

## Algoritmos de Ordenação

Algoritmos de ordenação são usados na ciência da computação para colocar elementos de uma sequência em uma ordem específica, facilitando a recuperação de dados em uma lista. Eles são usados em diversas aplicações, como bancos de dados, sistemas de busca e processamento de imagem. Existem diferentes algoritmos de ordenação, cada um com suas vantagens e desvantagens, e a escolha do algoritmo depende do tamanho e distribuição dos dados e do tempo disponível para a ordenação. Alguns dos algoritmos mais comuns são:

1. Bubble Sort;
2. Insertion Sort;
3. Selection Sort;
4. Quick Sort;
5. Merge Sort;

### Bubble Sort

O nome "bubble" vem da ideia de que os maiores elementos "flutuam" para o final do vetor, assim como bolhas de ar em um fluido. Trata-se de um algoritmo simples que percorre um vetor comparando cada elemento da posição  $i$  com o elemento da posição  $i+1$ , trocando-os de posição caso estejam na ordem errada. Esse processo é repetido até que não haja mais trocas a serem feitas. A complexidade do algoritmo Bubble Sort é  $O(n^2)$  no pior caso e caso médio, pois ele percorre o array várias vezes, comparando elementos adjacentes e trocando-os de posição se estiverem na ordem errada.

Exemplo: Suponha que temos o vetor [9, 7, 2, 1, 3]. O Bubble Sort iria comparar o 9 com o 7 e trocá-los de posição, resultando em [7, 9, 2, 1, 3]. Em seguida, ele compararia o 9 com o 2 e trocaria de posição, resultando em [7, 2, 9, 1, 3]. Depois disso, compararia o 9 com o 1 e trocaria de posição, resultando em [7, 2, 1, 9, 3]. Por fim, compararia o 9 com o 3 e trocaria de posição, resultando em [7, 2, 1, 3, 9]. Em seguida, ele começaria novamente a partir do primeiro elemento e repetiria esse processo até que não haja mais trocas a serem feitas.

$n = 5$  Verificação:  $i < n - i - 1$ ;  $A[i] > A[i+1]$

$i$	swapped	$j$	$A[j]$	$A[j+1]$	$A = \{9, 7, 2, 1, 3\}$
0	false; true	0; 1; 2; 3; 4	9; 9; 9; 9	7; 2; 1; 3	$\{7, 9, 2, 1, 3\}$ ; $\{7, 2, 9, 1, 3\}$ ; $\{7, 2, 1, 9, 3\}$ ; $\{7, 2, 1, 3, 9\}$
1	false; true	0; 1; 2; 3	7; 7; 7	2; 1; 3	$\{2, 7, 1, 3, 9\}$ ; $\{2, 1, 7, 3, 9\}$ ; $\{2, 1, 3, 7, 9\}$
2	false; true	0; 1; 2	2; 2	1; 3	$\{1, 2, 3, 7, 9\}$
3	false	0; 1	1	2	

**Figura 1:** Teste de mesa do algoritmo Bubble Sort.

## Insertion Sort

Insertion Sort é um algoritmo que percorre um vetor da esquerda para a direita e insere cada elemento em sua posição correta em uma sublista ordenada. Ele começa considerando o primeiro elemento como uma sublista ordenada e vai inserindo os elementos restantes na posição correta dessa sublista. A complexidade do algoritmo Insertion Sort é  $O(n^2)$  no pior caso e caso médio, pois ele percorre o array várias vezes, inserindo cada elemento em sua posição correta em relação aos elementos anteriores.

Exemplo: O Insertion Sort define o elemento chave a ser comparado e considera os elementos à esquerda dele como uma sublista ordenada; então, ele compara o elemento chave com os elementos da sublista, de modo a posicioná-lo na posição correta. Suponha que temos o vetor [9, 7, 2, 1, 3], o Insertion Sort começaria considerando o segundo elemento 7 como o elemento a ser comparado e o primeiro elemento 9 como uma sublista ordenada. Em seguida, compararia o elemento chave com os elementos da sublista, identificando a posição que deve ser posicionado; resultando no vetor [7, 9, 2, 1, 3]. Em seguida, ele pegaria o terceiro elemento 2, compararia com a sublista [7, 9] e o inseriria na posição correta da sublista ordenada, resultando em [2, 7, 9, 1, 3]. Na próxima iteração, ele pegaria o quarto elemento 1, compararia com a sublista [2, 7, 9] e o inseriria na posição correta, resultando em [1, 2, 7, 9, 3]. Por fim, ele pegaria o quinto elemento 3 e o inseriria na posição correta da sublista ordenada, resultando em [1, 2, 3, 7, 9].

$n = 5$  Verificação:  $i < n$ ;  $j > 0$  e  $A[j] > \text{key}$

$i$	key	$j$	$A[j]$	$A = \{9, 7, 2, 1, 3\}$
1	7	0; -1	9	$\{9, 9, 2, 1, 3\}; \{7, 9, 2, 1, 3\}$
2	2	1; 0; -1	9; 7	$\{7, 9, 9, 1, 3\}; \{7, 7, 9, 1, 3\};$ $\{2, 7, 9, 1, 3\}$
3	1	2; 1; 0; -1	9; 7; 2	$\{2, 7, 9, 9, 3\}; \{2, 7, 7, 9, 3\};$ $\{2, 2, 7, 9, 3\}; \{1, 2, 7, 9, 3\}$
4	3	3; 2; 1	9; 7; 2	$\{1, 2, 7, 9, 9\}; \{1, 2, 7, 7, 9\};$ $\{1, 2, 3, 7, 9\}$

**Figura 2:** Teste de mesa do algoritmo Insertion Sort.

## Selection Sort

Selection Sort é um algoritmo simples que percorre um vetor procurando pelo menor elemento e o troca de posição com o elemento que está na primeira posição. Em seguida, ele procura pelo menor elemento da sublista à direita do valor que foi ordenado na iteração anterior e o troca de posição com

o primeiro elemento dessa sublista. Seguindo dessa forma até que todo o vetor esteja ordenado. A complexidade do algoritmo Selection Sort é  $O(n^2)$  no pior caso e caso médio, pois ele percorre o array várias vezes, selecionando o menor elemento em cada iteração e trocando-o com o primeiro elemento não ordenado.

Exemplo: Suponha que temos o vetor [9, 7, 2, 1, 3]. O Selection Sort começaria procurando pelo menor elemento e encontraria o elemento 1. Ele o colocaria na primeira posição do vetor, resultando em [1, 7, 2, 9, 3]. Em seguida, ele procuraria pelo menor elemento na sublista [7, 2, 9, 3] e encontraria o valor 2. Ele o colocaria na primeira posição da sublista, resultando em [1, 2, 7, 9, 3]. Depois disso, ele procuraria pelo menor elemento na sublista [7, 9, 3] e o posicionaria na primeira posição, resultando em [1, 2, 3, 9, 7]. Por fim, ele analisaria a sublista [9, 7], identificaria o elemento 7 e o trocava de lugar com o elemento da primeira posição. Resultando na lista ordenada [1, 2, 3, 7, 9].

$n = 5$     Verificação:  $i < n-1; j < n; A[j] < A[\min]$

$i$	$\min$	$j$	$A[j]$	$A[\min]$	$A = \{9, 7, 2, 1, 3\}$
0	0; 1; 2; 3	1; 2; 3; 4	7; 2; 1; 3	9; 7; 2; 1	$\{1, 7, 2, 9, 3\}$
1	1; 2	2; 3; 4	2; 9; 3	7; 2	$\{1, 2, 7, 9, 3\}$
2	2; 4	3; 4	9; 3	7	$\{1, 2, 3, 9, 7\}$
3	3; 4	4	7	9	$\{1, 2, 3, 7, 9\}$

**Figura 3:** Teste de mesa do algoritmo Insertion Sort.

### Quick Sort

Quick Sort é um algoritmo eficiente que divide o vetor em duas partes menores, uma com elementos menores que um pivô escolhido e outra com elementos maiores. Em seguida, ele ordena recursivamente essas duas partes até que todo o vetor esteja ordenado. A complexidade do algoritmo Quick Sort é  $O(n \log n)$  no caso médio e  $O(n^2)$  no pior caso. No caso médio, o algoritmo divide o array em duas sublistas de tamanho aproximadamente igual em cada chamada recursiva, resultando em um tempo de execução de  $O(n \log n)$ . No pior caso, o pivô escolhido pode ser o menor ou maior elemento em cada chamada recursiva, resultando em uma divisão desbalanceada do array e um tempo de execução de  $O(n^2)$ .

Exemplo: Suponha que temos o vetor [9, 7, 2, 1, 3]. O Quick Sort escolheria um pivô, por exemplo, o elemento do meio (2), e dividiria o vetor em duas partes: [9, 7, 2] e [1, 3]. Em seguida, ele ordenaria recursivamente essas duas partes. Para a primeira parte, ele escolheria outro pivô (7) e dividiria em [9] e [2]. Em seguida, ordenaria recursivamente [9] e [2]. O vetor [9] já está ordenado e [2] é um único elemento, então não há mais divisão. Para a segunda parte, ele escolheria o pivô (3) e dividiria em [1]

e []. O vetor [] é vazio, então não há mais divisão. Por fim, ele combinaria as partes ordenadas na ordem correta: [1, 2, 3, 7, 9].

$n = 5$       Verificações:  $low < high$ ;  $A[i] < pivot$

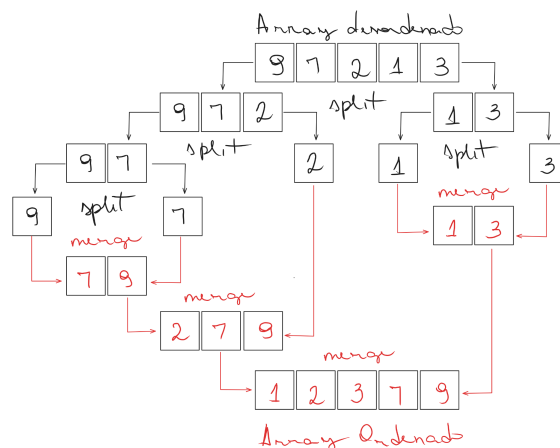
low	high	pivot	i	j	A[j]	A = {9, 7, 2, 1, 3}	return
0	4	3	-1; 0; 1	0; 1; 2; 3	9; 7; 2; 1	{2, 7, 9, 1, 3}, {2, 1, 9, 7, 3}, {2, 1, 3, 7, 9}	2
0	1	1	-1	0	2	{1, 2, 3, 7, 9}	0
3	4	9	2; 3	3	7	{1, 2, 3, 9, 7}, {1, 2, 3, 7, 9}	4

**Figura 4:** Teste de mesa do algoritmo Quick Sort.

### Merge Sort

Merge Sort é um algoritmo de ordenação que utiliza a estratégia de dividir para conquistar (dividir o problema em subproblemas menores, resolvê-los e combinar as soluções para resolver o problema original). Ele divide o array recursivamente em duas metades iguais, ordena cada metade e depois combina as metades ordenadas em um único array ordenado. A combinação das metades ordenadas é feita comparando os elementos das duas metades e colocando-os em ordem crescente em um novo array. Como a combinação das metades ordenadas leva tempo proporcional a  $n$ , a complexidade no pior caso e caso médio do algoritmo é  $O(n \log n)$ .

Exemplo: Suponha que temos o vetor [9, 7, 2, 1, 3]. O Merge Sort dividiria o vetor em duas partes iguais: [9, 7, 2] e [1, 3]. Em seguida, ele ordenaria recursivamente essas duas partes. Para a primeira parte, ele dividiria em [9] e [7, 2]. Em seguida, ordenaria recursivamente [9] e [7, 2]. O vetor [9] já está ordenado e [7, 2] seria dividido em [7] e [2]. Em seguida, ordenaria recursivamente [7] e [2]. O vetor [7] já está ordenado e [2] é um único elemento, então não há mais divisão. Depois disso, ele combinaria as partes ordenadas na ordem correta: [2, 7, 9]. Para a segunda parte, ele dividiria em [1] e [3]. Ambos os vetores já estão ordenados, então não há mais divisão. Por fim, ele combinaria as partes ordenadas na ordem correta: [1, 3]. Então ele combinaria as partes ordenadas na ordem correta novamente: [1, 2, 3, 7, 9].



**Figura 5:** Teste de mesa do algoritmo Merge Sort.

## Resultados e Discussão

Para a verificação prática do desempenho de cada um dos algoritmos apresentados, foram realizados testes de execução medindo a velocidade de processamento na ordenação de um arranjo com mil elementos. Os resultados obtidos podem ser verificados na Tabela 1.

**Tabela 1:** Tempo de execução durante a ordenação dos elementos do arquivo CSV original

Simulação	Métodos de Ordenação				
	Bubble Sort	Insertion Sort	Selection Sort	Quick Sort	Merge Sort
1	0,018897	0,008723	0,013020	0,002780	0,003012
2	0,026972	0,010933	0,011587	0,002939	0,002767
3	0,027132	0,011128	0,012613	0,002510	0,002741
4	0,026932	0,008341	0,012736	0,003314	0,003309
5	0,019661	0,011262	0,013207	0,002939	0,002770
Média (s)	0,023919	0,010077	0,012633	0,002896	0,002920

Através desses dados, com os elementos distribuídos sem uma ordem específica, pode-se notar que o *Quick Sort* apresenta o menor tempo de execução, seguido do *Merge Sort*, *Insertion Sort*, *Selection Sort* e, por fim, o *Bubble Sort* tendo o maior tempo de execução.

Em um segundo momento, foram realizados os testes com os elementos ordenados, a priori, no pior caso, os resultados podem ser observados na Tabela 2.

**Tabela 2:** Tempo de execução durante a ordenação dos elementos do arquivo CSV ordenado no pior caso

Simulação	Métodos de Ordenação				
	Bubble Sort	Insertion Sort	Selection Sort	Quick Sort	Merge Sort
1	0,031417	0,014268	0,013341	0,014063	0,002856
2	0,033205	0,016050	0,014578	0,018128	0,002621
3	0,032613	0,015016	0,014205	0,016707	0,003003
4	0,030581	0,014498	0,013591	0,013932	0,003344
5	0,032691	0,014694	0,015676	0,017594	0,002737
Média (s)	0,032101	0,014905	0,014278	0,016085	0,002912

Através dos resultados obtidos, nota-se que o algoritmo *Quick Sort* teve uma perda significativa em seu desempenho, o que corrobora o estudo de complexidade desse algoritmo. Já o algoritmo *Merge Sort* apresentou uma variação mínima no tempo de execução. O algoritmo *Bubble Sort* seguiu apresentando o pior desempenho durante os testes.