Ordenação

Maria Julia Prado Lazaroto

[1]PUCPR

Resumo. *Hoje, exploraremos algoritmos de ordenação, como o bubblesort, mergesort e shellsort, compreendendo seu funcionamento e determinando qual deles se adapta melhor a cada situação. Analisaremos suas características, eficiência e aplicações específicas, aprofundando nosso conhecimento sobre essas técnicas de classificação.*

1. BubbleSort

O bubblesort é um algoritmo de ordenação simples que percorre uma lista de elementos várias vezes, comparando pares adjacentes e trocando-os se estiverem fora de ordem. À medida que as iterações ocorrem, os elementos desordenados gradualmente se movem para suas posições corretas. Embora seja fácil de entender e implementar, o bubblesort é menos eficiente em comparação com outros algoritmos de ordenação, especialmente quando lidamos com conjuntos de dados grandes. Então, podemos concluir que ele é mais apropriado para listas menores.

Bubble Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
0.0	476	2450	50
0.0	622	2450	50
0.0	656	2450	50
0.0	641	2450	50
0.0	574	2450	50
Média			
0.0	593	2450	50

Bubble Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
2	61369	249500	500
2	60442	249500	500
2	61293	249500	500
2	60738	249500	500
2	63554	249500	500
Média			
2	61479	249500	500

Bubble Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
5	250441	999000	1000
5	248491	999000	1000
5	250852	999000	1000
5	238113	999000	1000
5	249962	999000	1000
Média			
5	247571	999000	1000

Bubble Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
27	6111950	24995000	5000
27	6154091	24995000	5000
27	6204304	24995000	5000
27	6236255	24995000	5000
27	6112106	24995000	5000
Média			
27	6163741	24995000	5000

Bubble Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
114	24873580	99990000	10000
114	24798885	99990000	10000
114	24403396	99990000	10000
114	24700050	99990000	10000
114	24692554	99990000	10000
Média			
114	24693693	99990000	5000

2. MergeSort

O processo do MergeSort ocorre em 2 passos importantes, a divisão e intercalação. Na divisão, o vetor é dividido ao meio, e este processo é repetido recursivamente até ficarem os números individuais em cada um. Após a divisão, o mesmo começa a mesclar os itens, para criar sublistas maiores ordenadas, assim, este processo vai acontecendo até tudo ficar ordenado em uma só lista. O Mergesort sempre funciona bem, não importa se a lista está toda bagunçada ou já meio ordenada. Ele também mantém a ordem dos números iguais, então é um jeito ”estável” de ordenar. A única desvantagem é que o Mergesort precisa de um pouco de espaço extra na memória para funcionar, o que pode ser um problema se você estiver ordenando uma lista consideravelmente grande. Mas, no geral, o Mergesort é um algoritmo de ordenação eficiente e estável que é adequado para a maioria das situações, especialmente quando a estabilidade da ordem é importante.

Merge Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
0	222	49	50
0	222	49	50
0	223	49	50
0	228	49	50
0	214	49	50
Média			
0	221	49	50

Merge Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
1	4084	548	500
1	4072	548	500
1	4066	548	500
1	4094	548	500
1	4067	548	500
Média			
1	4076	548	500

Merge Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
2	12821	1547	1000
2	12786	1547	1000
2	12796	1547	1000
2	12792	1547	1000
2	12774	1547	1000
Média			
2	12793	1547	1000

Merge Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
6	68038	6546	5000
6	67979	6546	5000
6	68002	6546	5000
6	68035	6546	5000
6	67983	6546	5000
Média			
6	68007	6546	5000

Merge Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
13	188451	16545	10000
13	188535	16545	10000
13	188491	16545	10000
13	188481	16545	10000
13	188335	16545	10000
Média			
13	188458	16545	10000

3. ShellSort

Primeiro, escolhemos um número que vai determinar o tamanho dos pedaços. Em seguida, dividimos a lista em pedaços usando esse número e organizamos cada pedaço separadamente usando o método de inserção. Depois, reduzimos o número, e fazemos tudo de novo. Os pedaços ficam menores, então a organização fica mais fácil. O algoritmo continua a reduzir o tamanho do intervalo e a ordenar as sublistas até que o intervalo se torne 1. Quando o intervalo é igual a 1, o Shell Sort funciona como um algoritmo de inserção padrão. O Shell Sort é uma ótima opção para organizar arrays desordenados, especialmente quando o mesmo é grande. É mais rápido do que o método de inserção puro. No entanto, escolher os tamanhos dos pedaços certos é importante para obter o melhor desempenho.

Shell Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
0	193	141	50
0	170	141	50
0	174	141	50
0	173	141	50
0	179	141	50
Média			
0	177	141	50

Shell Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
0	13148	1491	500
0	13556	1491	500
0	13773	1491	500
0	14240	1491	500
0	13601	1491	500
Média			
0	13663	1491	500

Shell Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
1	51886	2991	1000
1	55952	2991	1000
1	53254	2991	1000
1	50806	2991	1000
1	52047	2991	1000
Média			
1	52789	2991	1000

Shell Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
1	1267284	14991	5000
1	1265174	14991	5000
1	1269777	14991	5000
1	1304867	14991	5000
1	1274218	14991	5000
Média			
1	1276264	14991	5000

Shell Sort			
Tempo	Número de Trocas	Iterações	Quantidade de Elementos
1	5026007	29991	10000
1	5035487	29991	10000
1	5030121	29991	10000
1	5056916	29991	10000
1	5023990	29991	10000
Média			
1	5034504	29991	10000

4. Conclusão

A escolha entre esses algoritmos depende do tamanho do conjunto de dados, da estabilidade, do espaço disponível e do desempenho desejado. O BubbleSort é o menos eficiente, o MergeSort é altamente eficiente em grandes conjuntos de dados, e o ShellSort é uma escolha intermediária que pode ser eficaz em muitos cenários. Em resumo, a escolha do algoritmo de ordenação deve ser baseada nas características específicas do problema que você está enfrentando. É importante considerar o tamanho do conjunto de dados, a estabilidade, o espaço disponível e o desempenho desejado ao selecionar o algoritmo apropriado. Cada algoritmo tem suas próprias vantagens e desvantagens, e a escolha correta pode fazer uma grande diferença no desempenho de um programa ou sistema.