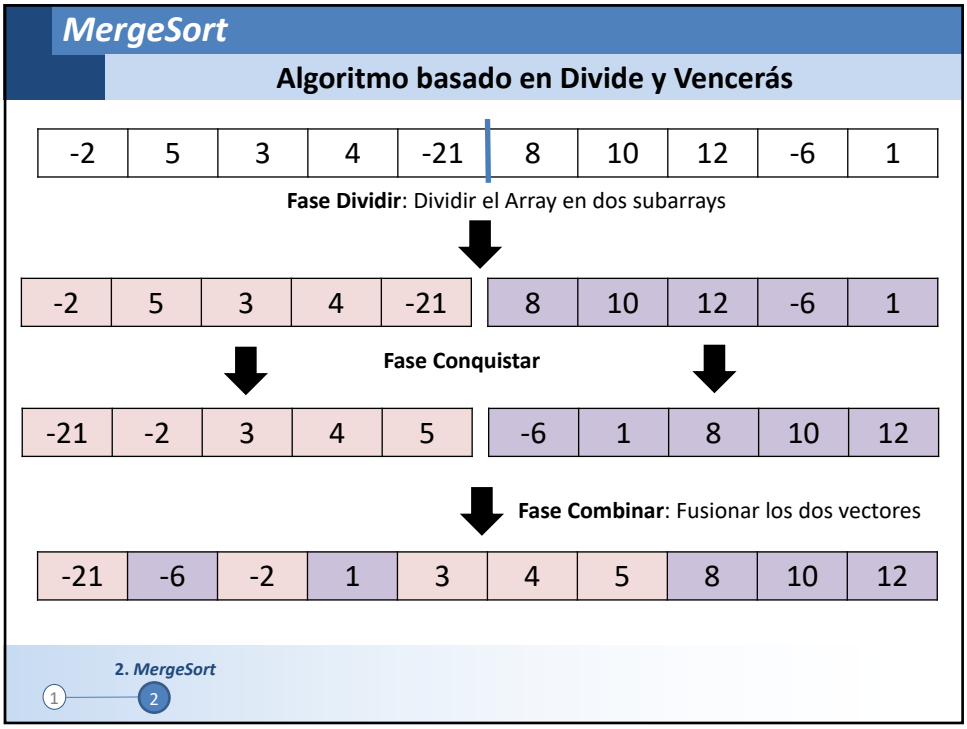
Algoritmos de Ordenación basados en Divide y Vencerás



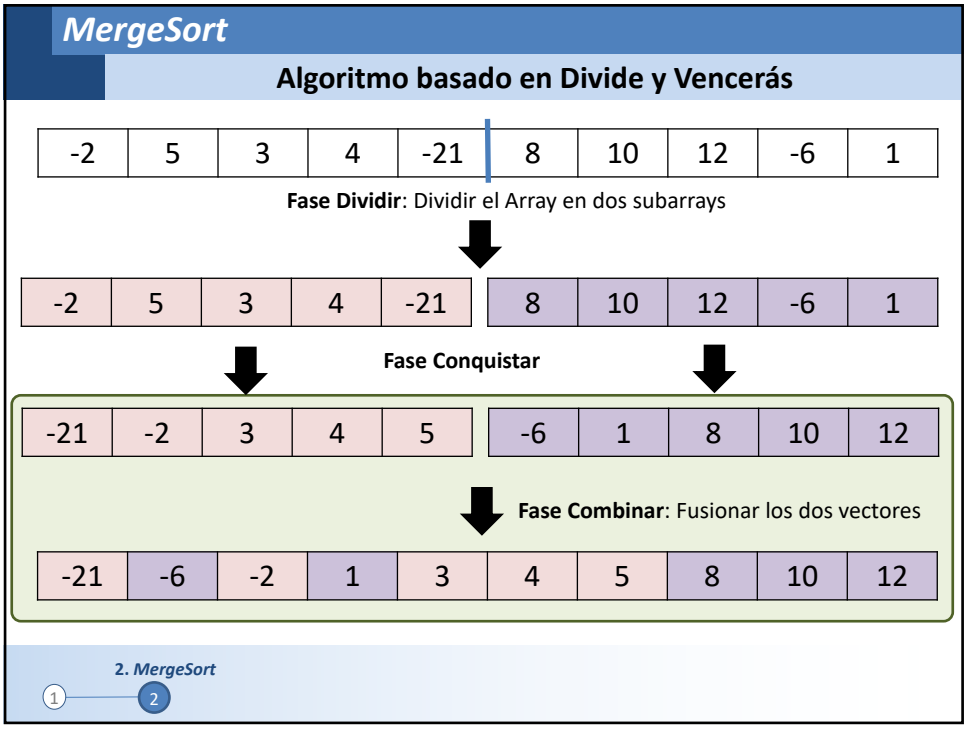
2. *MergeSort*

1 — 2

44


MergeSort
Algoritmo basado en Divide y Vencerás
-2 5 3 4 -21 8 10 12 -6 1
Fase Dividir: Dividir el Array en dos subarrays
-2 5 3 4 -21 8 10 12 -6 1
Fase Conquistar
-21 -2 3 4 5 -6 1 8 10 12
Fase Combinar: Fusionar los dos vectores
-21 -6 -2 1 3 4 5 8 10 12
2. MergeSort
1
2

45

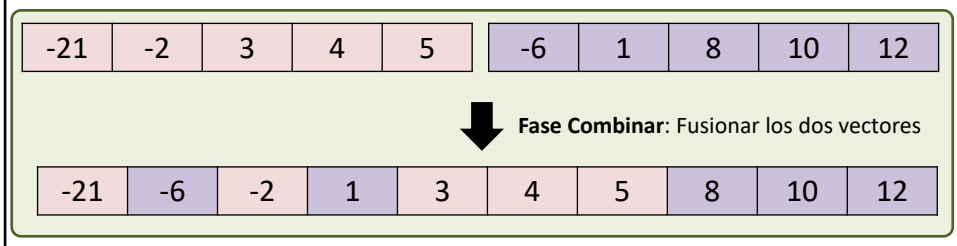

MergeSort
Algoritmo basado en Divide y Vencerás
-2 5 3 4 -21 8 10 12 -6 1
Fase Dividir: Dividir el Array en dos subarrays
-2 5 3 4 -21 8 10 12 -6 1
Fase Conquistar
-21 -2 3 4 5 -6 1 8 10 12
Fase Combinar: Fusionar los dos vectores
-21 -6 -2 1 3 4 5 8 10 12
2. MergeSort
1
2

46

MergeSort

Algoritmo basado en Divide y Vencerás

Actividad 6.7. Diseña e implementa un algoritmo tal que dado dos vectores ordenados, genere un vector ordenado que contenga los elementos de esos vectores.

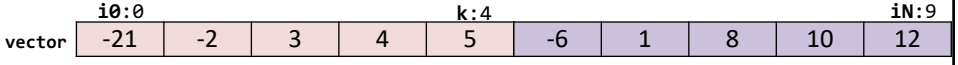


47

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```



48

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

	i:0				k:4	d:5				iN:9
vector	-21	-2	3	4	5	-6	1	8	10	12
aux										

f:0

2. MergeSort

1 2

49

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

	i:0				k:4	d:5				iN:9
vector	-21	-2	3	4	5	-6	1	8	10	12
aux										

f:0

2. MergeSort

1 2

50

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

	i:0				k:4	d:5				iN:9
vector	-21	-2	3	4	5	-6	1	8	10	12
aux										

f:0

2. MergeSort

1 2

51

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

	i:0				k:4	d:5				iN:9
vector	-21	-2	3	4	5	-6	1	8	10	12
aux										

f:0

2. MergeSort

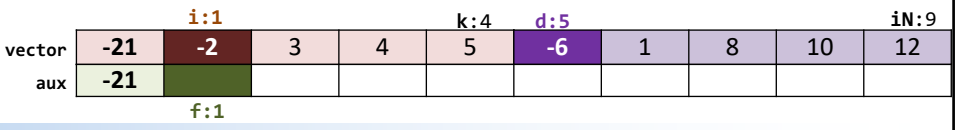
1 2

52

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```



2. MergeSort

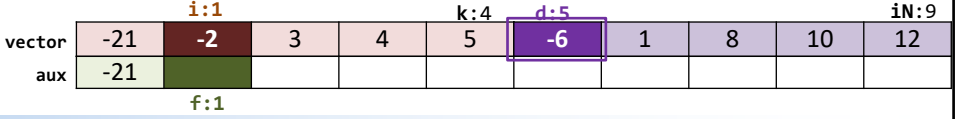
1 2

53

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```



2. MergeSort

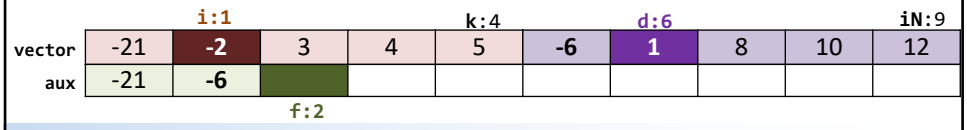
1 2

54

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```



2. MergeSort

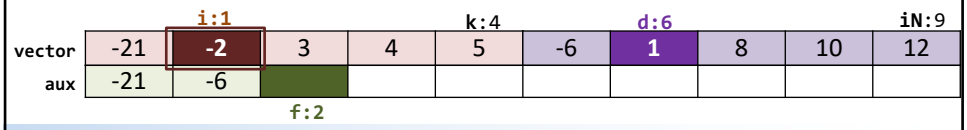
1 2

55

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```



2. MergeSort

1 2

56

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

			i:2		k:4		d:6			iN:9
vector	-21	-2	3	4	5	-6	1	8	10	12
aux	-21	-6	-2							
				f:3						

2. MergeSort

1 2

57

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

			i:2		k:4		d:6			iN:9
vector	-21	-2	3	4	5	-6	1	8	10	12
aux	-21	-6	-2							
				f:3						

2. MergeSort

1 2

58

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

			i:2		k:4			d:7		iN:9
vector	-21	-2	**3**	4	5	-6	**1**	**8**	10	12
aux	-21	-6	-2	**1**						

f:4

2. MergeSort

1 2

59

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

			i:2		k:4			d:7		iN:9
vector	-21	-2	**3**	4	5	-6	1	**8**	10	12
aux	-21	-6	-2	1						

f:4

2. MergeSort

1 2

60

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

				i:3	k:4			d:7	iN:9
vector	-21	-2	**3**	**4**	5	-6	1	**8**	10	12
aux	-21	-6	-2	1	**3**					

f:5

2. MergeSort

1 — 2

61

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

				i:3	k:4			d:7	iN:9
vector	-21	-2	3	**4**	5	-6	1	**8**	10	12
aux	-21	-6	-2	1	3					

f:5

2. MergeSort

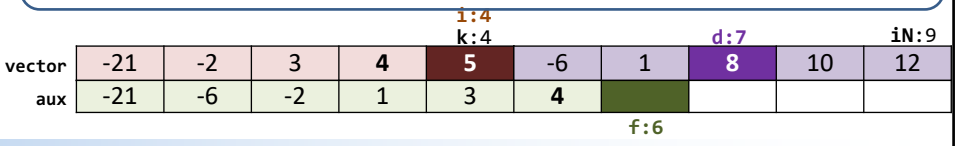
1 — 2

62

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```



2. MergeSort

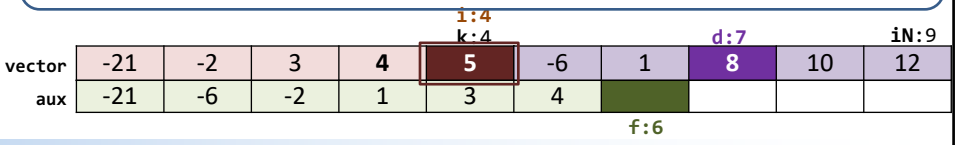
1 2

63

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```



2. MergeSort

1 2

64

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
➡     while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

					k:4	i:5		d:7		iN:9
vector	-21	-2	3	4	**5**	-6	1	**8**	10	12
aux	-21	-6	-2	1	3	4	**5**			
								f:7		

2. MergeSort

1 — 2

65

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
➡     for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

					k:4	i:5		d:7		iN:9
vector	-21	-2	3	4	5	-6	1	**8**	10	12
aux	-21	-6	-2	1	3	4	5			
								f:7		

2. MergeSort

1 — 2

66

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
➡     for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

					k:4			d:7		iN:9
vector	-21	-2	3	4	5	-6	1	8	10	12
aux	-21	-6	-2	1	3	4	5			
								f:7		

2. MergeSort

1 — 2

67

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
➡     for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

					k:4					iN:9
vector	-21	-2	3	4	5	-6	1	8	10	12
aux	-21	-6	-2	1	3	4	5	8	10	12

2. MergeSort

1 — 2

68

MergeSort

merge

```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

					k:4					iN:9
vector	-21	-6	-2	1	3	4	1	8	10	12
aux	-21	-6	-2	1	3	4	5	8	10	12
								f:7		

69

MergeSort

merge

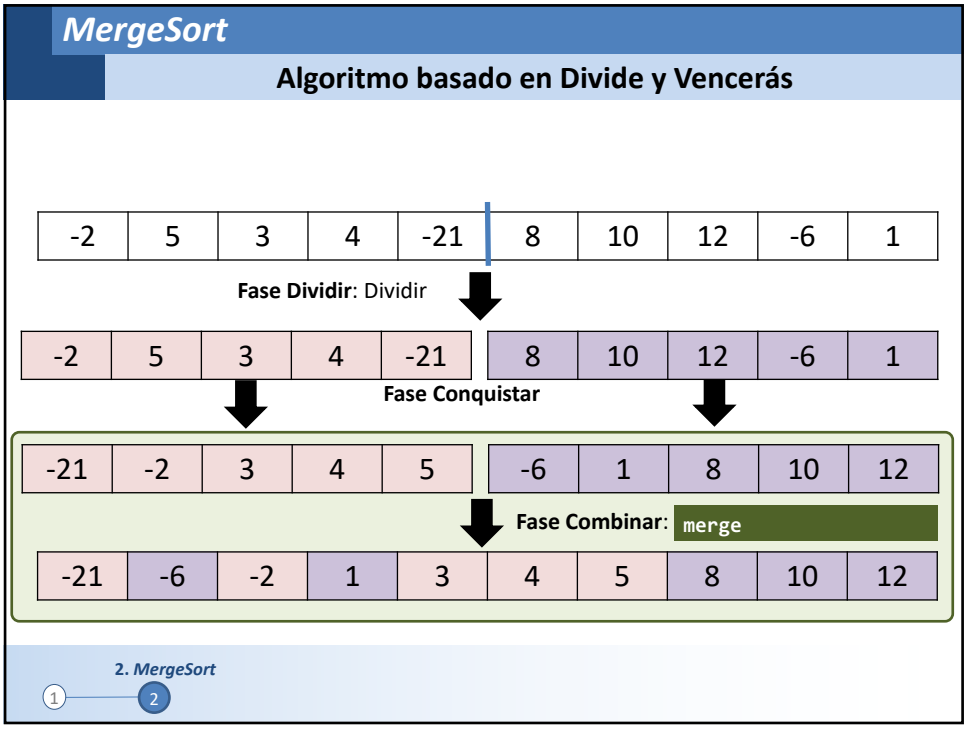
```
void merge(int[] vector, int i0, int k, int iN){
      int i=i0; int d=k+1;
      int[] aux=new int[iN-i0+1]; int f=0;
      while (i<=k && d<=iN) {
         if (vector[i]<=vector[d]) {
            aux[f] = vector[i];   i++; f++;
         }
         else {
            aux[f] = vector[d];   d++; f++;
         }
      }
      for (int a=i; a<=k; a++) {aux[f]= vector[a]; f++;}
      for (int a=d; a<=iN; a++) {aux[f]= vector[a]; f++;}
      for (int a=0; a<aux.length; a++) {vector[i0+a]= aux[a];}
}
```

Complejidad: $\Theta(N)$

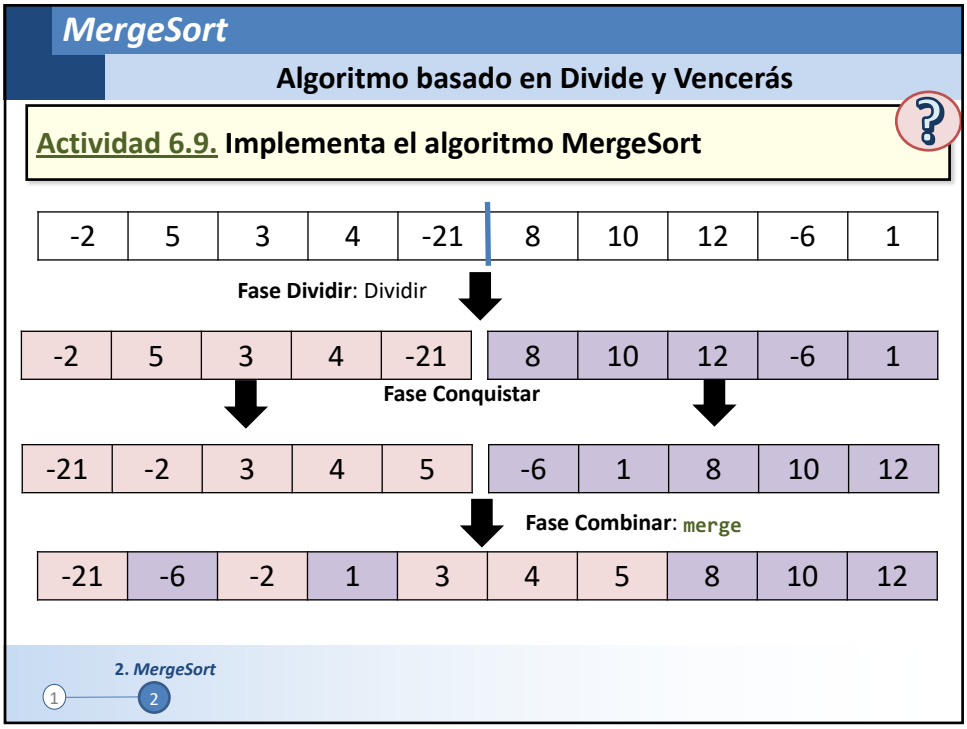
N Iteraciones

Actividad 6.8. Calcula la complejidad del algoritmo en función del tamaño del vector

70


MergeSort
Algoritmo basado en Divide y Vencerás
-2 5 3 4 -21 8 10 12 -6 1
Fase Dividir: Dividir
-2 5 3 4 -21 8 10 12 -6 1
Fase Conquistar
-21 -2 3 4 5 -6 1 8 10 12
Fase Combinar: merge
-21 -6 -2 1 3 4 5 8 10 12
2. MergeSort
1
2

71

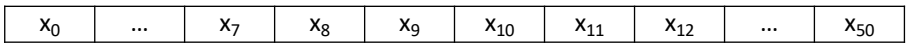

MergeSort
Algoritmo basado en Divide y Vencerás
Actividad 6.9. Implementa el algoritmo MergeSort
-2 5 3 4 -21 8 10 12 -6 1
Fase Dividir: Dividir
-2 5 3 4 -21 8 10 12 -6 1
Fase Conquistar
-21 -2 3 4 5 -6 1 8 10 12
Fase Combinar: merge
-21 -6 -2 1 3 4 5 8 10 12
2. MergeSort
1
2

72

MergeSort

Implementación

Actividad 6.9. Implementa el algoritmo MergeSort



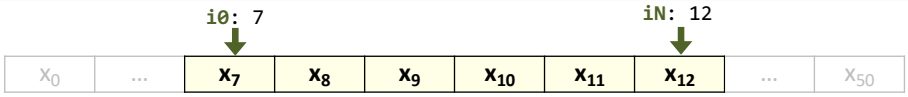

```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int k= (i0 + iN) /2;
         mergeSortAux(vector, i0, k);
         mergeSortAux(vector, k+1, iN);
         merge(vector, i0, k, iN);
      }
}
void mergeSort(int[] vector) {
      mergeSortAux(vector,0, vector.length-1);
}
```


73

MergeSort

Implementación

Actividad 6.9. Implementa el algoritmo MergeSort




```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int k= (i0 + iN) /2;
         mergeSortAux(vector, i0, k);
         mergeSortAux(vector, k+1, iN);
         merge(vector, i0, k, iN);
      }
}
void mergeSort(int[] vector) {
      mergeSortAux(vector,0, vector.length-1);
}
```


74

MergeSort

Implementación

Actividad 6.9. Implementa el algoritmo MergeSort


```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int k= (i0 + iN) /2;
         mergeSortAux(vector, i0, k);
         mergeSortAux(vector, k+1, iN);
         merge(vector, i0, k, iN);
      }
}
void mergeSort(int[] vector) {
      mergeSortAux(vector,0, vector.length-1);
}
```

2. MergeSort

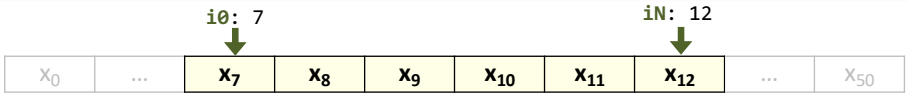
1 2

75

MergeSort

Implementación

Actividad 6.9. Implementa el algoritmo MergeSort




```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int k= (i0 + iN) /2;
         mergeSortAux(vector, i0, k);
         mergeSortAux(vector, k+1, iN);
         merge(vector, i0, k, iN);
      }
}
void mergeSort(int[] vector) {
      mergeSortAux(vector,0, vector.length-1);
}
```

2. MergeSort

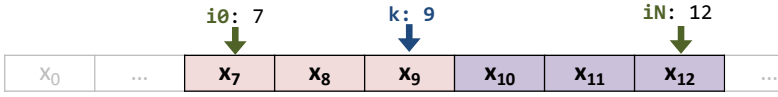
1 2

76

MergeSort

Implementación

Actividad 6.9. Implementa el algoritmo MergeSort




```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int k= (i0 + iN) /2;
         mergeSortAux(vector, i0, k);
         mergeSortAux(vector, k+1, iN);
         merge(vector, i0, k, iN);
      }
}
void mergeSort(int[] vector) {
      mergeSortAux(vector,0, vector.length-1);
}
```

Dividir

2. MergeSort

1 2

77

MergeSort

Implementación

Actividad 6.9. Implementa el algoritmo MergeSort


```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int k= (i0 + iN) /2;
         mergeSortAux(vector, i0, k);
         mergeSortAux(vector, k+1, iN);
         merge(vector, i0, k, iN);
      }
}
void mergeSort(int[] vector) {
      mergeSortAux(vector,0, vector.length-1);
}
```

Conquistar

2. MergeSort

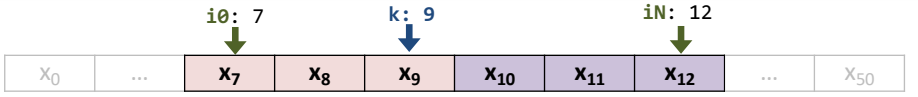
1 2

78

MergeSort

Implementación

Actividad 6.9. Implementa el algoritmo MergeSort




```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int k= (i0 + iN) /2;
         mergeSortAux(vector, i0, k);
         mergeSortAux(vector, k+1, iN);
         merge(vector, i0, k, iN);
      }
}
void mergeSort(int[] vector) {
      mergeSortAux(vector,0, vector.length-1);
}
```

Combinar

2. MergeSort

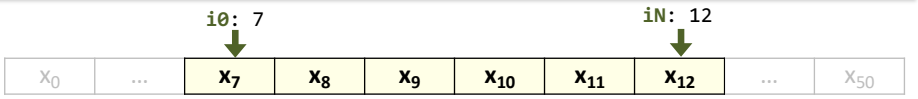
1 2

79

MergeSort

Complejidad

Actividad 6.10. Calcula la complejidad del algoritmo




```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int k= (i0 + iN) /2;
         mergeSortAux(vector, i0, k);
         mergeSortAux(vector, k+1, iN);
         merge(vector, i0, k, iN);
      }
}
```

2. MergeSort

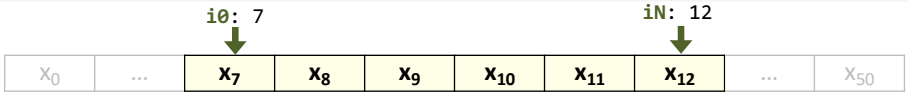
1 2

80

MergeSort

Complejidad

Actividad 6.10. Calcula la complejidad del algoritmo




```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
          int k= (i0 + iN) /2;
          mergeSortAux(vector, i0, k);
          mergeSortAux(vector, k+1, iN);
          merge(vector, i0, k, iN);
      }
}
```

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 2 \cdot T\left(\frac{N}{2}\right) + \Theta(N) & N > 1 \end{cases}$$

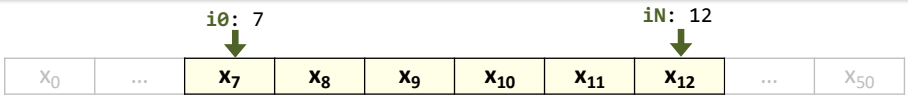
1 2

81

MergeSort

Complejidad

Actividad 6.10. Calcula la complejidad del algoritmo




```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
          int k= (i0 + iN) /2;
          mergeSortAux(vector, i0, k);
          mergeSortAux(vector, k+1, iN);
          merge(vector, i0, k, iN);
      }
}
```

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 2 \cdot T\left(\frac{N}{2}\right) + \Theta(N^1) & N > 1 \end{cases}$$

Teorema Maestro

(2º Caso)
$\text{Log}_2(2)=1$

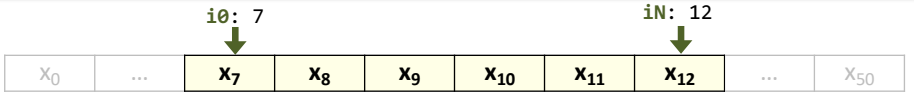
1 2

82

MergeSort

Complejidad

Actividad 6.10. Calcula la complejidad del algoritmo




```
void mergeSortAux (int[] vector, int i0, int iN){
      if (i0 >= iN)
          return;
      else {
         int k= (i0 + iN) /2;
         mergeSortAux(vector, i0, k);
         mergeSortAux(vector, k+1, iN);
         merge(vector, i0, k, iN);
      }
}
```

Complejidad $\Theta(N \cdot \log N)$

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 2 \cdot T\left(\frac{N}{2}\right) + \Theta(N) & N > 1 \end{cases}$$

Teorema Maestro

(2º Caso)
$\text{Log}_2(2)=1$

1 — 2

83

MergeSort

Algoritmos de Ordenación basados en Divide y Vencerás

Método	Complejidad Algorítmica en tiempo			Estabilidad
	Mejor	Promedio	Peor	
Selección	$O(N^2)$	$O(N^2)$	$O(N^2)$	× No
Inserción	$O(N)$	$O(N^2)$	$O(N^2)$	√Sí
Burbuja	$O(N)$	$O(N^2)$	$O(N^2)$	√Sí
CombSort	$O(N)$	$O(N^2)$	$O(N^2)$	× No
HeapSort	$O(N \cdot \log N)$	$O(N \cdot \log N)$	$O(N \cdot \log N)$	× No
QuickSort	$O(N \cdot \log N)$	$O(N \cdot \log N)$	$O(N^2)$	× No
MergeSort	$O(N \cdot \log N)$	$O(N \cdot \log N)$	$O(N \cdot \log N)$	√Sí

2. *MergeSort*

1 — 2

84