Bubble Sort:

O Bubble Sort é um algoritmo de classificação simples que funciona trocando repetidamente os elementos adjacentes se estiverem fora de ordem. No entanto, esse algoritmo não é eficiente para conjuntos de dados grandes, pois sua complexidade média e pior caso são altas.

No algoritmo de classificação de bolhas, percorremos a lista da esquerda para a direita, comparando os elementos adjacentes e movendo o maior elemento para a direita. Esse processo é repetido para encontrar o segundo maior elemento, e assim por diante, até que os dados estejam classificados.

O funcionamento do Bubble Sort pode ser entendido com a ajuda de um exemplo. Considere a entrada: arr[] = {6, 3, 0, 5}. Na primeira passagem, o maior elemento é colocado em sua posição correta, que é o final da matriz. Na segunda passagem, o segundo maior elemento é colocado em sua posição correta. Esse processo continua até que os dois elementos restantes estejam nas posições corretas.

O Bubble Sort possui algumas vantagens, como ser fácil de entender e implementar, não requerer espaço de memória adicional e ser um algoritmo de classificação estável, ou seja, mantém a ordem relativa dos elementos com o mesmo valor de chave na saída classificada.

No entanto, também possui algumas desvantagens. Sua complexidade temporal é O(N^2), tornando-o lento para conjuntos de dados grandes. Além disso, o Bubble Sort é um algoritmo de classificação baseado em comparação, o que significa que requer um operador de comparação para determinar a ordem dos elementos. Isso pode limitar sua eficiência em certos casos.

O Bubble Sort é frequentemente usado para introduzir o conceito de algoritmo de classificação devido à sua simplicidade. Na computação gráfica, ele é popular por sua capacidade de detectar pequenos erros em matrizes quase ordenadas e corrigi-los com complexidade linear. Um exemplo de uso do Bubble Sort é em algoritmos de preenchimento de polígonos, onde as linhas delimitadoras são classificadas por suas coordenadas x em uma linha de varredura específica e suas ordens mudam apenas em interseções de duas linhas com um incremento em y.

Selection Sort

O Selection Sort é um algoritmo simples e eficiente de classificação que funciona selecionando repetidamente o menor (ou maior) elemento da parte não classificada da lista e movendo-o para a parte classificada. O processo é repetido até que toda a lista esteja classificada.

O funcionamento do algoritmo é o seguinte: para cada posição na parte classificada da lista, percorre-se o restante da lista para encontrar o elemento mínimo. Esse elemento é então trocado com o primeiro elemento da parte não classificada. Esse processo é repetido para cada posição restante até que a lista esteja completamente classificada.

Por exemplo, considerando a matriz {64, 25, 12, 22, 11}, na primeira iteração, percorre-se a matriz e encontra-se o valor mínimo, que é 11. Esse valor é trocado com o elemento atualmente na primeira posição da lista. Na segunda iteração, encontra-se o segundo menor valor, que é 12, e o troca com o elemento na segunda posição. Esse processo continua até que a lista esteja completamente classificada.

A complexidade de tempo do Selection Sort é O(N^2), onde N é o tamanho da lista, devido aos dois loops aninhados. O algoritmo também utiliza um espaço auxiliar constante, pois faz trocas apenas entre dois elementos da lista.

Entre as vantagens do Selection Sort, destacam-se a simplicidade e facilidade de compreensão, além de funcionar bem com conjuntos de dados pequenos. No entanto, ele apresenta algumas desvantagens, como a complexidade de tempo O(N^2) em casos médios e piores, o que o torna ineficiente para grandes conjuntos de dados. Além disso, ele não preserva a ordem relativa dos itens com chaves iguais, o que significa que não é um algoritmo estável.

O Selection Sort pode ser usado em: Conjuntos de dados pequenos, Listas parcialmente ordenadas, Implementações simples e em Casos em que a memória é limitada.

Insertion Sort:

A classificação por inserção é um algoritmo simples que funciona de maneira semelhante à forma como você organiza as cartas de um baralho em suas mãos. O array é dividido virtualmente em uma parte classificada e uma parte não classificada. Os valores da parte não classificada são selecionados e colocados na posição correta na parte classificada.

O algoritmo de classificação por inserção funciona da seguinte maneira: para classificar uma matriz de tamanho N em ordem crescente, itera-se sobre a matriz e compara-se o elemento atual (chave) com seu antecessor. Se o elemento chave for menor que seu antecessor, ele é comparado com os elementos anteriores e os elementos maiores são movidos uma posição para cima para abrir espaço para o elemento a ser trocado.

Para ilustrar o funcionamento, vamos considerar um exemplo: arry = {12, 11, 13, 5, 6}.

Na primeira passagem, os dois primeiros elementos da matriz são comparados. Como 12 é maior que 11, eles não estão em ordem crescente, então ocorre uma troca entre 11 e 12. Assim, por enquanto, 11 está na submatriz classificada.

Na segunda passagem, passamos para os próximos dois elementos (12 e 13) e verificamos que estão em ordem crescente, portanto, nenhuma troca ocorre. 12 também é armazenado na submatriz classificada junto com 11.

Na terceira passagem, temos dois elementos na submatriz classificada (11 e 12) e avançamos para os próximos dois elementos (13 e 5). Como tanto o 5 quanto o 13 não estão em suas posições corretas, eles são trocados. Após a troca, os elementos 12 e 5 ainda não estão classificados, então ocorre outra troca. Novamente, 11 e 5

não estão classificados, então ocorre mais uma troca. Agora, 5 está em sua posição correta.

Na quarta passagem, os elementos presentes na submatriz classificada são 5, 11 e 12. Passando para os próximos dois elementos (13 e 6), verificamos que eles não estão classificados, então ocorre uma troca entre eles. Em seguida, 6 é menor que 12, então ocorre outra troca. Novamente, a troca faz com que 11 e 6 fiquem fora de ordem, então ocorre mais uma troca. Finalmente, a matriz está completamente classificada.

A complexidade de tempo da classificação por inserção é O(N^2) no pior caso e no caso médio. No melhor caso, a complexidade de tempo é O(N). A complexidade de espaço auxiliar é O(1), ou seja, não é necessário espaço adicional além do array original.

Algumas características da classificação por inserção são: é um algoritmo simples com implementação simples, é eficiente para conjuntos de dados pequenos e é adequado para conjuntos de dados parcialmente classificados.

Aplicabilidade da classificação por inserção:

A classificação por inserção é mais adequada para conjuntos de dados pequenos ou parcialmente classificados.

É útil quando a eficiência em termos de tempo de execução não é uma prioridade e a simplicidade e facilidade de implementação são mais importantes.

Vantagens da classificação por inserção:

Simplicidade: A classificação por inserção é um algoritmo fácil de entender e implementar. Requer menos código em comparação com outros algoritmos de classificação mais complexos.

Eficiência para pequenos conjuntos de dados: É eficiente em termos de tempo de execução para conjuntos de dados pequenos, pois possui uma complexidade de tempo de melhor caso de O(N), onde N é o tamanho da matriz.

Adaptação a dados parcialmente classificados: É uma escolha adequada quando o conjunto de dados já está parcialmente classificado, pois requer menos operações de troca em comparação com outros algoritmos.

Desvantagens da classificação por inserção:

Ineficiência para grandes conjuntos de dados: A complexidade de tempo no pior caso e no caso médio é O(N^2), o que significa que a classificação por inserção pode se tornar muito lenta e ineficiente quando aplicada a grandes conjuntos de dados.

Baixo desempenho em geral: Comparado a outros algoritmos de classificação, como o merge sort ou quicksort, a classificação por inserção geralmente apresenta um desempenho inferior. Portanto, em situações onde a eficiência é crucial, outros algoritmos são preferíveis.

Sensibilidade à ordem de entrada: A eficiência da classificação por inserção é afetada pela ordem de entrada dos elementos. Se a entrada já estiver quase completamente classificada em ordem decrescente, por exemplo, o algoritmo exigirá um grande número de operações de troca, tornando-o menos eficiente.

Merge Sort:

A classificação por mesclagem é um algoritmo que divide uma matriz em submatrizes menores, as classifica individualmente e, em seguida, mescla as submatrizes classificadas para obter a matriz final classificada.

De forma simplificada, o processo de classificação por mesclagem divide a matriz pela metade, classifica cada metade e, em seguida, mescla as metades classificadas. Esse processo é repetido até que toda a matriz esteja classificada.

A classificação por mesclagem é especialmente útil devido à sua eficiência. Possui uma complexidade de tempo de O(n log n), o que significa que pode classificar rapidamente matrizes grandes. Além disso, é uma classificação estável, preservando a ordem dos elementos iguais durante a classificação.

É uma opção popular para classificar conjuntos de dados extensos devido à sua eficiência e facilidade de implementação. Frequentemente, é usado em combinação com outros algoritmos, como o quicksort, para melhorar o desempenho geral de um processo de classificação.

O processo de trabalho da classificação por mesclagem envolve a divisão recursiva da matriz pela metade até que não seja mais possível dividi-la. Quando a matriz fica vazia ou contém apenas um elemento, a divisão é interrompida. Se a matriz tiver vários elementos, ela é dividida em metades e a classificação por mesclagem é invocada recursivamente em cada uma delas. Por fim, quando ambas as metades estão classificadas, ocorre a mesclagem, combinando duas submatrizes classificadas em uma maior.

A classificação por mesclagem é aplicada em várias situações. É comumente usada para classificar grandes conjuntos de dados devido à sua complexidade de tempo garantida. Também é adequada para classificação externa, onde os dados não cabem na memória principal. Além disso, é naturalmente paralelizável, tornando-a útil em aplicações que requerem alto desempenho computacional. Sua estabilidade e capacidade de lidar com diferentes distribuições de entrada são características valiosas em bancos de dados, sistemas financeiros e outros contextos.

Embora a classificação por mesclagem tenha várias vantagens, como estabilidade, bom desempenho no pior caso, paralelização, eficiência de memória, versatilidade e adaptabilidade, também possui algumas desvantagens. Requer memória adicional para armazenar submatrizes mescladas, pode resultar em um grande número de chamadas de função e uso de pilha para conjuntos de dados muito grandes, não é um algoritmo in-loco e pode ser mais complexo de implementar em comparação com outros algoritmos de classificação. Além disso, para conjuntos de dados pequenos, pode ser menos eficiente do que algoritmos como a classificação por inserção.

Quick Sort:

O processo chave do Quicksort é a função de particionamento. O objetivo do particionamento é posicionar o pivô corretamente na matriz classificada, colocando os elementos menores à esquerda e os elementos maiores à direita. Esse processo é feito recursivamente em cada lado do pivô, o que resulta na classificação final da matriz.

A escolha do pivô pode ser feita de diferentes maneiras, e o algoritmo de particionamento segue uma lógica simples. Ele começa com o elemento mais à esquerda e mantém um índice para os elementos menores ou iguais ao pivô (chamado de índice i). Durante a varredura, se um elemento menor é encontrado, ele é trocado com o elemento atual em arr[i]. Caso contrário, o elemento é ignorado.

O Quicksort é amplamente utilizado devido à sua rapidez e facilidade de implementação. Sua complexidade média de tempo é O(n log n), o que o torna eficiente para grandes conjuntos de dados. Além disso, ele pode ser implementado de forma recursiva ou iterativa e não requer espaço adicional, sendo in-place.

Embora o Quicksort tenha um pior caso de complexidade quadrática, esse cenário raramente ocorre na prática. Com a escolha inteligente do pivô e otimizações, como a seleção aleatória do pivô, o desempenho do Quicksort é geralmente bom.

No entanto, é importante destacar que existem outros algoritmos de ordenação que podem ser mais adequados em certos casos. Além disso, o Quicksort não preserva a ordem relativa de elementos iguais, o que pode ser relevante dependendo do contexto.

Aplicabilidade: O Quicksort é amplamente utilizado em uma variedade de situações para ordenar diferentes tipos de dados. Aqui estão alguns casos comuns em que o Quicksort é aplicado: Grande volume de dados, Ordenação em memória principal, Versatilidade, Desempenho médio, Implementação simples.

Vantagens do Quicksort incluem o fato de ser um algoritmo de "Dividir e Conquistar" que facilita a resolução de problemas, sua eficiência em grandes conjuntos de dados e a baixa sobrecarga de memória. Por outro lado, suas desvantagens incluem a complexidade de tempo pior em caso de escolha inadequada do pivô, não sendo ideal para pequenos conjuntos de dados, e a falta de estabilidade na ordenação, o que significa que a ordem relativa de elementos iguais pode não ser preservada na saída classificada.

Heap Sort:

A classificação por heap é um método de classificação baseado em comparações que utiliza a estrutura de dados do heap binário. É semelhante à classificação por seleção, onde encontramos primeiro o elemento mínimo e o colocamos no início. Repetimos esse processo para os elementos restantes.

O algoritmo de classificação por heap segue a seguinte ideia para resolver o problema:

Primeiro, convertemos a matriz em uma estrutura de dados de heap usando o processo de heapify. Em seguida, removemos um a um o nó raiz do heap máximo e

o substituímos pelo último nó no heap. Depois, realizamos o processo de heapify novamente na raiz do heap. Repetimos esse processo até que o tamanho do heap seja maior que 1.

Para realizar a classificação por heap, seguimos os seguintes passos: Criamos um heap a partir da matriz de entrada fornecida. Repetimos as seguintes etapas até que o heap contenha apenas um elemento: a. Trocamos o elemento raiz do heap (que é o maior elemento) pelo último elemento do heap. b. Removemos o último elemento do heap (que agora está na posição correta). c. Heapificamos os elementos restantes do heap. A matriz classificada é obtida invertendo a ordem dos elementos na matriz de entrada.

Para entender melhor o processo de classificação por heap, vamos utilizar um exemplo com uma matriz não classificada: arr[] = {4, 10, 3, 5, 1}.

Transformamos essa matriz em um heap máximo, onde o pai é sempre maior ou igual aos filhos. No exemplo, trocamos o pai 4 com o filho 10 para criar um heap máximo. Em seguida, trocamos o pai 4 com o filho 5, obtendo o seguinte heap e matriz resultante.

Em seguida, executamos a classificação por heap, removendo o elemento máximo em cada etapa e transformando os elementos restantes em um heap máximo. Após a remoção do elemento raiz (10) e a realização do processo de heapify, obtemos o seguinte heap e matriz resultante:

Repetimos as etapas anteriores até obtermos o seguinte resultado:

Agora, ao removermos a raiz (3) novamente e executarmos o heapify, obtemos a matriz classificada: arr[] = {1, 3, 4, 5, 10}.

Alguns pontos importantes sobre a classificação por heap:

A classificação por heap é um algoritmo in-place. Sua implementação típica não é estável, mas pode ser tornada estável. Normalmente é cerca de 2-3 vezes mais lenta do que o QuickSort bem implementado, devido à falta de localidade de referência.

O Heap Sort tem uma ampla aplicabilidade em diferentes cenários, especialmente quando se trata de grandes conjuntos de dados ou em situações em que a estabilidade não é uma preocupação. Algumas das aplicações mais comuns do Heap Sort incluem: Classificação em tempo real: O Heap Sort é um algoritmo in-place, o que significa que ele pode classificar os dados diretamente na memória principal, sem a necessidade de estruturas de dados adicionais. Isso o torna adequado para classificar dados em tempo real, onde a eficiência e o uso mínimo de memória são cruciais. Classificação de grandes volumes de dados: O Heap Sort tem um desempenho relativamente bom para conjuntos de dados muito grandes. Sua complexidade de tempo é O(n log n), o que o torna uma opção eficiente para classificar grandes volumes de dados quando comparado a outros algoritmos com complexidades superiores, como o Bubble Sort. Filas de prioridade: O Heap Sort é frequentemente usado como base para implementar estruturas de dados de fila de prioridade, onde os elementos são inseridos e removidos de acordo com suas

prioridades. A propriedade de heap permite que o Heap Sort seja usado para gerenciar eficientemente a fila de prioridade. Algoritmo auxiliar: Embora o Heap Sort possa ser mais lento do que outros algoritmos de classificação como o QuickSort em muitos casos, ele ainda é útil como um algoritmo auxiliar em algoritmos mais complexos. Por exemplo, o Heap Sort é usado como parte do algoritmo de ordenação por seleção em várias implementações.

Vantagens do Heap Sort: Eficiência: O tempo necessário para executar a classificação por heap aumenta logaritmicamente, tornando-o um algoritmo de classificação muito eficiente. Uso de memória: O uso de memória é mínimo, pois não requer espaço adicional além da lista inicial de itens a serem classificados. Simplicidade: É mais simples de entender do que outros algoritmos de classificação eficientes, pois não utiliza conceitos avançados de ciência da computação, como recursão.

Desvantagens do Heap Sort:Custo: A classificação por heap pode ser custosa. Instabilidade: A classificação por heap é instável, podendo reorganizar a ordem relativa dos elementos. Eficiência: Não é muito eficiente ao trabalhar com dados altamente complexos.

Referencias:

Bubble Sort - Tutoriais de Estrutura de Dados e Algoritmos - GeeksforGeeks

Selection Sort – Tutoriais de Estrutura de Dados e Algoritmos - GeeksforGeeks

Insertion Sort - Tutoriais de Estrutura de Dados e Algoritmos - GeeksforGeeks

Algoritmo de Classificação de Mesclagem - GeeksforGeeks

QuickSort - Tutoriais de Estrutura de Dados e Algoritmos - GeeksforGeeks

Heap Sort - Tutoriais de Estruturas de Dados e Algoritmos - GeeksforGeeks