Algoritmos

Pedro Hokama

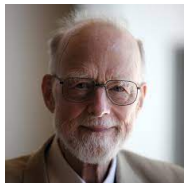
Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Sir Charles Antony Richard Hoare

- Cientista da Computação Britânico nascido em 1934
- Ganhador de diversos prêmios na Área da Computação
- I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.
- Inventou em 1959 (e publicou em 1961) o QuickSort.



QuickSort

- Porque ver e rever o QuickSort?
  - Um dos Greatest Hits da ciência da computação.
  - Muito usado na prática.
  - Memória extra constante.
  - Análise muito interessante.

Problema da Ordenação

Dado um arranjo de n inteiros distintos, encontrar o arranjo $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ que contenha os mesmos elementos mas ordenados de maneira não decrescente, ou seja, $\pi_i \leq \pi_j$ para qualquer $i < j$ e $i, j \in \{1, \ldots, n\}$.

Partição

O QuickSort depende fortemente da operação de Partição cuja a ideia é particionar o arranjo em torno de um pivô:

- Escolha um pivô.
- Reorganize os elementos do arranjo de forma que:
  - A esquerda do pivô tenha os elementos menores que o pivô.
  - A direita do pivô tenha os elementos maiores que o pivô.

Exemplo

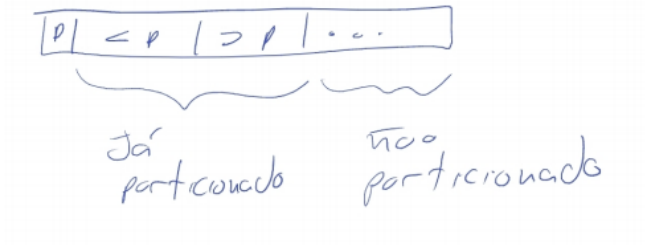
No vetor (3, 8, 2, 5, 1, 4, 7, 6), se escolhemos 3 como pivô.
Uma partição seria: (2, 1, 3, 6, 7, 4, 5, 8)

Observe que depois de uma partição o pivô estará na sua posição correta. Note também que não nos importamos com a posição relativa de cada uma das partes esquerda e direita.

- Como particionar em $O(n)$ usando memória extra?
- Podemos fazer a Partição em tempo $O(n)$ sem usar memória extra.

Ideia:

- Manter uma porção do arranjo já particionado, e o restante ainda não particionado



- Percorrer uma única vez o arranjo

Algoritmo 1: Partição
Entrada: Um arranjo A, índices l e r
Saída: O mesmo arranjo mas particionado

```
1 p = A[l];
2 i = l + 1;
3 para j = l + 1 até r faça
4     se A[j] < p então
5         Troca A[j] e A[i];
6         i = i + 1;
7 Troca A[l] e A[i − 1];
8 devolva A;
```

Provas por Indução - Revisão

Vamos revisar o que é uma prova por indução, que utiliza o seguinte axioma:

Principio da Indução Completa

Seja $P(n)$ uma sentença aberta sobre $\mathbb{N}$. Suponha que:

1. $P(0)$ é verdade, e
2. para todo $k \in \mathbb{N}$, $((\forall i \in \mathbb{N}) i \leq k \rightarrow P(i)) \rightarrow P(k+1)$

Então $P(n)$ é verdade para toda $n \in \mathbb{N}$.

Ou seja se $P(0), P(1), \ldots, P(k)$ é verdade então $P(k+1)$ também é verdade.

Para usar esse método podemos utilizar o seguinte roteiro:

- *Base da Indução*: Provar que $P(0)$ é verdade.
- *Hipótese de Indução*: Supor que $P(i)$ é verdade para todo $i \leq k \in \mathbb{N}$.
- *Passo da Indução*: Provar que $P(k+1)$ é verdade.

Exemplo: Provar que, para todo $n \geq 0$:

$$1 + 3 + 5 + \ldots + (2n+1) = (n+1)^2$$

- Base da Indução: $P(0)$ é verdade pois

$$1 = (0+1)^2 = 1$$

- Hipótese de Indução: Suponha que $P(i)$ é verdade para todo $i \leq k \in \mathbb{N}$.
- Passo da Indução: Provar que $P(k+1)$ é verdade, ou seja que:

$$1 + 3 + 5 + \ldots (2k+1) + (2(k+1)+1) = ((k+1)+1)^2$$

$$[1+3+5+\ldots+(2k+1)]+(2(k+1)+1)) = [(k+1)^2]+(2(k+1)+1))$$

$$= k^2+2k+1+2k+3 = k^2+4k+4 = (k+2)^2 = ((k+1)+1)^2 \quad \square$$

Corretude do algoritmo partição

Invariante de laço: Os elementos $A[l+1], \ldots, A[i-1]$ são todos menores que o pivô, e os elementos $A[i], \ldots, A[j-1]$ são todos maiores que o pivô.

- **Inicialização:** Por vacuidade, antes da primeira iteração a invariante vale.
- **Manutenção:** Em cada iteração verificamos se $A[j]$ é menor que p e nesse caso passamos ele para a primeira porção, incrementando o i. E dessa forma a invariante de mantêm.
- **Termino:** Ao final do laço trocamos o pivô pelo ultimo elemento da primeira porção (ou ele mesmo caso a primeira porção seja vazia) e o arranjo estará particionado.

QuickSort

Algoritmo 2: QuickSort

Entrada: Um arranjo A, o comprimento do arranjo n
Saída: Um arranjo com os mesmos elementos de A porém ordenados

1 **se** $n \leq 1$ **então** devolva A;
2 $p =$ EscolhePivo(A, n);
3 Particionar A em torno de p;
4 Recursivamente ordenar a parte esquerda;
5 Recursivamente ordenar a parte direita;
6 devolva A;

QuickSort

Algoritmo 3: QuickSort

Entrada: Um arranjo A, o comprimento do arranjo n
Saída: Um arranjo com os mesmos elementos de A porém ordenados

1 **se** $n \leq 1$ **então** devolva A;
2 $p =$ EscolhePivo(A, n);
3 Particionar A em torno de p;
4 Recursivamente ordenar a parte esquerda;
5 Recursivamente ordenar a parte direita;
6 devolva A;

Qual o tempo de execução da fase de combinação?

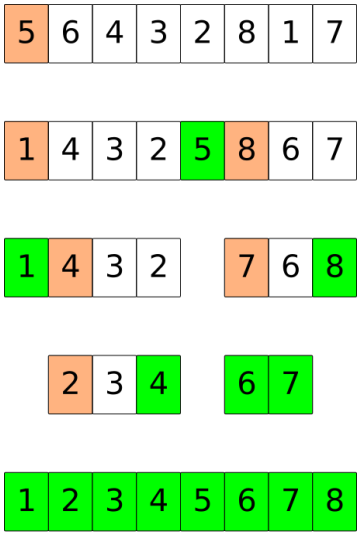
Teorema

O algoritmo QuickSort ordena corretamente um vetor.

Iremos provar a corretude do QuickSort por indução no comprimento n do arranjo. Considere a seguinte sentença aberta $P(n)$: "O QuickSort corretamente ordena arranjos de comprimento n.

- *Base da Indução*: Quando $n = 1$ o algoritmo não faz nada e o vetor está trivialmente ordenado. Portando $P(1)$ é verdade.
- *Hipótese de Indução*: $P(k)$ é verdade para $k < n$
- *Passo da Indução*: Agora queremos provar $P(k+1)$ é verdade.
  - O algoritmo escolhe algum pivô p e particiona o arranjo
  - O pivô termina já em seu lugar correto
  - O algoritmo recursivamente ordena a primeira e a segunda porção que necessariamente são menores que k, logo pela hipótese de indução, são ordenadas corretamente. □

QuickSort - revisão

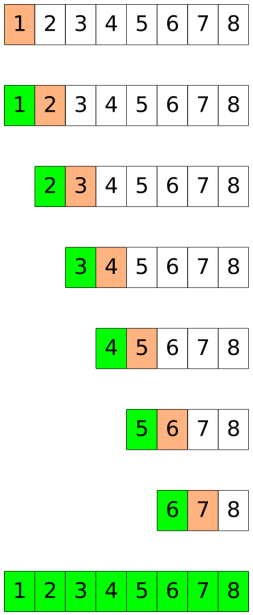


QuickSort

- Vamos analisar a complexidade do QuickSort.
- O que acontece com o QuickSort se a escolha de pivô for muito ruim?
- Como seria a árvore de recursão e complexidade no pior caso? Por exemplo se você tiver um arranjo já ordenado, e sempre escolhermos o primeiro elemento.

Suponha que implementamos o QuickSort, de forma que o pivô é sempre o primeiro elemento do arranjo. Qual é o tempo de execução desse algoritmo em um vetor de entrada que já está ordenado?

a. Não é possível estimar
b. $\Theta(n)$
c. $\Theta(n \log n)$
d. $\Theta(n^2)$



QuickSort

- Nesse caso iremos dividir em duas partes, uma vazia e uma com $n-1$ elementos. E o trabalho da partição será pelo menos:

$$n + (n-1) + (n-2) + \ldots + 1 = \Theta(n^2)$$

- O tempo de execução do QuickSort depende da qualidade do pivô escolhido.
- Um bom pivô é aquele que divide os problemas em partes de tamanho parecido,
- enquanto um pivô ruim é aquele que deixa duas partes muito desiguais.

QuickSort - o caso bom

- Se dividirmos o arranjo sempre no máximo, ou seja, na metade.
- Isso aconteceria se encontrássemos a mediana.
- Teremos duas partes com menos que metade dos elementos
- Podemos limitar superiormente pela mesma recorrência do MergeSort
- Sabemos então que no melhor caso QuickSort é $O(n \log n)$

Seja $T(n)$ o tempo de execução desse algoritmo em um vetor de tamanho n. Então

$$T(n) \leq 2T\left(\frac{n}{2}\right) + \Theta(n)$$

Pelo teorema mestre:

$$T(n) = O(n \log n)$$

Suponha que implementamos o QuickSort, de forma que (magicamente) sempre escolhemos a mediana como pivô. Qual é o tempo de execução desse algoritmo?

a. Não é possível estimar
b. $\Theta(n)$
c. $\Theta(n \log n)$
d. $\Theta(n^2)$

- Infelizmente não é possível encontrar a mediana em $O(1)$ para garantir esse tempo de execução.
- Felizmente não é necessário!

Aleatorização do QuickSort

- A aleatorização de algoritmos é uma ferramenta importante da bagagem de qualquer profissional da computação.
- Veremos como aplicar essa técnica no QuickSort e as vantagens que ela pode trazer.
- A ideia é escolher é escolher cada pivô aleatoriamente com a **mesma probabilidade**. E assim escolher um pivô "razoavelmente bom", em uma frequência "razoavelmente alta".
- Digamos que um pivô razoavelmente bom seria um que divida o arranjo em 25-75%, que é suficiente para garantir o tempo de execução de $O(n \log n)$
- Como seria a árvore de recursão nesse caso?
- Note que metade dos elementos, se escolhidos como pivô, resultam numa divisão de 25-75% ou melhor.

Revisão(zinha) de Probabilidade

Para completar a análise do QuickSort Aleatorizado e outros algoritmos, precisamos relembrar alguns conceitos de probabilidade:

- Espaço Amostral
- Eventos
- Variáveis Aleatórias
- Esperança

Espaço Amostral

- **Espaço Amostral** Ω = todos os possíveis resultados de um evento aleatório.
- cada resultado $i \in \Omega$ tem probabilidade $p(i) \geq 0$.
- **Restrição**: $\sum_{i \in \Omega} p(i) = 1$, ou seja, com certeza um dos resultados de Ω vai acontecer.

Exemplo 1

Rolar dois dados de 6 lados.
$\Omega = \{(1,1),(2,1),(3,1),\ldots,(5,6),(6,6)\}$ (pares ordenados)
$p(i) = \frac{1}{36}$ para todos $i \in \Omega$

Exemplo 2

Escolher o índice do pivô aleatório da primeira iteração do QuickSort.
$\Omega = \{1,2,\ldots,n\}$
$p(i) = \frac{1}{n}$ para todo $i \in \Omega$

Evento

- Um **evento** é um subconjunto $S \subseteq \Omega$.
- A probabilidade de um evento S é

$$\sum_{i \in S} p(i).$$

- Considere o evento "a soma dos dados é 7". Qual é a probabilidade desse evento?
  a. $1/36$
  b. $1/12$
  c. $1/6$
  d. $1/2$
- $S = \{(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)\}$
- $Pr[S] = 6/36 = 1/6$

- Considere o evento "o pivô aleatório escolhido induz uma divisão de pelo menos $25-75\%$ ou melhor". Qual é a probabilidade desse evento?
  a. $1/n$
  b. $1/4$
  c. $1/2$
  d. $3/4$
- S = qualquer escolha que não seja os $1/4$ menores ou os $1/4$ maiores.

$$Pr[S] = \frac{\frac{n}{2}}{n} = \frac{n}{2} \cdot \frac{1}{n} = \frac{1}{2}$$

Variáveis Aleatórias

- Uma **Variável Aleatória** X é uma função:

$$X : \Omega \to \mathbb{R}$$

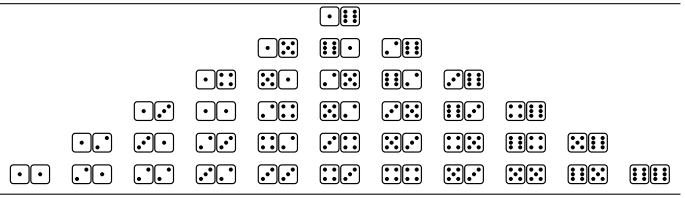
- Exemplo 1: A soma dos dois dados, ex: $X((2,5)) = 7$
- Exemplo 2: O valor do maior dado, ex: $Y((2,5)) = 5$
- Exemplo 3: Tamanho do subproblema passado na primeira chamada recursiva do QuickSort

Esperança

- Seja $X : \Omega \to \mathbb{R}$ uma Variável Aleatória.
- A **Esperança** $E[X]$ de X é o valor médio de X ponderado pela probabilidade.

$$E[X] = \sum_{i \in \Omega} X(i) \cdot p(i)$$

- Qual a esperança da soma de dois dados?
  a. 6.5
  b. 7
  c. 7.5
  d. 8

x	2	3	4	5	6	7	8	9	10	11	12
S'											
$\lvert S' \rvert$	1	2	3	4	5	6	5	4	3	2	1

$$E[X] = \frac{1}{36} \cdot 2 + \frac{2}{36} \cdot 3 + \frac{3}{36} \cdot 4 + \frac{4}{36} \cdot 5 + \frac{5}{36} \cdot 6 + \frac{6}{36} \cdot 7 + \frac{5}{36} \cdot 8 + \frac{4}{36} \cdot 9 + \frac{3}{36} \cdot 10 + \frac{2}{36} \cdot 11 + \frac{1}{36} \cdot 12$$

$$= \frac{2 + 6 + 12 + 20 + 30 + 42 + 40 + 36 + 30 + 22 + 12}{36}$$

$$= \frac{252}{36} = 7$$

- Qual a esperança do tamanho do subproblema passado na primeira chamada recursiva do QuickSort?
  a. $n/4$
  b. $n/3$
  c. $(n-1)/2$
  d. $3n/4$
- Seja X o tamanho do subproblema.

$$\begin{aligned} E[X] &= \frac{1}{n} \cdot 0 + \frac{1}{n} \cdot 1 + \frac{1}{n} \cdot 2 + \ldots + \frac{1}{n} \cdot (n-1) \\ &= \frac{1 + 2 + \ldots + (n-1)}{n} \\ &= \frac{(n-1)(1+n-1)/2}{n} \\ &= \frac{n^2 - n/2}{n} = \frac{n^2 - n}{2} \cdot \frac{1}{n} \\ &= \frac{n-1}{2} \end{aligned}$$

Linearidade da Esperança

Lema

Sejam $X_1, \ldots, X_n$ variáveis aleatórias definidas em Ω. Então:

$$E\left[\sum_{j=1}^{n} X_j\right] = \sum_{j=1}^{n} E\left[X_j\right]$$

- Vale mesmo quando as variáveis não são independentes!
- Exemplo: X_1 e X_2 são variáveis aleatórias que dizem o valor do primeiro e do segundo dado.

$$E[X_j] = \frac{1}{6}(1+2+3+4+5+6) = 3.5$$

- Qual o valor esperado para a soma de dois dados?

$$E[X] = E[X_1 + X_2] = E[X_1] + E[X_2] = 3.5 + 3.5 = 7.$$