Comparação de Algoritmos de Ordenação

Evelyn Melo de Matos
26/07/2024

O que vamos abordar:

- Implementação de cinco algoritmos de ordenação clássicos.
- Comparação de desempenho com diferentes tipos de dados.
- Análise das complexidades teóricas e desempenho prático.

Algoritmos analisados:

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Merge Sort
5. HeapSort

Bubble Sort

O algoritmo compara pares de elementos adjacentes e os troca se estiverem na ordem errada.

Complexidade Temporal (Pior Caso): $O(n^2)$

Complexidade Espacial: O(1) (in-place)

Vantagens

- Simples de implementar.
- Pode detectar se o array já está ordenado.

Desvantagens:

- Ineficiente para grandes conjuntos de dados.

Resultados dos Testes - Bubble Sort

O **Bubble Sort** é um algoritmo de ordenação simples e ineficiente em termos de complexidade de tempo. Ele tem um desempenho de **O(n²)**, o que significa que seu tempo de execução aumenta exponencialmente à medida que o tamanho do conjunto de dados cresce.
Os resultados dos testes confirmam que o **Bubble Sort** não é adequado para conjuntos de dados grandes devido ao seu crescimento. Para conjuntos pequenos, o algoritmo pode ser viável, mas para grandes volumes de dados, alternativas mais eficientes devem ser consideradas.

Insertion Sort

O algoritmo constrói a ordenação inserindo cada elemento na posição correta em um array já parcialmente ordenado.

Complexidade Temporal (Pior Caso): $O(n^2)$

Complexidade Espacial: O(1) (in-place)

Vantagens

- Razoavelmente eficiente para pequenos conjuntos de dados ou quase ordenados.
- Simples de implementar.

Desvantagens:

- Não é adequado para grandes volumes de dados desordenados.

Resultados dos Testes - Insertion Sort

Em nossos testes, observamos que o **Insertion Sort** está retornando **0 ms** em quase todos os casos, exceto em conjuntos "Crescentes com Repetição". Esse comportamento pode indicar que o algoritmo está lidando com dados quase ordenados (caso crescente) de forma eficiente ou que há um problema na medição do tempo.

Selection Sort

O algoritmo seleciona o menor (ou maior) elemento de uma lista e o move para a posição correta, repetidamente.

Complexidade Temporal (Pior Caso): $O(n^2)$

Complexidade Espacial: O(1) (in-place)

Vantagens

- Funciona bem em situações onde o custo da troca é irrelevante.

Desvantagens:

- Ineficiente em comparação a algoritmos como Merge Sort

Resultados dos Testes - Selection Sort

Embora o **Selection Sort** seja intuitivo e fácil de implementar, ele tem um desempenho relativamente pobre para grandes volumes de dados, especialmente em comparação com algoritmos mais eficientes, como **Merge Sort.**
O **Selection Sort** não é um algoritmo estável, o que significa que elementos com o mesmo valor podem ter sua ordem relativa modificada após a ordenação.

O **Selection Sort** é útil principalmente em casos onde a simplicidade e a memória constante são mais importantes do que o tempo de execução. Porém, devido ao seu comportamento **$O(n^2)$**, ele não é recomendável para grandes volumes de dados. Algoritmos como **Merge Sort**, **ou Heap Sort** oferecem desempenho significativamente melhor em cenários com conjuntos de dados grandes.

Merge Sort

Divide o array em duas metades, ordena cada metade e, em seguida, mescla as duas metades ordenadas.

Complexidade Temporal (Pior Caso): O(n log n)

Complexidade Espacial: O(n) (não in-place)

Vantagens

- Consistentemente eficiente, mesmo para grandes conjuntos de dados.
- Estável, preserva a ordem relativa de elementos iguais.

Desvantagens:

- Requer espaço adicional proporcional ao tamanho do array.

Resultados dos Testes - Merge Sort

O **Merge Sort** apresentou tempos de execução extremamente rápidos e consistentes, mesmo com o aumento do tamanho dos dados. Isso reflete a sua complexidade **O(n log n)**, que é uma das mais eficientes entre os algoritmos de ordenação comparativa.

O **Merge Sort** é **estável**, ou seja, ele mantém a ordem relativa dos elementos com valores iguais, o que pode ser uma vantagem em algumas aplicações que dependem dessa característica.

O **Merge Sort** é um algoritmo de ordenação altamente eficiente, especialmente adequado para grandes conjuntos de dados. Ele garante um desempenho **O(n log n)** consistente em todos os casos, tornando-o preferível em cenários onde a estabilidade é importante ou quando o tempo de execução é crítico. Embora consuma mais memória do que algoritmos **in-place**, seu tempo de execução justifica seu uso em muitas situações práticas.

HeapSort

Constrói uma heap máxima (ou mínima) e, em seguida, remove o maior (ou menor) elemento repetidamente.

Complexidade Temporal (Pior Caso): O(n log n)

Complexidade Espacial: O(1) (in-place)

Vantagens

- Não requer espaço adicional além da entrada original.
- Bom desempenho com grandes conjuntos de dados.

Desvantagens:

- Não é estável (não preserva a ordem relativa de elementos iguais).
- Implementação um pouco mais complexa que outros algoritmos simples.

Resultados dos Testes - HeapSort

O **Heap Sort** é um algoritmo de ordenação com complexidade **O(n log n)** em seu pior caso, assim como o **Merge Sort**. Entretanto, devido à natureza das operações de heapificação, ele pode ter um desempenho ligeiramente inferior ao **Merge Sort** em alguns cenários.

O **Heap Sort** é uma escolha confiável quando o foco é garantir desempenho consistente com grandes volumes de dados. Em comparação com outros algoritmos como **Bubble Sort** ou **Insertion Sort**, ele é significativamente mais rápido e mantém o desempenho em níveis aceitáveis à medida que o tamanho dos dados aumenta.

Comparação Geral dos Algoritmos

Tabela de Desempenho

Aleatório com Repetição	100000	160000	220000	280000	340000	400000	460000	520000	580000	640000	700000
Bubble Sort	11470 ms	29230 ms	55940 ms	90034 ms	134870 ms	186715 ms	244696 ms	306648 ms	390854 ms	479063 ms	562371 ms
Insertion Sort	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms
Selection Sort	730 ms	1832 ms	3577 ms	5523 ms	8838 ms	11824 ms	15303 ms	20503 ms	25922 ms	31604 ms	37690 ms
Merge Sort	3 ms	8 ms	7 ms	9 ms	15 ms	21 ms	23 ms	28 ms	29 ms	28 ms	26 ms
HeapSort	5 ms	8 ms	12 ms	17 ms	26 ms	24 ms	27 ms	35 ms	34 ms	40 ms	44 ms

Comparação Geral dos Algoritmos

Tabela de Desempenho

Aleatório sem Repetição	100000	160000	220000	280000	340000	400000	460000	520000	580000	640000	700000
Bubble Sort	11034 ms	29441 ms	56037 ms	90511 ms	135445 ms	188115 ms	249766 ms	306648 ms	390854 ms	481800 ms	483623 ms
Insertion Sort	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms
Selection Sort	690 ms	1791 ms	3471 ms	5963 ms	9367 ms	11782 ms	17312 ms	20503 ms	25922 ms	29329 ms	29450 ms
Merge Sort	2 ms	7 ms	11 ms	10 ms	9 ms	11 ms	13 ms	28 ms	29 ms	19 ms	20 ms
HeapSort	5 ms	9 ms	12 ms	15 ms	21 ms	24 ms	29 ms	35 ms	34 ms	40 ms	40 ms

Comparação Geral dos Algoritmos

Tabela de Desempenho

Crescente com Repetição	100000	160000	220000	280000	340000	400000	460000	520000	580000	640000	700000
Bubble Sort	939 ms	2348 ms	4446 ms	7323 ms	10763 ms	14895 ms	20153 ms	26098 ms	31749 ms	39341 ms	562371 ms
Insertion Sort	1 ms	1 ms	2 ms	2 ms	3 ms	3 ms	3 ms	3 ms	3 ms	3 ms	0 ms
Selection Sort	751 ms	1799 ms	3428 ms	6781 ms	8152 ms	13524 ms	18504 ms	23439 ms	28899 ms	30659 ms	37690 ms
Merge Sort	7 ms	11 ms	17 ms	17 ms	19 ms	22 ms	26 ms	29 ms	32 ms	36 ms	26 ms
HeapSort	11 ms	17 ms	24 ms	26 ms	30 ms	29 ms	35 ms	33 ms	38 ms	37 ms	44 ms

Comparação Geral dos Algoritmos

Tabela de Desempenho

Crescente sem Repetição	100000	160000	220000	280000	340000	400000	460000	520000	580000	640000	700000
Bubble Sort	1261 ms	3472 ms	6191 ms	9866 ms	14821 ms	20807 ms	26428 ms	34919 ms	43368 ms	54503 ms	483623 ms
Insertion Sort	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms
Selection Sort	712 ms	1906 ms	3408 ms	5581 ms	8536 ms	12091 ms	15646 ms	20579 ms	26561 ms	30018 ms	29450 ms
Merge Sort	6 ms	4 ms	10 ms	15 ms	17 ms	22 ms	18 ms	15 ms	17 ms	21 ms	20 ms
HeapSort	6 ms	9 ms	13 ms	16 ms	20 ms	24 ms	29 ms	33 ms	38 ms	41 ms	40 ms

Comparação Geral dos Algoritmos

Tabela de Desempenho

Decrescente com Repetição	100000	160000	220000	280000	340000	400000	460000	520000	580000	640000	700000
Bubble Sort	5588 ms	13994 ms	25876 ms	42421 ms	62288 ms	88723 ms	112796 ms	140324 ms	181895 ms	220328 ms	562371 ms
Insertion Sort	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms
Selection Sort	811 ms	1779 ms	3421 ms	6659 ms	8187 ms	14216 ms	18358 ms	22519 ms	30248 ms	30777 ms	37690 ms
Merge Sort	5 ms	7 ms	11 ms	13 ms	15 ms	15 ms	16 ms	21 ms	26 ms	29 ms	26 ms
HeapSort	5ms	8 ms	11 ms	14 ms	18 ms	21 ms	24 ms	27 ms	32 ms	35 ms	44 ms

Comparação Geral dos Algoritmos

Tabela de Desempenho

Decrescente sem Repetição	100000	160000	220000	280000	340000	400000	460000	520000	580000	640000	700000
Bubble Sort	5744 ms	15028 ms	28329 ms	46073 ms	65707 ms	94548 ms	124871 ms	159598 ms	196309 ms	246577 ms	483623 ms
Insertion Sort	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms
Selection Sort	695 ms	1804 ms	3493 ms	5820 ms	8415 ms	12940 ms	15967 ms	19482 ms	26008 ms	29590 ms	29450 ms
Merge Sort	4 ms	5 ms	10 ms	14 ms	13 ms	13 ms	14 ms	14 ms	18 ms	19 ms	20 ms
HeapSort	5 ms	9 ms	13 ms	17 ms	20 ms	26 ms	29 ms	32 ms	35 ms	40 ms	40 ms