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

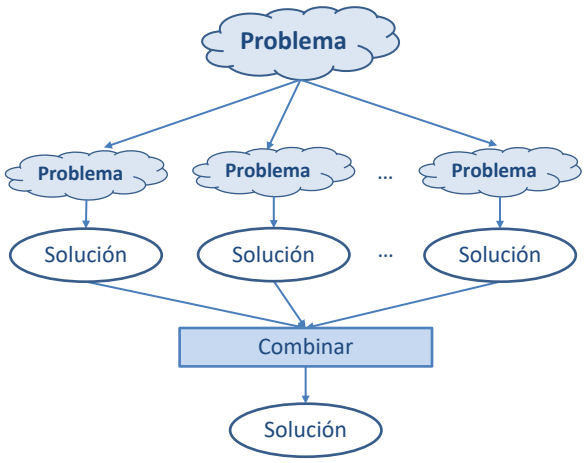
Tema 5. Esquema Divide y Vencerás

Algorítmica y Complejidad

1

Introducción

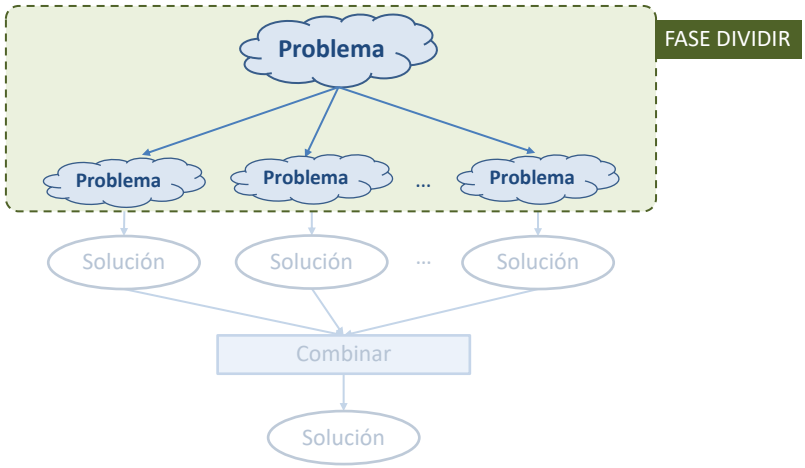
Esquema General



2

Introducción

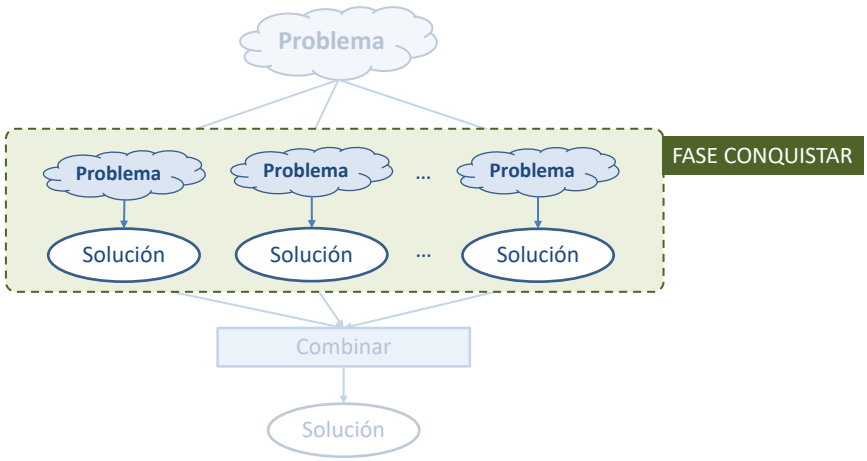
Esquema General



3

Introducción

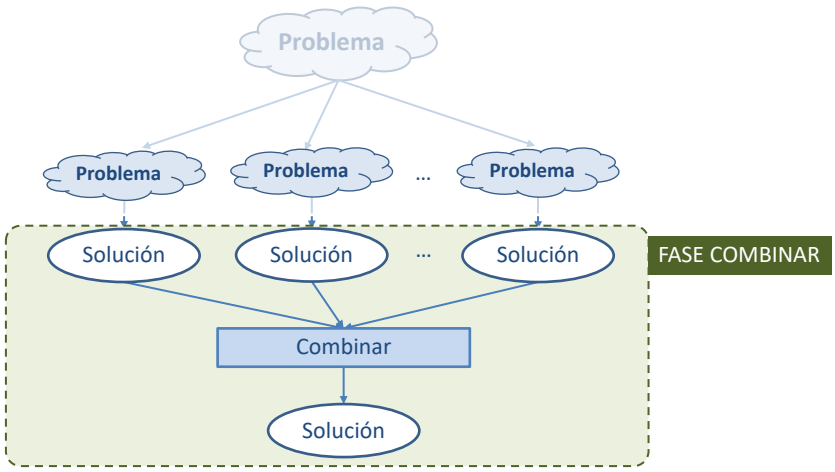
Esquema General



4

Introducción

Esquema General



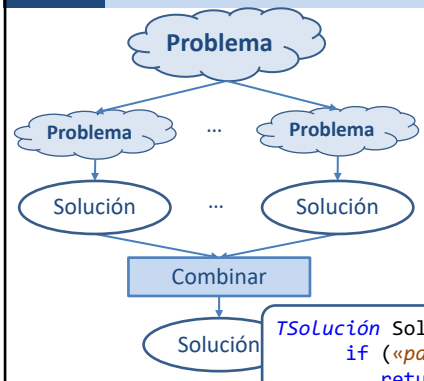
1. Introducción

1 2 3 4 5

5

Introducción

Esquema General




```
TSolución SoluciónProblema(Tentrada parámetros){
      if («parámetros pequeños»)
         return «Solucionar problema»
      else {
         «Dividir Problema en problemas P1,P2,…,Pn »
         TSolución sol1= SoluciónProblema(«P1»)
         …
         TSolución soln= SoluciónProblema(«Pn»)
         TSolución sol = «Combinar sol1, sol2,…, soln»
         return sol;
      }
}
```

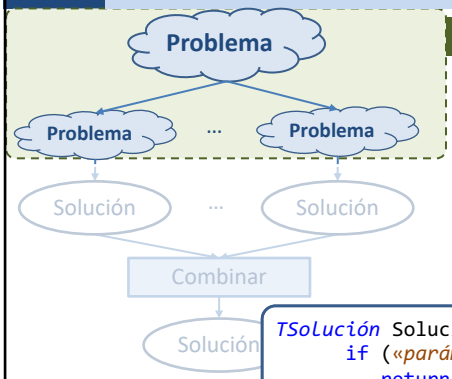
1. Introducción

1 2 3

6

Introducción

Esquema General



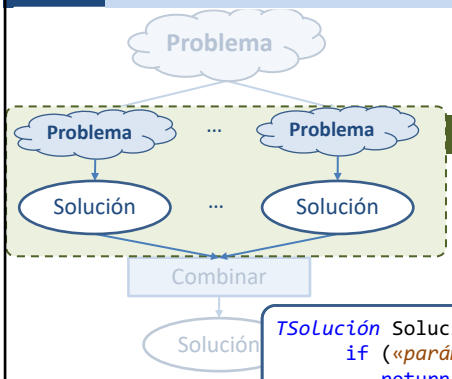

```
TSolución SoluciónProblema(Tentrada parámetros){
      if («parámetros pequeños»)
         return «Solucionar problema»
      else {
         «Dividir Problema en problemas P1,P2,…,Pn »
         TSolución sol1= SoluciónProblema(«P1»)
         …
         TSolución soln= SoluciónProblema(«Pn»)
         TSolución sol = «Combinar sol1, sol2,…, soln»
         return sol;
      }
}
```

1. Introducción

7

Introducción

Esquema General



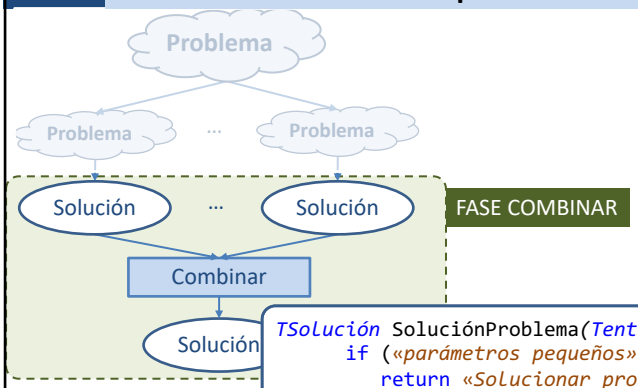

```
TSolución SoluciónProblema(Tentrada parámetros){
      if («parámetros pequeños»)
         return «Solucionar problema»
      else {
         «Dividir Problema en problemas P1,P2,…,Pn »
         TSolución sol1= SoluciónProblema(«P1»)
         …
         TSolución soln= SoluciónProblema(«Pn»)
         TSolución sol = «Combinar sol1, sol2,…, soln»
         return sol;
      }
}
```

1. Introducción

8

Introducción

Esquema General



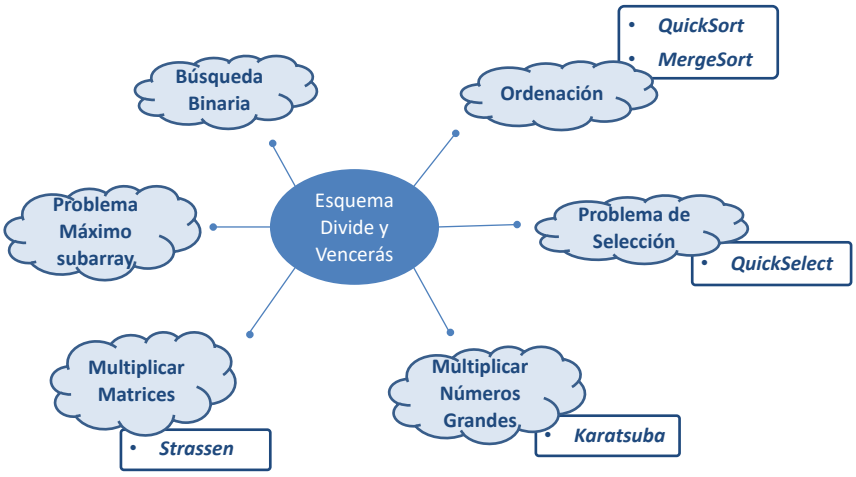

```
TSolución SoluciónProblema(Tentrada parámetros){
      if («parámetros pequeños»)
         return «Solucionar problema»
      else {
         «Dividir Problema en problemas P1,P2,…,Pn »
         TSolución sol1= SoluciónProblema(«P1»)
         …
         TSolución soln= SoluciónProblema(«Pn»)
         TSolución sol = «Combinar sol1, sol2,…, soln»
         return sol;
      }
}
```

1. Introducción

9

Introducción

Algunos problemas planteados

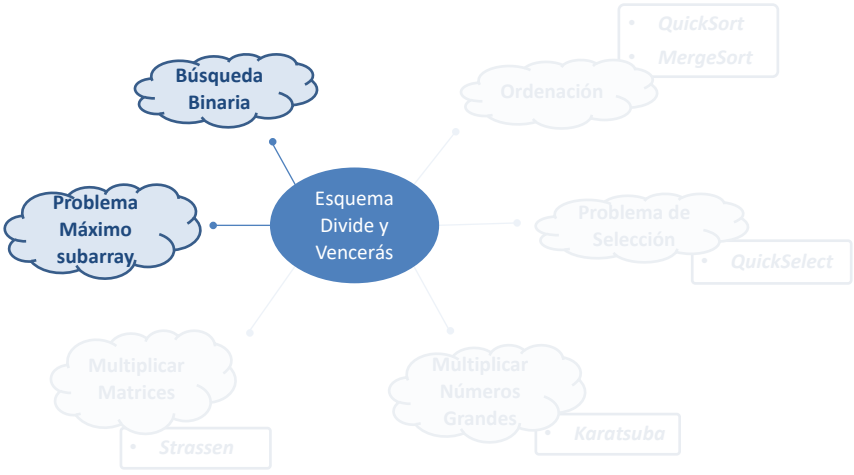


1. Introducción

10

Introducción

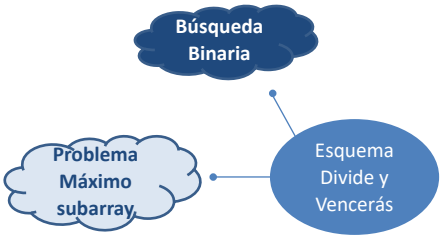
Algunos problemas planteados



11

Búsqueda Binaria

Algunos problemas planteados



12

Búsqueda Binaria

Enunciado del Problema

Dado un *array* **ordenado**: Y un número: **25**

-10	-2	1	8	12	15	20	**25**	36	51

Comprobar si el número se encuentra en el *array*

true

13

Búsqueda Binaria

Enunciado del Problema

Dado un *array* **ordenado**: Y un número: **30**

-10	-2	1	8	12	15	20	25	36	51

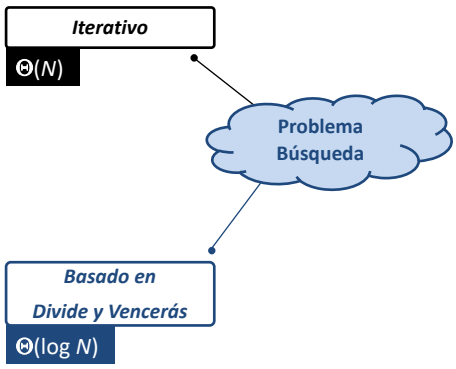
Comprobar si el número se encuentra en el *array*

false

14

Búsqueda Binaria

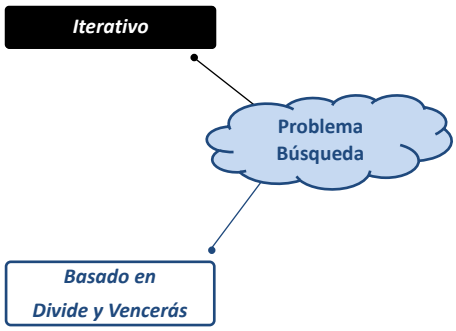
Algoritmos de Resolución



15

Búsqueda Binaria

Algoritmos de Resolución



16

Búsqueda Binaria

Algoritmo Simple

`elemento: 8`

`i:0`

-10	-2	1	8	12	15	20	25	36	51

Si `vector[i]<elemento`, entonces incrementamos `i`

2. Búsqueda Binaria

1 — 2 — 3 — 4 — 5

17

Búsqueda Binaria

Algoritmo Simple

`elemento: 8`

`i:1`

-10	**-2**	1	8	12	15	20	25	36	51

Si `vector[i]<elemento`, entonces incrementamos `i`

2. Búsqueda Binaria

1 — 2 — 3 — 4 — 5

18

Búsqueda Binaria

Algoritmo Simple

`elemento: 8`

`i:2`

-10	-2	1	8	12	15	20	25	36	51

Si `vector[i]<elemento`, entonces incrementamos `i`

19

Búsqueda Binaria

Algoritmo Simple

`elemento: 8`

`i:3`

-10	-2	1	8	12	15	20	25	36	51

Si `vector[i]<elemento`, entonces incrementamos `i`

Si `vector[i]==elemento`, entonces devolvemos **true**

20

Búsqueda Binaria

Algoritmo Simple

`elemento: 13`

`i:3`

-10	-2	1	**8**	12	15	20	25	36	51

Si `vector[i]<elemento`, entonces incrementamos `i`

Si `vector[i]==elemento`, entonces devolvemos **true**

2. Búsqueda Binaria

1 — 2 — 3 — 4 — 5

21

Búsqueda Binaria

Algoritmo Simple

`elemento: 13`

`i:4`

-10	-2	1	8	**12**	15	20	25	36	51

Si `vector[i]<elemento`, entonces incrementamos `i`

Si `vector[i]==elemento`, entonces devolvemos **true**

2. Búsqueda Binaria

1 — 2 — 3 — 4 — 5

22

Búsqueda Binaria

Algoritmo Simple

elemento: 13

i:5

-10	-2	1	8	12	15	20	25	36	51

Si `vector[i]<elemento`, entonces incrementamos `i`

Si `vector[i]==elemento`, entonces devolvemos **true**

Si `vector[i]>elemento`, entonces devolvemos **false**

23

Búsqueda Binaria

Actividad 5.1. Implementa un algoritmo que compruebe si hay un elemento en un *array* ordenado.

elemento: 13

i:5

-10	-2	1	8	12	15	20	25	36	51

```
boolean contieneValor (int[] vector, int elemento){
   int i=0;
   while ((vector[i] < elemento) && (i<vector.lenght))
         i++;
   if ((i<vector.lenght)&&(vector[i]==elemento))
        return true;
   else return false;
}
```


24

Búsqueda Binaria

Actividad 5.2. **Calcula la complejidad del algoritmo en el caso peor.**

elemento: 52 i:9

-10	-2	1	8	12	15	20	25	36	51

```
boolean contieneValor (int[] vector, int elemento
   int i=0;
   while ((vector[i] < elemento) && (i<vector.le
         i++;
   if ((i<vector.lenght)&&(vector[i]==elemento))
        return true;
   else return false;
}
```

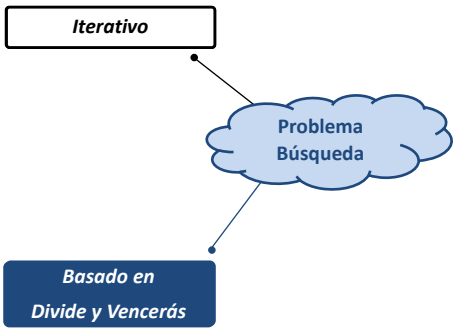
Caso peor:
El elemento buscado es mayor que todos los del *array*.

Complejidad: $\Theta(N)$

25

Búsqueda Binaria

Algoritmos de Resolución



26

Búsqueda Binaria

Algoritmo basado en Divide y Vencerás

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

Fase Dividir: Dividir el Array en dos subarrays

x_0	x_1	x_2	x_3	x_4

x_5	x_6	x_7	x_8	x_9

valor <= x_4

Fase Conquistar:

valor > x_4

x_0	x_1	x_2	x_3	x_4

x_5	x_6	x_7	x_8	x_9

Fase Combinar:

Resultado del subproblema resuelto

27

Búsqueda Binaria

Algoritmo basado en Divide y Vencerás: Implementación

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

```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
boolean contiene(int[] vector, int elemento){
      return contieneAux(vector,0, vector.length-1,elemento);
}
```


28

Búsqueda Binaria

Algoritmo basado en Divide y Vencerás: Implementación

i0: 7 iN: 12

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

```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
boolean contiene(int[] vector, int elemento){
      return contieneAux(vector,0, vector.length-1,elemento);
}
```

2. Búsqueda Binaria

1 2 3 4 5

29

Búsqueda Binaria

Algoritmo basado en Divide y Vencerás: Implementación

i0:7 iN:7

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

```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
boolean contiene(int[] vector, int elemento){
      return contieneAux(vector,0, vector.length-1,elemento);
}
```

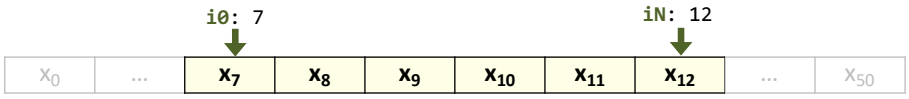
2. Búsqueda Binaria

1 2 3 4 5

30

Búsqueda Binaria

Algoritmo basado en Divide y Vencerás: Implementación




```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
boolean contiene(int[] vector, int elemento){
      return contieneAux(vector,0, vector.length-1,elemento);
}
```

2. Búsqueda Binaria

1 — 2 — 3 — 4 — 5

31

Búsqueda Binaria

Algoritmo basado en Divide y Vencerás: Implementación


```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;                                  } Dividir
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
boolean contiene(int[] vector, int elemento){
      return contieneAux(vector,0, vector.length-1,elemento);
}
```

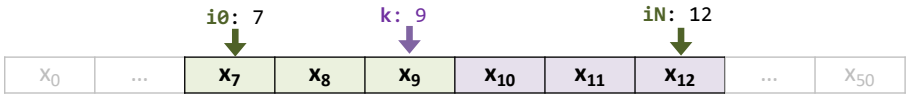
2. Búsqueda Binaria

1 — 2 — 3 — 4 — 5

32

Búsqueda Binaria

Algoritmo basado en Divide y Vencerás: Implementación




```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
boolean contiene(int[] vector, int elemento){
      return contieneAux(vector,0, vector.length-1,elemento);
}
```

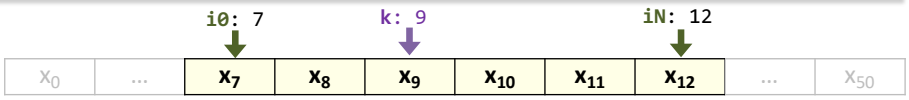
Conquistar

2. Búsqueda Binaria

1 2 3 4 5

33

Actividad 5.3. Calcula la complejidad de `contieneAux` en función del tamaño (`iN-i0+1`) del vector sobre el que se busca el elemento



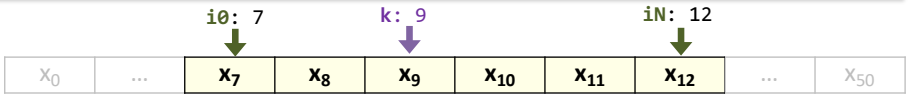

```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
boolean contiene(int[] vector, int elemento){
      return contieneAux(vector,0, vector.length-1,elemento);
}
```

2. Búsqueda Binaria

1 2 3 4 5

34

Actividad 5.3. Calcula la complejidad de `contieneAux` en función del tamaño (`iN-i0+1`) del vector sobre el que se busca el elemento




```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
```

Ecuación de Recurrencia

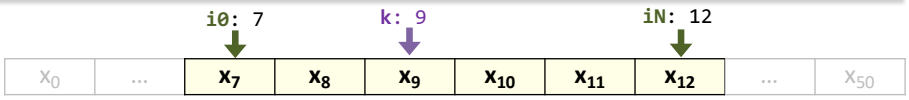
$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ T\left(\frac{N}{2}\right) + \Theta(1) & N > 1 \end{cases}$$

2. Búsq

1 2 3 4 5

35

Actividad 5.3. Calcula la complejidad de `contieneAux` en función del tamaño (`iN-i0+1`) del vector sobre el que se busca el elemento




```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
```

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 1 \cdot T\left(\frac{N}{2}\right) + \Theta(N^0) & N > 1 \end{cases}$$

➡ **Teorema Maestro**

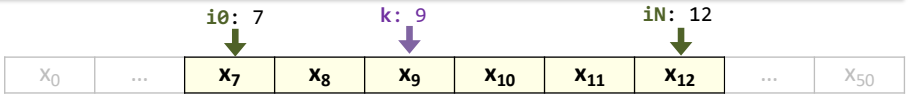
(2º Caso)
$\text{Log}_2(1)=0$

2. Búsq

1 2 3 4 5

36

Actividad 5.3. Calcula la complejidad de `contieneAux` en función del tamaño (`iN-i0+1`) del vector sobre el que se busca el elemento




```
boolean contieneAux(int[] vector, int i0, int iN, int elemento){
      if (i0 == iN)
         return (vector[i0]==elemento);
      else {
         int k= (i0 + iN) / 2;
         if (vector[k]>elemento)
             return contieneAux(vector, i0, k, elemento);
         else if (vector[k]==elemento)
             return true;
         else
             return contieneAux(vector, k+1, iN, elemento);
      }
}
```

Complejidad: $\Theta(\log N)$

Ecuación de Recurrencia

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

Teorema Maestro

(2º Caso)
$\text{Log}_2(1)=0$

2. Búsq

1 2 3 4 5

37

Búsqueda Binaria

Algoritmos de Resolución

Algoritmos basados en comparaciones $\Omega(\log N)$

Búsqueda lineal: Iterativo

$\Theta(N)$

Problema Búsqueda

Búsqueda Binaria: Basado en Divide y Vencerás

$\Theta(\log N)$

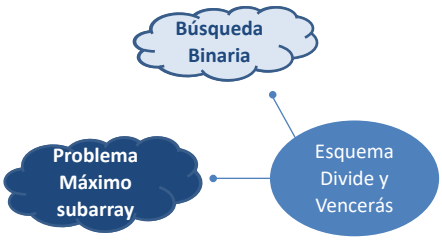
2. Búsqueda Binaria

1 2 3 4 5

38

Problema del Máximo Subarray

Algunos problemas planteados



39

Problema del Máximo Subarray

Enunciado del Problema

Dado un *array*:

-2	12	3	4	-31	8	12	5	-6	1

Encontrar la suma máxima de cualquiera de sus *subarrays*

40

Problema del Máximo Subarray

Enunciado del Problema

Dado un *array*:

-2	12	3	4	-31	8	12	5	-6	1

19

Encontrar la suma máxima de cualquiera de sus *subarrays*

41

Problema del Máximo Subarray

Enunciado del Problema

Dado un *array*:

-2	12	3	4	-31	8	12	5	-6	1

-4

Encontrar la suma máxima de cualquiera de sus *subarrays*

42

Problema del Máximo Subarray

Enunciado del Problema

Dado un *array*:

-2	12	3	4	-31	8	12	5	-6	1

-11

Encontrar la suma máxima de cualquiera de sus *subarrays*

43

Problema del Máximo Subarray

Enunciado del Problema

Dado un *array*:

-2	12	3	4	-31	8	12	5	-6	1

25

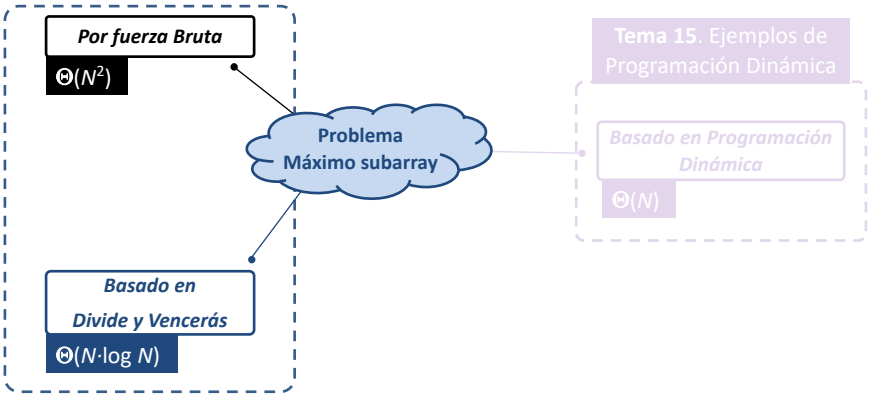
Encontrar la suma máxima de cualquiera de sus *subarrays*

Solución: **25**

44

Problema del Máximo Subarray

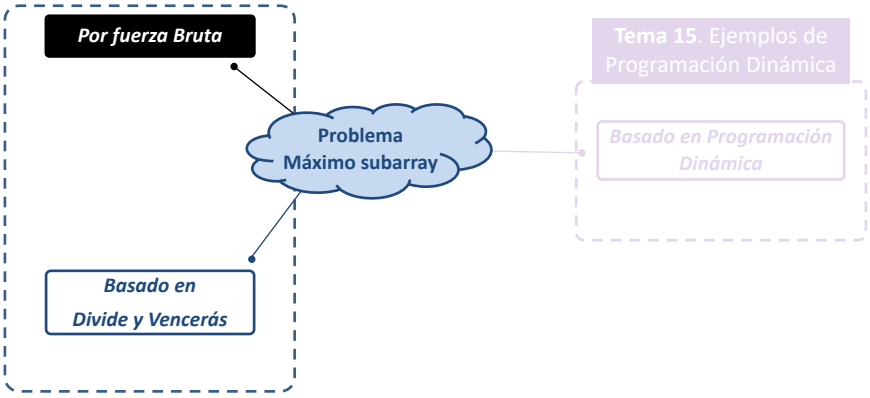
Algoritmos de Resolución



45

Problema del Máximo Subarray

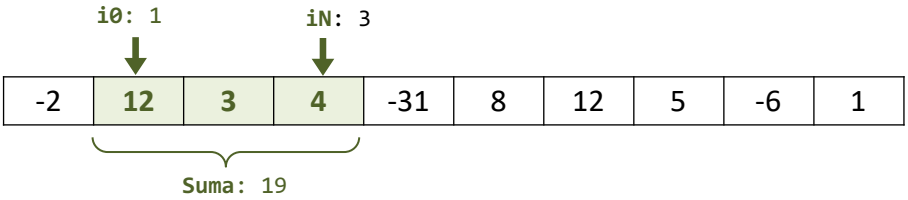
Algoritmos de Resolución



46

Problema del Máximo Subarray

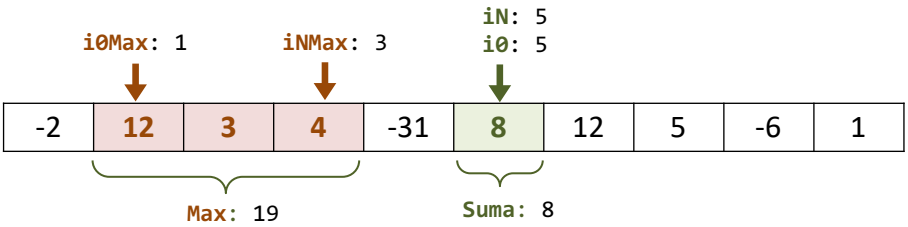
Algoritmo por Fuerza Bruta



47

Problema del Máximo Subarray

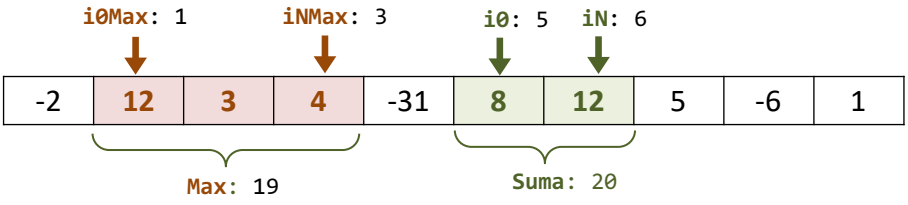
Algoritmo por Fuerza Bruta



48

Problema del Máximo Subarray

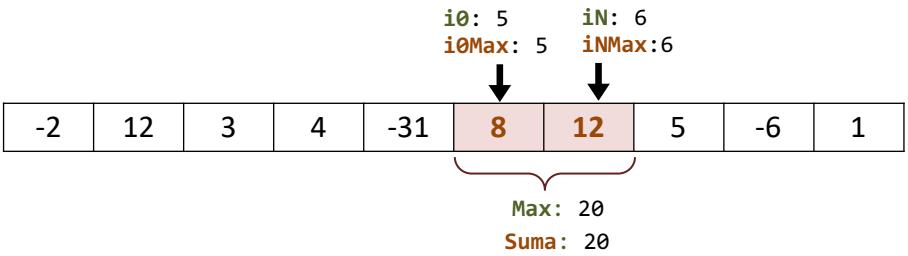
Algoritmo por Fuerza Bruta



49

Problema del Máximo Subarray

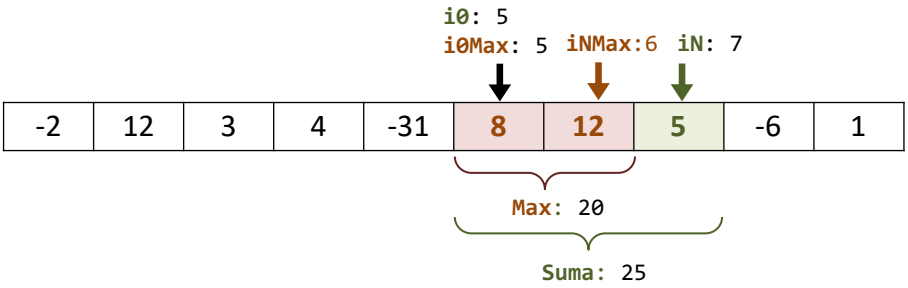
Algoritmo por Fuerza Bruta



50

Problema del Máximo Subarray

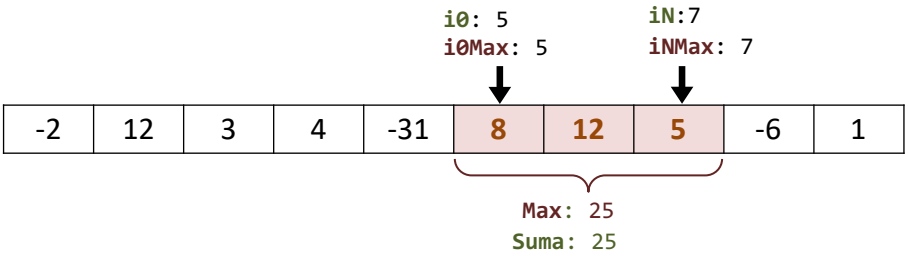
Algoritmo por Fuerza Bruta



51

Problema del Máximo Subarray

Algoritmo por Fuerza Bruta

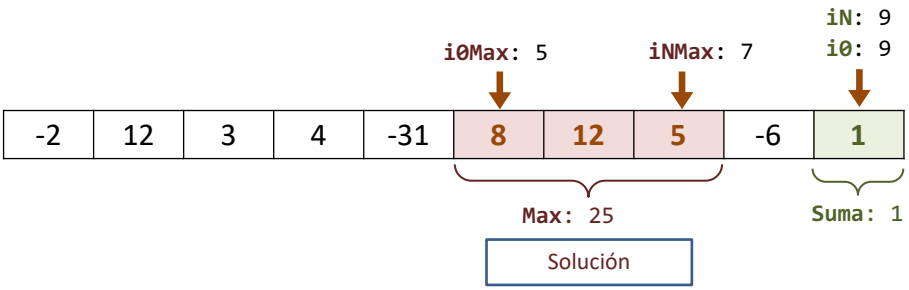


52

Problema del Máximo Subarray

Algoritmo por Fuerza Bruta

Actividad 5.4. Diseña e implementa un algoritmo que calcule el máximo subarray de un array.



53

Problema del Máximo Subarray

Algoritmo por Fuerza Bruta

Actividad 5.4. Diseña e implementa un algoritmo que calcule el máximo subarray de un array.

```
int maxSubarray (int[] vector){
      int max=Integer.MIN_VALUE;
      int i0Max=0;
      int iNMax=0;
      for (int i0=0; i0<vector.length; i0++) {
          int suma = 0;
          for (int iN=i0; iN<vector.length; iN++) {
             suma+=vector[iN];
             if (suma > max) {
                max = suma;
                i0Max = i0;
                iNMax = iN;
             }
          }
      }
      return max;
}
```


54

Problema del Máximo Subarray

Algoritmo por Fuerza Bruta

Actividad 5.5. Calcula la complejidad del algoritmo.

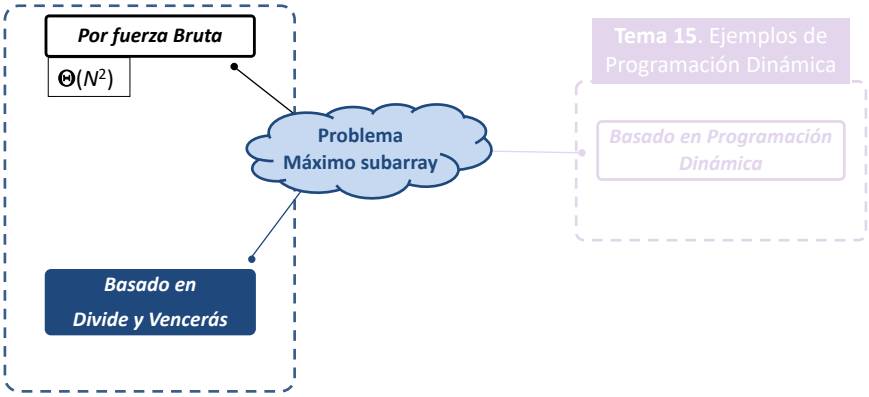
```
int maxSubarray (int[] vector){
      int max=Integer.MIN_VALUE;
      int i0Max=0;
      int iNMax=0;
      for (int i0=0; i0<vector.length; i0++) {
          int suma = 0;
          for (int iN=i0; iN<vector.length; iN++) {
             suma+=vector[iN];
             if (suma > max) {
                max = suma;
                i0Max = i0;
                iNMax = iN;
             }
          }
      }
      return max;
}
```

Complejidad: $\Theta(N^2)$

55

Problema del Máximo Subarray

Algoritmos de Resolución



56

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

Fase Dividir: Dividir el Array en dos subarrays

x_0	x_1	x_2	x_3	x_4

x_5	x_6	x_7	x_8	x_9

Fase Conquistar: Encontrar la solución de cada subarray

x_0	x_1	x_2	x_3	x_4

x_5	x_6	x_7	x_8	x_9

Fase Combinar: Combinar las soluciones de los subproblemas

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

3. Problema del Máximo Subarray

1 2 3 4 5

57

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

Fase Dividir: Dividir el Array en dos subarrays

x_0	x_1	x_2	x_3	x_4

x_5	x_6	x_7	x_8	x_9

Fase Conquistar: Encontrar la solución de cada subarray

x_0	x_1	x_2	x_3	x_4

x_5	x_6	x_7	x_8	x_9

Fase Combinar: Combinar las soluciones de los subproblemas

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

3. Problema del Máximo Subarray

1 2 3 4 5

58

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Combinar

Caso 1

-2	12	3	4	-31	-6	12	5	8	-1

Caso 2

10	5	-16	1	8	12	5	-20	16	2

59

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Combinar

Caso 1

-2	12	3	4	-31	-6	**12**	**5**	**8**	-1

Caso 2

10	5	-16	1	8	12	5	-20	16	2

60

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Combinar

Caso 1 El subarray máximo permanence en alguna de las divisiones

-2	12	3	4	-31	-6	12	5	8	-1

-2	12	3	4	-31	-6	12	5	8	-1

Caso 2

10	5	-16	1	8	12	5	-20	16	2

61

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Combinar

Caso 1 El subarray máximo permanence en alguna de las divisiones

-2	12	3	4	-31	-6	12	5	8	-1

-2	12	3	4	-31	-6	12	5	8	-1

El subarray máximo **es uno de los subarrays máximos** de los subproblemas.

Caso 2

10	5	-16	1	8	12	5	-20	16	2

62

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Combinar

Caso 1 El subarray máximo permanence en alguna de las divisiones

-2	12	3	4	-31	-6	12	5	8	-1

-2	12	3	4	-31	-6	12	5	8	-1

19 25

Caso 2

10	5	-16	1	8	12	5	-20	16	2

63

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Combinar

Caso 1 El subarray máximo permanence en alguna de las divisiones

-2	12	3	4	-31	-6	12	5	8	-1

-2	12	3	4	-31	-6	12	5	8	-1

19 25

Caso 2 El subarray máximo se encuentra en mitad de las división

10	5	-16	1	8	12	5	-20	16	2

10	5	-16	1	8	12	5	-20	16	2

64

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Combinar

Caso 1 El subarray máximo permanence en alguna de las divisiones

-2	12	3	4	-31	-6	12	5	8	1

-2	12	3	4	-31	-6	12	5	8	1

19 25

Caso 2 El subarray máximo se encuentra en mitad de las división

10	5	-16	1	8	12	5	-20	16	2

26

10	5	-16	1	8	12	5	-20	16	2

15 18

3. Problema del Máximo Subarray

1 2 3 4 5

65

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Combinar

Caso 1 El subarray máximo permanence en alguna de las divisiones

-2	12	3	4	-31	-6	12	5	8	1

-2	12	3	4	-31	-6	12	5	8	1

El subarray máximo **incluye el elemento por el que hemos dividido**

Caso 2

10	5	-16	1	8	12	5	-20	16	2

26

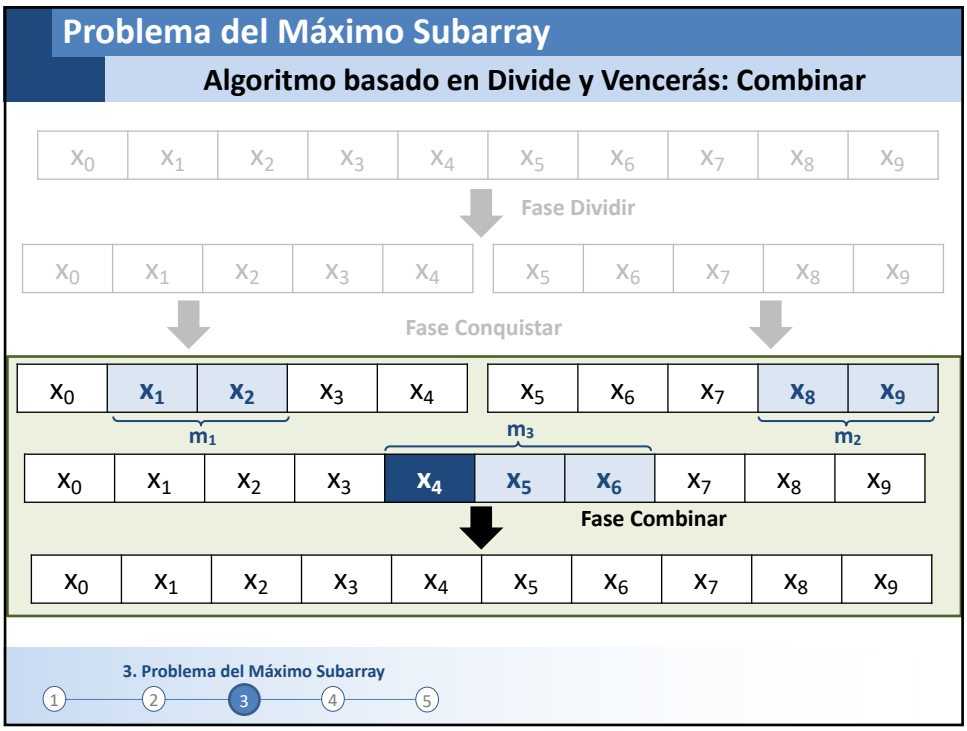
10	5	-16	1	8	12	5	-20	16	2

15 18

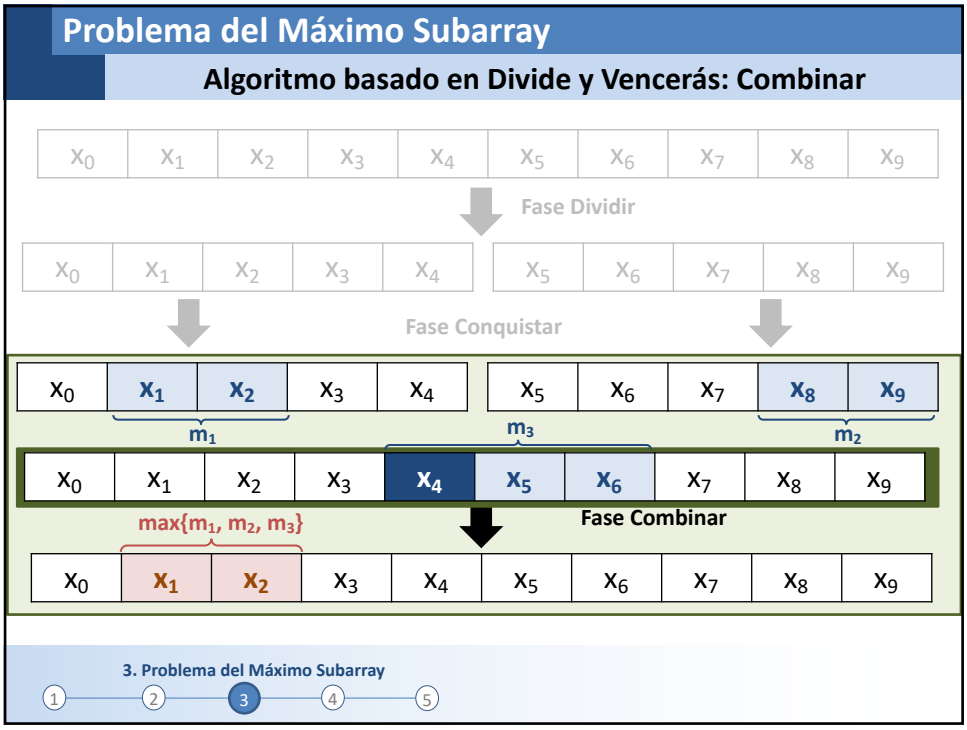
3. Problema del Máximo Subarray

1 2 3 4 5

66


Problema del Máximo Subarray
Algoritmo basado en Divide y Vencerás: Combinar
Fase Dividir
Fase Conquistar
m1
m3
m2
Fase Combinar
3. Problema del Máximo Subarray

67

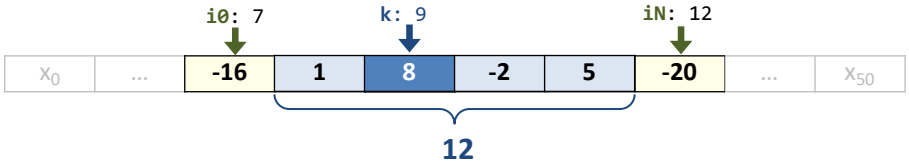

Problema del Máximo Subarray
Algoritmo basado en Divide y Vencerás: Combinar
Fase Dividir
Fase Conquistar
m1
m3
m2
max{m1, m2, m3}
Fase Combinar
3. Problema del Máximo Subarray

68

Problema del Máximo Subarray

Actividad 5.6. Diseña e implementa un algoritmo que calcule el máximo subarray que incluye el pivote.

Dado un *array*: y un pivote *k*:

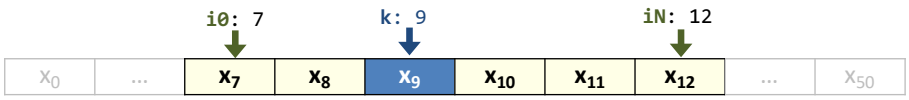


Encontrar la suma **máxima** de cualquiera de los *subarrays* que **incluyan el pivote**

69

Problema del Máximo Subarray

Implementación `maxSubarrayCruzada`

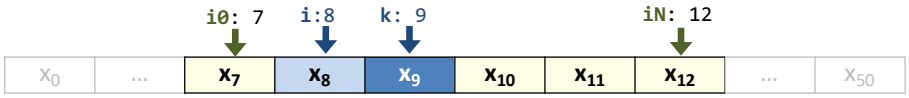



```
int maxSubarrayCruzada(int[] vector, int i0, int k, int iN){
      int max=Integer.MIN_VALUE;     int suma=0;
      int iMax=k;      int jMax=k;
      for (int i=k; i>= i0; i--) {
          suma += vector[i];
          if (suma > max) {
             max=suma; iMax=i;
          }
      }
      suma=max;
      for (int j=k+1; j<= iN; j++) {
          suma += vector[j];
          if (suma > max) {
             max = suma; jMax = j;
          }
      }
      return max;
}
```


70

Problema del Máximo Subarray

Implementación maxSubarrayCruzada



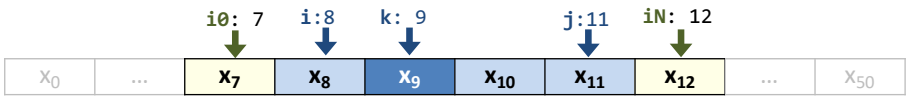

```
int maxSubarrayCruzada(int[] vector, int i0, int k, int iN){
      int max=Integer.MIN_VALUE;     int suma=0;
      int iMax=k;      int jMax=k;
      for (int i=k; i>= i0; i--) {
          suma += vector[i];
          if (suma > max) {
             max=suma; iMax=i;
          }
      }
      suma=max;
      for (int j=k+1; j<= iN; j++) {
          suma += vector[j];
          if (suma > max) {
             max = suma; jMax = j;
          }
      }
      return max;
}
```

Se busca por la izquierda de *k*:
el mayor subarray desde i=i0 hasta k

71

Problema del Máximo Subarray

Implementación maxSubarrayCruzada




```
int maxSubarrayCruzada(int[] vector, int i0, int k, int iN){
      int max=Integer.MIN_VALUE;     int suma=0;
      int iMax=k;      int jMax=k;
      for (int i=k; i>= i0; i--) {
          suma += vector[i];
          if (suma > max) {
             max=suma; iMax=i;
          }
      }
      suma=max;
      for (int j=k+1; j<= iN; j++) {
          suma += vector[j];
          if (suma > max) {
             max = suma; jMax = j;
          }
      }
      return max;
}
```

Se busca desde la derecha de *k*:
el mayor subarray desde j=k+1 hasta iN

72

Problema del Máximo Subarray

Complejidad `maxSubarrayCruzada`

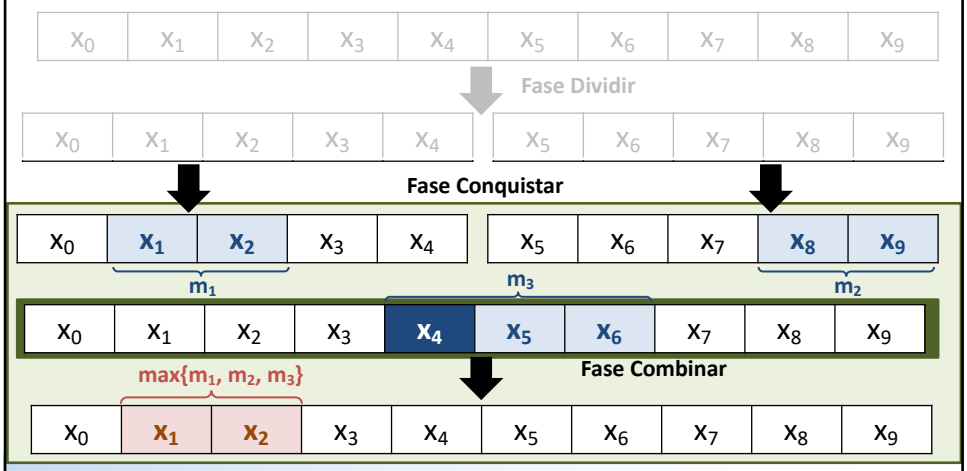
Actividad 5.7. Calcula la complejidad del algoritmo.

```
int maxSubarrayCruzada(int[] vector, int i0, int k, int iN){
      int max=Integer.MIN_VALUE;     int suma=0;
      int iMax=k;      int jMax=k;
      for (int i=k; i>= i0; i--) {
          suma += vector[i];
          if (suma > max) {
             max=suma; iMax=i;
          }
      }
      suma=max;
      for (int j=k+1; j<= iN; j++) {
          suma += vector[j];
          if (suma > max) {
             max = suma; jMax = j;
          }
      }
      return max;
}
```

Complejidad: $\Theta(N)$

Problema del Máximo Subarray

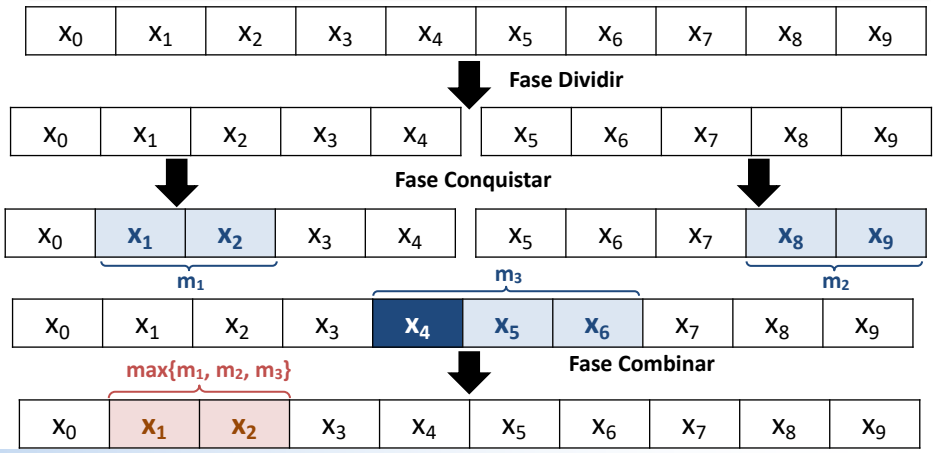
Algoritmo basado en Divide y Vencerás: Combinar



Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás

Actividad 5.8. Implementa el algoritmo basado en Divide y Vencerás.



75

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

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

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length-1);
}
```


76

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

i0: 7 iN: 12

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

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length-1);
}
```

3. Problema del Máximo Subarray

1 2 3 4 5

77

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

i0: 7 iN: 12

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

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length-1);
}
```

3. Problema del Máximo Subarray

1 2 3 4 5

78

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

i0: 7 iN: 7

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

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length-1);
}
```

Caso Base

3. Problema del Máximo Subarray

1 2 3 4 5

79

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

i0: 7 iN: 12

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

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length);
}
```

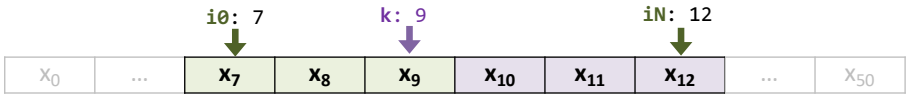
3. Problema del Máximo Subarray

1 2 3 4 5

80

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación




```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length-1);
}
```

Dividir

81

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

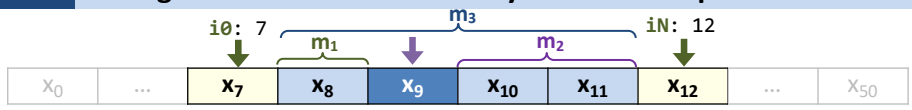

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length-1);
}
```

Conquistar

82

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación



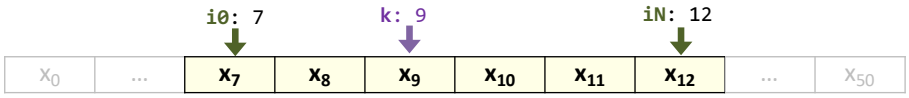

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length-1);
}
```

Combinar

83

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación




```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length-1);
}
```

Complejidad: $\Theta(N)$

84

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.9. Calcula la complejidad del algoritmo en función del tamaño del vector

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}

int maxSubarray(int[] vector){
      return maxSubarrayAux(vector,0, vector.length-1);
}
```


85

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.9. Calcula la complejidad del algoritmo en función del tamaño del vector

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}
```

Ecuación de Recurrencia

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

86

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

i0: 7 iN: 12

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

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}
```

Ecuación de Recurrencia

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

Teorema Maestro

(2º Caso)
$\text{Log}_2(2)=1$

87

Problema del Máximo Subarray

Algoritmo basado en Divide y Vencerás: Implementación

i0: 7 iN: 12

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

```
int maxSubarrayAux(int[] vector, int i0, int iN){
      if (i0 == iN)
         return vector[i0];
      else {
         int k= (i0 + iN) / 2;
         int m1 = maxSubarrayAux(vector, i0, k);
         int m2 = maxSubarrayAux(vector, k + 1, iN);
         int m3 = maxSubarrayCruzada(vector, i0, k, iN);
         return Math.max(m1, Math.max(m2, m3));
      }
}
```

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

Ecuación de Recurrencia

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

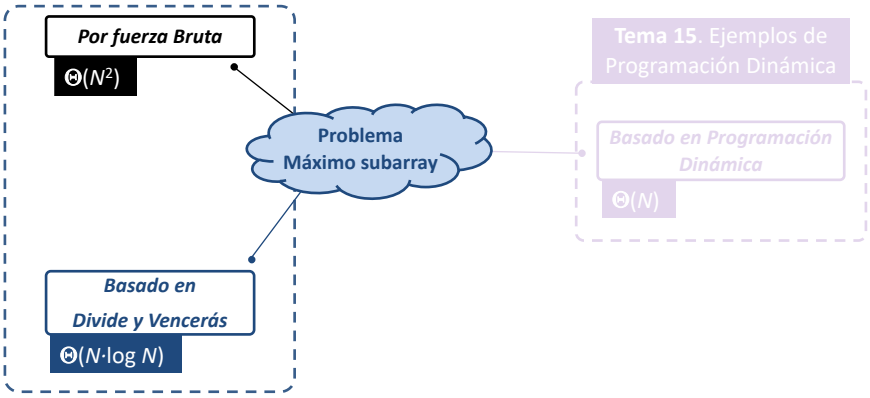
Teorema Maestro

(2º Caso)
$\text{Log}_2(2)=1$

88

Problema del Máximo Subarray

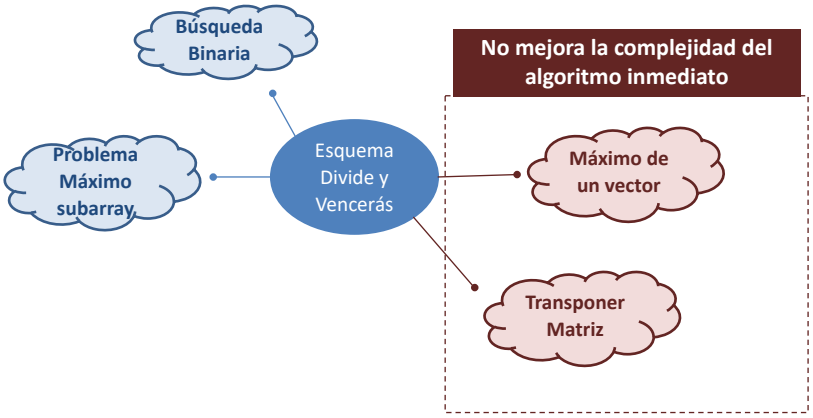
Algoritmos de Resolución



89

Problema del Máximo Subarray

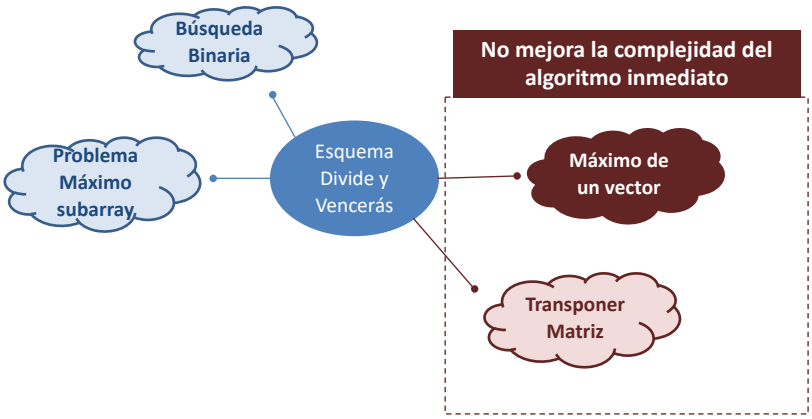
Algunos problemas planteados



90

Máximo de un vector

Algunos problemas planteados



91

Máximo de un vector

Enunciado del Problema

Dado un *array*:

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

Devolver el elemento **mayor** del *array*

12

92

Máximo de un vector

Enunciado del Problema

Actividad 5.10. Diseña e implementa un algoritmo que calcule el máximo de un vector

Dado un *array*:

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

Devolver el elemento **mayor** del *array*

12

93

Máximo de un vector

Algoritmo sencillo

Actividad 5.10. Diseña e implementa un algoritmo que calcule el máximo de un vector

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

```
int max(int[] vector){
   int m= Integer.MIN_VALUE;
   for (int i=0; i<vector.length; i++) {
      if (vector[i]> m) m = vector[i];
   }
   return m;
}
```


94

Máximo de un vector

Complejidad del algoritmo sencillo

Actividad 5.11. Calcula la complejidad del algoritmo

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

```
int max(int[] vector){
   int m= Integer.MIN_VALUE;
   for (int i=0; i<vector.length; i++) {
      if (vector[i]> m) m = vector[i];
   }
   return m;
}
```

Complejidad: $\Theta(N)$

4. Problema del Máximo de un vector

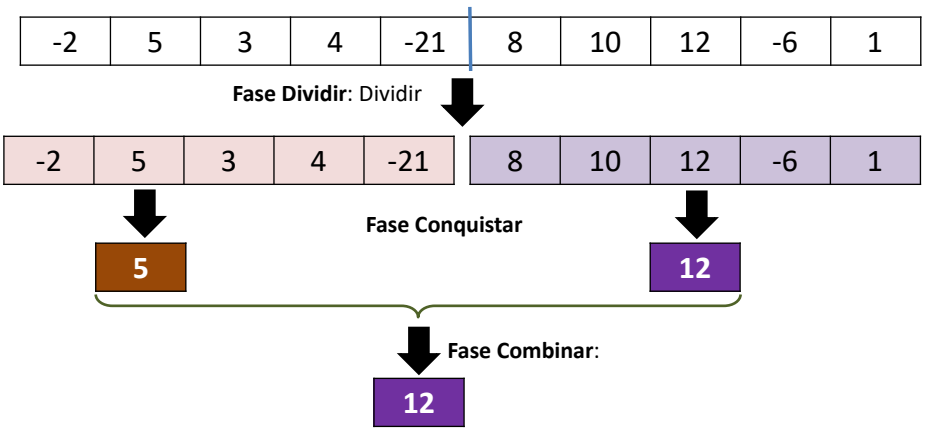
1 2 3 4 5

95

Máximo de un vector

Algoritmo basado en Divide y Vencerás

Actividad 5.12. Implementa el algoritmo basado en Divide y Vencerás



4. Problema del Máximo de un vector

1 2 3 4 5

96

Máximo de un vector

Implementación

Actividad 5.12. Implementa el algoritmo basado en Divide y Vencerás

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

```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
int max(int[] vector) {
      return maxAux(vector,0, vector.length-1);
}
```


97

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.12. Implementa el algoritmo basado en Divide y Vencerás

i0: 7 iN: 12

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

```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
int max(int[] vector) {
      return maxAux(vector,0, vector.length-1);
}
```


98

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.12. Implementa el algoritmo basado en Divide y Vencerás

i0: 7 iN: 7

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

```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
int max(int[] vector) {
      return maxAux(vector,0, vector.length-1);
}
```

4. Problema del Máximo de un vector

1 2 3 4 5

99

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.12. Implementa el algoritmo basado en Divide y Vencerás

i0: 7 iN: 12

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

```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
int max(int[] vector) {
      return maxAux(vector,0, vector.length-1);
}
```

4. Problema del Máximo de un vector

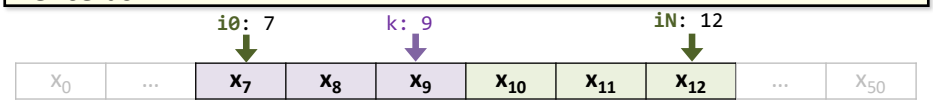
1 2 3 4 5

100

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.12. Implementa el algoritmo basado en Divide y Vencerás




```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
int max(int[] vector) {
      return maxAux(vector,0, vector.length-1);
}
```

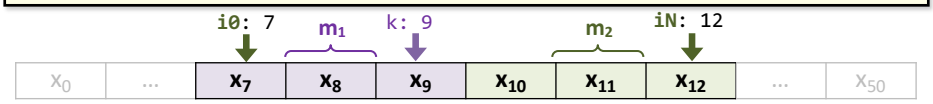
Dividir

101

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.12. Implementa el algoritmo basado en Divide y Vencerás




```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
int max(int[] vector) {
      return maxAux(vector,0, vector.length-1);
}
```

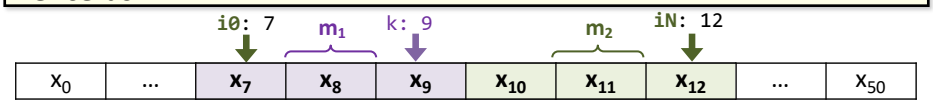
Conquistar

102

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.12. Implementa el algoritmo basado en Divide y Vencerás




```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
int max(int[] vector) {
      return maxAux(vector,0, vector.length-1);
}
```

Combinar

4. Problema del Máximo de un vector

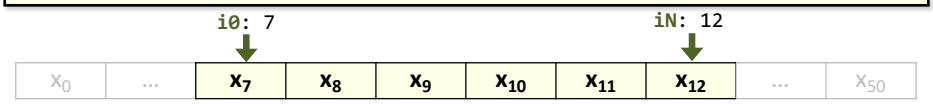
1 2 3 4 5

103

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.12. Implementa el algoritmo basado en Divide y Vencerás




```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
int max(int[] vector) {
      return maxAux(vector,0, vector.length-1);
}
```

4. Problema del Máximo de un vector

1 2 3 4 5

104

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.13. Calcula la complejidad del algoritmo

i0: 7 iN: 12

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

```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}

int max(int[] vector) {
      return maxAux(vector,0, vector.length-1);
}
```

4. Problema del Máximo de un vector

1 2 3 4 5

105

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.13. Calcula la complejidad del algoritmo

i0: 7 iN: 12

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

```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
```

Ecuación de Recurrencia

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

1 2 3 4 5

106

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.13. Calcula la complejidad del algoritmo

i0: 7 iN: 12

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

```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
```

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 2 \cdot T\left(\frac{N}{2}\right) + \Theta(N^0) & N > 1 \end{cases}$$

Teorema Maestro

(1er Caso)
$\text{Log}_2(2)=1 > 0$

1 2 3 4 5

107

Máximo de un vector

Algoritmo basado en Divide y Vencerás: Implementación

Actividad 5.13. Calcula la complejidad del algoritmo

i0: 7 iN: 12

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

```
int maxAux (int[] vector, int i0, int iN){
      if (i0 == iN)
          return vector[i0];
      else {
         int k = (i0 + iN) /2;
         int m1 = maxAux(vector, i0, k);
         int m2 = maxAux(vector, k+1, iN);
         return Math.max(m1, m2);
      }
}
```

Ecuación de Recurrencia

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

Teorema Maestro

(1er Caso)
$\text{Log}_2(2)=1 > 0$

Complejidad: $\Theta(N)$

1 2 3 4 5

108

Máximo de un vector

Algoritmo basado en Divide y Vencerás

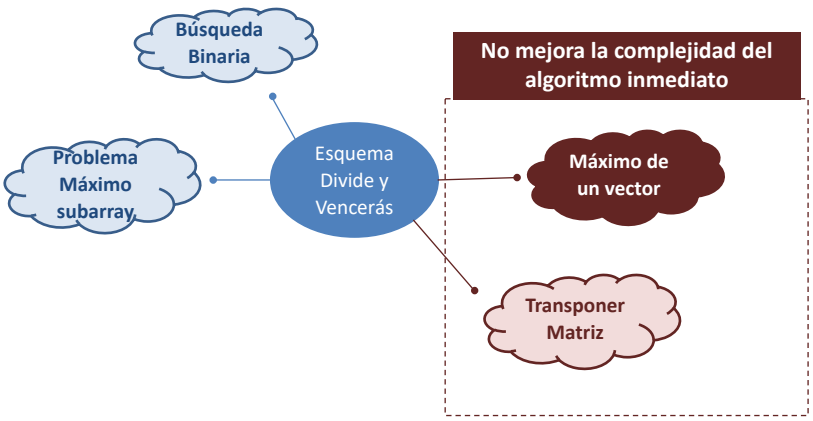
Actividad 5.14. Compara la complejidad del algoritmo sencillo con el algoritmo basado en Divide y Vencerás. ¿Qué conclusiones puedes sacar?

Algoritmo Sencillo	Un esquema basado en Divide y Vencerás no tiene por qué mejorar la estrategia natural	Algoritmo basado en Divide y Vencerás
$\Theta(N)$		$\Theta(N)$

109

Máximo de un vector

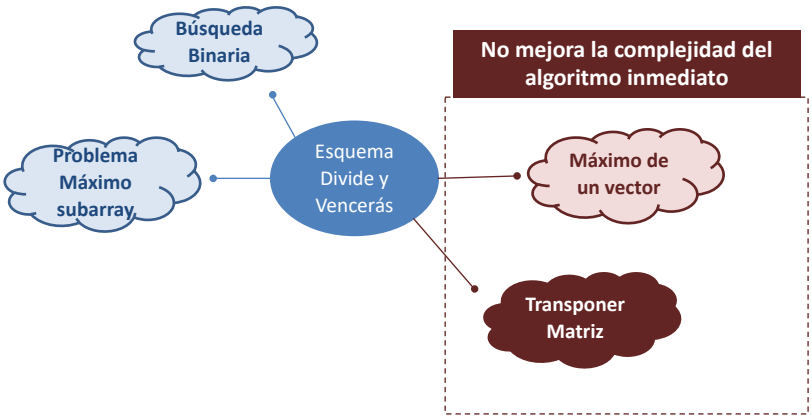
Algunos problemas planteados



110

Transponer una Matriz

Algunos problemas planteados



111

Transponer una Matriz

Enunciado del Problema

Dada una matriz cuadrada:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Transponerla:

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

112

Transponer una Matriz

Algoritmo Sencillo

Actividad 5.15. Diseña e implementa un algoritmo que transponga una matriz

```
void transponer(int[][] matriz){
  for (int f=0; f<matriz.length; f++)
    for (int c=0; c<f; c++) {
      int aux = matriz[f][c];
      matriz[f][c]=matriz[c][f];
      matriz[c][f]=aux;
    }
}
```


Transponer una Matriz

Algoritmo Sencillo

Actividad 5.16. Calcula la complejidad del Algoritmo

```
void transponer(int[][] matriz){
  for (int f=0; f<matriz.length; f++)
    for (int c=0; c<f; c++) {
      int aux = matriz[f][c];
      matriz[f][c]=matriz[c][f];
      matriz[c][f]=aux;
    }
}
```

Complejidad: $\Theta(N^2)$

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

Vamos a considerar una matriz cuadrada de dimensión 2^k

Fase Dividir **Fase Conquistar** **Fase Combinar**

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	**5**	3	**7**
2	6	**4**	8
9	**13**	11	**15**
10	14	**12**	16

1	5	**9**	**13**
2	6	**10**	**14**
3	**7**	11	15
4	**8**	12	16

5. Transponer una Matriz

1 2 3 4 5

115

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

Actividad 5.17. **Diseña e implementa el algoritmo basado en divide y vencerás para matrices cuadradas de dimensión 2^k.**

Fase Dividir **Fase Conquistar** **Fase Combinar**

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	5	3	7
2	6	4	8
9	13	11	15
10	14	12	16

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

5. Transponer una Matriz

1 2 3 4 5

116

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

```
void transponer(int[][] matriz){
   transponerAux(matriz, 0, matriz.length-1, 0, matriz[0].length-1);
}

void transponerAux(int[][] matriz, int f0, int fN, int c0, int cN){
    if (f0>=fN || c0>=cN)
       return;
    else {
       int fk = (f0 + fN)/2; int ck = (c0 + cN)/2;
       transponerAux(matriz, f0,fk, c0,ck);
       transponerAux(matriz, f0,fk, ck+1, cN);
       transponerAux(matriz, fk+1, fN, c0, ck);
       transponerAux(matriz, fk+1, fN, ck+1, cN);
       for (int f=0; f<fk-f0+1; f++)
         for (int c=0; c<ck-c0+1; c++){
           int a=matriz[f+fk+1][c+c0];
           matriz[f+fk+1][c+c0]=matriz[f+f0][c+ck+1];
           matriz[f+f0][c+ck+1] = a;
         }
    }
}
```

	c0	ck		cN
f0	1	2	3	4
fk	5	6	7	8
	9	10	11	12
fN	13	14	15	16

117

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

```
void transponerAux(int[][] matriz, int f0, int fN, int c0, int cN){
    if (f0>=fN || c0>=cN)
       return;
    else {
       int fk = (f0 + fN)/2; int ck = (c0 + cN)/2;
       transponerAux(matriz, f0,fk, c0,ck);
       transponerAux(matriz, f0,fk, ck+1, cN);
       transponerAux(matriz, fk+1, fN, c0, ck);
       transponerAux(matriz, fk+1, fN, ck+1, cN);
       for (int f=0; f<fk-f0+1; f++)
         for (int c=0; c<ck-c0+1; c++){
           int a=matriz[f+fk+1][c+c0];
           matriz[f+fk+1][c+c0]=matriz[f+f0][c+ck+1];
           matriz[f+f0][c+ck+1] = a;
         }
    }
}
```

Caso Base

118

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

```
void transponerAux(int[][] matriz, int f0, int fN, int c0, int cN){
    if (f0>=fN || c0>=cN)
       return;
    else {
       int fk = (f0 + fN)/2; int ck = (c0 + cN)/2;
       transponerAux(matriz, f0,fk, c0,ck);
       transponerAux(matriz, f0,fk, ck+1, cN);
       transponerAux(matriz, fk+1, fN, c0, ck);
       transponerAux(matriz, fk+1, fN, ck+1, cN);
       for (int f=0; f<fk-f0+1; f++)
         for (int c=0; c<ck-c0+1; c++){
           int a=matriz[f+fk+1][c+c0];
           matriz[f+fk+1][c+c0]=matriz[f+f0][c+ck+1];
           matriz[f+f0][c+ck+1] = a;
         }
    }
}
```

} Dividir

5. Transponer una Matriz

1 2 3 4 5

119

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

```
void transponerAux(int[][] matriz, int f0, int fN, int c0, int cN){
    if (f0>=fN || c0>=cN)
       return;
    else {
       int fk = (f0 + fN)/2; int ck = (c0 + cN)/2;
       transponerAux(matriz, f0,fk, c0,ck);
       transponerAux(matriz, f0,fk, ck+1, cN);
       transponerAux(matriz, fk+1, fN, c0, ck);
       transponerAux(matriz, fk+1, fN, ck+1, cN);
       for (int f=0; f<fk-f0+1; f++)
         for (int c=0; c<ck-c0+1; c++){
           int a=matriz[f+fk+1][c+c0];
           matriz[f+fk+1][c+c0]=matriz[f+f0][c+ck+1];
           matriz[f+f0][c+ck+1] = a;
         }
    }
}
```

} Conquistar

5. Transponer una Matriz

1 2 3 4 5

120

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

```
void transponerAux(int[][] matriz, int f0, int fN, int c0, int cN){
    if (f0>=fN || c0>=cN)
       return;
    else {
       int fk = (f0 + fN)/2; int ck = (c0 + cN)/2;
       transponerAux(matriz, f0,fk, c0,ck);
       transponerAux(matriz, f0,fk, ck+1, cN);
       transponerAux(matriz, fk+1, fN, c0, ck);
       transponerAux(matriz, fk+1, fN, ck+1, cN);
       for (int f=0; f<fk-f0+1; f++)
         for (int c=0; c<ck-c0+1; c++){
           int a=matriz[f+fk+1][c+c0];
           matriz[f+fk+1][c+c0]=matriz[f+f0][c+ck+1];
           matriz[f+f0][c+ck+1] = a;
         }
    }
}
```

Combinar

1 2 3 4 5

121

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

Actividad 5.18. Calcula la complejidad del algoritmo

```
void transponerAux(int[][] matriz, int f0, int fN, int c0, int cN){
    if (f0>=fN || c0>=cN)
       return;
    else {
       int fk = (f0 + fN)/2; int ck = (c0 + cN)/2;
       transponerAux(matriz, f0,fk, c0,ck);
       transponerAux(matriz, f0,fk, ck+1, cN);
       transponerAux(matriz, fk+1, fN, c0, ck);
       transponerAux(matriz, fk+1, fN, ck+1, cN);
       for (int f=0; f<fk-f0+1; f++)
         for (int c=0; c<ck-c0+1; c++){
           int a=matriz[f+fk+1][c+c0];
           matriz[f+fk+1][c+c0]=matriz[f+f0][c+ck+1];
           matriz[f+f0][c+ck+1] = a;
         }
    }
}
```

1 2 3 4 5

122

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

Actividad 5.18. Calcula la complejidad del algoritmo

```
void transponerAux(int[][] matriz, int f0, int fN, int c0, int cN){
    if (f0>=fN || c0>=cN)
       return;
    else {
       int fk = (f0 + fN)/2; int ck = (c0 + cN)/2;
       transponerAux(matriz, f0,fk, c0,ck);
       transponerAux(matriz, f0,fk, ck+1, cN);
       transponerAux(matriz, fk+1, fN, c0, ck);
       transponerAux(matriz, fk+1, fN, ck+1, cN);
       for (int f=0; f<fk-f0+1; f++)
         for (int c=0; c<ck-c0+1; c++){
           int a=matriz[f+fk+1][c+c0];
           matriz[f+fk+1][c+c0]=matriz[f+f0][c+ck+1];
           matriz[f+f0][c+ck+1] = a;
         }
    }
}
```

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 4 \cdot T\left(\frac{N}{2}\right) + \Theta(N^2) & N > 1 \end{cases}$$

1 2 3 4 5

123

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

Actividad 5.18. Calcula la complejidad del algoritmo

```
void transponerAux(int[][] matriz, int f0, int fN, int c0, int cN){
    if (f0>=fN || c0>=cN)
       return;
    else {
       int fk = (f0 + fN)/2; int ck = (c0 + cN)/2;
       transponerAux(matriz, f0,fk, c0,ck);
       transponerAux(matriz, f0,fk, ck+1, cN);
       transponerAux(matriz, fk+1, fN, c0, ck);
       transponerAux(matriz, fk+1, fN, ck+1, cN);
       for (int f=0; f<fk-f0+1; f++)
         for (int c=0; c<ck-c0+1; c++){
           int a=matriz[f+fk+1][c+c0];
           matriz[f+fk+1][c+c0]=matriz[f+f0][c+ck+1];
           matriz[f+f0][c+ck+1] = a;
         }
    }
}
```

Teorema Maestro

(2º Caso)
$\text{Log}_2(4)=2$

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 4 \cdot T\left(\frac{N}{2}\right) + \Theta(N^2) & N > 1 \end{cases}$$

1 2 3 4 5

124

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

Actividad 5.18. Calcula la complejidad del algoritmo

```
void transponerAux(int[][] matriz, int f0, int fN, int c0,
    if (f0>=fN || c0>=cN)
       return;
    else {
       int fk = (f0 + fN)/2; int ck = (c0 + cN)/2;
       transponerAux(matriz, f0,fk, c0,ck);
       transponerAux(matriz, f0,fk, ck+1, cN);
       transponerAux(matriz, fk+1, fN, c0, ck);
       transponerAux(matriz, fk+1, fN, ck+1, cN);
       for (int f=0; f<fk-f0+1; f++)
         for (int c=0; c<ck-c0+1; c++){
           int a=matriz[f+fk+1][c+c0];
           matriz[f+fk+1][c+c0]=matriz[f+f0][c+ck+1];
           matriz[f+f0][c+ck+1] = a;
         }
    }
}
```

Complejidad: $\Theta(N^2 \cdot \log N)$

Teorema Maestro

(2º Caso)
$\text{Log}_2(4)=2$

Ecuación de Recurrencia

$$T(N) = \begin{cases} \Theta(1) & N = 1 \\ 4 \cdot T\left(\frac{N}{2}\right) + \Theta(N^2) & N > 1 \end{cases}$$

125

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

Actividad 5.19. Compara la complejidad del algoritmo sencillo con el algoritmo basado en Divide y Vencerás. ¿Qué conclusiones puedes sacar?

Algoritmo Sencillo	Un esquema basado en Divide y Vencerás puede ser **peor** que la estrategia natural	Algoritmo basado en Divide y Vencerás
$\Theta(N^2)$		$\Theta(N^2 \cdot \log N)$

126

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

Actividad 5.19. **Compara la complejidad del algoritmo sencillo con el algoritmo basado en Divide y Vencerás. ¿Qué conclusiones puedes sacar?**

Algoritmo Sencillo	Un esquema basado en Divide y Vencerás puede ser **peor** que la estrategia natural	Algoritmo basado en Divide y Vencerás
$\Theta(N^2)$		$\Theta(N^2 \cdot \log N)$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

127

Transponer una Matriz

Algoritmo basado en Divide y Vencerás

Actividad 5.19. **Compara la complejidad del algoritmo sencillo con el algoritmo basado en Divide y Vencerás. ¿Qué conclusiones puedes sacar?**

Algoritmo Sencillo	Un esquema basado en Divide y Vencerás puede ser **peor** que la estrategia natural	Algoritmo basado en Divide y Vencerás
$\Theta(N^2)$		$\Theta(N^2 \cdot \log N)$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	5	3	7
2	6	4	8
9	13	11	15
10	14	12	16

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

128