# Notas de Aula de QuickSort - Técnicas de Busca e Ordenação 2004/01

Mike Wesley Blunk

UFES - Vitóia, ES - 22/12/2004

## 1 Introdução

O *Quicksort* figura entre os dez mais famosos algortimos. O *Quicksort* é um bom exemplo de aplicação do *Métododo Dividir e Conquistar*.

### 1.1 Dividir e Conquistar

*Dividir:* O vetor $A[p..r]$ é particionado em dois sub-vetores não vazios $A[p..q]$ e $A[q+1..r]$ tal que cada elemento de A[p..q] é menor ou igual a cada elemento de A[q+1..r].

*Conquistar:* Ambos os vetores são ordenados utilizando o *Quicksort* .

*Combinar:* Não faça nada.

**Algorithm 1** O algoritmo do QuickSort

```
1: procedure QUICKSORT(A, p, r)
2:     if p < r then
3:         q ← Particionar(A, p, r)
4:         QuickSort(A, p, q)
5:         QuickSort(A, q + 1, r)
6:     end if
7: end procedure
```

**Algorithm 2** O algoritmo de Particionamento do Quicksort

```
1: procedure PARTICIONAR(A, p, r)
2:     x ← A[p]
3:     i ← p − 1
4:     j ← r + 1
5:     while true do
6:         repeat
7:             j ← j − 1
8:         until A[j] ≤ x
9:         repeat
10:            i ← i + 1
11:        until A[i] ≥ x
12:        if i < j then
13:            Troque(A[i], A[j])
14:        else
15:            return j
16:        end if
17:    end while
18: end procedure
```

*Observação:* Não escolha o último elemento do vetor como pivô, pois se o último elemento for o maior elemento do vetor o *Quicksort* entra num laço infinto na primeira chamada recursiva do algoritmo.

# 2 Análise do Algoritmo do *Quicksort*

O tempo de execução do *Quicksort* depende se o particionamento é balanceado ou não. O que por sua vez depende da escolha do elemento utilizado como pivô.

## 2.1 Desempenho no Pior Caso (Vetor Ordenado)

O pior caso da ordenação de um vetor ocorre quando o mesmo já está ordenado.

$$T(n) = T(n-1) + T(1) + \theta(n) \tag{1}$$

$$T(n) \quad = \quad 2 + \sum_{i=3}^{n}(i) = 2 + \frac{(n+3)(n-2)}{2} + n \tag{2}$$

$$T(n) = 2 + \frac{n^2 + 3n - 2n - 6}{2} + n \quad (3)$$

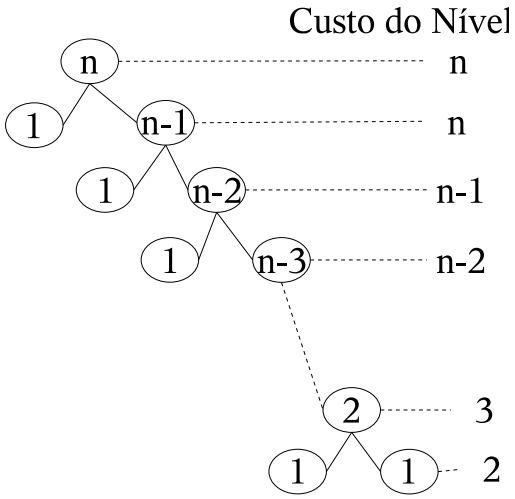


Figure 1: Pior Caso do *Quicksort*

## 2.2 Desempenho no Melhor Caso

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n) \quad (4)$$
$$T(n) = (n \cdot \lg n) \quad (5)$$

*Observação:* As constantes e os termos de menor ordem do *Quicksort* são menores que os do *MergeSort*.

## 2.3 Desempenho 9/10

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + \theta(n) \quad (6)$$
$$h = \log_{10/9} n \quad (7)$$
$$T(n) \; leq \; n \cdot \log_{10/9} n \quad (8)$$
$$T(n) \; leq \; O(n \cdot \lg n) \quad (9)$$
$$(10)$$

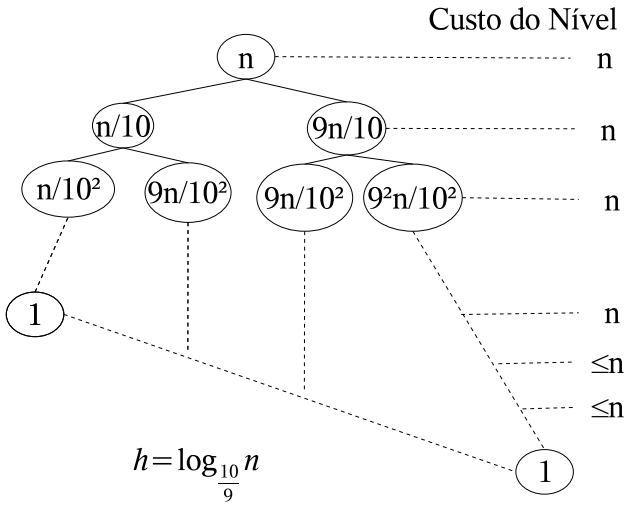


Figure 2: Desempenho 9/10 do *Quicksort*

# 3 *Quicksort* Aleatório

Um algortimo é aleatório se o seu comportamento é determinado não apenas pela entrada, mas também pelos valores produzidos por um gerador de números aleatórios. Nenhuma entrada particular garante o comportamento no pior caso. Duas versões possíveis do *Quicksort*

- A primeira versão embaralha a entrada, de modo a assegurar que toda a permutação é igualemnte provável.
- A segunda vesão troca $A[p]$ com um elemento escolhido aleatoriamente de $A[p..r]$. Isto assegura que o elemento pivô é igualmente provável para todos os elementos existentes na entrada.

Estas vesões não melhoram o tempo de execução do *Quicksort* , mas tornam o tempo de execução independente da ordenação da entrada.

**Algorithm 3** O algoritmo do QuickSort Aleatório

```
1: procedure QUICKSORT_ALEATORIO(A, p, r)
2:     if p < r then
3:         q ← Particionar_Aleatorio(A, p, r)
4:         QuickSort_Aleatorio(A, p, q)
5:         QuickSort_Aleatorio(A, q + 1, r)
6:     end if
7: end procedure
```

**Algorithm 4** O algoritmo de Particionamento do Quicksort Aleatório

```
1: procedure PARTICIONAR_ALEATORIO(A, p, r)
2:     i ← Random(p, r)
3:     Troque(A[p], A[i])
4:     return Particionar(A, p, r)
5: end procedure
```

# 4 Análise do *Quicksort* Aleatório

Para $1 \leq i \leq n$, $S_{(i)}$ denota o elemento de rank $i$ (o $i$-ésimo menor elemento) no conjunto $S$. Considere a variável $X_{ij}$, que assume o valor 1 se $S_{(i)}$ e $S_{(j)}$ são comparados durante a execução do *Quicksort* , e assume o valor 0 (zero) caso contrário. Dessa forma $X_{ij}$ é um contador do número de comparações. Logo o número total de comparações é:

$$TOTALCOMP = \sum_{i=1}^{n} \left( \sum_{j>i}^{n} (X_{ij}) \right) \tag{11}$$

Nós estamos interessados no número médio de comparações

$$E[TOTALCOMP] = E\left[ \sum_{i=1}^{n} \left( \sum_{j>i}^{n} (X_{ij}) \right) \right] = \sum_{i=1}^{n} \left( \sum_{j>i}^{n} (E[X_{ij}]) \right) \tag{12}$$

Assuma que $p_{ij}$ é a probabilidade de $S_{(i)}$ ser comparado a $S_{(j)}$ em uma execução do *Quicksort* .

$$E[X_{ij}] = 1 \cdot p_{ij} + 0 \cdot (1 - p_{ij}) \tag{13}$$

$$E[X_{ij}] = p_{ij} \tag{14}$$

$$E[TOTALCOMP] = E\left[ \sum_{i=1}^{n} \left( \sum_{j>i}^{n} (p_{ij}) \right) \right] \tag{15}$$

Para facilitar a determinação de $p_{ij}$, nós consideramos a execução de do *Quicksort* como uma árvore binária de pesquisa $T$, cada nó da árvore é rotulado com um elemento de $S$.

S raiz $y$ é comparada com os elementos nas duas sub-árvores, mas *nenhuma* comparação é executada entre um elemento da sub-árvore esquerda e um elemento da sub-árvore direita. Dessa forma existe uma comparação entre $S_{(i)}$ e $S_{(j)}$, se e somente se $S_{(i)}$ é antecessor de $S_{(j)}$ ou vice-versa.

Para computar $p_{ij}$ nós fazemos duas observações:

1. Existe uma comparação entre $S_{(i)}$ e $S_{(j)}$, se e somente se $S_{(i)}$ ou $S_{(j)}$ ocorre primeiro na permutação pi que qualquer elemento de $S_{(k)}$ tal que $i < k < j$

2. Qualquer um dos elementos $S_{(i)}$, $S_{(i+1)}$, $\ldots$, $S_{(j)}$ é igualmente provável de ser o primeiro elemento a ser escolhido como pivô, e assim aparecer primeiro em pi. Dessa forma, a probabilidade que este elemento seja $S_{(i)}$ ou $S_{(j)}$ é:

$$\frac{2}{j-i+1} = p_{ij} \tag{16}$$

$$\begin{aligned}
E\left[TOTALCOMP\right] &= \sum_{i=1}^{n}\left(\sum_{j>i}^{n}\left(\frac{2}{j-i+1}\right)\right) && (17)\\
&= \sum_{i=1}^{n}\left(\sum_{k=1}^{n-i}\left(\frac{2}{k+1}\right)\right) && (18)\\
&= 2\sum_{i=1}^{n}\left(\sum_{k=1}^{n-i}\left(\frac{1}{k+1}\right)\right) && (19)\\
&\leq 2\sum_{i=1}^{n}\left(\sum_{k=1}^{n-i}\left(\frac{1}{k}\right)\right) && (20)\\
&\leq 2\sum_{i=1}^{n}\left(\sum_{k=1}^{n}\left(\frac{1}{k}\right)\right) && (21)\\
E\left[TOTALCOMP\right] &= 2\sum_{i=1}^{n}\left(H_n\right) && (22)\\
& && (23)
\end{aligned}$$

*Observação:* $H_n$ é chamado de *número harmônico*, denotado por:

$$
\begin{aligned}
H_n &= \sum_{k=1}^{n}(\frac{1}{k}) && (24)\\
&= \frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n} && (25)\\
H_n &= O(\lg n) && (26)
\end{aligned}
$$

Logo:

$$
\begin{aligned}
E\left[TOTALCOMP\right] &\leq 2n \cdot H_n && (27)\\
E\left[TOTALCOMP\right] &\leq 2c \cdot n \cdot H_n && (28)\\
E\left[TOTALCOMP\right] &= O(n \cdot \lg n) && (29)
\end{aligned}
$$