

Estrutura de Dados

Estrutura de Dados
Algoritmos de Ordenação

Sumário

- 1 Definição
- 2 Bubble Sort
- 3 Insertion Sort
- 4 Selection Sort

- 5 Divisão e Consquista
- 6 Merge Sort
- 7 Quick Sort
- 8 Heap Sort
- 9 Complexidade

Definição

Algoritmo de Ordenação

Um algoritmo de ordenação é um conjunto de instruções projetadas para reorganizar os elementos de uma coleção (como um vetor ou lista) em uma sequência específica, geralmente em ordem crescente ou decrescente.

Objetivo

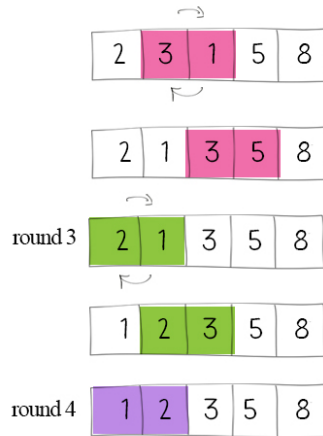
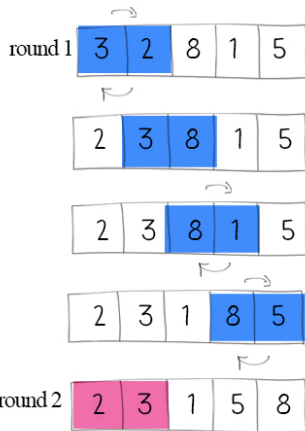
Facilitar a busca, comparação e análise dos dados, reduzindo a complexidade de operações posteriores como a busca binária.

Bubble Sort

Definição

O Bubble Sort compara pares de elementos adjacentes e os troca se estiverem fora de ordem, repetindo esse processo até que toda a coleção esteja ordenada.

Bubble Sort II



```
1 void bubble_sort(int *vetor, int n) {  
2     for (int i = 0; i < n - 1; i++) {  
3         for (int j = 0; j < n - 1 - i; j++) {  
4             if (vetor[j] > vetor[j + 1]) {  
5                 swap(&vetor[j], &vetor[j + 1]);  
6             }  
7         }  
8     }  
9 }
```

Código 1: Implementação do algoritmo Bubble Sort.

Insertion Sort

Definição

O Insertion Sort constrói a ordenação final um elemento por vez, inserindo cada novo elemento na posição correta em relação aos anteriores.

Insertion Sort II

input

3	2	8	1	5
---	---	---	---	---

round 1

3	2	8	1	5
---	---	---	---	---

round 2

2	3	8	1	5
---	---	---	---	---

round 3

2	3	8	1	5
---	---	---	---	---

round 4

1	2	3	8	5
---	---	---	---	---

result

1	2	3	5	8
---	---	---	---	---

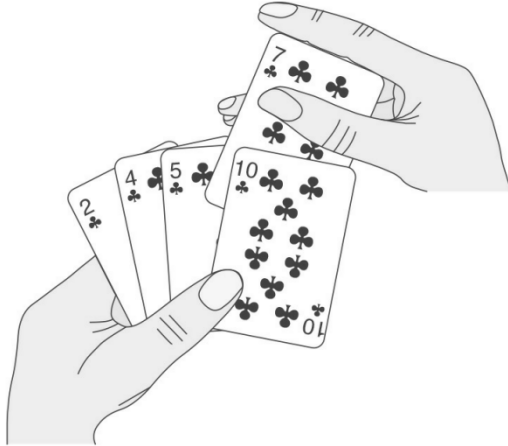
```
1 void insertion_sort(int *vetor, int n) {  
2     for (int i = 1; i < n; i++) {  
3         int chave = v[i];  
4         int j = i - 1;  
5         while (j >= 0 && v[j] > chave) {  
6             vetor[j + 1] = v[j];  
7             j--;  
8         }  
9         vetor[j + 1] = chave;  
10    }  
11 }
```

Código 2: Implementação do algoritmo Insertion Sort.

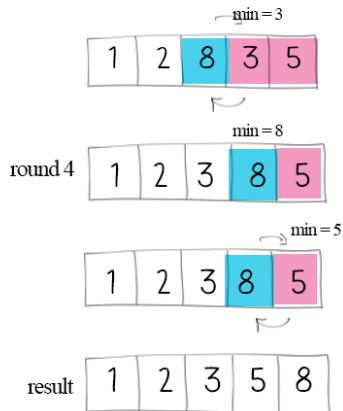
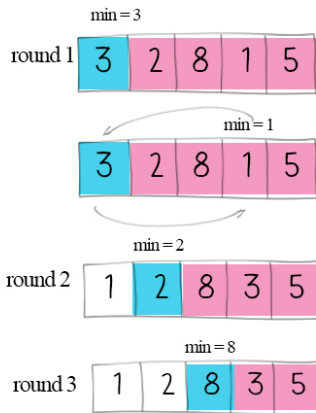
Selection Sort

Definição

O Selection Sort percorre repetidamente o vetor, selecionando o menor (ou maior) elemento do subvetor não ordenado e trocando com o elemento da posição atual, construindo a ordenação final.



Selection Sort III



```
1 void selection_sort(int *vetor, int n) {  
2     for (int i = 0; i < n - 1; i++) {  
3         int min_idx = i;  
4         for (int j = i + 1; j < n; j++) {  
5             if (vetor[j] < vetor[min_idx]) {  
6                 min_idx = j;  
7             }  
8         }  
9         if (min_idx != i) {  
10            swap(&vetor[i], &vetor[min_idx]);  
11        }  
12    }  
13 }
```

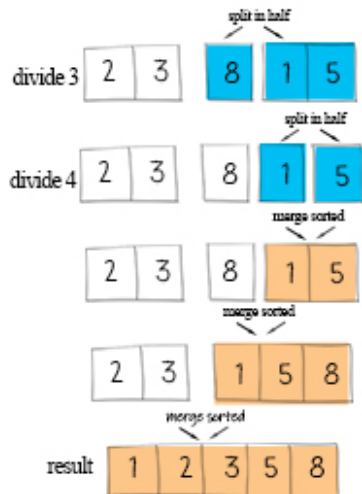
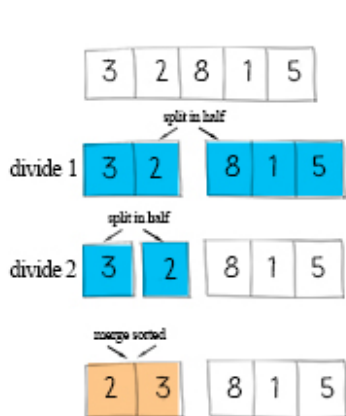
Código 3: Implementação do algoritmo Selection Sort.

Divisão e Consquista

Merge Sort

Definição

O Merge Sort é um algoritmo de ordenação recursivo baseado no paradigma *Dividir para Conquistar*, que divide o vetor em partes menores, ordena e depois combina os resultados.



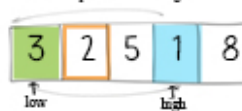
```
1 void merge_sort(int *vetor, int inicio, int fim) {  
2     if (inicio < fim) {  
3         int meio = (inicio + fim) / 2;  
4         merge_sort(vetor, inicio, meio);  
5         merge_sort(vetor, meio + 1, fim);  
6         merge(vetor, inicio, meio, fim);  
7     }  
8 }
```

Código 4: Implementação do algoritmo Merge Sort.

Quick Sort

Definição

O Quick Sort é um algoritmo recursivo que seleciona um pivô e reorganiza os elementos de modo que os menores fiquem à esquerda e os maiores à direita, repetindo o processo recursivamente.



```
1 void quick_sort(int *vetor, int inicio, int fim) {  
2     if (inicio < fim) {  
3         int pivo = particionar(vetor, inicio, fim);  
4         quick_sort(vetor, inicio, pivo - 1);  
5         quick_sort(vetor, pivo + 1, fim);  
6     }  
7 }
```

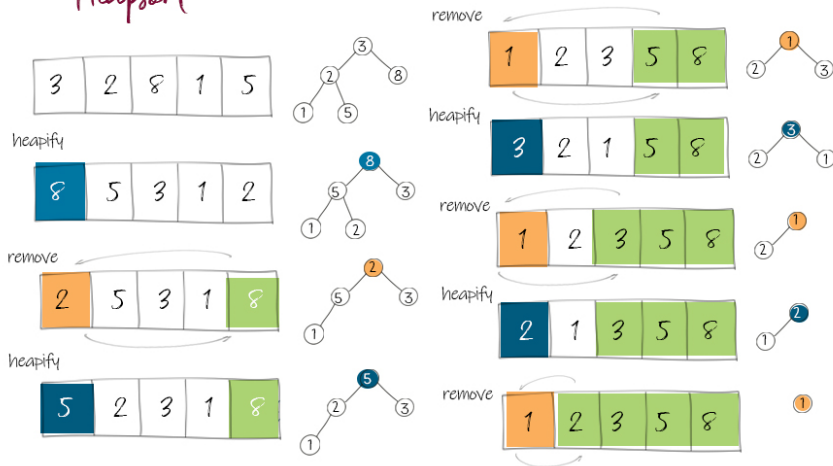
Código 5: Implementação do algoritmo Quick Sort.

Heap Sort

Definição

O Heap Sort utiliza uma estrutura de dados chamada *heap* para ordenar os elementos, construindo um heap máximo e extraíndo repetidamente o maior elemento.

Heapsort



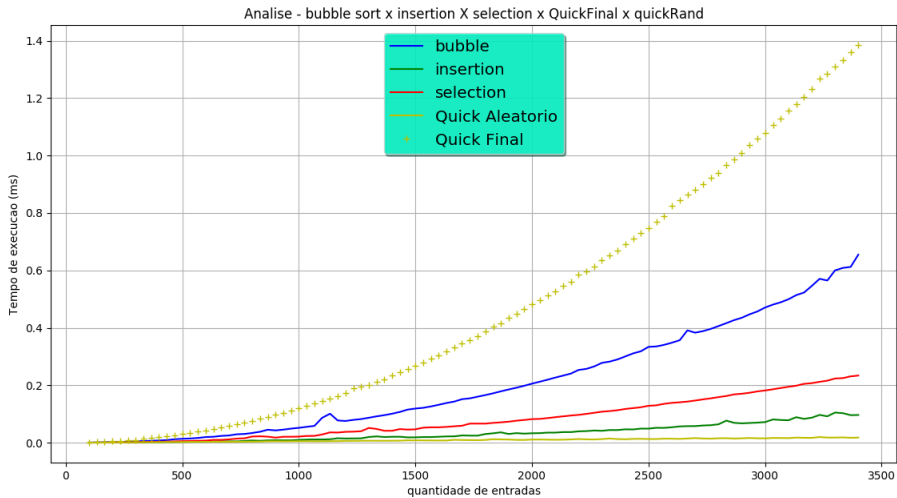
```
1 void heapify(int *vetor, int n, int i) {  
2     int maior = i;  
3     int esquerda = 2*i + 1;  
4     int direita = 2*i + 2;  
5     if (esquerda < n && vetor[esquerda] > vetor[maior])  
6         maior = esquerda;  
7     if (direita < n && vetor[direita] > vetor[maior])  
8         maior = direita;  
9     if (maior != i) {  
10         swap(&vetor[i], &vetor[maior]);  
11         heapify(vetor, n, maior);  
12     }  
13 }
```

Código 6: Implementação do algoritmo Heap Sort.

Complexidade

Complexidade dos Algoritmos de Ordenação I

Complexidade dos Algoritmos de Ordenação II

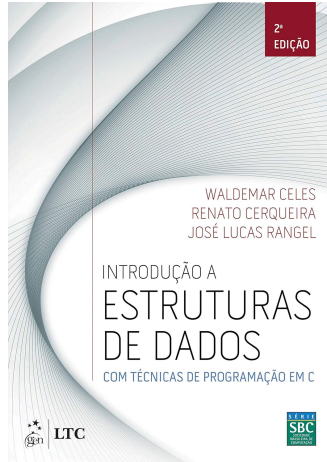


Complexidade dos Algoritmos de Ordenação III

Algoritmo	Melhor Caso	Pior Caso	Complexidade Média
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Tabela 1: Complexidade dos algoritmos de ordenação

(CELES; CERQUEIRA; RANGEL, 2004) - Capítulo 11



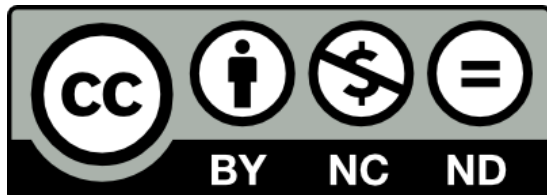
CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. **Introdução a estruturas de dados: com técnicas de programação em C**. Rio de Janeiro: Elsevier, 2004.

DEITEL, Paul; DEITEL, Harvey. **C: Como Programar**. 6. ed. São Paulo: Pearson Universidades, 2011.

PUGA, Sandra Gavioli; RISSETTI, Gerson. **Lógica de programação e estrutura de dados: com aplicações em Java**. 2. ed. São Paulo, SP: Pearson, 2009. E-book. Disponível em: <https://plataforma.bvirtual.com.br>. Acesso em: 24 jul. 2025.

SCHILDT, Herbert. **C Completo e Total: O Guia Definitivo para Programação em C**. 3. ed. São Paulo: Makron Books, 1996. ISBN 978-8534606928.

Estes slides estão protegidos por uma licença Creative Commons



Este modelo foi adaptado de Maxime Chupin.

Estrutura de Dados

Estrutura de Dados
Algoritmos de Ordenação