# Quicksort

Guillermo Palma

Universidad Simón Bolívar
Departamento de Computación y T.I.

CI-2612: Algoritmos y Estructuras de Datos II



## Plan

# Sobre Quicksort

- Algoritmo de ordenamiento que aplica la técnica Divide-and-conquer
- Inventado por Tony Hoare (1961)
- Popularizado por Robert Sedgewick (1976)
- El algoritmo en la práctica es más rápido que los otros algoritmos de ordenamientos basados en comparación de claves (Heapsort, Mergesort, Insertion-Sort, etc.)
- Tiempo de Quicksort para el caso promedio $\Theta(n \log n)$
- Tiempo de Quicksort para el peor caso es $\Theta(n^2)$





# Idea de Quicksort

**Divide**

Dado un elemento $A[q]$ (pivote), $A[p..r]$ se divide en $A[p..q-1]$ y $A[q+1..r]$, tal que todos los elementos $A[p..q-1]$ son menores o iguales a $A[q]$ y todos elementos $A[q+1..r]$ son mayores que $A[q]$

**Conquer**

Se ordenan $A[p..q-1]$ y $A[q+1..r]$ con llamadas recursivas a Quicksort

**Combine**

Como los subarreglos que se obtienen están ordenados, el arreglo $A[p..r]$ termina estando ordenado

# Procedimiento Quicksort

**Procedimiento** QUICKSORT(A, p, r)

**inicio**
  **si** $p < r$ **entonces**
    $q \leftarrow$ `PARTITION` $(A, p, r)$ ;
    `QUICKSORT` $(A, p, q - 1)$ ;
    `QUICKSORT` $(A, q + 1, r)$ ;

- La llamada inicial del algoritmo es `QUICKSORT`$(A, 1, A.length)$.
- Recurrencia $T(n) = T(q) + T(n - q) + g(n)$, donde $g(n)$ es el tiempo de PARTITION





# Partición de un arreglo

**Función** PARTITION(A, p, r)

**inicio**
  $x \leftarrow A[r]$ ;
  $i \leftarrow p - 1$ ;
  **para** $j \leftarrow p$ **a** $r - 1$ **hacer**
    **si** $A[j] \leq x$ **entonces**
      $i \leftarrow i + 1$;
      `SWAP` $(A[i], A[j])$ ;
  `SWAP` $(A[i + 1], A[r])$ ;
  **devolver** $i + 1$ ;

- Tiempo de ejecución: $\Theta(n)$, donde $n = r - p + 1$

# Invariante en la partición del arreglo

En las líneas 4-7 de la función `PARTITION`, se encuentra el ciclo **para** y en él se cumple que:

- Si $p \leq k \leq i$, entonces $A[k] \leq x$
- Si $i + 1 \leq k \leq j - 1$, entonces $A[k] > x$
- Si $k = r$, entonces $A[k] = x$





# Invariante en la partición del arreglo

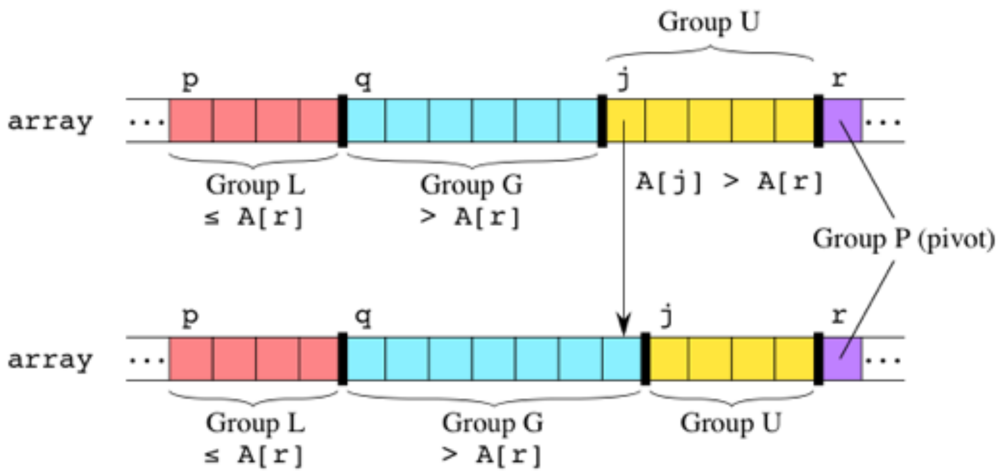



Figura: Caso $A[j] > A[r]$. Fuente: [1]

# Invariante en la partición del arreglo

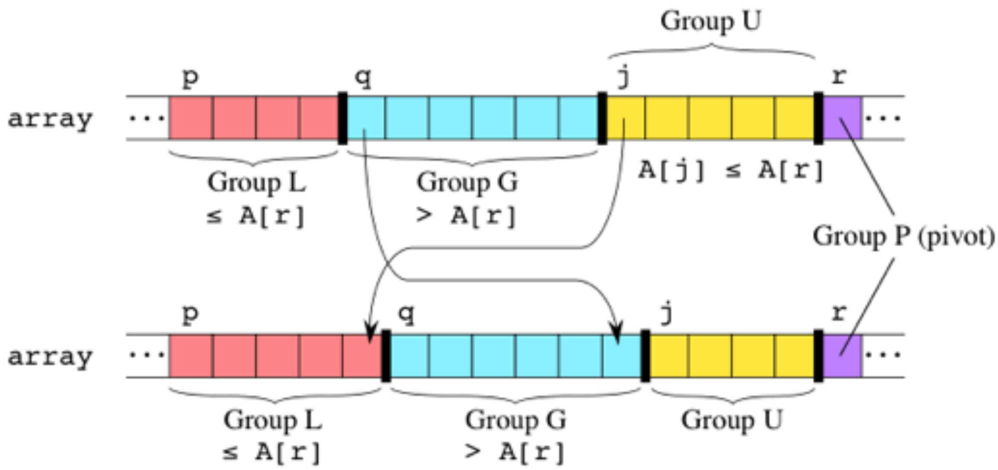



Figura: Caso $A[j] \leq A[r]$. Fuente: [1]





# Ejemplo de la partición del arreglo

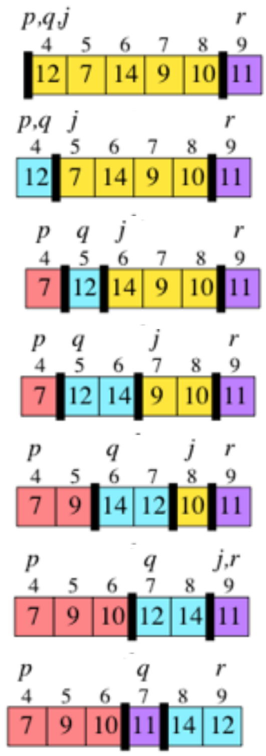



Figura: Ilustración de la función `Partition`. Fuente: [1]

# Ejemplo de Quicksort

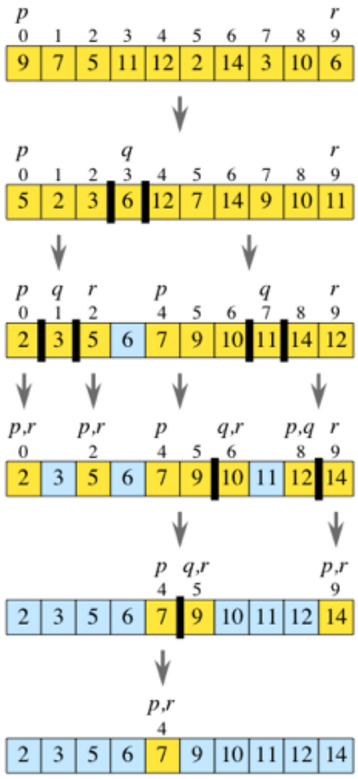



Figura: Ilustración del algoritmo `Quicksort`. Fuente: [1]





# Tiempo del peor caso de la partición

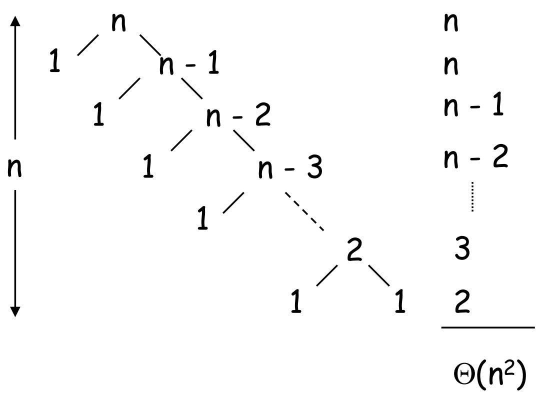



Figura: Árbol de recursión de `Quicksort` con una partición de 1 a $n - 1$ [1]

- El subarreglo de un lado del pivote $q$ tiene 1 elemento y el otro subarreglo $n - 1$ elementos
- Recurrencia: $q = 1$ , entonces $T(n) = T(1) + T(n - 1) + n$
- $T(1) = \Theta(1)$
- $T(n) = T(n - 1) + n = n + (n - 1) + (n - 2) + \cdots + 2 + n = \Theta(n^2)$
- Por lo tanto el peor caso es $\Theta(n^2)$

# Tiempo del mejor caso de la partición

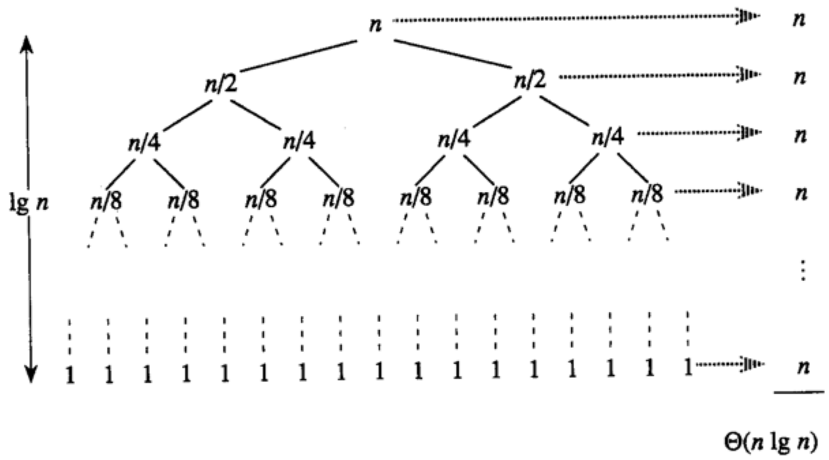



Figura: Árbol de recursión de `Quicksort` con una partición de $n/2$ [1]

- Los subarreglos izquierdo y derecho del pivote $q$ tienen el mismo tamaño, esto es $n/2$
- Recurrencia: $q = n/2$ , entonces $T(n) = 2T(n/2) + \Theta(n)$
- $T(n) = \Theta(n \log n)$
- Por lo tanto el mejor caso es $\Theta(n \log n)$





# Tiempo de un caso entre el mejor y el peor caso

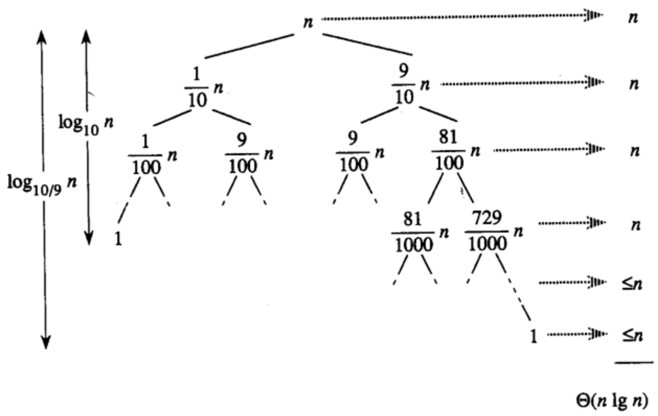
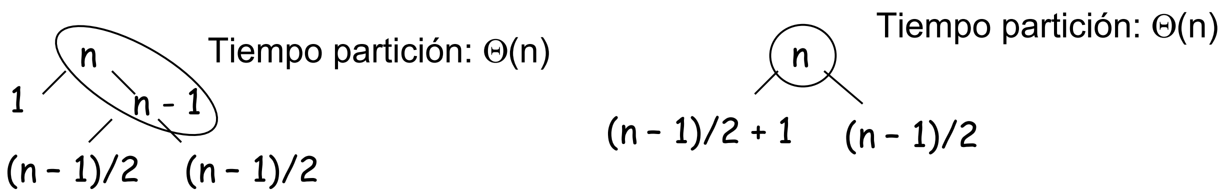



Figura: Árbol de recursión de `Quicksort` con una partición de 9 a 1 [1]

- Los subarreglos izquierdo y derecho del pivote $q$ tienen una diferencia de 9 a 1.
- Recurrencia: $T(n) = T(9n/10) + T(n/10) + \Theta(n)$
- Cota superior: $T(n) \leq n \sum_{i=0}^{\log_{10/9}} 1 = n(\log_{10/9} n + 1) = C_1 n \log n$
- Cota inferior: $T(n) \geq n \sum_{i=0}^{\log_{10}} 1 = n(\log_{10} n + 1) = C_2 n \log n$
- Por lo tanto $T(n) = \Theta(n \log n)$

# Intuición del caso promedio




Se alterna entre buenas y malas particiones

Patición casi balanceada

Figura: A la izquierda dos niveles del árbol de recursión de `Quicksort`, a la derecha un nivel. Fuente: [2]

- Todas las permutaciones del arreglo de entrada son igualmente probables
- En una entrada aleatoria, una partición que pueden ser balanceada (buena) o no balanceada (mala)
- Los pivotes buenos y malos se encuentran aleatoriamente distribuidos en el árbol
- El tiempo de Quicksort cuando las llamadas recursivas alternan entre balanceadas o no balanceadas particiones es $O(n \log n)$





# Idea de la versión aleatoria de Quicksort

- Se quiere evitar que un adversario provoque el peor caso de Quicksort
- Solución 1: Se permutan aleatoriamente la secuencia de entrada
- Solución 2: Se escoge cada vez un pivote $A[i]$ al azar
  - Dado una secuencia $A[p \dots r]$, se intercambia $A[r]$ por un elemento $A[i]$ escogido al azar
  - Todos los elementos $r - p + 1$ de la secuencia $A[p \dots r]$ son igualmente probable de ser el pivote
- Solo se logra el peor caso si el generador de números aleatorios genera dicha esa secuencia
- No se hace que desaparezca el peor caso, pero se hace menos probable que se genere

# Partición Aleatoria

**Función** RANDOMIZED-PARTITION(A, p, r)

**inicio**
  $i \leftarrow$ `RANDOM`$(p, r)$ ;
  `SWAP`$(A[r], A[i])$ ;
  **devolver** `PARTITION`$(A, p, r)$ ;





# Procedimiento Quicksort Aleatorio

**Procedimiento** RANDOMIZED-QUICKSORT(A, p, r)

**inicio**
  **si** $p < r$ **entonces**
    $q \leftarrow$ `RANDOMIZED-PARTITION`$(A, p, r)$ ;
    `RANDOMIZED-QUICKSORT`$(A, p, q-1)$ ;
    `RANDOMIZED-QUICKSORT`$(A, q+1, r)$ ;

La llamada inicial del algoritmo es
`RANDOMIZED-QUICKSORT`$(A, 1, A.length)$.

# Tiempo del peor caso de Quicksort

- Sea $T(n)$ el tiempo del peor caso
- Se tiene que $T(n) = \text{máx}_{1 \leq q \leq n-1}(T(q) + T(n-q)) + \Theta(n)$
- Usando el método de sustitución se puede probar que el tiempo del peor caso es $O(n^2)$
  - Suponga que $T(n) = O(n^2)$ (Por lo visto anteriormente)
  - Hipótesis inductiva: $T(k) \leq ck^2$ para cualquier $k < n$
  - Se quiere probar que: $T(n) \leq cn^2$ para algún $c > 0$ y $n \geq n_o$



# Referencias


T. Cormen, D. Balkcom, and Khan Academy Team.
`shorturl.at/sBNS7`, septiembre 2019.

T. Cormen, C. Leirserson, R. Rivest, and C. Stein.
*Introduction to Algorithms*.
McGraw Hill, 3ra edition, 2009.