Quicksort

Manassés Ribeiro
manasses.ribeiro@ifc.edu.br

Agenda

- Conceitos preliminares
- Passos do algoritmo
- Exemplo de ordenação
- Método de partição de Lomuto
- Desempenho
- Comparação com outros algoritmos
- Outras propostas do quicksort
- Referências

Conceitos preliminares

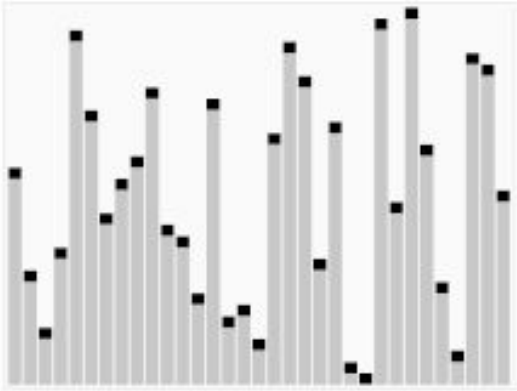
- Método de ordenação rápido e eficiente*
- Inventado por Charles Antony Richard Hoare em 1960 e publicado oficialmente em 1962

* apesar de não ser consenso na literatura

Conceitos preliminares

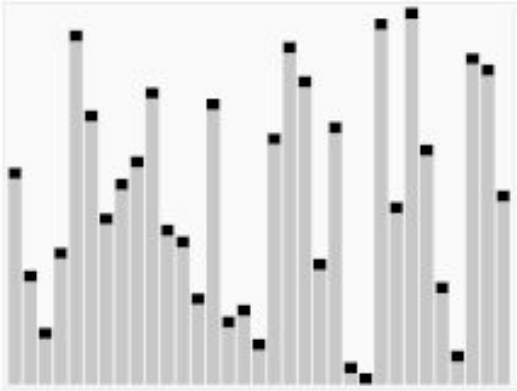
A ideia principal do método utiliza a estratégia de **divisão e conquista**:

- Consiste em rearranjar os itens de um vetor de modo que **os menores precedem os maiores;**
- Na sequência, **duas sublistas são geradas e ordenadas recursivamente**.



Passos do algoritmo

1. Escolha
2. Partição
3. Ordenação (recursiva)



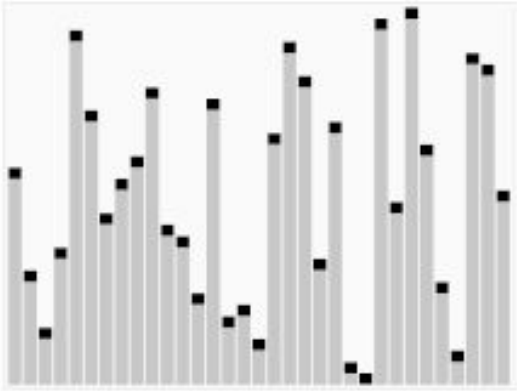
➔ Devido ao processo de recursão, são as listas de tamanho zero ou um, que estão sempre ordenadas;

➔ O processo é finito

- ◆ a cada iteração pelo menos um elemento é posto em sua posição final;
- ◆ e não será mais manipulado na iteração seguinte.

Passos do algoritmo

1. Escolha
2. Partição
3. Ordenação

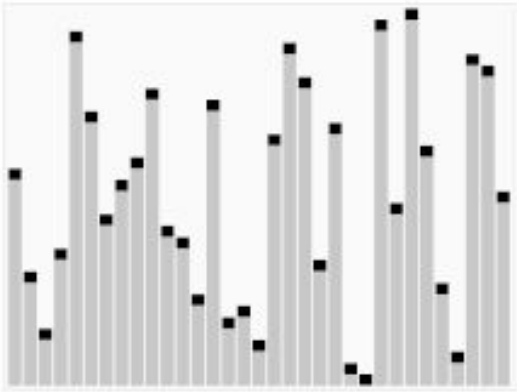


➔ Tanto a escolha do pivô quanto os passos da **Partição** podem ser realizados de diferentes formas:

- ◆ a escolha de uma implementação específica afeta fortemente a performance do algoritmo.

Passos do algoritmo

1. Escolha
   a. etapa da escolha do elemento pivô;
   b. existem diversas abordagens de escolhas;
   c. exemplo de escolha pelo meio da lista:

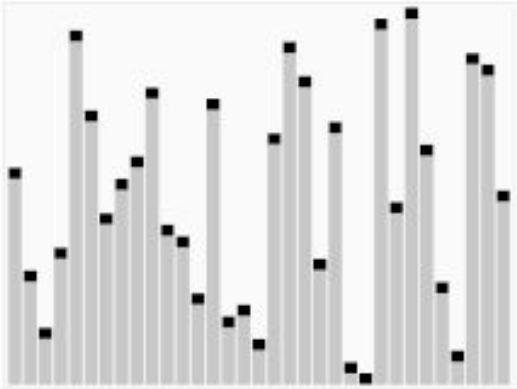


```
pivo <- X[(IniVet + FimVet) div 2]
```

Retorna

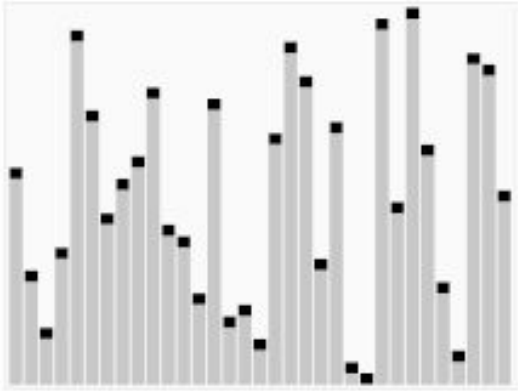
Passos do algoritmo

2. Partição de **Hoare**
    a. Etapa de rearranjo da lista com os elementos menores à esquerda do pivô e os elementos maiores à direita;
        i. ao final, o pivô estará no lugar e as duas sublistas (da esquerda e da direita) ainda estarão desordenadas



Passos do algoritmo (Hoare)

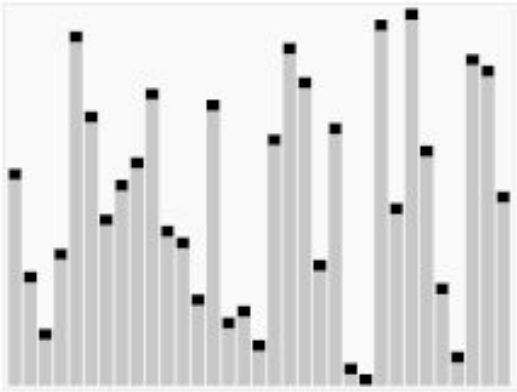
```
i <- IniVet
j <- FimVet
enquanto(i <= j)
  |       enquanto (X[i] < pivo) faça
  |        |    i <- i + 1
  |       fimEnquanto
  |       enquanto (X[j] > pivo) faça
  |        |    j <- j - 1
  |       fimEnquanto
  |       se (i <= j) então
  |        |    troca(X[i], X[j])
  |        |    i <- i + 1
  |        |    j <- j - 1
  |       fimSe
fimEnquanto
```



Retorna

Passos do algoritmo

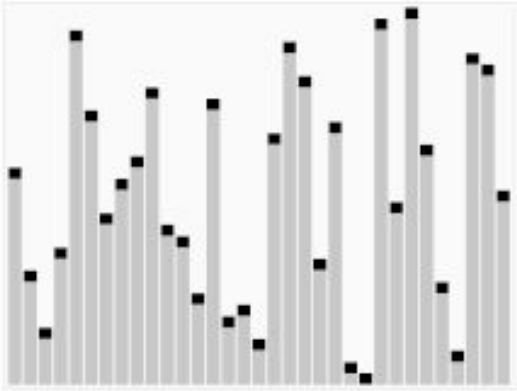
3. Ordenação
    a. etapa onde as sublistas dos elementos menores e maiores são ordenadas **recursivamente.**



Passos do algoritmo

```
    se (IniVet < j) então
     |  QuickSort(X, IniVet, j)
    fimSe

    se (i < FimVet) então
     |  QuickSort(X, i, FimVet)
    fimSe
```



Retorna

Passos do algoritmo de Hoare (Completo)

```
procedimento QuickSort(X[], IniVet, FimVet)
var
   i, j, pivo
início
   i <- IniVet
   j <- FimVet
   pivo <- X[(IniVet + FimVet) div 2]
   enquanto(i <= j)
    |      enquanto (X[i] < pivo) faça
    |       |   i <- i + 1
    |      fimEnquanto
    |      enquanto (X[j] > pivo) faça
    |       |   j <- j - 1
    |      fimEnquanto
    |      se (i <= j) então
    |       |   troca(X[i], X[j])
    |       |   i <- i + 1
    |       |   j <- j - 1
    |      fimSe
   fimEnquanto
   se (IniVet < j) então
    |  QuickSort(X, IniVet, j)
   fimSe
   se (i < FimVet) então
    |  QuickSort(X, i, FimVet)
   fimSe
fimProcedimento
```

Exemplo de ordenação

A = {3, 4, 9, 1, 7, 0, 5, 2, 6, 8}

Método de partição de Lomuto

- Método atribuído a Nico **Lomuto**
  - popularizado por Bentley no livro *Programming Pearls* e por Cormen et al. no livro *Introduction to Algorithms*;
- Escolhe um pivô tipicamente no início ou no final do array;
- **É a maneira mais simples e fácil de entender o algoritmo, entretanto é menos eficiente que o método Hoare**;
- Este Método decai para **O(n2)** quando o array já está ordenado ou quando só possui elementos iguais.

Passos do algoritmo de Lomuto (Completo)

```
procedimento QuickSort(X[], IniVet, FimVet)
var
   pi
início
   se (IniVet < FimVet) então
    |      pi = Particao(X[], IniVet, FimVet)
    |      QuickSort(X, IniVet, pi - 1)
    |      QuickSort(X, pi + 1, FimVet)
   fimSe
fimProcedimento
```

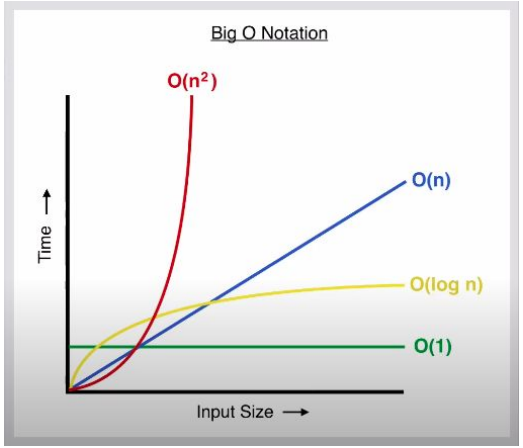
```
função Particao(X[], IniVet, FimVet)
var
   i, j, pivo
início
   i <- IniVet - 1
   pivo <- X[FimVet]
   para j de IniVet até FimVet-1 passo 1) faça
    |       se (X[j] <= pivo) então
    |       |       i <- i + 1
    |       |       troca(X[i], X[j])
    |       fimSe
   fimPara
   troca(X[i + 1], X[FimVet])
   retorna (i + 1)
fimFunção
```

Desempenho de tempo

- **Comportamento no pior caso**
  - ocorre quando o elemento pivô divide a lista de forma desbalanceada
    - uma com tamanho 0 e outra com tamanho n - 1, onde *n* é o tamanho da lista original;

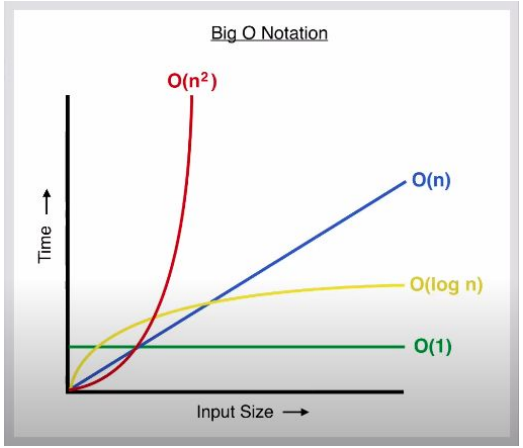
Desempenho de tempo

- isso pode ocorrer quando o elemento pivô é o maior ou menor elemento da lista
  - ou seja, quando a lista já está ordenada, ou está inversamente ordenada;
- neste caso o algoritmo terá tempo de execução igual à **$O(n^2)$**.



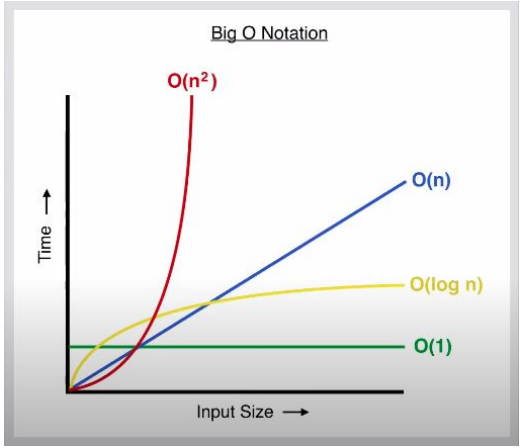
Desempenho de tempo

- Comportamento no melhor caso
  - O melhor caso de particionamento acontece quando é produzido duas listas de tamanho não maior que n/2
    - uma lista terá tamanho [n/2] e outra tamanho [n/2] - 1;
    - nesse caso, o desempenho de tempo é **O(n log(n))**.



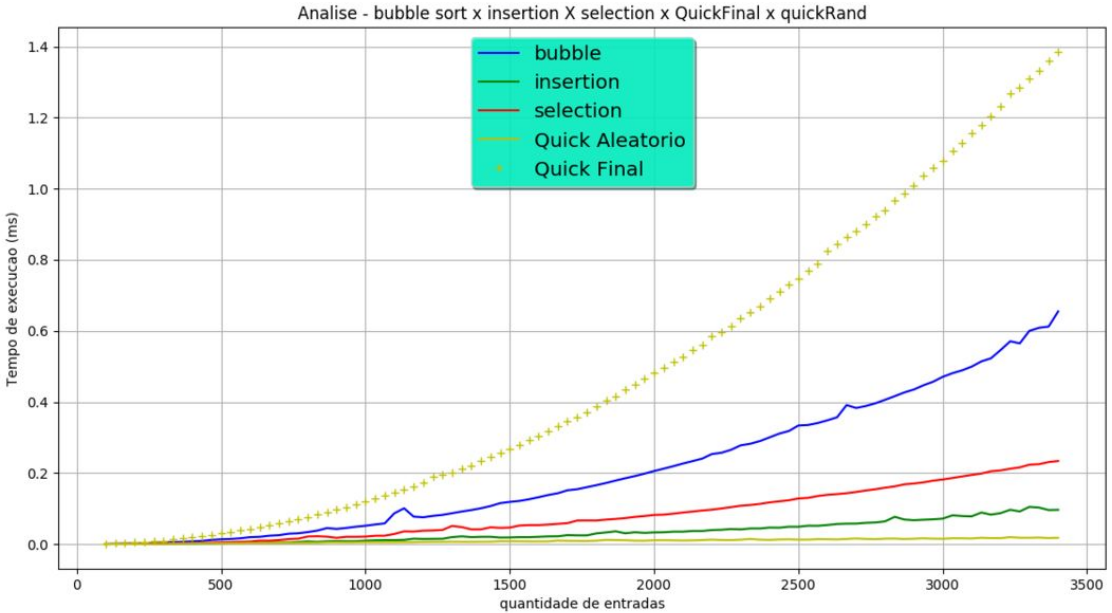
Desempenho de espaço

- Em relação ao desempenho de espaço
  - **O($\log_2$n)** no melhor caso e no caso médio;
  - **O(n)** no pior caso;
  - **Atenção**: dependendo da implementação da recursão, o desempenho pode ser **O(n2).**



Comparação com outros algoritmos

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)



Comparação com outros algoritmos

- O **quicksort** é uma versão optimizada de uma **árvore binária** ordenada (conteúdo de ED II)
  - algoritmo bastante dependente da escolha correta do pivô.
- **Heapsort**: o pior caso neste algoritmo é **$O(n \log_2 n)$**;
  - Em média trata-se de um algoritmo mais lento que o quicksort, embora haja consenso.

Comparação com outros algoritmos

- **Mergesort**: outro algoritmo de ordenação recursiva;
  - no pior caso **O(n log n);**
  - é estável e pode facilmente ser adaptado para operar em listas encadeadas e em listas grandes armazenadas em dispositivos de acesso lento (discos);
  - a maior desvantagem é que quando opera em arrays, requer **O(n)** de espaço para o melhor caso;
    - o quicksort com um particionamento espacial e com recursão utiliza apenas **O (log n)** de espaço.

Comparação com outros algoritmos

- **Bucket sort:**
  - com dois buckets é muito parecido ao quicksort;
  - o pivô neste caso é o valor do meio do vetor.

Outras propostas de Quicksort

- Quicksort utilizando dois pivôs:
  - *Dual-Pivot Quicksort*: proposto por Yaroslavskiy (2009):
    - são utilizados 2 pivôs, particionando um array de entrada em 3 partes;
    - Yaroslavskiy demonstra que o uso de dois pivôs é mais eficaz, especialmente quando possui uma quantidade maior de dados de entrada.

Outras propostas de Quicksort

- Múltiplos pivôs:
  - Budiman, Zamzami e Rachmawati (2017) pontuam que o quicksort com múltiplos pivôs é mais eficiente:
    - analisando o uso de até 5 pivôs foi verificado que quanto mais pivôs são utilizados em um algoritmo quicksort, mais rápido seu desempenho se torna;
    - contudo, o desempenho resultante da adição de mais pivôs tende a diminuir gradualmente.

Referências

- **Hoare's vs Lomuto partition scheme in QuickSort**. https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/. Acesso em 2022.
- **Quicksort algorithm using Hoare's partitioning scheme**. https://www.techiedelight.com/quick-sort-using-hoares-partitioning-scheme/. Acesso em 2022.
- **An Overview of QuickSort Algorithm**. https://www.baeldung.com/cs/algorithm-quicksort. Acesso em 2022.
- **Know Thy Complexities!** https://www.bigocheatsheet.com/. Acesso em 2022.
- YAROSLAVSKIY, V. **Dual-pivot quicksort. Research Disclosure**, 2009.
- BUDIMAN, M.; ZAMZAMI, E.; RACHMAWATI, D. **Multi-pivot quicksort: an experiment with single, dual, triple, quad, and penta-pivot quicksort algorithms in python**. In: IOP PUBLISHING.IOP Conference Series: Materials Science and Engineering. 2017.

Outros materiais

- **Hoare on inventing Quicksort** (vídeo)
- **Quicksort** (Wikipedia)
- Quick Sort - Data Structures & Algorithms Tutorial Python #15 (vídeo)
- Iniciando com a notação Big O
- O que é Big O Notation?
- Simulador Quicksort

Resumo da aula

- O que foi visto nesta aula:
  - Conceitos preliminares
  - Passos do algoritmo
  - Exemplo de ordenação
  - Método de partição de Lomuto
  - Complexidade
  - Comparação com outros algoritmos
  - Outras propostas do Quicksort

Quicksort

Manassés Ribeiro
manasses.ribeiro@ifc.edu.br