Curso: Desenvolvimento de Software Multiplataforma
Disciplina: Estrutura de Dados – 2º semestre
Aluno: João Paulo Falcuci Teixeira **R.A.:** 1091392223005

Complexidade de algoritmos

A análise de complexidade de algoritmos é uma forma de medir a eficiência de um algoritmo em termos de tempo de execução e uso de recursos, como memória. Essa análise nos ajuda a entender como o desempenho de um algoritmo é afetado pelo tamanho dos dados de entrada. Existem duas medidas principais de complexidade de algoritmos: complexidade de tempo e complexidade de espaço. A complexidade de tempo mede quanto tempo um algoritmo leva para ser executado, enquanto a complexidade de espaço mede quanto espaço de memória é necessário para executar o algoritmo.

Complexidade de tempo

A complexidade de tempo é geralmente expressa usando a notação Big O, que fornece uma aproximação do comportamento do algoritmo à medida que o tamanho dos dados de entrada aumenta. Por exemplo, se um algoritmo tem uma complexidade de tempo O(n), isso significa que o tempo de execução do algoritmo cresce linearmente com o tamanho dos dados de entrada.

Exemplos de complexidades de tempo comuns:

- O(1) - Tempo constante: Isso significa que o tempo de execução do algoritmo é constante, independentemente do tamanho dos dados de entrada. Um exemplo simples disso é acessar um elemento em um array pelo seu índice.
- O(n) - Tempo linear: Isso significa que o tempo de execução do algoritmo cresce linearmente com o tamanho dos dados de entrada. Um exemplo é percorrer todos os elementos de um array para realizar uma determinada operação.

- O(n^2) - Tempo quadrático: Isso significa que o tempo de execução do algoritmo cresce quadraticamente com o tamanho dos dados de entrada. Um exemplo é um algoritmo de ordenação por seleção, que compara todos os pares de elementos do array.
- O(log n) - Tempo logarítmico: Isso significa que o tempo de execução do algoritmo cresce de forma logarítmica com o tamanho dos dados de entrada. Um exemplo é a busca binária em um array ordenado.
- O(n log n) - Tempo linearítmico: Isso significa que o tempo de execução do algoritmo cresce de forma linear em relação ao tamanho dos dados de entrada multiplicado pelo logaritmo desse tamanho. Um exemplo é o algoritmo Quick Sort, que divide repetidamente a lista em sub-listas menores e as ordena de forma recursiva.

É importante observar que a complexidade de tempo nos dá uma ideia de como o tempo de execução do algoritmo aumenta à medida que o tamanho dos dados de entrada aumenta, mas não nos diz o tempo exato que o algoritmo levará para ser executado.

Complexidade de espaço

A complexidade de espaço em um algoritmo mede a quantidade de memória necessária para executar o algoritmo em relação ao tamanho dos dados de entrada. Isso nos ajuda a entender o uso de recursos de memória do algoritmo. Assim como a complexidade de tempo, a complexidade de espaço também é expressa usando a notação Big O.

Exemplos de complexidades de espaço comuns:

- O(1) - Espaço constante: Isso significa que o algoritmo utiliza uma quantidade fixa de espaço, independentemente do tamanho dos dados de entrada. Um exemplo simples disso é uma variável inteira que armazena um único valor.
- O(n) - Espaço linear: Isso significa que o espaço necessário pelo algoritmo aumenta linearmente com o tamanho dos dados de entrada. Um exemplo é um

algoritmo que cria um novo array para armazenar os elementos de entrada em uma nova ordem.

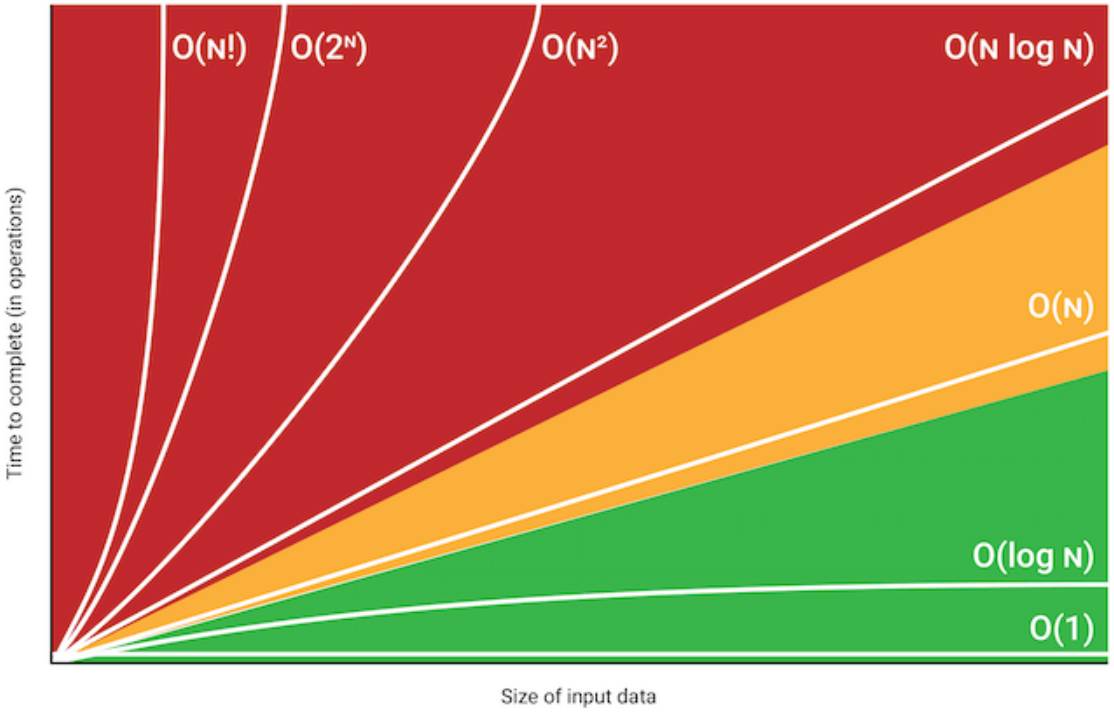
- O(n^2) - Espaço quadrático: Isso significa que o espaço necessário pelo algoritmo cresce quadraticamente com o tamanho dos dados de entrada. Um exemplo é um algoritmo que cria uma matriz bidimensional para armazenar todas as combinações possíveis de elementos em um conjunto.
- O(log n) - Espaço logarítmico: Isso significa que o espaço necessário pelo algoritmo aumenta de forma logarítmica com o tamanho dos dados de entrada. Um exemplo é um algoritmo que utiliza uma pilha para realizar uma operação de busca em uma árvore binária.

É importante notar que a complexidade de espaço também pode ser influenciada por fatores como o tamanho de estruturas de dados auxiliares usadas pelo algoritmo, alocação dinâmica de memória, entre outros.

Ao analisar a complexidade de espaço, é importante considerar tanto o espaço utilizado pelo algoritmo em si quanto o espaço necessário para armazenar os dados de entrada.

Em resumo, a complexidade de espaço nos fornece informações sobre a quantidade de memória necessária para executar um algoritmo em relação ao tamanho dos dados de entrada. Isso nos ajuda a avaliar o uso de recursos de memória do algoritmo e escolher a melhor abordagem para resolver um determinado problema.

Big O Notation – Complexidade (Gráfico)



Análise dos algoritmos de ordenação

Bubble Sort:

Complexidade de tempo no pior caso: O(n^2)

O Bubble Sort compara pares de elementos adjacentes e realiza trocas até que a lista esteja ordenada. No pior caso, quando a lista está completamente desordenada, o Bubble Sort requer várias iterações para ordená-la completamente. A cada iteração, o algoritmo percorre toda a lista uma vez, resultando em um tempo de execução proporcional ao quadrado do número de elementos (O(n^2))

Complexidade de espaço no pior caso: O(1)

Em termos de espaço, o Bubble Sort utiliza uma quantidade constante de espaço adicional, pois não requer estruturas de dados extras para armazenar os elementos (O(1)).

Selection Sort:

Complexidade de tempo no pior caso: O(n^2)

O Selection Sort encontra o menor elemento e o coloca na posição correta, repetindo esse processo para o restante da lista. No pior caso, são necessárias várias iterações para ordenar a lista. Em cada iteração, o algoritmo percorre a parte não ordenada da lista em busca do menor elemento, resultando em um tempo de execução proporcional ao quadrado do número de elementos (O(n^2)).

Complexidade de espaço no pior caso: O(1)

Assim como o Bubble Sort, o Selection Sort também utiliza uma quantidade constante de espaço adicional, pois não requer estruturas de dados extras (O(1)).

Merge Sort:

Complexidade de tempo no pior caso: O(n log n)

O Merge Sort utiliza a estratégia de divisão e conquista. Ele divide repetidamente a lista em duas metades, ordena cada metade separadamente e, em seguida, mescla as duas metades ordenadas para obter a lista final ordenada. A etapa de mesclagem é a parte essencial do algoritmo e é realizada de forma eficiente. Como resultado, o tempo de execução cresce de forma proporcional ao produto do tamanho dos dados de entrada (n) e seu logaritmo (log n), resultando em uma complexidade de tempo de O(n log n).

Complexidade de espaço no pior caso: O(n)

Durante o processo de mesclagem, o Merge Sort cria uma estrutura de dados temporária, geralmente um array, para armazenar os elementos mesclados. Essa estrutura de dados temporária pode ocupar espaço adicional proporcional ao tamanho dos dados de entrada (O(n)), já que todas as metades da lista precisam ser armazenadas temporariamente.

Quick Sort:

Complexidade de tempo no pior caso: O(n^2)

O Quick Sort escolhe um elemento chamado pivô e particiona a lista em duas sub-listas, uma com elementos menores que o pivô e outra com elementos maiores. Esse processo é realizado recursivamente até que a lista esteja ordenada. No pior caso, quando o pivô escolhido não divide a lista de forma equilibrada, o tempo de execução pode se tornar quadrático (O(n^2)). No entanto, em média, o Quick Sort

tem um desempenho muito melhor, com uma complexidade de tempo média de O(n log n).

Complexidade de espaço no pior caso: O(n)

Quanto ao espaço, durante o processo de particionamento, o Quick Sort precisa armazenar temporariamente os elementos em diferentes partições, requerendo espaço adicional proporcional ao tamanho dos dados de entrada (O(n)). Além disso, o Quick Sort utiliza a recursão para dividir a lista em sub-listas menores, consumindo espaço adicional proporcional ao número de elementos (O(n)). Portanto, a complexidade de espaço no pior caso é O(n).

Classificação dos algoritmos de ordenação

Bubble Sort: É o menos eficiente tanto em termos de tempo quanto de espaço. Possui uma complexidade de tempo quadrática (O(n^2)) e utiliza apenas uma quantidade constante de espaço adicional (O(1)).

Selection Sort: Também é considerado ineficiente tanto em tempo quanto em espaço. Assim como o Bubble Sort, possui uma complexidade de tempo quadrática (O(n^2)) e utiliza uma quantidade constante de espaço adicional (O(1)).

Merge Sort: É mais eficiente do que o Bubble Sort e o Selection Sort. Possui uma complexidade de tempo de O(n log n) no pior caso, o que indica um desempenho melhor. No entanto, utiliza uma quantidade adicional de espaço proporcional ao tamanho dos dados de entrada (O(n)) devido à criação de uma estrutura temporária durante o processo de mesclagem.

Quick Sort: É o mais eficiente em termos de tempo na maioria dos casos, mas no pior caso pode ser menos eficiente do que o Merge Sort. Possui uma complexidade de tempo quadrática (O(n^2)) no pior caso, mas em média tem uma complexidade de tempo de O(n log n). Quanto ao espaço, também utiliza uma quantidade adicional de espaço proporcional ao tamanho dos dados de entrada (O(n)).

Em resumo, o Bubble Sort e o Selection Sort têm complexidade de tempo e espaço no pior caso de O(n^2), enquanto o Merge Sort tem complexidade de tempo de O(n log n) e complexidade de espaço de O(n). O Quick Sort tem complexidade de tempo no pior caso de O(n^2) e complexidade de espaço de O(n). Considerando apenas o

pior caso, o Merge Sort é mais eficiente em termos de tempo, seguido pelo Quick Sort. O Bubble Sort e o Selection Sort são os menos eficientes. Em termos de espaço, o Merge Sort utiliza mais espaço adicional devido à criação da estrutura temporária durante o processo de mesclagem, enquanto os outros algoritmos (Bubble Sort, Selection Sort e Quick Sort) utilizam uma quantidade constante de espaço adicional.

Referências

Estruturas de Dados - Algoritmos, Análise de Complexidade em Java e C/C++ (Ana Fernanda Gomes Ascencio & Graziela Santos de Araújo)

O Problema de 1 MILHÃO de DÓLARES - Ciência Todo Dia

Introdução à Complexidade de Algoritmos - medium.com

Análise da Complexidade de Algoritmos - iugu.com/iugu4devs

Complexidade de Algoritmos - lapix.ufsc.br

Big O Notation: What Is It? - pub.towardsai.net

O que é Big O Notation? - medium.com