

Projeto e Análise de Algoritmos

Flávio L. C. de Moura*

10 de março de 2022

1 Quicksort

O algoritmo Quicksort, assim como *merge sort*, utiliza o paradigma de 'divisão e conquista' para ordenar um vetor $A[1..n]$. As etapas para ordenar um subvetor $A[p..r]$ ($1 \leq p \leq r \leq n$) são as seguintes:

1. **Dividir:** Esta etapa consiste em particionar o vetor $A[p..r]$ em dois subvetores (possivelmente vazios) $A[p..q-1]$ e $A[q+1..r]$ tais que cada elemento do vetor $A[p..q-1]$ é menor ou igual a $A[q]$, que por sua vez também é menor ou igual do que cada elemento do vetor $A[q+1..r]$. Esta etapa também computa o índice q do particionamento.
2. **Conquistar:** Esta etapa consiste em recursivamente ordenar os subvetores $A[p..q-1]$ e $A[q+1..r]$.

Assim, a ideia do algoritmo pode ser apresentada a partir do pseudocódigo a seguir:

Algorithm 1: Quicksort(A, p, r)

```
1 if  $p < r$  then
2    $q = \text{Partition}(A, p, r)$ ;
3   Quicksort( $A, p, q - 1$ );
4   Quicksort( $A, q + 1, r$ );
5 end
```

A ordenação do vetor $A[1..n]$ é, então, obtida pela chamada $\text{Quicksort}(A, 1, n)$.

O particionamento do vetor consiste na principal etapa do algoritmo Quicksort:

Algorithm 2: Partition(A, p, r)

```
1  $x = A[r]$ ;
2  $i = p - 1$ ;
3 for  $j = p$  to  $r - 1$  do
4   if  $A[j] \leq x$  then
5      $i = i + 1$ ;
6     exchange  $A[i]$  com  $A[j]$ ;
7   end
8 end
9 exchange  $A[i + 1]$  with  $A[r]$ ;
10 return  $i + 1$ ;
```

Exercício 1.1. Prove a seguinte invariante de laço, e conclua que o algoritmo Partition é correto:

Antes de cada iteração do laço **for** (linhas 3-8), para todo k , temos:

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

*flavio@flaviomoura.info

3. Se $k = r$, então $A[k] = x$.

O número de comparações (linha 4) feitas por Partition em um vetor com n elementos é $\Theta(n)$. Qual seria, então, o tempo de execução de Quicksort?

O tempo de execução $T(n)$ de Quicksort, no pior caso, para entradas de tamanho n ocorre quando o particionamento gera um vetor vazio, e outro com $n - 1$ elementos. Neste caso dizemos que o particionamento é *desbalanceado*. Se assumirmos que este desbalanceamento ocorre em cada chamada recursiva, podemos modelar o processo pela seguinte equação de recorrência:

$$T_w(n) = T_w(n - 1) + T_w(0) + \Theta(n)$$

Como não há trabalho a fazer em um vetor vazio, temos que $T(0) = \Theta(1)$, e portanto a recorrência acima pode ser reescrita como:

$$T_w(n) = T_w(n - 1) + \Theta(n) \tag{1}$$

Como exercício, mostre que $T_w(n) = \Theta(n^2)$, e observe que esta situação ocorre, por exemplo, quando o vetor dado como argumento já está ordenado.

Por outro lado, quando o particionamento é balanceado temos o melhor caso para Quicksort. Agora, o particionamento divide o problema original em dois subproblemas sendo um de tamanho $\lfloor \frac{n}{2} \rfloor$, e outro de tamanho $\lceil \frac{n}{2} - 1 \rceil$, o que nos dá a recorrência:

$$T_b(n) = T_b(\lfloor \frac{n}{2} \rfloor) + T_b(\lceil \frac{n}{2} - 1 \rceil) + \Theta(n)$$

Se ignorarmos as funções de aproximação e a subtração por 1, temos a recorrência:

$$T_b(n) = 2.T_b(\frac{n}{2}) + \Theta(n)$$

que, pelo Teorema Mestre, tem solução $T_b(n) = \Theta(n \lg n)$.

Assim, o tempo de execução de Quicksort depende se o particionamento está balanceado ou não: Em caso afirmativo, Quicksort é assintoticamente tão rápido quanto *mergesort*, mas se o particionamento não estiver balanceado, Quicksort tem o mesmo comportamento assintótico de *Insertion sort* no pior caso.

O que ocorre se o particionamento é sempre da ordem de 9-1, *i.e.* 90% dos elementos são menores ou iguais ao pivô, e apenas 10% são maiores do que o pivô? (Exercício!) A resposta desta pergunta sugere que a complexidade do caso médio para Quicksort está mais próxima do melhor caso do que do pior caso.

Exercício 1.2. *Mostre que a complexidade de Quicksort, no melhor caso, é $\Omega(n \lg n)$.*

Exercício 1.3. *Mostre que o algoritmo Quicksort é correto.*