"Analysis of Algorithms"

de J.J. McConnell

Índice

[Capítulo 1: Analysis Basics 2](#_Toc151224411)

[1.1 WHAT ANALYSIS? 2](#_Toc151224412)

[1.2 WHAT TO COUNT AND CONSIDER 2](#_Toc151224413)

[**1.2.1 Casos a Considerar:** 2](#_Toc151224414)

[1.4 RATES OF GROWTH 3](#_Toc151224415)

[1.5 DIVIDE AND CONQUER ALGORITHMS 6](#_Toc151224416)

[1.5.1 Método do Torneio (Exemplificado, mas não Aplicado): 8](#_Toc151224417)

[1.5.2 Limites Inferiores: Árvores de Decisão e Complexidade Ótima 9](#_Toc151224418)

[Capítulo 2: Searching and Selection Algorithms 10](#_Toc151224419)

[2.1 Sequential Search 10](#_Toc151224420)

[2.1.1 Análise do Pior Caso 11](#_Toc151224421)

[2.1.2 Análise do Caso Médio 11](#_Toc151224422)

[2.2 Binary Search 13](#_Toc151224423)

[2.2.1 Árvore de Decisão 14](#_Toc151224424)

[2.2.2 Análise do Caso Médio 15](#_Toc151224425)

[2.3 Seleção 16](#_Toc151224426)

[Algoritmo FindKthLargest 16](#_Toc151224427)

[Algoritmo KthLargestRecursive 18](#_Toc151224428)

[Capítulo 3: Sorting Algorithms 20](#_Toc151224429)

[3.1 Ordenação por Inserção 20](#_Toc151224430)

[3.1.1 Análise do Pior Caso 22](#_Toc151224431)

[3.1.2 Análise do Caso Médio 22](#_Toc151224432)

[3.2: Bubble Sort 24](#_Toc151224433)

[3.2.1 Análise do Melhor Caso 25](#_Toc151224434)

[3.2.2 Análise do Pior Caso 25](#_Toc151224435)

[3.2.3 Análise do Caso Médio 25](#_Toc151224436)

# Capítulo 1: Analysis Basics

## WHAT ANALYSIS?

Este capítulo trata da análise de algoritmos e estruturas de dados, podendo ser realizada em termos de tempo (quantidade de operações executadas) ou espaço (quantidade de memória necessária).

## WHAT TO COUNT AND CONSIDER

A decisão do que contar em uma análise algorítmica envolve duas etapas: escolher as operações significativas e distinguir entre aqueles essenciais para o algoritmo e as que são excessivas ou de manutenção.

Existem duas classes de operações geralmente escolhidas como operações significativas: comparação e aritmética. Operadores de comparação, como igual, não igual, menor que, maior que, menor ou igual e maior ou igual, são essenciais em algoritmos de busca e ordenação.

As operações aritméticas são contadas em dois grupos: aditivas (adição, subtração, incremento e decremento) e multiplicativas (multiplicação, divisão e módulo). Multiplicações são consideradas mais demoradas que adições. Além disso, outras operações, como logaritmos e funções geométricas, podem ser mais demoradas.

### **1.2.1 Casos a Considerar:**

A escolha do input ao analisar um algoritmo pode impactar seu desempenho. Não se analisa apenas um conjunto de inputs; procura-se aqueles que permitem ao algoritmo ser mais rápido e mais lento, considerando também o desempenho médio.

* **Melhor Caso:**
  + Entrada que exige o menor tempo possível.
  + Exemplo: Em um algoritmo de busca, a chave é encontrada na primeira comparação.
* **Pior Caso:**
  + Entrada que requer o maior esforço possível.
  + Exemplo: Em um algoritmo de busca, a chave está na última posição ou não está na lista.
* **Caso Médio:**
  + Análise complexa envolvendo diferentes grupos de inputs e suas probabilidades.

## RATES OF GROWTH

Na análise de algoritmos, o número exato de operações não é crucial. Em vez disso, foca-se na taxa de aumento das operações à medida que o tamanho do problema aumenta, conhecida como a taxa de crescimento do algoritmo. O interesse reside nas tendências gerais quando os conjuntos de dados são grandes.

Ao analisar algoritmos, concentra-se na taxa geral de crescimento, não nos detalhes. Funções baseadas em x² crescem lentamente inicialmente, mas rapidamente à medida que o tamanho do problema aumenta. Funções baseadas em x crescem a uma taxa constante. Funções baseadas em log x crescem muito lentamente.

Uma imagem com texto, file, captura de ecrã, Gráfico

Descrição gerada automaticamente

Na análise de algoritmos, interessa mais a classe de crescimento do algoritmo do que o número exato de operações. As classes comuns incluem funções quadráticas, lineares, logarítmicas, etc.

**Classificação de Crescimento:**

1. **Big Omega (Ω):**
   * Representa funções que crescem pelo menos tão rápido quanto outra função (limite inferior).
   * Exemplo: Ω(n²) inclui funções que crescem pelo menos tão rápido quanto n².
2. **Big Oh (O):**
   * Representa funções que não crescem mais rápido que outra função (limite superior).
   * Exemplo: O(n²) inclui funções que crescem no máximo tão rápido quanto n².
3. **Big Theta (θ):**
   * Representa funções que crescem ao mesmo ritmo que outra função.
   * É a sobreposição de Big Omega e Big Oh.

**Encontrar Big Oh:**

* A função g(n) pertence a O(f(n)) se o limite de g(n) / f(n) for um número real menor que um valor constante para n suficientemente grande.

Nota: Referencia para + a frente, "g = O(f)" é equivalente a "g ∈ O(f)".

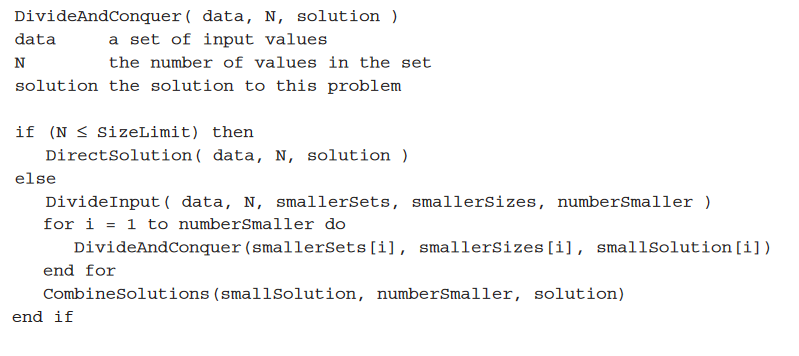
Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Uma imagem com texto, Tipo de letra, escrita à mão, captura de ecrã

Descrição gerada automaticamente

## 1.5 DIVIDE AND CONQUER ALGORITHMS



Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

1. **Exemplo: Algoritmo Fatorial Recursivo:**

Uma imagem com texto, captura de ecrã, Tipo de letra, algebra

Descrição gerada automaticamente

* **Descrição:**
  + Resolve problemas dividindo-os em subproblemas menores.
  + Utiliza chamadas recursivas para tratar subconjuntos de dados.
  + Combina as soluções dos subproblemas para formar a solução final.

**Eficiência Recursiva do Algoritmo:**

* A eficiência é determinada pelos casos diretos, divisão do input, chamadas recursivas e combinação das soluções.

Uma imagem com texto, Tipo de letra, captura de ecrã, algebra

Descrição gerada automaticamente

Uma imagem com texto, Tipo de letra, branco, file

Descrição gerada automaticamente

* + Análise do exemplo do fatorial em relação à relação de recorrência.

Uma imagem com texto, Tipo de letra, branco, tipografia

Descrição gerada automaticamente

### **1.5.1 Método do Torneio:**

* **Construção da Árvore de Decisão:**
  + Cada comparação na árvore resulta em um "vencedor" e um "perdedor".
  + A busca descendente identifica os elementos que perderam para o maior.
* **Eficiência do Método do Torneio:**
  + Destaca a estrutura equilibrada da árvore.
  + Explica como encontrar o segundo maior elemento usando o método.

Cada comparação produz um “vencedor” e um “perdedor”. Os perdedores são eliminados e apenas os vencedores sobem na árvore. Cada elemento, exceto o maior, deve “perder” uma comparação. Portanto, construir a árvore do torneio levará N - 1 comparações. O segundo maior elemento só poderia ter perdido para o maior elemento. Descemos a árvore e obtemos o conjunto de elementos que perderam para o maior.

Uma imagem com diagrama, círculo, file, design

Descrição gerada automaticamente

### **1.5.2 Limites Inferiores: Árvores de Decisão e Complexidade Ótima**

* **Árvores de Decisão em Algoritmos de Ordenação:**
  + Uso de árvores de decisão para analisar algoritmos de ordenação.
  + Cada nó interno representa uma comparação entre dois elementos.
  + o caminho mais longo representa o pior caso. O melhor caso é o caminho mais curto. O caso médio é o número total de arestas na árvore de decisão dividido pelo número de folhas na árvore.

Uma imagem com diagrama, círculo, desenho, file

Descrição gerada automaticamente

Podemos novamente usar uma árvore binária para nos ajudar a analisar o processo de ordenação de uma lista de três números. Podemos construir uma árvore binária para o processo de ordenação rotulando cada nó interno com os dois elementos da lista que seriam comparados.

* **Análise de Limites Inferiores para Algoritmos de Ordenação:**
  + Destaca a necessidade de determinar o número mínimo absoluto de operações para resolver um problema.
  + Utilização de uma árvore de decisão para analisar a ordenação de uma lista de três elementos.
  + Expressa a complexidade mínima como O(N lg N), onde N é o número de elementos na lista.

# Capítulo 2: Searching and Selection Algorithms

## 2.1 Sequential Search

* **Objetivo:** Encontrar um elemento específico (alvo) em uma lista.
* **Abordagem:** Percorre a lista elemento por elemento até encontrar uma correspondência.
* **Caso Padrão:** Lista não ordenada, pois existem algoritmos mais eficientes para listas ordenadas.
* **Retorno:** Índice do elemento encontrado; 0 se o alvo não estiver na lista.

Uma imagem com texto, captura de ecrã, Tipo de letra, algebra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

### 2.1.1 Análise do Pior Caso

* **Cenários de Pior Caso:**
  1. O alvo é o último elemento na lista.
  2. O alvo não está na lista.
* **Número de Comparações:** Sempre leva N comparações, onde N é o número de elementos na lista.

### 2.1.2 Análise do Caso Médio

* **Caso Bem-Sucedido:**
  + Probabilidade igual de o alvo estar em qualquer posição da lista.
  + Número médio de comparações é (N + 1) / 2.

Uma imagem com Tipo de letra, texto, diagrama, file

Descrição gerada automaticamente

Foto do livro

* **Incluindo Possibilidade de Falha:**
  + Número médio de comparações é (N + 1) / 2 + 1/2.

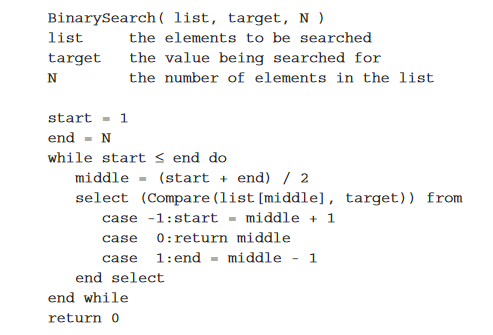
Uma imagem com texto, Tipo de letra, escrita à mão, caligrafia

Descrição gerada automaticamente

Foto do livro

## 2.2 Binary Search

* **Objetivo:** Buscar em uma lista ordenada.
* **Abordagem:** Compara o alvo com o elemento no meio e elimina metade da lista a cada iteração.
* **Loop:** Continua até encontrar o alvo ou reduzir a lista a zero.
* **Retorno:** Índice do elemento encontrado; 0 se o alvo não estiver na lista.



Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

### 2.2.1 Árvore de Decisão

* **Decisão Estruturada:** Construir uma árvore de decisão para o processo de busca.
* **Nós da Árvore:** Elementos verificados em cada passo.
* **Subárvores:** Elementos verificados quando o alvo é menor ou maior que o elemento atual.
* **Equilíbrio da Árvore:** A escolha do meio da lista garante uma árvore relativamente equilibrada.
* **Número de Comparativos:** No máximo, lg(N + 1) comparações, onde N é o número de elementos na lista.

Uma imagem com círculo, captura de ecrã, Tipo de letra, texto

Descrição gerada automaticamente

Foto do livro

### 2.2.2 Análise do Caso Médio

* **Situações Analisadas:**
  1. O alvo está sempre na lista.
  2. O alvo pode não estar na lista.
* **Probabilidades:** Caso 1 - Cada localização tem probabilidade 1/N.
* **Número Médio de Comparativos:**
  1. A análise é baseada na estrutura da árvore de decisão.
  2. Equação para o caso em que o alvo está na lista:

Uma imagem com texto, Tipo de letra, recibo, captura de ecrã

Descrição gerada automaticamente

Foto do livro

* 1. Equação para o caso em que o alvo não está na lista:

Uma imagem com Tipo de letra, escrita à mão, texto, branco

Descrição gerada automaticamente

Foto do livro

## 2.3 Seleção

* **Objetivo:**
  + Exploração de situações em que a busca é baseada em propriedades específicas, não apenas em valores.
  + Foco na identificação do maior, menor ou valor mediano, especialmente na busca pelo KK-ésimo maior valor em uma lista.

### Algoritmo FindKthLargest

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

* **Descrição:**
  + Proposta de algoritmo para encontrar o K-ésimo maior valor em uma lista.
  + Estratégia inicial: ordenar a lista em ordem decrescente e selecionar o valor na posição K.
  + Alternativa mais eficiente: seleção iterativa, movendo os maiores valores para o final da lista.
* **Complexidade:**
  + Cada passo faz N−(i+1) comparações.

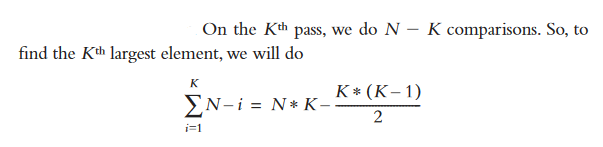
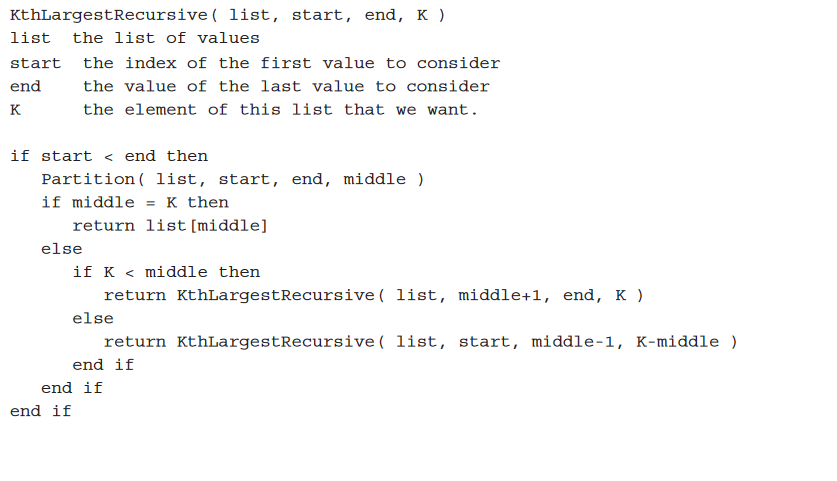


Foto do livro

* + No total, O(K⋅N)O(K⋅N) comparações.
  + Observação: Eficiente para K próximo às extremidades da lista; estratégia mais eficaz para K no meio:

### Algoritmo KthLargestRecursive



Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

* **Descrição:**
  + Algoritmo recursivo mais eficiente para encontrar o K-ésimo maior valor.
  + Utiliza o conceito de partição para dividir a lista, reduzindo a complexidade.
  + Exploração da partição média: aproximadamente 2N comparações.

#### Observações Finais

* **Eficiência:**
  + Estratégias mais rápidas para K próximo ao início ou final da lista.
  + Algoritmo recursivo eficaz para K no meio, explorando a partição média.
* **Partição:**
  + Prévia menção sobre a exploração detalhada da partição no quicksort no Capítulo 3.

# Capítulo 3: Sorting Algorithms

## 3.1 Ordenação por Inserção

* **Ideia Básica:**
  + Adição eficiente de um novo elemento a uma lista ordenada.
  + Evitar adicionar em qualquer lugar e, em seguida, reordenar a lista inteira.
  + Considera a primeira posição da lista como uma lista ordenada de tamanho 1.

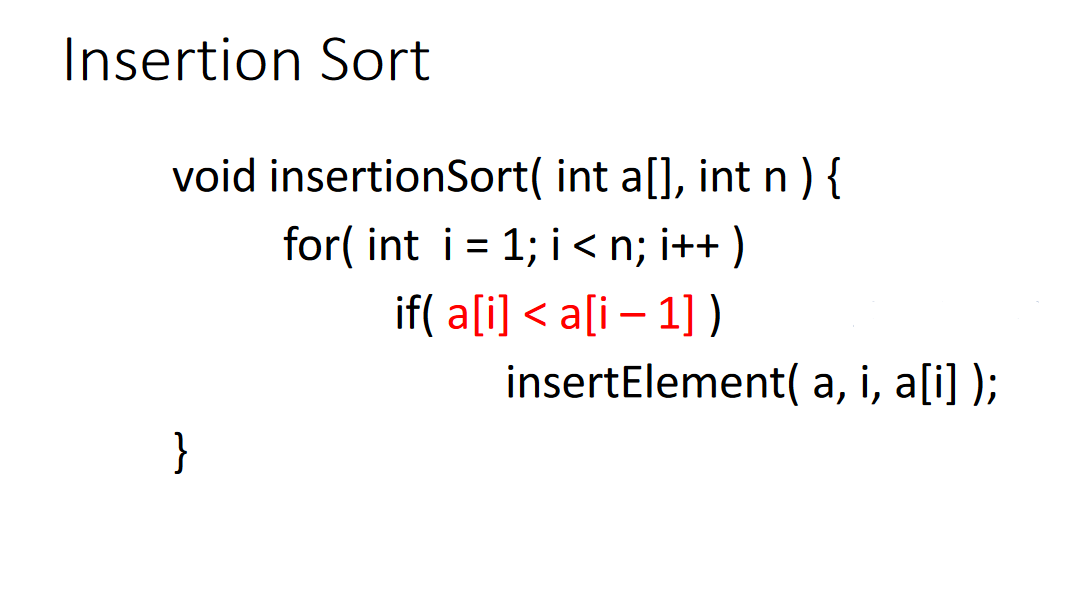
#### Algoritmo InsertionSort

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente



Talvez mais fácil de entender que a anterior.

* **Descrição:**
  + Itera sobre os elementos, inserindo cada novo elemento na posição correta.
  + O processo envolve deslocar elementos maiores para abrir espaço.
* **Complexidade:**
  + Cada iteração compara o novo elemento a, no máximo, ii elementos anteriores.
  + Pior caso: O(N^2) comparações.
  + Pior caso ocorre quando a lista está em ordem decrescente inicialmente.

### 3.1.1 Análise do Pior Caso

* **Caso Pior:**
  + Cada novo elemento adicionado é menor que todos os elementos já ordenados.
  + O loop interno executa o máximo de trabalho quando o novo elemento é adicionado no início da lista.
* **Complexidade no Pior Caso:**
  + Soma das comparações: N+(N−1)+(N−2)+…+1N+(N−1)+(N−2)+…+1.

Uma imagem com texto, Tipo de letra, file, número

Descrição gerada automaticamente

Foto do livro

### 3.1.2 Análise do Caso Médio

A análise do caso médio é realizada em duas etapas. Primeiro, determinamos o número médio de comparações necessário para posicionar um elemento. Em seguida, usamos esse resultado para calcular o número médio total de operações.

* **Número Médio de Comparações para o i-ésimo Elemento:**
  + Adicionando o i-ésimo elemento à parte ordenada, leva no máximo i comparações.
  + O pior caso ocorre quando o novo elemento é menor que todos os elementos já ordenados.
  + Em média, há i+1 posições possíveis para o i-ésimo elemento.

Uma imagem com Tipo de letra, tipografia, design

Descrição gerada automaticamente

Foto do chat 1.1, talvez duvidoso. Há no livro, mas é mais confuso.

* **Número Médio Total de Operações:**
* Somamos os resultados para todos os elementos de 1 a N−1.

Uma imagem com texto

Descrição gerada automaticamente

Foto do chat1.2, talvez duvidoso. Há no livro, mas é mais confuso.

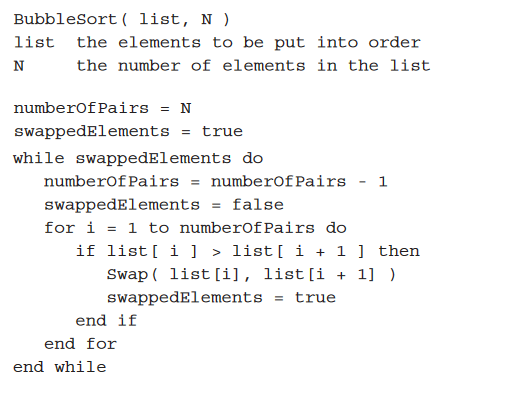
* Isso é equivalente a O(N^2), mas a análise detalhada mostra uma constante menor.
* **Simplificação:**

Uma imagem com texto, Tipo de letra, tipografia, design

Descrição gerada automaticamente

Foto do chat1.3, talvez duvidoso. Há no livro, mas é mais confuso.

## 3.2: Bubble Sort



Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

O algoritmo Bubble Sort faz várias passagens pela lista de elementos, comparando valores adjacentes e trocando-os se estiverem fora de ordem. A cada passagem, os valores mais altos se movem para o final da lista, enquanto os menores se movem para o início. O processo continua até que nenhum elemento seja trocado em uma passagem, indicando que a lista está ordenada.

### 3.2.1 Análise do Melhor Caso

O melhor caso ocorre quando a lista já está ordenada. Nesse caso, o algoritmo faz N−1N−1 comparações na primeira passagem. As comparações diminuem em cada passagem subsequente, e se nenhuma troca ocorrer em uma passagem, o algoritmo para. Assim, o melhor caso é N−1 comparações.

### 3.2.2 Análise do Pior Caso

O pior caso ocorre quando a lista está em ordem reversa. Cada passagem move o maior elemento para a posição correta, mas o próximo maior elemento é movido para a segunda posição, e assim por diante. Isso resulta em N(N−1)/ 2 comparações no pior caso.

Uma imagem com texto, Tipo de letra, escrita à mão, file

Descrição gerada automaticamente

Foto do livro

### 3.2.3 Análise do Caso Médio

Para o caso médio, consideramos que é igualmente provável que em cada passagem não haja trocas. O número médio de comparações é dado por uma série, resultando em O(N^2)), mas com uma constante menor do que o pior caso.

Uma imagem com Tipo de letra, branco, diagrama, texto

Descrição gerada automaticamente

Uma imagem com Tipo de letra, texto, file, escrita à mão

Descrição gerada automaticamente

Uma imagem com Tipo de letra, file, diagrama, branco

Descrição gerada automaticamente

Uma imagem com Tipo de letra, tipografia, escrita à mão, texto

Descrição gerada automaticamente

Uma imagem com Tipo de letra, file, número, texto

Descrição gerada automaticamente

Uma imagem com texto, Tipo de letra, escrita à mão, número

Descrição gerada automaticamente

Foto do livro que só um DEUS percebe.