

# Dividir y conquistar

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos  
Tecnológico de Monterrey

*pperezm@tec.mx*

01-2020

# Contenido

Introducción

Ejemplos clásicos

Otros ejemplos

Más ejemplos

# Definición

Es una técnica que permite encontrar la solución de un problema descomponiéndolo en subproblemas más pequeños (dividir) y que tienen la misma naturaleza del problema original, es decir, son similares a este. Luego resuelve cada uno de los subproblemas recursivamente hasta llegar a problemas de solución trivial o conocida con antelación (conquistar) para, finalmente, unir las diferentes soluciones (combinar) y así conformar la solución global al problema.

## Forma general

---

### Procedure 1 DIVIDE\_AND\_CONQUER

---

**Input:**  $X$

**if**  $X$  is simple or known **then**

**return**  $SOLUTION(X)$

**else**

    Decompose  $X$  into smaller problems  $x_1, x_2, \dots, x_n$

**for**  $i \leftarrow 1$  **to**  $n$  **do**

$y_i \leftarrow DIVIDE\_AND\_CONQUER(x_i)$

**end for**

    Combine the  $y_i$  to get the  $Y$  that is solution of  $X$

**return**  $Y$

**end if**

---

# Búsqueda binaria

Buscar un elemento  $x$  en un arreglo ordenado  $A$  de  $n$  elementos.

---

## Procedure 2 BINARY\_SEARCH

---

**Input:**  $A$  : Array,  $low$  : Index,  $high$  : Index,  $k$  : Key

**if**  $low > high$  **then**

**return**  $-1$

**else**

$mid \leftarrow \text{FLOOR}((high + low)/2)$

**if**  $k = A[mid]$  **then**

**return**  $mid$

**else if**  $k < A[mid]$  **then**

**return**  $\text{BINARY\_SEARCH}(A, low, mid - 1, key)$

**else if**  $k > A[mid]$  **then**

**return**  $\text{BINARY\_SEARCH}(A, mid + 1, high, key)$

**end if**

**end if**

---

# Permutaciones

Hallar todas las permutaciones de un número. Por ejemplo, las permutaciones de 123 son: {123, 231, 321, 312, 132, 213, 123}.

---

### Procedure 3 PERMUTATION

---

Input:  $S : \text{String}$ ,  $pos : \text{Integer}$

```
if  $pos > 1$  then
  for  $i \leftarrow 1$  to  $pos$  do
     $SWAP(number, i, pos)$ 
     $PERMUTATION(number, pos - 1)$ 
     $SWAP(number, i, pos)$ 
  end for
else
  print  $S$ 
end if
```

---



# Torres de Hanoi

En este problema hay tres ejes verticales. En uno de los ejes se acomoda un número indeterminado de discos, todos de diferente tamaño y ordenados de abajo hacia arriba del más grande al más pequeño.

El reto consiste en mover todos los discos del eje en el que se encuentran a un eje destino utilizando el otro eje como auxiliar., de acuerdo con las siguientes reglas: <sup>a</sup>

- ▶ Solo se puede mover un disco a la vez.
- ▶ Solo se pueden mover los discos que están en los toques de los ejes.
- ▶ No puede quedar un disco más grande sobre uno más pequeño.



# Torres de Hanoi

En este problema hay tres ejes verticales. En uno de los ejes se acomoda un número indeterminado de discos, todos de diferente tamaño y ordenados de abajo hacia arreglo del más grande al más pequeño.



El reto consiste en mover todos los discos del eje en el que se encuentran a un eje destino utilizando el otro eje como auxiliar., de acuerdo con las siguientes reglas: <sup>1</sup>

- ▶ Solo se puede mover un disco a la vez.
- ▶ Solo se pueden mover los discos que están en los topes de los ejes.
- ▶ No puede quedar un disco más grande sobre uno más pequeño.

---

<sup>1</sup><https://goo.gl/bsLeq3>

El problema de pasar  $n$  discos del eje inicial al eje final se puede dividir en el problema de pasar  $n - 1$  discos del inicial a un eje auxiliar, luego pasar un disco al poste final y finalmente pasar los  $n - 1$  del eje auxiliar al final (con el mismo algoritmo).

---

## Procedure 4 HANOI

---

**Input:**  $n : \text{Integer}, \text{start} : \text{Index}, \text{aux} : \text{Index}, \text{end} : \text{Index}$

if  $n > 1$  then

$\text{HANOI}(n - 1, \text{start}, \text{end}, \text{aux})$

    print Move from start to end

$\text{HANOI}(n - 1, \text{aux}, \text{start}, \text{end})$

end if

---

# Exponenciación rápida

Dados dos números,  $x$  y  $n$ , calcular el resultado de  $x^n$ , haciendo uso de la técnica de dividir y conquistar.

---

## Procedure 5 FAST\_POW

---

**Input:**  $x : \text{Real}, n : \text{Integer}$

**if**  $n < 0$  **then**

**return**  $\text{FAST\_POW}(1/x, -n)$

**else if**  $n == 0$  **then**

**return** 1

**else if**  $n == 1$  **then**

**return**  $x$

**else if**  $n \bmod 2 = 0$  **then**

**return**  $\text{FAST\_POW}(x * x, n/2)$

**else if**  $n \bmod 2 = 1$  **then**

**return**  $x * \text{FAST\_POW}(x * x, (n - 1)/2)$

**end if**

---

## Prefijo común más largo

Dado un conjunto de cadenas, encontrar el prefijo común más largo. Por ejemplo, dadas la siguiente cadenas: “geeksforgeeks”, “geeks”, “geek”, “geezer”, el resultado esperado es: “gee”.<sup>2</sup>

---

<sup>2</sup><https://goo.gl/rqjV76>



---

## Procedure 6 FIND\_PREFIX

---

**Input:**  $A : \text{String}, B : \text{String}$

$result \leftarrow ""$

$i \leftarrow 1$

$j \leftarrow 1$

**while**  $i < A.length$  **and**  $j < B.length$  **do**

**if**  $A[i] \neq B[j]$  **then**

**break**

**end if**

$result \leftarrow result + A[i]$

$i \leftarrow i + 1$

$j \leftarrow j + 1$

**end while**

**return**  $result$

---

---

## Procedure 7 COMMON\_PREFIX

---

**Input:**  $A$  : Array,  $low$  : Index,  $high$  : Index

```
if  $low == high$  then
    return  $A[low]$ 
end if
if  $low < high$  then
     $mid \leftarrow FLOOR((high + low)/2)$ 
     $str1 \leftarrow COMMON\_PREFIX(A, low, mid)$ 
     $str2 \leftarrow COMMON\_PREFIX(A, mid + 1, high)$ 
    return  $FIND\_PREFIX(str1, str2)$ 
end if
```

---

## Contando inversiones

Dado arreglo de números enteros distintos,  $A$ , determinar el número de inversiones que existen. Decimos que dos índices  $i < j$  forma una inversión si  $A[i] > A[j]$ .

El número de inversiones de un arreglo indica: a qué distancia (o qué tan cerca) está el arreglo de estar ordenado. Si el arreglo ya está ordenado, el conteo de inversión es 0. Si el arreglo está ordenado en orden inverso, el conteo de inversión es el máximo.

Ejemplo: ¿cuántas inversiones hay en el siguiente arreglo: 2, 4, 1, 3, 5? Tiene tres inversiones (2, 1), (4, 1), (4, 3).<sup>3</sup>

---

<sup>3</sup><https://goo.gl/DxqxBe>



---

## Procedure 9 MERGE\_AND\_COUNT

---

**Input:**  $A, B : \text{Array}, \text{low}, \text{mid}, \text{high} : \text{Index}$

...

**while**  $\text{left} \leq \text{mid}$  AND  $\text{right} \leq \text{high}$  **do**

**if**  $A[\text{left}] < A[\text{right}]$  **then**

$B[i] \leftarrow A[\text{left}]$

$\text{left} \leftarrow \text{left} + 1$

**else**

$B[j] \leftarrow A[\text{right}]$

$\text{right} \leftarrow \text{right} + 1$

$\text{count} \leftarrow \text{count} + (\text{mid} - \text{left})$

**end if**

$i \leftarrow i + 1$

**end while**

...

**return**  $\text{count}$

---

## Par más cercano

Dado un arreglo de  $N$  puntos en un plano, el problema es encontrar el par de puntos más cercano del arreglo. Este problema lo podemos encontrar en una serie de aplicaciones. Por ejemplo, en el control de tráfico aéreo, se requiere monitorear los aviones que se acercan demasiado, ya que esto puede indicar una posible colisión.

La solución a fuerza bruta, de  $O(n^2)$ , calcula la distancia existente entre cada par de puntos y devuelve el más cercano. Podemos calcular la distancia más cercana en  $O(n \log n)$  utilizando la estrategia de Dividir y Conquistar.<sup>4</sup>

---

<sup>4</sup><https://goo.gl/36zwaA>

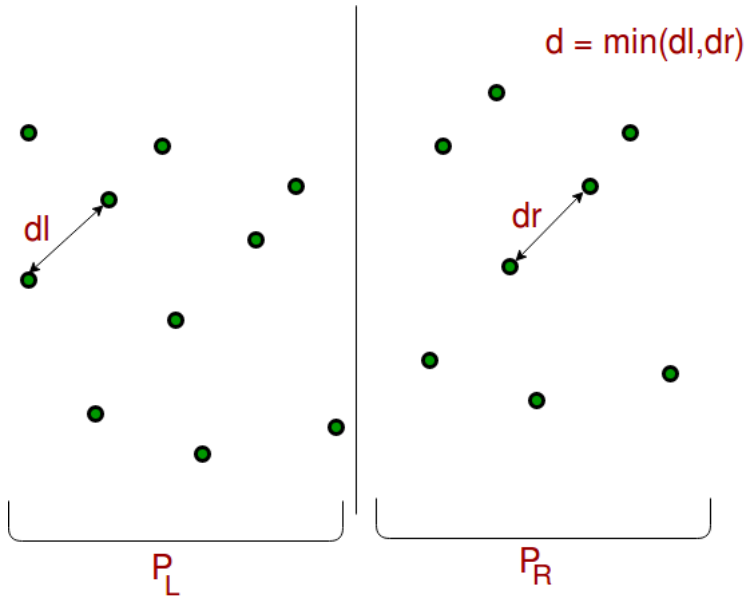
---

## Procedure 10 CLOSEST\_PAIR

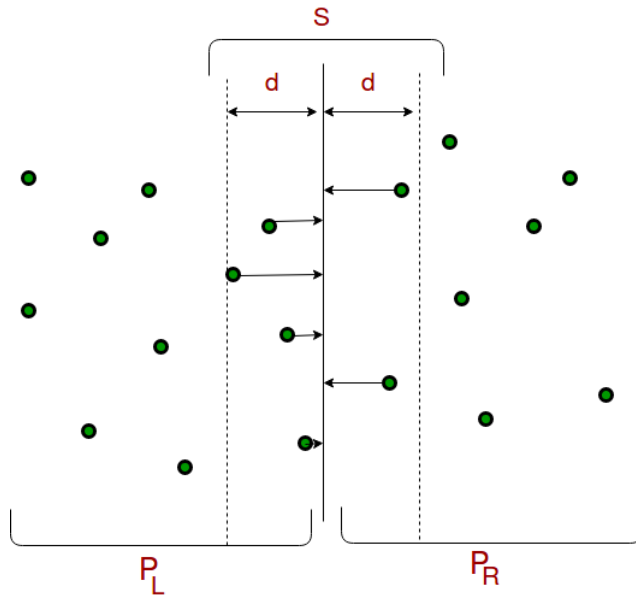
---

**Input:**  $A : \text{Array} < \text{Point} >$

1. Ordenamos todos los puntos según sus coordenadas  $x$ .
  2. Divide todos los puntos en dos mitades.
  3. Encuentra, recursivamente, las distancias más pequeñas en ambas mitades.
  4. Tomamos el mínimo de ambas distancias,  $d$ .
  5. Crea un arreglo que almacene todos los puntos que se encuentran a una distancia máxima de la línea media que divide ambas mitades.
  6. Encuentramos la distancia más corta de este subconjunto.
  7. Devolvemos el mínimo,  $d$ , y la distancia más pequeña calculada en 4.
-







# Encontrar el número más cercano en el arreglo

Dado un arreglo de números enteros ordenados. Necesitamos encontrar el valor más cercano al número dado. El arreglo puede contener valores duplicados y números negativos. <sup>5</sup>

---

<sup>5</sup><https://goo.gl/XTgBzX>

---

## Procedure 11 FIND\_CLOSEST

---

Input:  $A$  : Array,  $target$  : Key

if  $target < A[1]$  then

return  $A[1]$

end if

if  $target > A[A.length]$  then

return  $A[A.length]$

end if

$low \leftarrow 1$

$high \leftarrow A.length$

while  $low < high$  do

NEXT SLIDE

end while

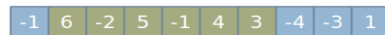
---

---

```
mid ← FLOOR((high + low)/2)
if target = A[mid] then
    return A[mid]
else if target < A[mid] then
    if mid > 0 and target > A[mid − 1] then
        return CLOSEST(A[mid − 1], A[mid], target)
    end if
    high ← mid
else
    if mid < A.length and target < A[mid + 1] then
        return CLOSEST(A[mid], A[mid + 1], target)
    end if
    high ← mid
end if
```

---

Dado un arreglo de  $n$  números enteros positivos y negativos, encontrar los  $i$  elementos del arreglo cuya suma se la máxima posible.



La secuencia máxima se encuentra comprendida entre la posición 1 y 6. La suma máxima es 15.

---

## Procedure 12 MAX\_SUM

---

**Input:**  $A$  : Array,  $low$ ,  $high$  : Index

if  $(high - low + 1) = 1$  then

return  $A[low]$

else

$mid \leftarrow FLOOR((high - low)/2)$

$left \leftarrow MAX\_SUM(A, low, mid)$

$right \leftarrow MAX\_SUM(A, mid + 1, high)$

$center \leftarrow MAX\_AUX(A, low, high)$

return  $MAX(left, right, center)$

end if

---

