Implementação Algorítmica

Quick Sort

Carlos H. A. Higa

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

FACOM-UFMS

UFMS

Motivação

- O QUICKSORT também é chamado de **algoritmo de ordenação por separação**;
- O QUICKSORT tem tempo de pior caso $\Theta(n^2)$ sobre um vetor de entrada com n elementos;
- É o algoritmo de ordenação mais usado na prática devido a sua eficiência na média: seu tempo de execução esperado é $\Theta(n \lg n)$ e os fatores constantes escondidos na notação assintótica são muito pequenos;
- Usa espaço auxiliar constante para executar a ordenação.

FACOM-UFMS

Quicksort

FACOM-UFMS

Divisão-e-conquista

Divida: Particione o vetor $A[p\,..\,r]$ em dois subvetores $A[p\,..\,q-1]$ e $A[q+1\,..\,r]$ tal que cada elemento de $A[p\,..\,q-1]$ é menor ou igual a $A[q]$, que é, por sua vez, menor ou igual a cada elemento de $A[q+1\,..\,r]$. Compute o índice q como parte deste processo de partição.

Conquiste: Ordene os dois subvetores $A[p\,..\,q-1]$ e $A[q+1\,..\,r]$ por chamadas recursivas ao QUICKSORT.

Combine: "Cole" os subvetores $A[p\,..\,q-1]$, $A[q]$ e $A[q+1\,..\,r]$ e obtenha imediatamente o vetor $A[p\,..\,r]$ ordenado

QUICKSORT

FACOM-UFMS

QUICKSORT(A, p, r)

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

Chamada inicial: QUICKSORT$(A, 1, A.\mathit{length})$.

PARTITION

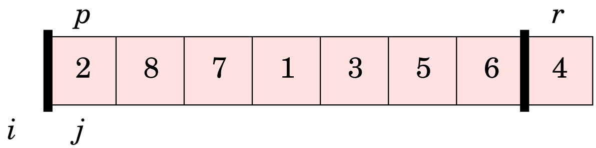
O ponto chave do algoritmo é o procedimento PARTITION, que particiona o vetor $A[p\,..\,r]$.

PARTITION

PARTITION(A, p, r)

$x = A[r]$
$i = p - 1$
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 $i + 1$

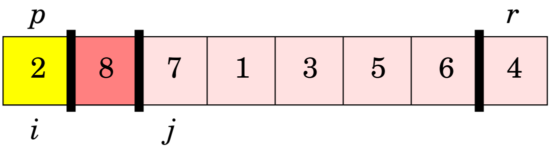
Partition



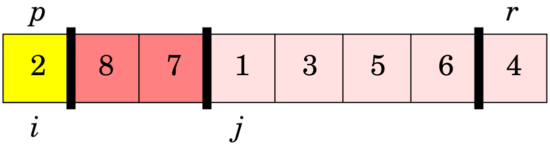
PARTITION

p							r
2	8	7	1	3	5	6	4
i	j						

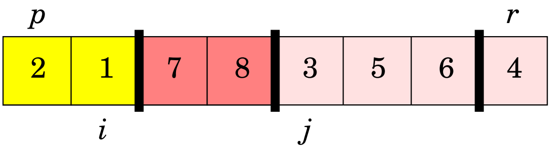
PARTITION



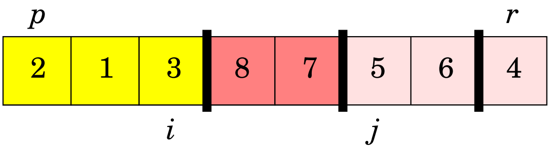
PARTITION



Partition

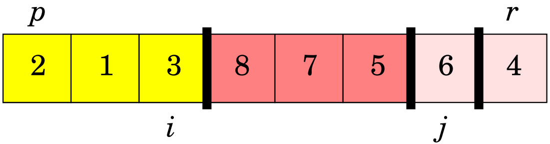


PARTITION

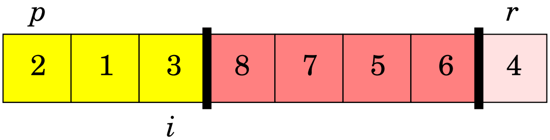


FACOM-UFMS

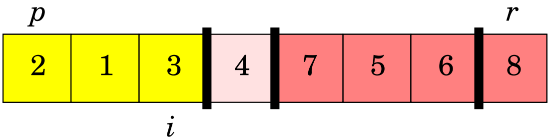
PARTITION



Partition



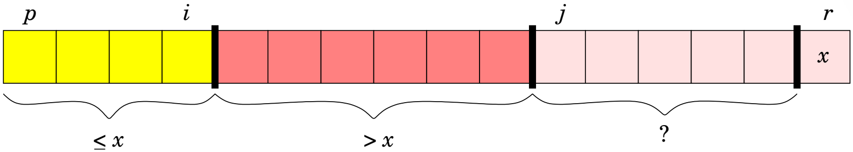
Partition



PARTITION

- O algoritmo PARTITION sempre seleciona um elemento $x = A[r]$ como um **pivô** em torno do qual a separação do vetor $A[p \mathinner{..} r]$ será realizada;
- Durante a execução do algoritmo PARTITION, o vetor $A[p \mathinner{..} r]$ é particionado em 4 regiões;
- No começo de cada iteração da estrutura de repetição das linhas $3 - 6$, as regiões satisfazem certas propriedades, que podem ser descritas como o invariante da estrutura de repetição.

PARTITION



QUICKSORT

FACOM-UFMS

Invariante:
No começo de cada iteração da estrutura de repetição das linhas $3-6$, para qualquer índice k do vetor A,

1. Se $p \leq k \leq i$, então $A[k] \leq x$.
2. Se $i+1 \leq k \leq j-1$, então $A[k] > x$.
3. Se $k = r$, então $A[k] = x$.

- Os índices entre j e $r-1$ não são cobertos por qualquer dos 3 casos e os valores nesses compartimentos não têm relação direta com o pivô x;
- Precisamos mostrar que este invariante é verdadeiro antes da primeira iteração, que cada iteração da estrutura de repetição mantém o invariante e que o invariante fornece uma propriedade útil para mostrar a correção do algoritmo quando a estrutura de repetição termina.

PARTITION

Correção do algoritmo PARTITION

Inicialização: Antes da primeira iteração da estrutura de repetição, $i = p - 1$ e $j = p$. Como não há valores armazenados no vetor A entre p e i e entre $i + 1$ e $j - 1$, as primeiras duas condições do invariante são trivialmente satisfeitas. A atribuição na linha 1 satisfaz a terceira condição.

FACOM-UFMS

Partition

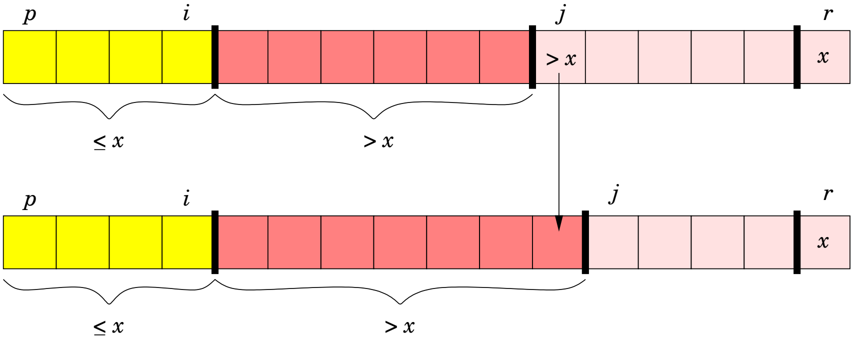
Correção do algoritmo Partition

Manutenção: Temos de considerar dois casos que dependem do resultado do teste da linha 4.

Primeiro caso: se $A[j] > x$ então j é incrementado. Depois que j é incrementado, a condição 2 é satisfeita para $A[j-1]$ e todas os outros elementos permanecem não modificados.

Partition

Primeiro caso:



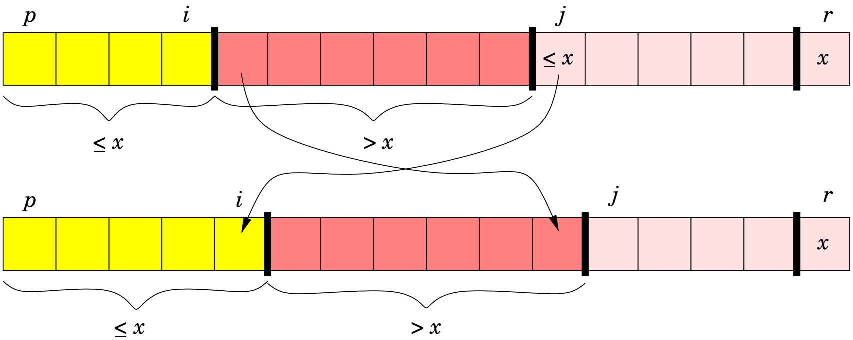
PARTITION

Correção do algoritmo PARTITION

Manutenção: Segundo caso: se $A[j] \leq x$ então i é incrementado, $A[i]$ é trocado com $A[j]$ e depois j é incrementado. Devido à troca, temos agora que $A[i] \leq x$ e a condição 1 é satisfeita. Similarmente, temos também que $A[j-1] > x$, já que o elemento que foi trocado em $A[j-1]$ é, pelo invariante, maior que x.

PARTITION

Segundo caso:



PARTITION

Correção do algoritmo PARTITION

Término: No final, $j = r$. Portanto, toda entrada no vetor está em um dos 3 conjuntos descritos pelo invariante e temos separado os valores do vetor em 3 conjuntos: aqueles menores ou iguais a x, aqueles maiores que x e um conjunto unitário contendo x.

FACOM-UFMS

PARTITION

Correção do algoritmo PARTITION

- As duas últimas linhas do algoritmo PARTITION trocam o pivô com o elemento mais à esquerda do conjunto dos elementos maiores que x, movendo portanto o pivô para a posição correta no vetor particionado e devolvendo o novo índice do pivô;
- A saída do algoritmo PARTITION satisfaz, então, as especificações da etapa da divisão.

PARTITION

Tempo de execução do algoritmo PARTITION

- Como cada elemento do subvetor $A[p\,..\,r-1]$ é comparado uma única vez, na linha 4, com o pivô $A[r]=x$, o tempo de execução do algoritmo PARTITION é $\Theta(n)$, onde $n=r-p+1$.

FACOM-UFMS

Desempenho do QUICKSORT

- O desempenho do QUICKSORT depende se a separação é balanceada ou não, que por sua vez depende de quais elementos são usados para realizar a separação;
- Se a separação é balanceada, o algoritmo é assintoticamente tão rápido quanto o MERGESORT;
- Se a separação não é balanceada, o algoritmo pode ser assintoticamente tão lento quanto o INSERTIONSORT.

Desempenho do QUICKSORT

Separação de pior caso

- O comportamento de pior caso do QUICKSORT ocorre quando a separação produz um subproblema com $n-1$ elementos e um outro com 0 elementos;
- Considere que esta separação desbalanceada ocorra em cada chamada recursiva;
- O custo da separação é $\Theta(n)$;
- Como uma chamada recursiva ao QUICKSORT sobre um vetor de tamanho 0 nada faz, temos que $T(0) = \Theta(1)$.

Desempenho do QUICKSORT

- Então a recorrência do tempo de execução de pior caso é

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \,, \\ &= T(n-1) + \Theta(n) \,. \end{aligned}$$

- Podemos usar o método da substituição para mostrar que $T(n) = T(n-1) + \Theta(n)$ é tal que $T(n) = \Theta(n^2)$;
- Isso significa que se a separação é maximamente desbalanceada em todo nível da recursão, o tempo de execução é $\Theta(n^2)$;
- Portanto o tempo de execução de pior caso do QUICKSORT não é melhor que do INSERTIONSORT
- Além disso, o tempo de execução $\Theta(n^2)$ ocorre quando o vetor de entrada está ordenado, uma situação em que o INSERTIONSORT tem tempo de execução $O(n)$.

Desempenho do QUICKSORT

FACOM-UFMS

Separação de melhor caso

- No outro extremo, a separação produz dois subproblemas, cada um de tamanho não maior que $n/2$, já que um é de tamanho $\lfloor n/2 \rfloor$ e o outro é $\lceil n/2 \rceil - 1$;
- A recorrência para o tempo de execução é então

$$T(n) = 2T(n/2) + \Theta(n)$$

- Podemos usar o caso 2 do Teorema Mestre e obter a solução $T(n) = \Theta(n \lg n)$;
- Se os lados da partição são balanceados igualmente em cada nível da recursão, temos um algoritmo assintoticamente mais rápido.

Desempenho do QUICKSORT

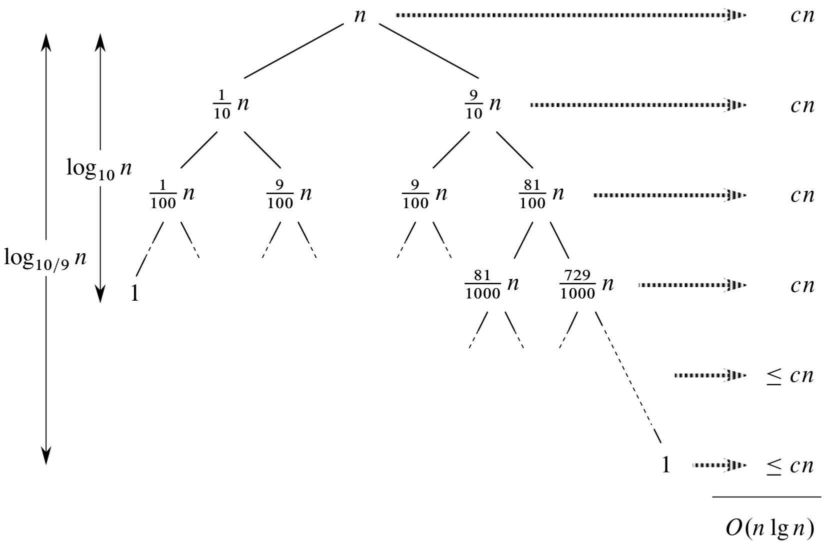
Separação balanceada

- O tempo de execução do caso médio do QUICKSORT é muito mais próximo do melhor caso do que do pior caso;
- É necessário compreender como o balanceamento da separação é refletido na recorrência que descreve o tempo de execução;
- suponha, por exemplo, que o algoritmo de separação sempre produz uma quebra de proporção 9-para-1, que parece bastante desbalanceada;
- Obtemos então a recorrência

$$T(n) = T(9n/10) + T(n/10) + cn \, ,$$

para o tempo de execução do QUICKSORT.

Desempenho do QUICKSORT



Desempenho do QUICKSORT

Separação balanceada

- Assim, com uma quebra proporcional a 9-para-1 em todo nível da recursão, o QUICKSORT tem tempo de execução $O(n \lg n)$;
- Este tempo é assintoticamente o mesmo se a quebra fosse sempre no meio do vetor;
- Mesmo com uma quebra proporcional a 99-para-1, ainda assim o tempo de execução é $O(n \lg n)$;
- Qualquer quebra de proprocionalidade **constante** fornece uma árvore de recursão de profundidade $\Theta(\lg n)$, onde o custo de cada nível é $O(n)$.

Desempenho do QUICKSORT

FACOM-UFMS

Intuição para o caso médio

- Considere que todas as permutações dos números de entrada são igualmente prováveis;
- Quando executamos o QUICKSORT sobre um vetor aleatório de entrada, é altamente improvável que a separação ocorra da mesma forma em todos os níveis;
- Esperamos que algumas das separações sejam razoavelmente balanceadas e que algumas sejam bastante desbalanceadas;
- No caso médio, o algoritmo PARTITION produz um misto de quebras "boas" e "ruins";
- Na árvore de recursão para uma execução de caso médio do algoritmo PARTITION, as quebras boas e ruins são distribuídas aleatoriamente pela árvore.