

A background network diagram consisting of numerous small, semi-transparent colored circles (nodes) in shades of blue, green, orange, purple, and grey. These nodes are interconnected by a web of thin, light-grey lines, creating a complex, organic structure that fills the entire frame.

UADE

A horizontal bar at the bottom of the image, composed of several segments of different colors: green, dark blue, light blue, purple, red, orange, and yellow.

Programación III – Diseño y Análisis de Algoritmos



Temas a desarrollar

1

Repaso Clase Anterior

2

Revisión Ejercicios Práctico

3

Métodos de Ordenamiento

4

Merge Sort

5

Quick Sort



Repaso Clase Anterior





Conceptos Básicos

- Un algoritmo es una secuencia ordenada y finita de acciones que transforman un conjunto de datos de entrada en datos de salida y que resuelven un problema específico. Tiene cero o más datos de entrada, al menos una salida, cada paso esta unívocamente determinado y debe terminar en un número finito de pasos.
- La algoritmia se ocupa del estudio de los algoritmos y abarca el análisis de la **correctitud** y de la **eficiencia**, es decir, analiza si un algoritmo resuelve un problema y la cantidad de recursos necesarios para llegar a la solución.
- La eficiencia nos permite comparar diferentes algoritmos/programas entre sí en cuanto a los usos de recursos. Existen tres tipos de análisis: peor caso, caso promedio y mejor caso.

Cálculos de la complejidad temporal

- Complejidad temporal: funciones recursivas

$$T(n) = \begin{cases} c & \text{si } 0 \leq n < b \\ aT(n-b) + p(n) & \text{si } n \geq b \end{cases}$$



$$T(n) \in \begin{cases} \Theta(n^k) & \text{si } a < 1 \\ \Theta(n^{k+1}) & \text{si } a = 1 \\ \Theta(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

$$T(n) = \begin{cases} c & \text{si } 0 \leq n < b \\ aT(n/b) + f(n) & \text{si } n \geq b \end{cases}$$



$$T(n) \in \begin{cases} \Theta(n^k) & \text{si } a < b^k \\ \Theta(n^k \log(n)) & \text{si } a = b^k \\ \Theta(n^{\log_b(a)}) & \text{si } a > b^k \end{cases}$$

Divide y Conquista

- **Divide y Conquista** es una técnica de diseño de algoritmos que consiste en dividir un problema “grande” en problemas más “pequeños”, de resolución más sencilla, y luego combinar las soluciones de los problemas más “pequeños” para hallar la solución del problema “grande”.
- La estrategia de solución de un problema de divide y conquista se debe basar en identificar los casos bases, los subproblemas independientes y la forma de unificar las soluciones de cada uno de sus subproblemas.

ALGORITMO DYC

Entrada: x

si *esPequeno*(x)

 devolver *solucion_trivial*(x)

sino

$\langle x_1, \dots, x_n \rangle \leftarrow \text{descomponer}(x)$

 para $i = 1$ hasta n

$y_i \leftarrow \text{DyV}(x_i)$

 fin para

 devolver *combinar*(y_1, \dots, y_n)

fin si

Síntesis Búsqueda Binaria

- El problema de la búsqueda binaria consiste en buscar un número **x** dado en una secuencia ordenada en forma creciente de números **S** que también es un dato de entrada.
- La estrategia por divide y conquista se puede describir como sigue:

Caso Base

1. Longitud UNO
2. Valor ubicado en la MITAD

Subproblemas

1. Si $x > \text{valor en mitad} \rightarrow$ buscar entre $(n/2)+1$ y n
2. Si $x < \text{valor en mitad} \rightarrow$ buscar entre 1 y $(n/2)-1$

Combinación

1. Se devuelve el valor de la mitad en que se busco.

- La complejidad temporal de la resolución es de $\Theta(\log n)$, siendo **n** la cantidad de elementos de la secuencia sobre la cual buscar.

Síntesis Problema de una Palabra Palíndromo

- Una palabra se dice que es Palíndromo si se lee igual en un sentido que en otro, es decir, si se mira de izquierda a derecha es igual que de derecha a izquierda. Si se trata de números en lugar de letras, se llama capicúa.
- La estrategia por Divide y Conquista es la siguiente:

Caso Base

Si la secuencia tiene longitud cero o uno, la solución trivial es devolver verdadero dado que la secuencia se considera palíndromo.

Subproblemas

Comparar los extremos de la secuencia, si no coinciden se devuelve falso, y sino se reduce la secuencia en dos extremos, y se resuelve si es palíndromo esa secuencia menor.

Combinación

Devolver el resultado de la secuencia de $n-2$.

- La complejidad temporal de la resolución es $\Theta(n)$ siendo n la longitud de la palabra a analizar.



Revisión Ejercicios Prácticos



Ejercicios a Resolver

Lea detenidamente cada problema, por cada uno de ellos deberá responder los siguientes puntos:

- a) Estrategia de la solución aplicando Divide y Conquista e indicando cada uno de los elementos que se requieren para la resolución de esa técnica.*
- b) Pseudocódigo del Algoritmo de la resolución del problema, el cual debe seguir la estrategia definida en el punto a)*
- c) Cálculo de la Complejidad Temporal, justificando la misma en forma detallada.*

1. Diseñar un algoritmo que determine si una secuencia de n caracteres está ordenada alfabéticamente.
2. Diseñar un algoritmo que calcule a^n cuando n es una potencia de 2.
3. Sea $A[1..n]$, $n \geq 1$, un vector de enteros diferentes y ordenados crecientemente, tal que algunos de los valores pueden ser negativos. Diseñar un algoritmo que devuelva un índice natural k , $1 \leq k \leq n$, tal que $A[k] = k$, siempre que tal índice exista.



Métodos de Ordenamiento



Métodos de Ordenamiento

- Existen métodos de ordenamiento que denominaremos “tradicionales”, en los cuales con alguna estrategia iterativa se puede llegar al ordenamiento, como son los siguientes:

SELECCIÓN

Consiste en buscar en toda la secuencia de elementos el menor y ponerlo en la posición 1, luego de los restantes el menor en la posición 2, y así hasta llegar a ordenar todos los elementos.

INSERCIÓN

Consiste en ir ubicando cada elemento en su posición. Es decir que en cada iteración i se obtiene una secuencia ordenada de $i-1$ elementos en la primera parte del arreglo y se inserta el elemento de la posición i .

Métodos de Ordenamiento - Selección

ALGORITMO SELECCION

Entrada: S : Vector<entero>

$n \leftarrow longitud(S)$

para $i = 0$ hasta $n - 1$

$m \leftarrow i$

para $j = i + 1$ hasta $n - 1$

si $S[j] < S[m]$

$m \leftarrow j$

fin si

fin para

$aux \leftarrow S[i];$

$S[i] \leftarrow S[m];$

$S[m] \leftarrow aux;$

fin para

Un ciclo que itera n veces en el peor de los casos, dentro de otro ciclo que itera n veces, nos da una complejidad de:

$$\mathcal{O}(n^2).$$

Métodos de Ordenamiento - Inserción

ALGORITMO INSERCION

Entrada: S : Vector<entero>

$n \leftarrow longitud(S)$

para $i = 1$ hasta $n - 1$

 para $j = i$ hasta 1 decrementando 1

 si $S[j] < S[j - 1]$

$aux \leftarrow S[j - 1];$

$S[j - 1] \leftarrow S[j];$

$S[j] \leftarrow aux;$

 fin si

 fin para

fin para

6	2	9	7	11	3
---	---	---	---	----	---

2	6	9	7	11	3
---	---	---	---	----	---

2	6	9	7	11	3
---	---	---	---	----	---

2	6	7	9	11	3
---	---	---	---	----	---

2	6	7	9	11	3
---	---	---	---	----	---

2	6	7	9	11	3
---	---	---	---	----	---

2	6	7	9	3	11
---	---	---	---	---	----

2	6	7	3	9	11
---	---	---	---	---	----

2	6	3	7	9	11
---	---	---	---	---	----

2	3	6	7	9	11
---	---	---	---	---	----

Métodos de Ordenamiento - Inserción

ALGORITMO INSERCIÓN

Entrada: S : Vector<entero>

$n \leftarrow longitud(S)$

para $i = 1$ hasta $n - 1$

 para $j = i$ hasta 1 decrementando 1

 si $S[j] < S[j - 1]$

$aux \leftarrow S[j - 1];$

$S[j - 1] \leftarrow S[j];$

$S[j] \leftarrow aux;$

 fin si

 fin para

fin para

Un ciclo que itera n veces en el peor de los casos, dentro de otro ciclo que itera n veces, nos da una complejidad de:

$$O(n^2).$$

Métodos de Ordenamiento – Divide y Conquista

- Dentro de los métodos de ordenamiento por Divide y Conquista, se van a estudiar dos algoritmos que son los siguientes:
 - a. Merge-Sort*
 - b. Quick-Sort*
- Cada uno de los algoritmos será analizado en cuanto a su estrategia y complejidad temporal.

Nota: Por ordenamiento se entiende “ordenar” una secuencia en forma creciente.



Merge Sort



Merge Sort – Estrategia

La estrategia por Divide y Conquista del Merge Sort se puede definir como sigue:

Caso Base

Una secuencia de un solo elemento, dado que ya por defecto se considera ordenada.

Subproblemas

Dividir la secuencia de entrada en dos mitades, y ordenar cada una de las mitades por separado

Combinación

Se deben mezclar las dos mitades ordenadas en una nueva secuencia ordenada, lo cual dará como resultado la secuencia total ordenada.

Merge Sort – Ejemplo usando la Estrategia

6	3	11	15	4	7	9	2
---	---	----	----	---	---	---	---

6	3	11	15
---	---	----	----

4	7	9	2
---	---	---	---

6	3
---	---

11	15
----	----

4	7
---	---

9	2
---	---

6

3

11

15

4

7

9

2

Casos Bases

3	6
---	---

11	15
----	----

4	7
---	---

2	9
---	---

Se coloca el menor, y se avanza en la primera parte, y se repite el proceso

3	6	11	15
---	---	----	----

2	4	7	9
---	---	---	---

2	3	4	6	7	9	11	15
---	---	---	---	---	---	----	----

Comienzo a combinar las dos subsecuencias, comparando el primero de cada una, el menor va al resultado, y así avanza en la parte del cual se tomo el valor, y nuevamente se compara.

Merge Sort – Algoritmo

ALGORITMO MERGESORT

Entrada: S : Vector<entero>, $inicio$: entero, fin : entero

si $inicio < fin$

entero $medio \leftarrow (fin + inicio)/2$

$MergeSort(S, inicio, medio)$

$MergeSort(S, medio + 1, fin)$

$Merge(S, inicio, fin)$

fin si

Nota: como se puede observar la combinación de las dos partes ordenadas se hace en un algoritmo adicional, el cual se describirá a continuación.

Merge Sort – Algoritmo

ALGORITMO MERGE

Entrada: S : Vector<entero>, $inicio$: entero, fin : entero

Vector $R \leftarrow inicializarVector(fin - inicio + 1)$

entero $medio \leftarrow (inicio + fin)/2$

$i \leftarrow inicio$

$j \leftarrow medio + 1$

para entero $k = 0$ hasta $fin - inicio$

si $(j > fin \text{ O } S[i] \leq S[j]) \text{ Y } i \leq medio$

$R[k] \leftarrow S[i]$

$i \leftarrow i + 1$

sino

$R[k] \leftarrow S[j]$

$j \leftarrow j + 1$

fin si

fin para

para entero $k = 0$ hasta $fin - inicio$

$S[inicio + k] \leftarrow R[k]$

fin para

Merge Sort – Complejidad Temporal

ALGORITMO MERGESORT

Entrada: S : Vector<entero>, $inicio$: entero, fin : entero

si $inicio < fin$

entero $medio \leftarrow (fin + inicio)/2$

$MergeSort(S, inicio, medio)$

$MergeSort(S, medio + 1, fin)$

$Merge(S, inicio, fin)$

fin si

Existen dos llamados recursivos en donde por cada parte el vector se divide en dos, por lo cual:

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 2T(n/2) + P^1(n) & \text{si } n > 1 \end{cases} \quad a = 2, b = 2$$

Debemos analizar la complejidad de $Merge$ para determinar el k .

Merge Sort – Complejidad Temporal

ALGORITMO MERGE

Entrada: S : Vector<entero>, $inicio$: entero, fin : entero

Vector $R \leftarrow inicializarVector(fin - inicio + 1)$

entero $medio \leftarrow (inicio + fin)/2$

$i \leftarrow inicio$

$j \leftarrow medio + 1$

para entero $k = 0$ hasta $fin - inicio$

si $(j > fin \text{ O } S[i] \leq S[j]) \text{ Y } i \leq medio$

$R[k] \leftarrow S[i]$

$i \leftarrow i + 1$

sino

$R[k] \leftarrow S[j]$

$j \leftarrow j + 1$

fin si

fin para

para entero $k = 0$ hasta $fin - inicio$

$S[inicio + k] \leftarrow R[k]$

fin para

La complejidad esta dada por un ciclo en el peor de los casos de n , por lo tanto:

$$\mathcal{O}(n)$$

Merge Sort – Complejidad temporal

ALGORITMO MERGESORT

Entrada: S : Vector<entero>, $inicio$: entero, fin : entero

si $inicio < fin$

entero $medio \leftarrow (fin + inicio)/2$

$MergeSort(S, inicio, medio)$

$MergeSort(S, medio + 1, fin)$

$Merge(S, inicio, fin)$

fin si

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 2T(n/2) + P^1(n) & \text{si } n > 1 \end{cases} \quad a = 2, \bar{b} = 2 \text{ y } k = 1,$$

$$T(n) \in \begin{cases} \Theta(n^k) & \text{si } a < b^k \\ \Theta(n^k \log(n)) & \text{si } a = b^k \\ \Theta(n^{\log_b(a)}) & \text{si } a > b^k \end{cases}$$




$$\Theta(n \log(n))$$



Quick Sort





Quick Sort – Definición

- La estrategia del Quick Sort consiste en seleccionar un elemento al azar denominado *Pivot*.
- A partir de ese Pivot se divide la secuencia en dos partes, a la izquierda del pivot todos los elementos menores y a la derecha del pivot todos los elementos mayores al mismo.
- Una vez hecho esto, el pivot estará ubicado en su posición definitiva en la secuencia y se aplicara en forma recursiva el método en las dos partes resultantes.



Quick Sort – Pivot

- Existen distintas técnicas para seleccionar el pivot, algunas mas sencillas y otras más complejas. El criterio seleccionada deberá redundar en una solución más eficiente.
- Cuanto mas cercano al elemento mediano se encuentre el pivot, mas parejas serán las dos partes de la secuencia y mejor la solución, cuanto mas cercano a alguno de los extremos se encuentre el pivot, más desparejas serán las dos partes y menos eficiente la solución.
- Un criterio simple es considerar como pivot el primer elemento de la secuencia inicial.

Quick Sort – Estrategia

- La estrategia de resolución por Divide y Conquista se puede enumerar como sigue:

Caso Base

Una secuencia que tenga una o dos posiciones. En nuestro caso se va a considerar una sola posición.

Subproblemas

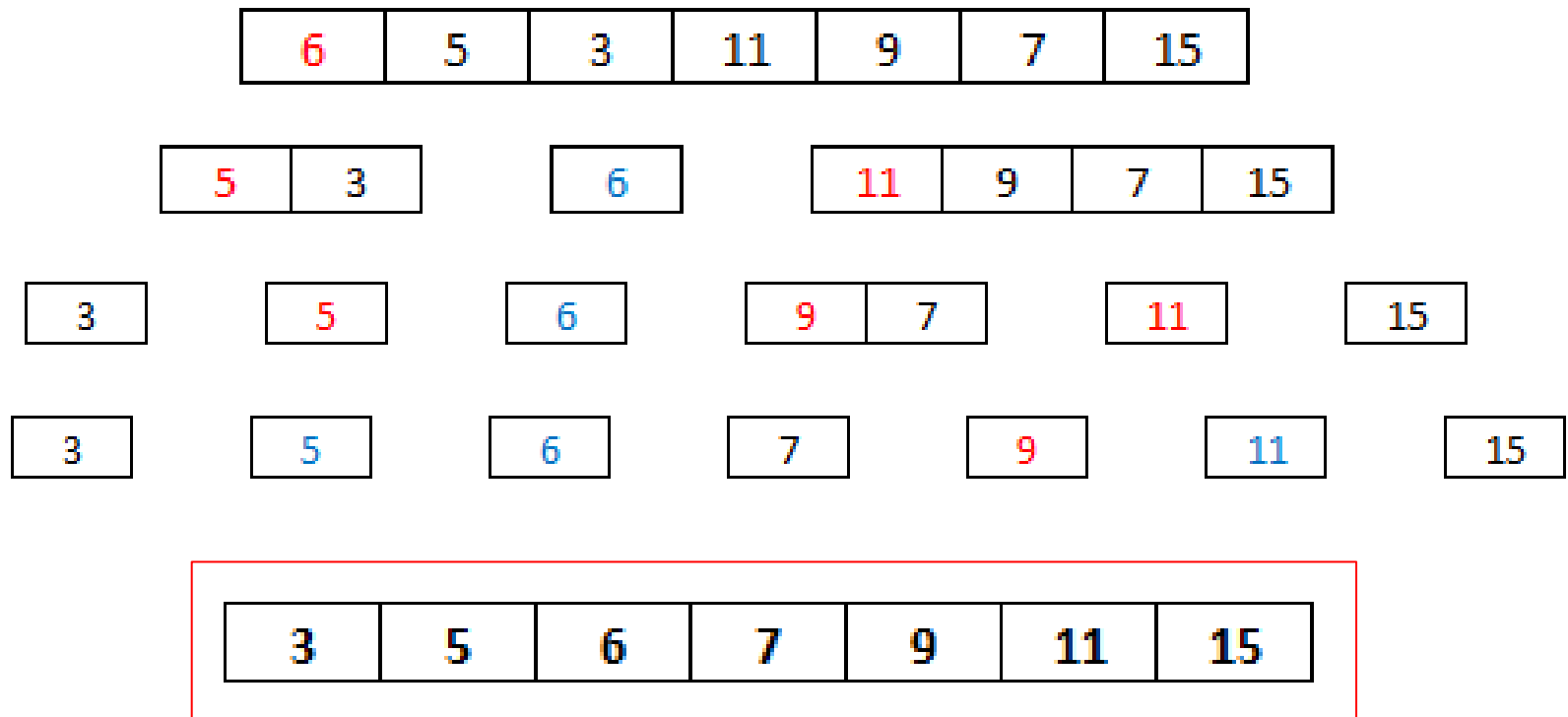
Dividir en dos subproblemas en base al Pivot, enviando a ordenar cada una de esas dos partes.

Combinación

No se requiere.

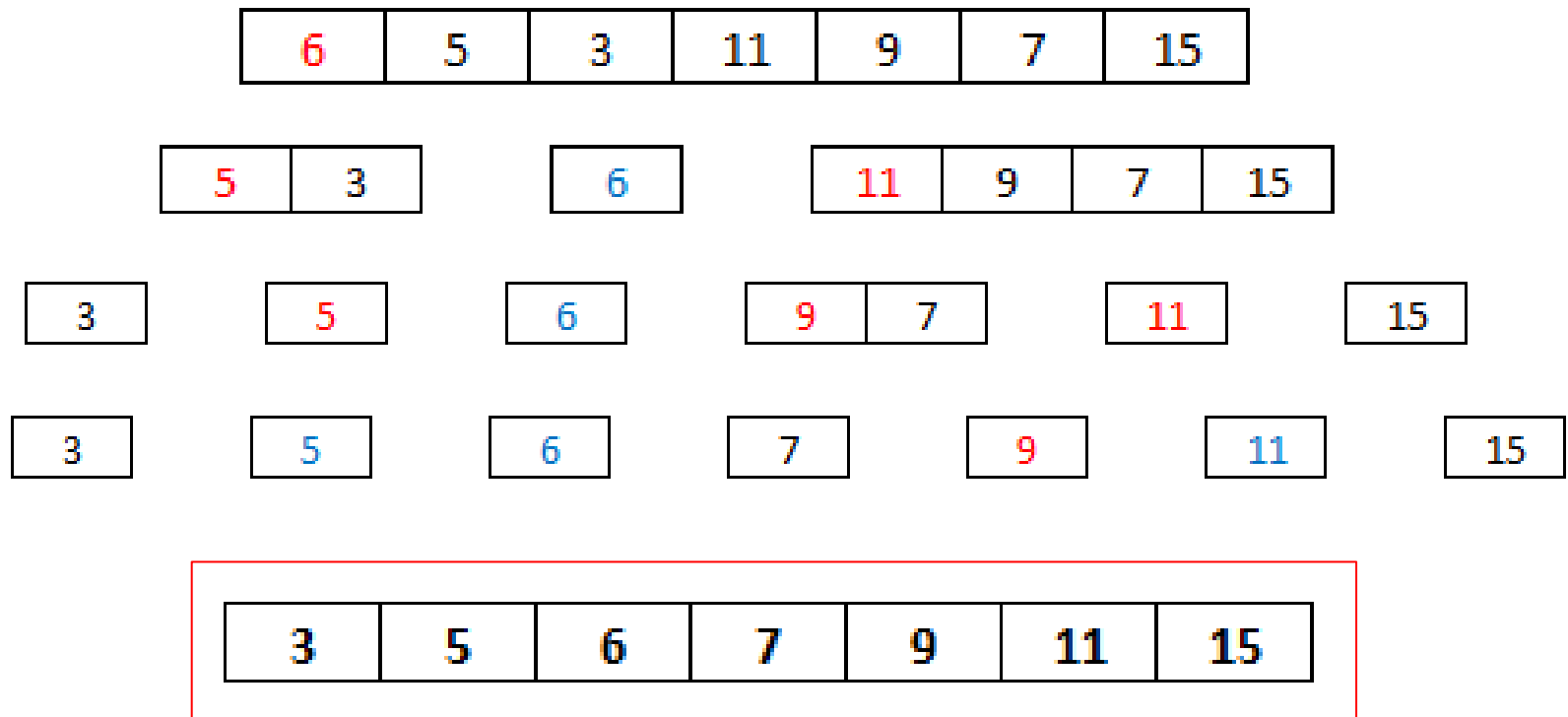
Quick Sort – Ejemplo

A partir del siguiente ejemplo se mostrará la estrategia del Quick Sort definida. En rojo se marca el elemento seleccionado como Pivot en cada iteración, y en azul como queda ubicado luego:



Quick Sort – Ejemplo

A partir del siguiente ejemplo se mostrará la estrategia del Quick Sort definida. En rojo se marca el elemento seleccionado como Pivot en cada iteración, y en azul como queda ubicado luego:



Quick Sort – Algoritmo

ALGORITMO QUICKSORT

Entrada: S : Vector<entero>, $inicio$:entero, fin :entero

si $inicio < fin$

$p \leftarrow pivot(S, inicio, fin)$

$QuickSort(S, inicio, p - 1)$

$QuickSort(S, p + 1, fin)$

fin si

Nota: como se puede observar el método *Pivot* se realiza en forma externa, y como resultado devuelve la posición en donde se encuentra ese elemento **Pivot** luego de haber ubicado a su izquierda los menores y a su derecha los mayores. A continuación se muestra el algoritmo del mismo.

Quick Sort – Algoritmo

ALGORITMO PIVOT

Entrada: S : Vector<entero>, $inicio$:entero, fin :entero

Salida: p : entero

```
entero  $p \leftarrow S[inicio]$ 
entero  $k \leftarrow inicio + 1$ 
entero  $l \leftarrow fin$ 
mientras  $S[k] \leq p$  Y  $k < fin$ 
     $k \leftarrow k + 1$ 
fin mientras
mientras  $S[l] > p$ 
     $l \leftarrow l - 1$ 
fin mientras
mientras  $k < l$ 
     $aux \leftarrow S[k]$ 
     $S[k] \leftarrow S[l]$ 
     $S[l] \leftarrow aux$ 
    mientras  $S[k] \leq p$ 
         $k \leftarrow k + 1$ 
    fin mientras
    mientras  $S[l] > p$ 
         $l \leftarrow l - 1$ 
    fin mientras
fin mientras
 $aux \leftarrow S[inicio]$ 
 $S[inicio] \leftarrow S[l]$ 
 $S[l] \leftarrow aux$ 
devolver  $l$ 
```

Quick Sort – Complejidad Temporal

ALGORITMO QUICKSORT

Entrada: S : Vector<entero>, $inicio$:entero, fin :entero

si $inicio < fin$

$p \leftarrow pivot(S, inicio, fin)$

$QuickSort(S, inicio, p - 1)$

$QuickSort(S, p + 1, fin)$

fin si

Debido a que el pivot no se sabe si queda a la mitad o en un extremo se analizan las dos posibles situaciones, caso promedio que divida a la mitad, y peor caso que quede en alguno de los extremos.

Caso Promedio

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 2T(n/2) + P^1(n) & \text{si } n > 1 \end{cases}$$

$$a = 2, b = 2$$

Peor Caso

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ T(n - 1) + P^1(n) & \text{si } n > 1 \end{cases}$$

$$a = 1, b = 1$$

En ambos casos se debe analizar la complejidad de *Pivot*

Quick Sort – Complejidad Temporal

ALGORITMO PIVOT

Entrada: S : Vector<entero>, $inicio$:entero, fin :entero

Salida: p : entero

```
entero  $p \leftarrow S[inicio]$   
entero  $k \leftarrow inicio + 1$   
entero  $l \leftarrow fin$ 
```

```
mientras  $S[k] \leq p \text{ Y } k < fin$   
   $k \leftarrow k + 1$ 
```

```
fin mientras
```

```
mientras  $S[l] > p$   
   $l \leftarrow l - 1$ 
```

```
fin mientras
```

```
mientras  $k < l$ 
```

```
   $aux \leftarrow S[k]$   
   $S[k] \leftarrow S[l]$   
   $S[l] \leftarrow aux$ 
```

```
  mientras  $S[k] \leq p$   
     $k \leftarrow k + 1$ 
```

```
  fin mientras
```

```
  mientras  $S[l] > p$   
     $l \leftarrow l - 1$ 
```

```
  fin mientras
```

```
fin mientras  
 $aux \leftarrow S[inicio]$   
 $S[inicio] \leftarrow S[l]$   
 $S[l] \leftarrow aux$   
devolver  $l$ 
```

La complejidad esta dado por diferentes ciclos en el peor de los casos esos ciclos iteran n veces. Los ciclos anidados por la combinación de índices no iteran más que n veces en conjunto, por lo tanto

$$O(n)$$

Quick Sort – Complejidad Temporal

ALGORITMO QUICKSORT

Entrada: S : Vector<entero>, $inicio$:entero, fin :entero

si $inicio < fin$

$p \leftarrow pivot(S, inicio, fin)$

$QuickSort(S, inicio, p - 1)$

$QuickSort(S, p + 1, fin)$

fin si

$\mathcal{O}(n)$

Caso Promedio

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 2T(n/2) + P^1(n) & \text{si } n > 1 \end{cases}$$

$$a = 2, \bar{b} = 2 \text{ y } k = 1,$$

$\Theta(n \log(n))$

Peor Caso

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ T(n - 1) + P^1(n) & \text{si } n > 1 \end{cases}$$

$$a = 1, b = 1 \text{ y } k = 1,$$

$\Theta(n^2)$



Síntesis Métodos de Ordenamiento



Síntesis Métodos de Ordenamiento

- Los métodos de ordenamiento denominados iterativos como ser “Selección” o “Inserción” tienen una complejidad temporal de $\Theta(n^2)$, siendo n la cantidad de elementos a ordenar.
- En los métodos de ordenamiento que siguen una estrategia por Divide y Conquista, como ser Merge Sort o Quick Sort, se pueden llegar a complejidades temporales de $\Theta(n \log n)$, siendo n la cantidad de elementos a ordenar.

Síntesis Métodos de Ordenamiento

- **Merge Sort**, consiste en dividir la secuencia de entrada en dos mitades, ordenar cada una de las mitades y luego mezclar las mitades ordenadas en una nueva secuencia ordenada. La complejidad temporal está dada por (siendo n la cantidad de elementos a ordenar):

$$\Theta(n \log(n))$$

- **Quick Sort**, El método consiste en seleccionar un elemento al azar denominado *Pivot*. A partir de ese *Pivot* se divide la secuencia en dos partes, a la izquierda del *Pivot* todos los elementos menores y a la derecha del *Pivot* todos los elementos mayores al mismo. La complejidad temporal está dada por (siendo n la cantidad de elementos a ordenar):

Caso Promedio

$$\Theta(n \log(n))$$

Peor Caso

$$\Theta(n^2)$$



¡Muchas gracias!

