Quicksort

Fernando Lobo

Algoritmos e Estrutura de Dados

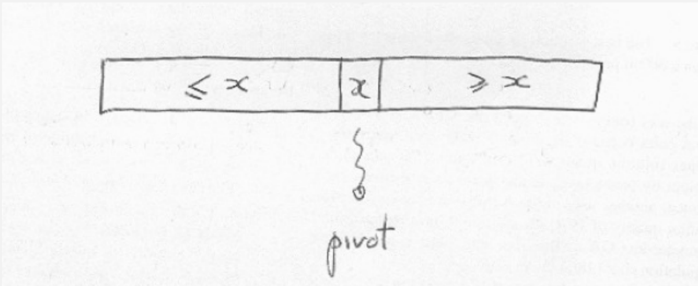
— Algumas figuras retiradas do livro Introduction to Algorithms, 3rd Edition. —

Quicksort

- Inventado por Hoare em 1962.
- É um algoritmo de D&C.
- Muito usado na prática. A maior parte dos algoritmos de ordenação incorporados em bibliotecas de linguagens de programação são baseados no quicksort.
- Complexidade no pior caso é $\Theta(n^2)$ mas o valor esperado da complexidade para uma versão que usa aleatoriedade é $\Theta(n \lg n)$.

Divisão

- Dividir o array em 2 subarrays usando um elemento x como *pivot* de tal forma que os elementos do subarray esquerdo $\leq x \leq$ os elementos do subarray direito.



Conquista (e Combina)

- Conquista: ordenar recursivamente os 2 subarrays.

- Combina: não faz nada.

Quicksort vs. MergeSort

- MergeSort faz o trabalho no passo de combinar.
- QuickSort faz o trabalho no passo de dividir.
- QuickSort ordena no próprio array, MergeSort requer memória auxiliar.

Pseudocódigo

$\textsc{Quicksort}(A, p, r)$
    if $p < r$
        $q = \textsc{Partition}(A, p, r)$
        $\textsc{Quicksort}(A, p, q - 1)$
        $\textsc{Quicksort}(A, q + 1, r)$

Chamada inicial: $\textsc{Quicksort}(A, 1, n)$

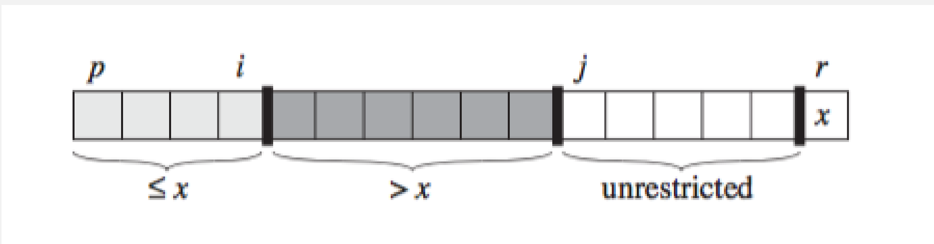
Pseudocódigo para Partition

PARTITION(A, p, r)

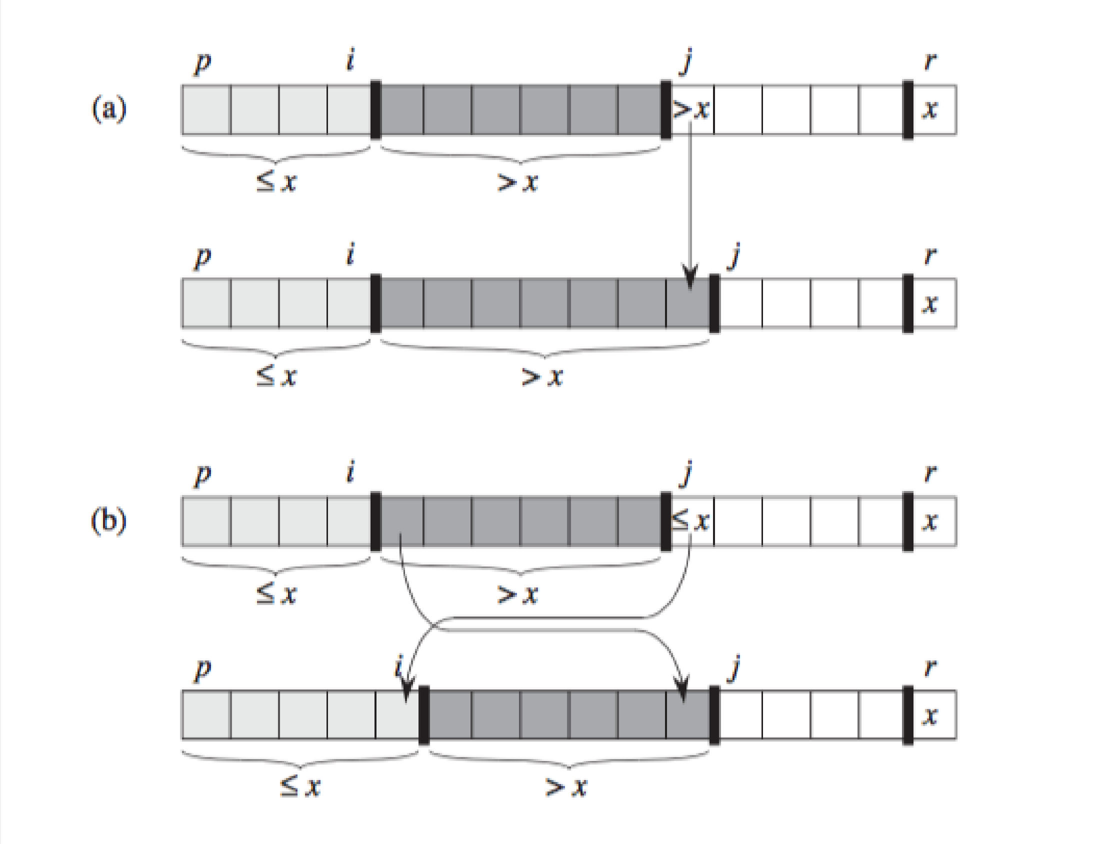
$x = A[r]$ // pivot
$i = p - 1$
for $j = p$ **to** $r - 1$
  if $A[j] \leq x$
    $i = i + 1$
    exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$

Partition

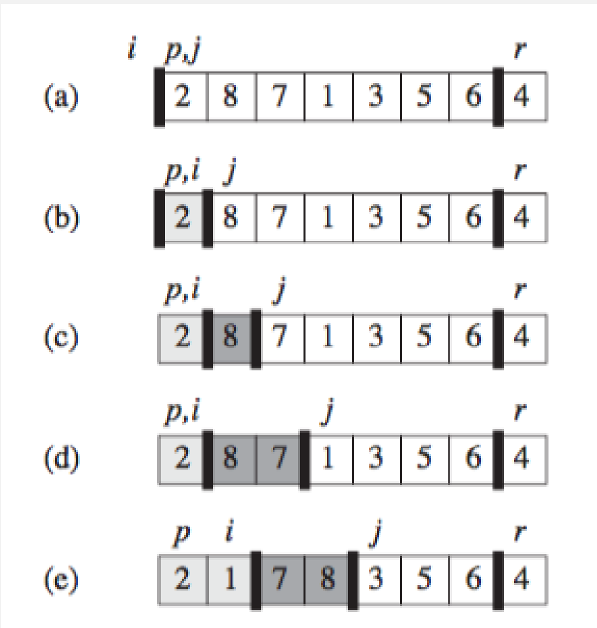
- Durante a execução, divide o array em 4 regiões (que poderão ser vazias).



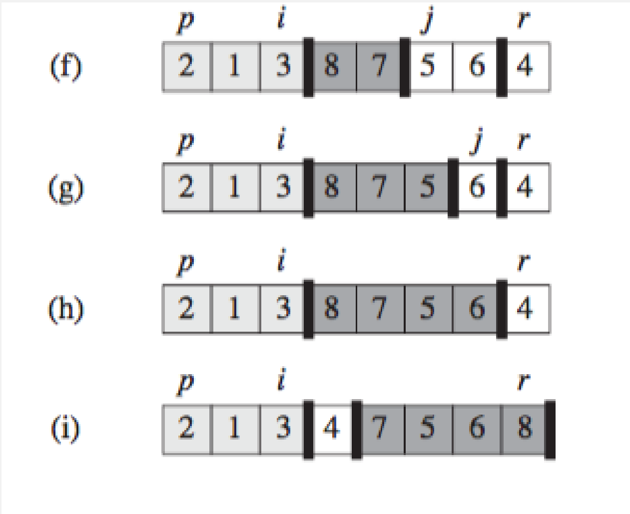
Partition: manutenção de invariante



Partition: exemplo



Partition: exemplo



Análise: pior caso

- Pior caso: o array está ordenado (ou ordenado por ordem inversa).

$$\begin{aligned} T(n) &= T(n-1) + T(1) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

Análise: melhor caso

- Apenas como intuição. O melhor caso não é relevante.
- Melhor caso: os 2 subarrays são do mesmo tamanho.

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

- Igual à recorrência do MergeSort.

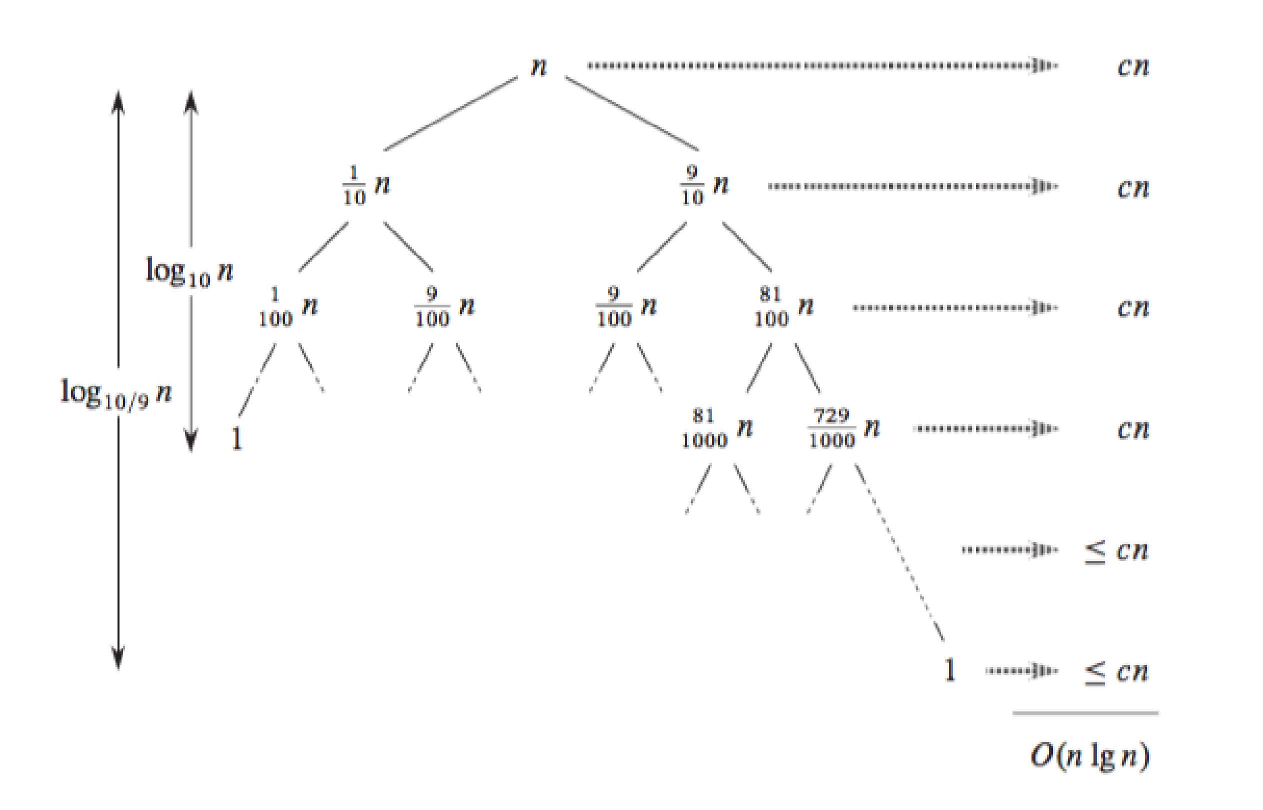
Análise de um caso bastante mau

- Imaginemos que a divisão era sempre feita em 2 subarrays de tamanho $n/10$, $9n/10$.
- A partição é bastante desequilibrada, o que é mau para o algoritmo.

$$T(n) = T(n/10) + T(9n/10) + \Theta(n)$$

- Não se pode aplicar o teorema Mestre.
- Podemos expandir a árvore de recorrências para ganhar uma intuição e tentar provar depois pelo método de substituição.

Expansão da árvore de recorrência



- O ramo mais à direita demora mais tempo a atingir o caso base.
- Vai multiplicando por $\frac{9}{10}$ (ou seja, dividindo por $\frac{10}{9}$).
- Atinge 1 quando,

$$n\left(\frac{9}{10}\right)^h = 1 \implies h = \log_{\frac{10}{9}} n$$

- Base do logaritmo é irrelevante em termos assimptóticos porque,
  - $\log_b a = \frac{\log_c a}{\log_c b}$

Análise

- Pode-se provar pelo método de substituição que $T(n) = \Theta(n \lg n)$.
- Continuaria a ser $\Theta(n \lg n)$ se os 2 subarrays tivessem tamanho $n/1000$, $999n/1000$.
- $T(n) = \Theta(n \lg n)$ sempre que a divisão tenha uma proporcionalidade constante.
- Daria $\Theta(n^2)$ se os 2 subarrays tivessem tamanho k, $n - k$, para uma constante k.

Randomized Quicksort

- Podemos introduzir aleatoriedade para obter boa performance praticamente sempre.
- Ideia: baralhar o input antes de ordenar.
- Ideia mais simples: escolher o pivot de forma aleatória.
- A performance deixa de ser afectada por maus inputs.

Pseudocódigo

RANDOMIZED-PARTITION(A, p, r)

```
    i = RANDOM(p, r)        // random int drawn unif. from [p . . r]
    exchange A[r] with A[i]
    return PARTITION(A, p, r)
```

RANDOMIZED-QUICKSORT(A, p, r)

```
    if p < r
        q = RANDOMIZED-PARTITION(A, p, r)
        RANDOMIZED-QUICKSORT(A, p, q − 1)
        RANDOMIZED-QUICKSORT(A, q + 1, r)
```


Análise de Randomized Quicksort

- No pior caso continua a ser $T(n) = \Theta(n^2)$
- Mas devido à aleatoriedade, o pior caso é praticamente impossível de ocorrer.
- Note a diferença para com a versão standard (não aleatória).
  - array já ordenado ou quase ordenado pode ocorrer com muita frequência na prática.
  - com a aleatoriedade esses casos tem uma probabilidade de ocorrer próxima de zero.

Análise do valor esperado do tempo de execução

- É possível mostrar que o valor esperado do tempo de execução de RANDOMIZED-QUICKSORT é de $O(n \lg n)$.
- Não vos vou exigir que saibam a demonstração.
- Mas está no livro, caso tenham interesse.