Capítulo 7

O algoritmo Quicksort

O algoritmo Quicksort, assim como *merge sort*, utiliza o paradigma de dividir para conquistar para ordenar um vetor $A[1..n]$. As etapas para ordenar um subvetor $A[p..r]$ 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 12: Quicksort(A, p, r)

```
1 if p < r then
2 |   q = 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 Quicksort$(A, 1, n)$.

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

Algorithm 13: Partition(A, p, r)

```
1  x = A[r];
2  i = p − 1;
3  for j = p to r − 1 do
4  |  if A[j] ≤ 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;
```

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?

Afirmação: O pior caso para o tempo de execução $T(n)$ de Quicksort para entradas de tamanho n ocorre quando o particionamento gera um vetor vazio, e outro com $n-1$ elementos.

Provaremos esta afirmação posteriormente, mas agora vejamos a análise do algoritmo quando temos o particionamento *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:

$$T_w(n) = T_w(n-1) + \Theta(n) \tag{7.1}$$

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 *merge sort*, mas se o particionamento não estiver balanceado, Quicksort tem o mesmo comportamento assintótico de *Insertion sort* no pior caso.

Agora faremos a prova da afirmação feita acima, isto é, a prova de que a análise do pior caso se dá quando o particionamento é completamente desbalanceado (um dos subvetores não possui elementos). Iniciamos a partir da seguinte recorrência:

$$T(n) \leq \max_{0 \leq q < n} (T(q) + T(n-q-1)) + \Theta(n) \tag{7.2}$$

A partir da solução vista anteriormente quando $q = 0$, parece razoável acreditar que $T(n) \leq c.n^2$ para alguma constante positiva c. Substituindo esta possível solução, temos

$$T(n) \leq \max_{0 \leq q < n} (c(q^2 + (n-q-1)^2) + \Theta(n) = c. \max_{0 \leq q < n} (q^2 + (n-q-1)^2) + \Theta(n)$$

e a expressão $q^2 + (n-q-1)^2$ assume valor máximo quando $q = 0$ ou $q = n-1$ (Exercício!), portanto

$$T(n) \leq c. \max_{0 \leq q < n} (q^2 + (n-q-1)^2) + \Theta(n) \leq (n-1)^2$$

Assim, podemos escrever $T(n) \leq c.n^2 - c.(2n-1) + \Theta(n) \leq c.n^2$, já que podemos tomar c grande o suficiente para dominar $\Theta(n)$. Ou seja, $T(n) = O(n^2)$.

Exercício 7.0.1. *Para a recorrência* (7.2)*, mostre* $T(n) = \Omega(n^2)$ *e conclua que o tempo de execução de Quicksort, no pior caso, é* $\Theta(n^2)$.