Ordenação e Análise: Merge Sort e Quick sort

Rafael Alves da Costa

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
FATEC Carapicuíba

Aula 13 - Estrutura de Dados

11/2025

Sumário

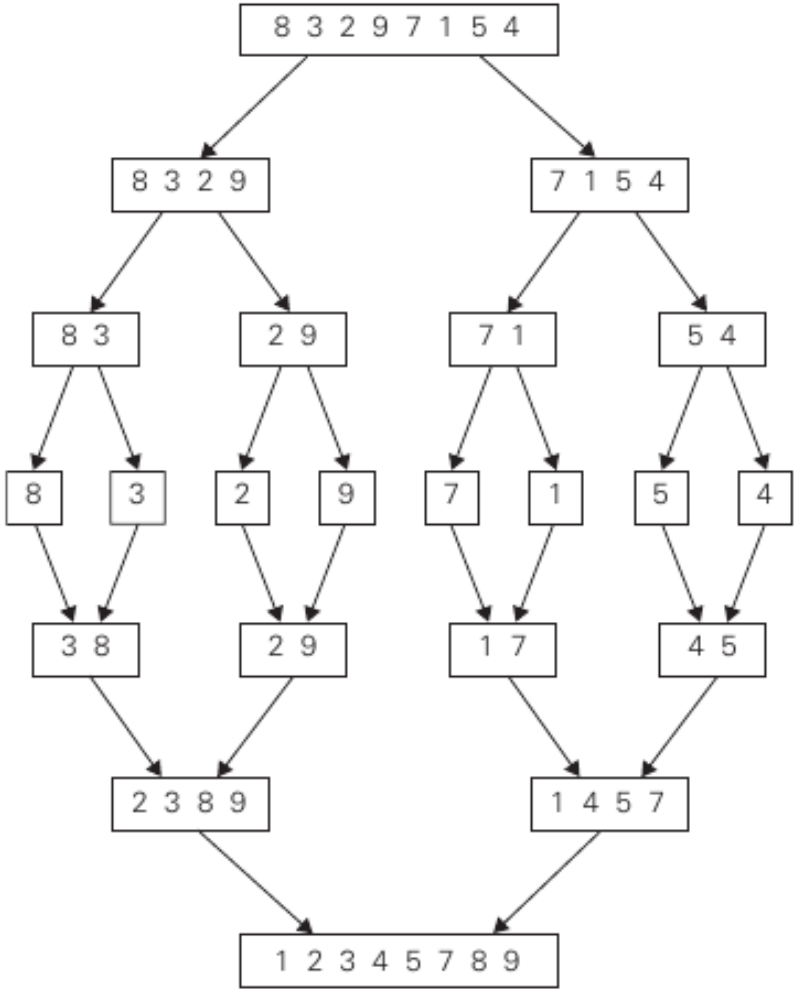
Algoritmos: Dividir para Conquistar (Conceito)

Conceito de Dividir e Conquistar

- O conceito de "Dividir e Conquista" é amplamente conhecido, talvez pelo seu nome atrativo, mas a fama é bem merecida: diversos algoritmos eficientes são implementações específicas dessa estratégia geral.
- Algoritmos de Dividir e Conquistar seguem o seguinte plano geral:

1. O problema é dividido em vários subproblemas do mesmo tipo, idealmente de tamanho aproximadamente igual.
2. Os subproblemas são resolvidos (tipicamente de forma recursiva, embora às vezes um algoritmo diferente seja empregado, especialmente quando os subproblemas se tornam pequenos o suficiente).
3. Se necessário, as soluções dos subproblemas são combinadas para obter uma solução para o problema original.

Exemplo Merge Sort



- **Vamos ver uma animação!!!**

Fatec Carapicuíba

Algoritmo: Mergesort

Algorithm Mergesort

1: **Entrada:** Um array $A[0..n-1]$ de elementos ordenáveis
2: **Saída:** Array $A[0..n-1]$ ordenado em ordem crescente
3: **if** $n > 1$ **then** $O(1)$
4: Copie $A[0..\lfloor n/2 \rfloor - 1]$ para $B[0..\lfloor n/2 \rfloor - 1]$ $O(n/2) = O(n)$
5: Copie $A[\lfloor n/2 \rfloor..n-1]$ para $C[0..\lceil n/2 \rceil - 1]$ $O(n/2) = O(n)$
6: **Mergesort**$(B[0..\lfloor n/2 \rfloor - 1])$ $T(n/2)$
7: **Mergesort**$(C[0..\lceil n/2 \rceil - 1])$ $T(n/2)$
8: **Merge**(B, C, A) $O(n)$
9: **end if**

Análise de Custo do Algoritmo

- **Linha 3:** Condicional $n > 1$, que possui custo constante $O(1)$.
- **Linha 4 e 5:** Cópias do array A para subarrays B e C. Cada cópia possui custo $O(n/2) = O(n)$.
- **Linha 6 e 7:** Chamadas recursivas para resolver cada subarray ($T(n/2)$ para cada chamada).
- **Linha 8:** Operação de mesclagem (**Merge**) dos subarrays B e C em A, com custo $O(n)$.

Complexidade Temporal Total

- A complexidade do Mergesort é descrita pela relação de recorrência:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

  onde $2T(n/2)$ representa o custo das duas chamadas recursivas e $O(n)$ o custo das operações de cópia e mesclagem.
- Usando o Teorema Mestre[1], obtemos:

$$T(n) = O(n \log n)$$

- Portanto, a complexidade temporal de Mergesort é $O(n \log n)$ no pior, melhor e caso médio.

[1] Teorema Mestre

Resumo da Análise de Custo

- **Complexidade Temporal:** $O(n \log n)$ em todos os casos.
- **Espaço Auxiliar:** $O(n)$ devido à necessidade de espaço adicional para os subarrays B e C em cada nível de recursão.
- Mergesort é um algoritmo eficiente e estável[2], particularmente adequado para ordenação de grandes listas, mas exige espaço adicional.

[2] "Um algoritmo é estável se preserva a ordem relativa de elementos iguais."

Introdução ao Quicksort

- Quicksort é outro algoritmo importante de ordenação baseado na abordagem de "Dividir e Conquistar".
- Diferente do Mergesort, que divide os elementos de entrada de acordo com suas posições no array, o Quicksort os divide de acordo com seus valores.
- A técnica de partição de arrays, utilizada no Quicksort, organiza os elementos de forma que:
  - Todos os elementos à esquerda de um elemento $A[s]$ são menores ou iguais a $A[s]$.
  - Todos os elementos à direita de $A[s]$ são maiores ou iguais a $A[s]$.

Exemplo de Partição do Array

Partição do Array:

$$A[0] \ldots A[s-1]\ A[s]\ A[s+1] \ldots A[n-1]$$

- **Todos à esquerda** de $A[s]$: $\leq A[s]$
- **Todos à direita** de $A[s]$: $\geq A[s]$

- Após a partição, $A[s]$ estará em sua posição final no array ordenado.
- Podemos então ordenar os subarrays à esquerda e à direita de $A[s]$ de forma independente (usando o mesmo método).
- **Vamos ver uma animação!!!**

Comparação com o Mergesort

- **Divisão no Mergesort:** A divisão do problema em dois subproblemas é direta, e o trabalho principal ocorre ao combinar as soluções dos subproblemas.
- **Divisão no Quicksort:** Todo o trabalho é realizado na etapa de divisão (partição), e não é necessário trabalho adicional para combinar as soluções dos subproblemas.
- **Resumo:** No Mergesort, o trabalho ocorre na fase de combinação, enquanto no Quicksort, o esforço se concentra na fase de divisão.

Algoritmo: Quicksort

Algorithm Quicksort

1: **Entrada:** Um subarray $A[l..r]$ de um array $A[0..n-1]$, definido pelos índices l e r
2: **Saída:** Subarray $A[l..r]$ ordenado em ordem crescente
3: **if** $l < r$ **then** $O(1)$
4: $s \leftarrow$ **Partition**$(A[l..r])$ $O(n)$ no pior caso
5: **Quicksort**$(A[l..s-1])$ $T(s-l)$
6: **Quicksort**$(A[s+1..r])$ $T(r-s)$
7: **end if**

Análise de Custo do Algoritmo

- **Linha 3:** Condicional $l < r$, com custo constante $O(1)$.
- **Linha 4:** Função de partição (Partition), que organiza o subarray $A[l..r]$ em torno de um pivô e tem custo $O(n)$ no pior caso.
- **Linhas 5 e 6:** Chamadas recursivas para os subarrays à esquerda ($T(s-l)$) e à direita ($T(r-s)$) do pivô $A[s]$.

Complexidade Temporal Total

- A complexidade de Quicksort depende da escolha do pivô em cada chamada recursiva:
  - **Melhor caso:** O pivô divide o array de forma balanceada, levando a uma recorrência de $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$, que resolve para $O(n \log n)$.
  - **Pior caso:** O pivô é o menor ou maior elemento em cada partição, resultando em uma recorrência de $T(n) = T(n-1) + O(n)$, que resolve para $O(n^2)$.
- O melhor caso é alcançado quando o pivô divide o array aproximadamente ao meio em cada etapa.
- O pior caso pode ser mitigado com técnicas de seleção de pivô, como escolher o pivô aleatoriamente ou usar a mediana de três elementos.

Resumo da Análise de Custo

- **Complexidade Temporal:**
  - **Melhor Caso**: $O(n \log n)$
  - **Caso Médio**: $O(n \log n)$
  - **Pior Caso**: $O(n^2)$
- **Espaço Auxiliar:** $O(\log n)$ em média, devido à profundidade da pilha de recursão.
- Quicksort é um algoritmo eficiente e in-place[3], mas seu desempenho depende da escolha do pivô.

[3] "In-place significa que o algoritmo reorganiza os elementos no próprio array, sem a necessidade de arrays auxiliares significativos."

Comparação: Merge Sort vs Quick Sort

Algoritmo	Complexidade	Espaço Auxiliar	Estabilidade	Tipo de Ordenação
Merge Sort	$O(n \log n)$	$O(n)$	Estável	Não in-place
Quick Sort	$O(n \log n)^*$	$O(\log n)$	Não Estável	In-place

*Pior caso do Quick Sort: $O(n^2)$

Considerações

Considerações

- **Recordando!!!**
  - Merge Sort.
  - Quick Sort.
  - Exercícios.

Considerações

- **Recordando!!!**
  - Merge Sort.
  - Quick Sort.
  - Exercícios.
- **Referências utilizadas!!!**

  - Cormen, T. H. et al. Algoritmos: Teoria e prática, 3a edição. Elsevier, 2012.
  - Levitin, A. Introduction to the Design and Analysis of Algorithms, 2007. (inclusive Figuras)
  - Goodrich, M. T., Tamassia, R., Goldwasser, M. H. (2013). Data structures and algorithms in Python.