

# Algoritmos y estructuras de datos

## Dividir y Conquistar

CEIS

Escuela Colombiana de Ingeniería

2022-2

# Agenda

## 1 Merge Sort

Formulación

Diseño

Análisis

## 2 Teorema Maestro

## 3 Subarreglo máximo

Formulación

Diseño

## 4 Busqueda

Formulación

Busqueda Secuencial

Busqueda Binaria

## 5 Ejercicios

# Agenda

## 1 Merge Sort

Formulación

Diseño

Análisis

## 2 Teorema Maestro

## 3 Subarreglo máximo

Formulación

Diseño

## 4 Busqueda

Formulación

Busqueda Secuencial

Busqueda Binaria

## 5 Ejercicios

## Formulación

- ▶ **Entrada:** Una secuencia de  $n$  números  $\langle a_1, a_2, \dots, a_n \rangle$
- ▶ **Salida:** Una permutación  $\langle a'_1, a'_2, \dots, a'_n \rangle$  tal que  
 $a'_1 \leq a'_2 \leq \dots \leq a'_n$

**Algoritmo Correcto:** Una algoritmo se dice correcto si para cada instancia de la entrada,

entrega una salida correcta.

# Dividir y Conquistar

- ▶ Dividir : El problema se divide en diversos subproblemas similares al original pero de un tamaño menor.
- ▶ Conquistar: Se resuelven los subproblemas de forma recursiva. Si el tamaño del problema es lo suficientemente pequeño se resuelve de forma directa sin recursión(Caso base).
- ▶ Combinar: Se unen las soluciones de los subproblemas para obtener una solución al problema original.

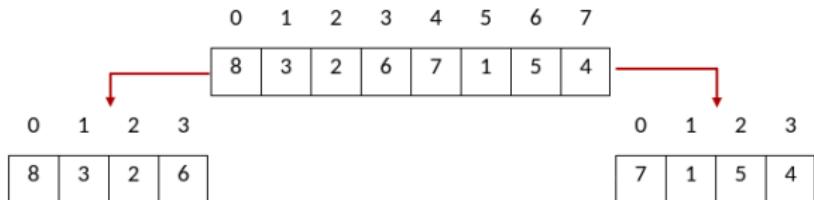
## Dividir y Conquistar para Ordenar

- ▶ Es un algoritmo de ordenamiento basado en el paradigma ‘Divide y conquista’. Para ordenar un arreglo A, merge sort realiza las siguientes actividades:
  - ▶ Dividir: Determina la mitad del arreglo
  - ▶ Conquistar: Ordena recurrentemente el arreglo para la primera mitad y para la segunda mitad.
  - ▶ Combinar: Mezcla ordenadamente los dos subarreglos ordenados.

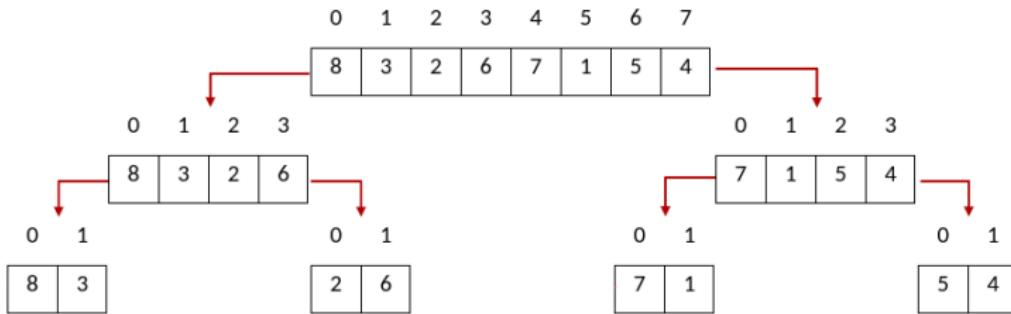
# Dividir y Conquistar para Ordenar

0	1	2	3	4	5	6	7
8	3	2	6	7	1	5	4

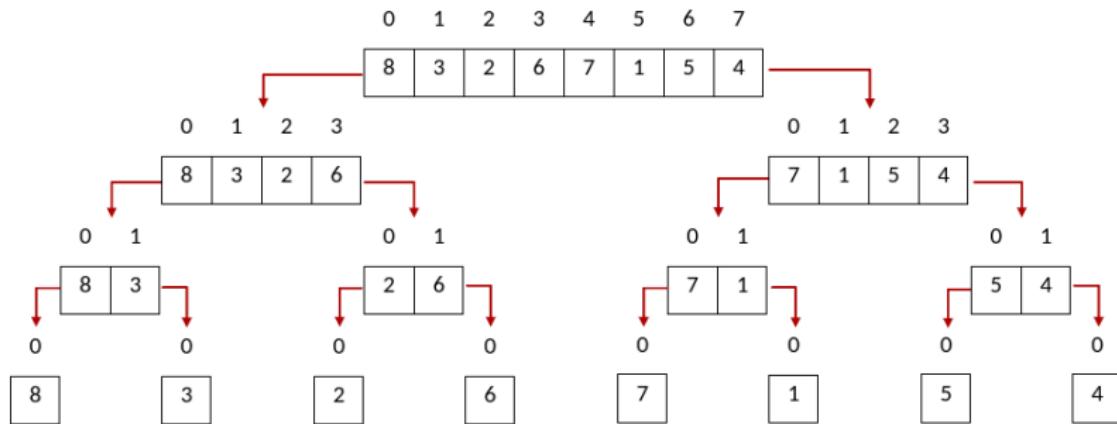
# Dividir y Conquistar para Ordenar



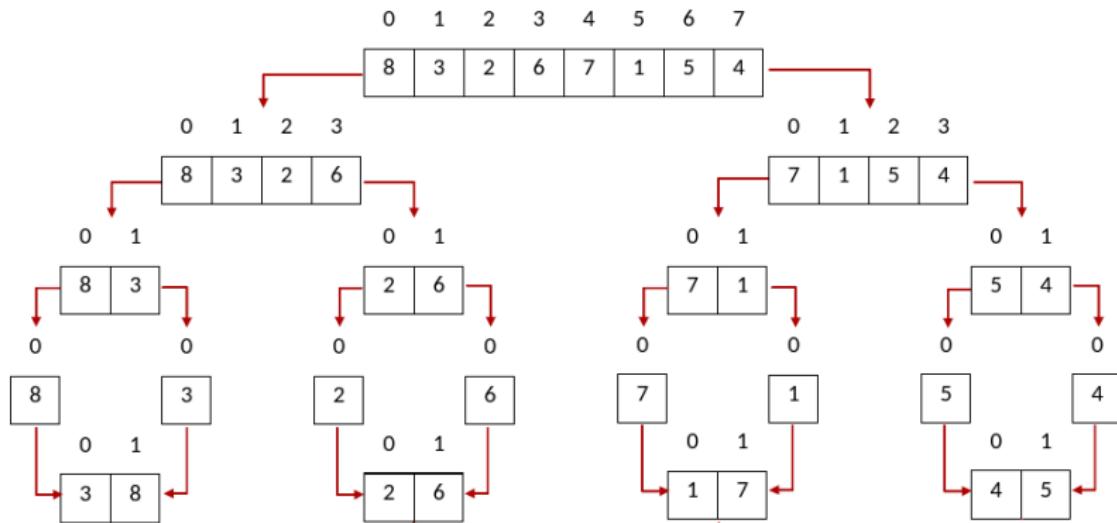
# Dividir y Conquistar para Ordenar



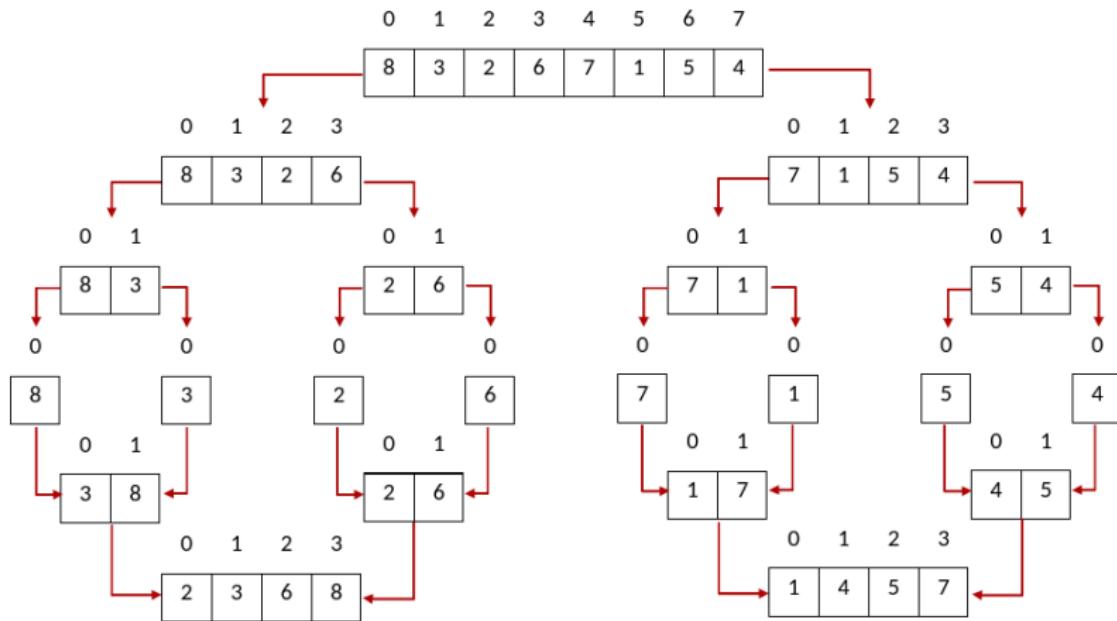
# Dividir y Conquistar para Ordenar



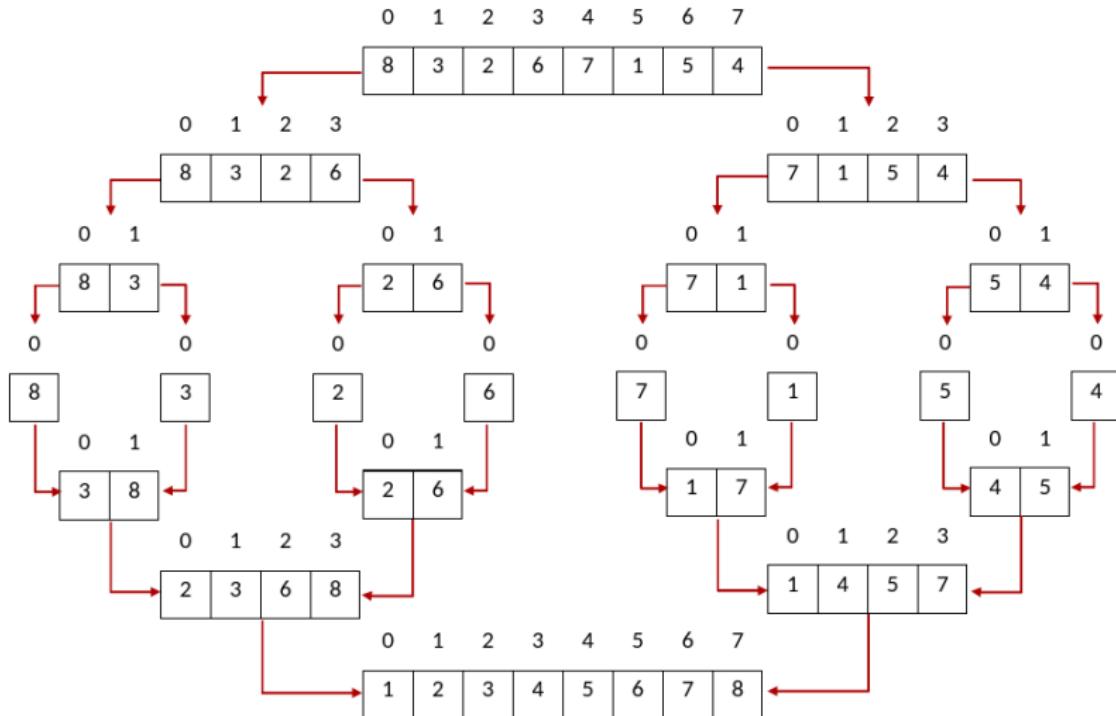
# Dividir y Conquistar para Ordenar



# Dividir y Conquistar para Ordenar



# Dividir y Conquistar para Ordenar



## Merge Sort

**MERGE-SORT( $A, p, r$ )**

- 1   **if**  $p < r$
- 2        $q = \lfloor (p + r)/2 \rfloor$
- 3       **MERGE-SORT( $A, p, q$ )**
- 4       **MERGE-SORT( $A, q + 1, r$ )**
- 5       **MERGE( $A, p, q, r$ )**

Para ordenar una secuencia A, usamos:

**MERGE-SORT( $A, 1, A.length$ )**

# Merge

MERGE( $A, p, q, r$ )

```
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$ 
4  be new arrays
5  for  $i = 1$  to  $n_1$ 
6     $L[i] = A[p + i - 1]$ 
7  for  $j = 1$  to  $n_2$ 
8     $R[j] = A[q + j]$ 
9   $L[n_1 + 1] = \infty$ 
10  $R[n_2 + 1] = \infty$ 
11  $i = 1$ 
12  $j = 1$ 
13 for  $k = p$  to  $r$ 
14   if  $L[i] \leq R[j]$ 
15      $A[k] = L[i]$ 
16      $i = i + 1$ 
17   else  $A[k] = R[j]$ 
         $j = j + 1$ 
```

# Merge

## Ejemplo

MERGE( $A, p, q, r$ )

```
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$ 
4  be new arrays
5  for  $i = 1$  to  $n_1$ 
6     $L[i] = A[p + i - 1]$ 
7  for  $j = 1$  to  $n_2$ 
8     $R[j] = A[q + j]$ 
9   $L[n_1 + 1] = \infty$ 
10  $R[n_2 + 1] = \infty$ 
11  $i = 1$ 
12  $j = 1$ 
13 for  $k = p$  to  $r$ 
14   if  $L[i] \leq R[j]$ 
15      $A[k] = L[i]$ 
16      $i = i + 1$ 
17   else  $A[k] = R[j]$ 
18      $j = j + 1$ 
```

$A$	8	9	10	11	12	13	14	15	16	17	...
	...	2	4	5	7	1	2	3	6	...	
$k$											
$L$	1	2	3	4	5						
	2	4	5	7	$\infty$						
$i$											
$R$	1	2	3	4	5						
	1	2	3	6	$\infty$						
$j$											

(a)





# Merge

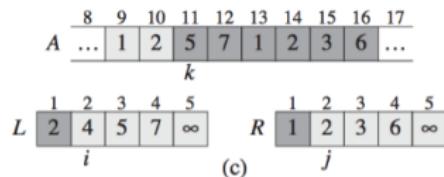
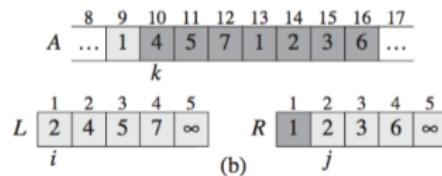
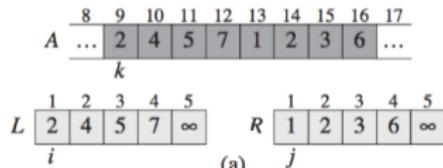
## Ejemplo

MERGE( $A, p, q, r$ )

```

1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$ 
4  be new arrays
5  for  $i = 1$  to  $n_1$ 
6     $L[i] = A[p + i - 1]$ 
7  for  $j = 1$  to  $n_2$ 
8     $R[j] = A[q + j]$ 
9   $L[n_1 + 1] = \infty$ 
10  $R[n_2 + 1] = \infty$ 
11  $i = 1$ 
12  $j = 1$ 
13 for  $k = p$  to  $r$ 
14   if  $L[i] \leq R[j]$ 
15      $A[k] = L[i]$ 
16      $i = i + 1$ 
17   else  $A[k] = R[j]$ 
            $j = j + 1$ 

```



# Merge

MERGE( $A, p, q, r$ )

```
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$ 
4  be new arrays
5  for  $i = 1$  to  $n_1$ 
6     $L[i] = A[p + i - 1]$ 
7  for  $j = 1$  to  $n_2$ 
8     $R[j] = A[q + j]$ 
9   $L[n_1 + 1] = \infty$ 
10  $R[n_2 + 1] = \infty$ 
11  $i = 1$ 
12  $j = 1$ 
13 for  $k = p$  to  $r$ 
14   if  $L[i] \leq R[j]$ 
15      $A[k] = L[i]$ 
16      $i = i + 1$ 
17   else  $A[k] = R[j]$ 
         $j = j + 1$ 
```

## Invariante

$A[p \dots k - 1]$  contiene los  $k - p$  elementos más pequeños de  $L[1 \dots n_1 + 1]$  y  $R[1 \dots n_2 + 1]$ , ordenados ascendenteamente.

$L[i]$  y  $R[j]$  son los elementos más pequeños en los arreglos respectivos que aún no han sido copiados en  $A$ .

- ¿Inicialización?
- ¿Estabilidad?
- ¿Terminación?

# Merge Sort

MERGE-SORT( $A, p, r$ )

```
1  if  $p < r$ 
2       $q = \lfloor (p + r)/2 \rfloor$ 
3      MERGE-SORT( $A, p, q$ )
4      MERGE-SORT( $A, q + 1, r$ )
5      MERGE( $A, p, q, r$ )
```

- ▶ El algoritmo Merge-Sort( $A, p, r$ ) tiene dos caminos de ejecución:
  - ▶  $p \geq r$ : El algoritmo no hace nada, entonces solo se considera el costo de la comparación que se supone constante.
  - ▶  $p < r$ : El algoritmo se invoca recurrentemente con dos instancias del problema cuyos tamaños corresponden a la mitad del problema inicial y luego mezcla las respuestas parciales.

$$T_{ms}(n) = \begin{cases} c_1 & , \text{ si } n = 1, \\ c_2 + 2 \cdot T_{ms}(n/2) + T_m(n) & , \text{ si } n > 1. \end{cases}$$

# Agenda

## 1 Merge Sort

Formulación

Diseño

Análisis

## 2 Teorema Maestro

## 3 Subarreglo máximo

Formulación

Diseño

## 4 Busqueda

Formulación

Busqueda Secuencial

Busqueda Binaria

## 5 Ejercicios

## Teorema Maestro

El teorema maestro provee una receta para resolver recurrencias de la forma:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Donde  $a \geq 1$  y  $b > 1$  son constantes y  $f(n)$  es una función asintótica positiva.  
Para usar el teorema maestro se deben memorizar tres casos.

## Teorema Maestro

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Donde  $a \geq 1$  y  $b > 1$  son constantes  
y  $f(n)$  es una función asintótica positiva.

- Cuando  $f(n) = O(n^c)$  y  $c < \log_b a$   
entonces  $T(n) = \Theta(n^{\log_b a})$
- Cuando  $f(n) = \Theta(n^c \log^k n)$  donde  $c = \log_b a$  y  $k \geq 0$   
entonces  $T(n) = \Theta(n^c \log^{k+1} n)$
- Cuando  $f(n) = \Omega(n^c)$  donde  $c > \log_b a$  y  $af\left(\frac{n}{b}\right) \leq kf(n)$  para algún  $k < 1$ ,  
entonces  $T(n) = \Theta(f(n))$

## Teorema Maestro

$$T(n) = 9T(n/3) + n$$

$$T(n) = T(2n/3) + 1$$

$$T(n) = 3T(n/4) + n \lg n$$

## Teorema Maestro

MERGE-SORT( $A, p, r$ )

```
1  if  $p < r$ 
2       $q = \lfloor (p + r)/2 \rfloor$ 
3      MERGE-SORT( $A, p, q$ )
4      MERGE-SORT( $A, q + 1, r$ )
5      MERGE( $A, p, q, r$ )
```

- ▶ El algoritmo Merge-Sort( $A, p, r$ ) tiene dos caminos de ejecución:
  - ▶  $p \geq r$ : El algoritmo no hace nada, entonces solo se considera el costo de la comparación que se supone constante.
  - ▶  $p < r$ : El algoritmo se invoca recurrentemente con dos instancias del problema cuyos tamaños corresponden a la mitad del problema inicial y luego mezcla las respuestas parciales.

$$T_{ms}(n) = \begin{cases} c_1 & , \text{ si } n = 1, \\ c_2 + 2 \cdot T_{ms}(n/2) + T_m(n) & , \text{ si } n > 1. \end{cases}$$

¿Complejidad?

# Agenda

## 1 Merge Sort

Formulación

Diseño

Análisis

## 2 Teorema Maestro

## 3 Subarreglo máximo

Formulación

Diseño

## 4 Busqueda

Formulación

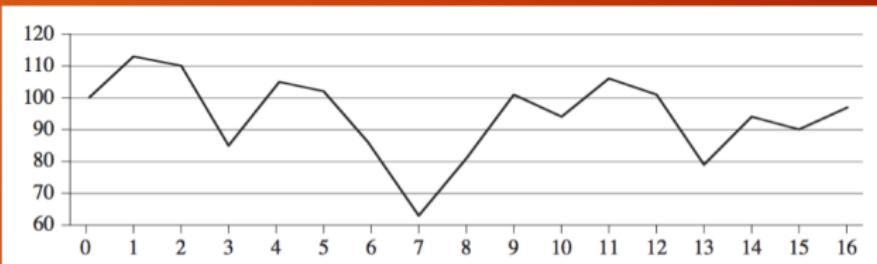
Busqueda Secuencial

Busqueda Binaria

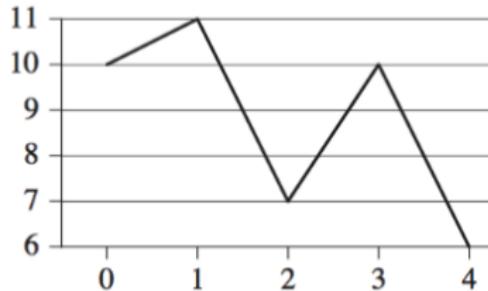
## 5 Ejercicios

## Formulación

Supongamos que tenemos la oportunidad de invertir en el mercado bursátil, la idea principal es maximizar nuestra ganancia al invertir cuando la acción está barata y venderla cuando ha aumentado de precio. Adicionalmente tenemos la información del precio de la acción para un lapso de tiempo.



## Diseño

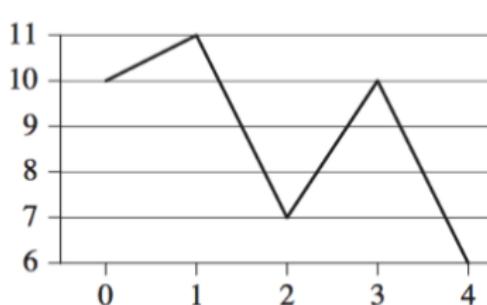


Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	-4	3	-4

### Estrategia

Tal vez es posible maximizar la ganancia si se compra al menor precio o si se vende al mayor precio.

# Diseño



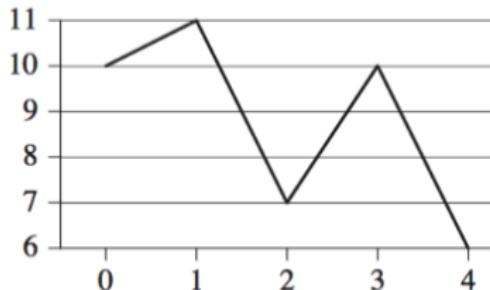
Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	-4	3	-4

## Fuerza Bruta

Se puede buscar una solución por fuerza bruta, tratando cada una de las posibles parejas para comprar y vender, donde la fecha de compra precede a la fecha de venta.

Para un periodo de  $n$  días, tenemos  $\binom{n}{2}$  posibles parejas. Evaluar  $\binom{n}{2}$  es  $\Theta(n^2)$

# Diseño



Day	0	1	2	3	4
Price	10	11	7	10	6
Change	1	-4	3	-4	

## Fuerza Bruta

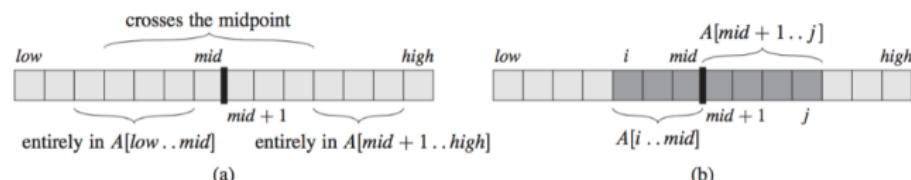
Podemos buscar una secuencia de días sobre los que el cambio neto desde el primer día hasta el último es máximo.

En vez de pensar en los precios de forma individual, consideraremos el cambio diario en el precio. La diferencia en precios al inicio y después del día  $i$ .

## Dividir y Conquistar

Tenemos tres casos donde podemos hallar el sub-arreglo de suma máxima:

- Enteramente en el sub-arreglo  $A[\text{low}..\text{mid}]$
- Enteramente en el sub-arreglo  $A[\text{mid}..\text{high}]$
- Cruzando el punto intermedio  $\text{low} \leq i < j \leq \text{high}$



FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )

```
1  left-sum = -∞
2  sum = 0
3  for i = mid downto low
4      sum = sum + A[i]
5      if sum > left-sum
6          left-sum = sum
7          max-left = i
8  right-sum = -∞
9  sum = 0
10 for j = mid + 1 to high
11     sum = sum + A[j]
12     if sum > right-sum
13         right-sum = sum
14         max-right = j
15 return (max-left, max-right, left-sum + right-sum)
```

# Diseño

FIND-MAXIMUM-SUBARRAY( $A, low, high$ )

```
1  if  $high == low$ 
2      return ( $low, high, A[low]$ )           // base case: only one element
3  else  $mid = \lfloor (low + high)/2 \rfloor$ 
4      ( $left-low, left-high, left-sum$ ) =
            FIND-MAXIMUM-SUBARRAY( $A, low, mid$ )
5      ( $right-low, right-high, right-sum$ ) =
            FIND-MAXIMUM-SUBARRAY( $A, mid + 1, high$ )
6      ( $cross-low, cross-high, cross-sum$ ) =
            FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )
7      if  $left-sum \geq right-sum$  and  $left-sum \geq cross-sum$ 
8          return ( $left-low, left-high, left-sum$ )
9      elseif  $right-sum \geq left-sum$  and  $right-sum \geq cross-sum$ 
10         return ( $right-low, right-high, right-sum$ )
11     else return ( $cross-low, cross-high, cross-sum$ )
```

# Agenda

## 1 Merge Sort

Formulación

Diseño

Análisis

## 2 Teorema Maestro

## 3 Subarreglo máximo

Formulación

Diseño

## 4 Busqueda

Formulación

Busqueda Secuencial

Busqueda Binaria

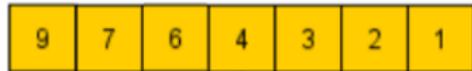
## 5 Ejercicios

## Formulación

Buscar la posición en la que está un elemento en una secuencia.

- Problema: Se desea hallar el índice  $i$  tal que  $A[i] = k$ . Si este índice  $i$  no existe, se retorna el valor de -1.
- Entrada: Un arreglo  $A[1..n]$  ordenado de forma descendente y un valor  $k$ .
- Salida: El valor del índice  $i$  tal que  $A[i] = k$  ó -1 si no  $k$  no existe en el arreglo  $A[1..n]$

# Busqueda Secuencial



```
def lineal_search(A, k):
    ans = -1
    for i in range(0, len(A)):
        if A[i] == k:
            return i
    return ans
```

## Busqueda Secuencial

```
def lineal_search(A, k):
    ans = -1
    for i in range(0, len(A)):
        if A[i] == k:
            return i
    return ans
```

- Diseño: ¿Invariantes?

## Busqueda Secuencial

```
def lineal_search(A, k):
    ans = -1
    for i in range(0, len(A)):
        if A[i] == k:
            return i
    return ans
```

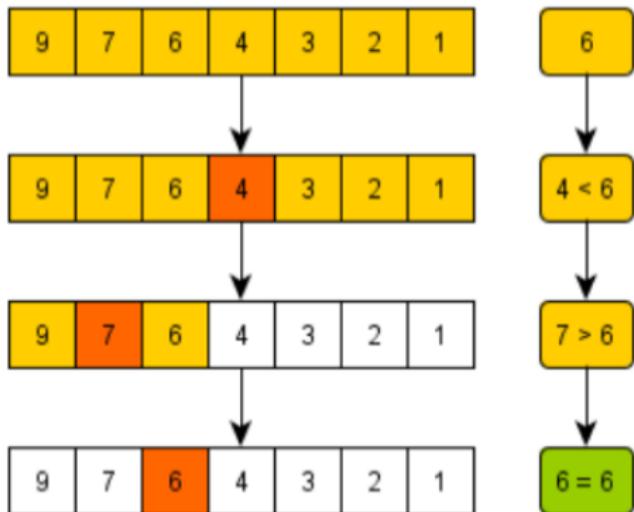
- Diseño: ¿Invariantes?
- Análisis: ¿Complejidad?

## Busqueda Secuencial

```
def lineal_search(A, k):
    ans = -1
    for i in range(0, len(A)):
        if A[i] == k:
            return i
    return ans
```

- Diseño: ¿Invariantes?
- Análisis: ¿Complejidad?
- ¿Aprovechar el ordenamiento?

# Busqueda Binaria



```
def binary_search(A, k):  
    low = 0  
    high = len(A)  
    while low < high:  
        mid = low + (high - low) // 2  
        if A[mid] == k:  
            return mid  
        else:  
            if A[mid] > k:  
                high = mid  
            else:  
                low = mid + 1
```

## Busqueda Secuencial

```
def binary_search(A, k):
    low = 0
    high = len(A)
    while low < high:
        mid = low + (high - low) // 2
        if A[mid] == k:
            return mid
        else:
            if A[mid] > k:
                high = mid
            else:
                low = mid + 1
```

- Diseño: ¿Invariantes?

# Busqueda Secuencial

```
def binary_search(A, k):
    low = 0
    high = len(A)
    while low < high:
        mid = low + (high - low) // 2
        if A[mid] == k:
            return mid
        else:
            if A[mid] > k:
                high = mid
            else:
                low = mid + 1
```

- Diseño: ¿Invariantes?
- Análisis: ¿Complejidad?

# Agenda

## 1 Merge Sort

Formulación

Diseño

Análisis

## 2 Teorema Maestro

## 3 Subarreglo máximo

Formulación

Diseño

## 4 Busqueda

Formulación

Busqueda Secuencial

Busqueda Binaria

## 5 Ejercicios

# Ejercicios

1. Dado un arreglo de **N** enteros, cuyos valores van en decremento y luego incremento, encontrar el menor número en el arreglo.

$$N = [2, 1, 2, 3, 4]$$

**min = 1**

$$N = [8, 5, 4, 3, 4, 10]$$

**min = 3**

2. Dado un arreglo de **N-1** enteros ordenados, cuyos valores están en el rango 1 a **N**, encontrar el entero faltante dado que uno de ellos no está presente en el arreglo.

$$N = [1, 2, 3, 4, 6, 7, 8]$$

**missing = 5**

$$N = [1, 2, 3, 4, 5, 6, 8, 9]$$

**missing = 7**

3. Diseñar un algoritmo D&Q para calcular el exponente de un número  $a^n$

4. Aplicar **mergesort** para ordenar 'ALGORITMO' en orden alfabético

5. Dado un número **X**, encontrar el menor número **N** tal que la suma de los bits de cada número desde 1 hasta **N** sea al menos **X**.

$$X = 5$$

**N = 4**

```
sum_bits(1) = 1
sum_bits(2) = 1
sum_bits(3) = 2
sum_bits(4) = 1
```