UNIFESP

Programa de Pós Graduação em Ciência da Computação

Campus São José dos Campos

Disciplina:

Análise de Algoritmos e Estrutura de Dados

Discentes:

Wagner Lopes Cardozo

Docente Orientadora:

DRa. Lilian Berton

UNIFESP

Tema da Pesquisa:

Estudo das Características, Complexidade Assintótica Big (O), Tempo Usado e Memória para Execução de Alguns Métodos Usados em Algoritmos de Ordenação

Sumário:

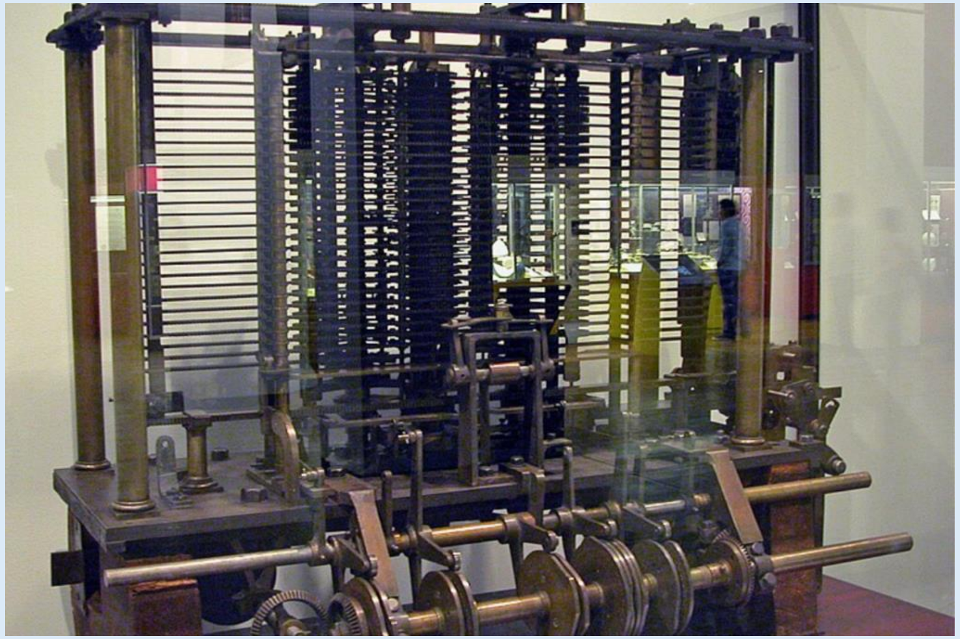
Quando tudo começou:

- Ada Lovelace, matemática e escritora

- Tinha a filosofia que a metafísica era tão importante quanto a matemática

- Sendo essencial para investigar “mundos invisíveis ao nosso redor”

- Trabalhou com Charles Babbage no Projeto da Máquina Analitica



Máquina analítica proposta por Babbage, London Science Museum

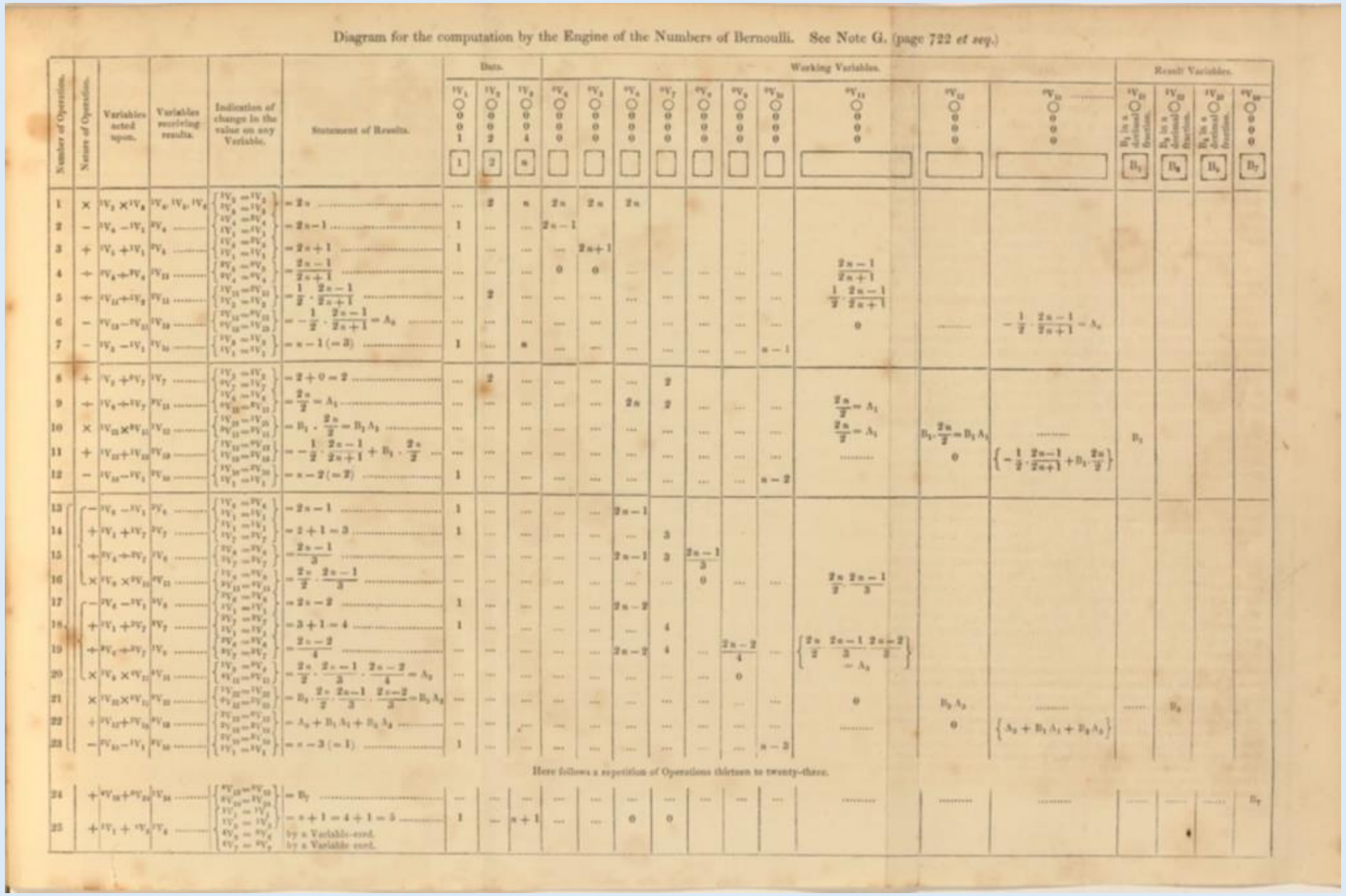
Fonte Fig.: Espaço do Conhecimento UFMG, Ada Lovelace: A Primeira Programadaora da História, 11 julho 2023

UNIFESP

Quando tudo começou:

- Escrever um algoritmo para que a máquina pudesse computar a Sequência de Bernoulli

- Que foi posteriormente reconhecido pela academia científica como sendo o primeiro programa de computador da história



Nota G contendo o primeiro algoritmo de computador da história. Fonte: Note G © Magdalen College Libraries and Archives, Daubeny 90.A.11

Fonte Fig.: Espaço do Conhecimento UFMG, Ada Lovelace: A Primeira Programadaora da História, 11 julho 2023

Matemática para Explicar os Algoritmos

“Tudo o que é construído dentro da programação tem um grande background matemático” (TAPES, G. 2022)

Mas por que é importante entender esses conceitos matemáticos por trás da programação?

“Programar tem muito mais a ver com matemática” (TAPES, G. 2022)

“Com a matemática aprendemos lógica, que é a base fundamental para algorítmos e por sua vez para a programação’ (TAPES, G. 2022)

“Um programador que não sabe lógica não passa de uma máquina de escrever código, sem nenhum critério” (TAPES, G. 2022)

UNIFESP

Métodos Matemáticos Para Explicar os Algoritmos

Método da Substituição

- É usado para provar a complexidade de recorrências de algoritmos recursivos. Aqui, você supõe uma solução T(n), substitui na recorrência e verifica se ela satisfaz a equação. Por exemplo, para um algoritmo como Merge Sort, cuja recorrência é:

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

- Pode assumir que T(n) = O (n Log n), substituir esta suposição na equação, e verificar se a equação é verdadeira.

Métodos Matemáticos Para Explicar os Algoritmos

Árvore de Recorrência

- É usado para demostar a complexidade de algoritmos recursivos ao visualizar a quantidade de processamento realizado em cada nível da recursão.
    - ✓ Nível 1 – O (n).
    - ✓ Nível 2 – cada subproblema é de tamanho (n/2), e há dois subproblemas, então o processamento é 2 x O (n/2) = O (n).
    - ✓ Em geral, há Log n níveis, cada um realizando O (n) processamento, então o total é O (n Log n).

Métodos Matemáticos Para Explicar os Algoritmos

Método da Interação

- É similar à árvore de recorrência, mas resolve a equação passo a passo sem a representação visual da árvore.

- Dado T (n) = T (n/2) + O (n), e possível iterar a equação substituindo recursivamente:

$$T(n) = T(n/2) + O(n)T(n/2) = T(n/4) + O(n/2)$$

- Somando, chega-se a O (n Log n).

Métodos Matemáticos Para Explicar os Algoritmos

Teorema Mestre

- Fornece uma fórmula para resolver recorrências do tipo:

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

- O método do Teorema Mestre possui três casos dependendo do valor relativo de a, b e d:

Se $a > b^d$, então $T(n) = O(n^{\log_b a})$.

Se $a = b^d$, então $T(n) = O(n^d \log n)$.

Se $a < b^d$, então $T(n) = O(n^d)$.

Métodos Matemáticos Para Explicar os Algoritmos

Simplificação de Polinômio

- É um método onde desprezamos termos de ordem inferior, porque eles têm um impacto menos importante do que termos de ordem superior para entradas muito grandes no algoritmo. Assim, por exemplo o polinômio $T(n) = 5n^2 + 3n + 100$, o termo dominante é n^2, então $T(n) = O(n^2)$

Métodos Matemáticos Para Explicar os Algoritmos

Análise de Limites Superiores e Inferiores

- Limites ajudam a definir limites superiores e inferiores para a complexidade de um algoritmo, frequentemente utilizando cálculo de limites.
  - ✓ Define-se uma função que descreve o número de operações do algoritmo.
  - ✓ Calcula-se o limite da função à medida que $n \to \infty$, e usa-se para determinar o comportamento assintótico.
  - ✓ Limite superio -

UNIFESP

Métodos Matemáticos Para Explicar os Algoritmos

Análise de Limites Superiores e Inferiores

- **Limite superio Big (O)** - Representa um **limite superior assintótico** para o tempo de execução de um algoritmo. Fornece uma garantia de que o algoritmo **não será mais lento** do que um certo limite, para entradas suficientemente grandes.

- **Limite inferio Big (Ω)** - Descreve um **limite inferior assintótico** para o tempo de execução de um algoritmo. Dá uma garantia de que o algoritmo **não será mais rápido** do que um certo limite, ou seja, ele precisará de **pelo menos** Ω (f (n)) tempo para ser executado em entradas suficientemente grandes.

- **Limite Exato Big (Θ)** - O limite superior quanto o limite inferior iguais para a mesma função f (n), podemos dizer que o algoritmo tem complexidade Θ (f (n)), o que significa que o tempo de execução **cresce precisamente** com a função f (n), independentemente do caso analisado (melhor ou pior caso).

UNIFESP

Matemática para Explicar os Algoritmos

Divisão Assintótica dos Algoritmos

- ✓ **Linear O (n) -** Analisar o número de operações simples executadas em sequência
- ✓ **Constante O (1) -** Verificar se o algoritmo realiza um número fixo de operações independente de n
- ✓ **Quadrático O (n^2) -** Contagem de pares de operações aninhadas, como em laços duplos
- ✓ **Logaritmico O (Log n) -** Analisar algoritmos que dividem o problema repetidamente por metade
- ✓ **Exponencial O (2^n) -** Expansão da árvore de recursão que gera 2^n subproblemas
- ✓ **Fatorial O (n!) -** Contagem direta de permutações ou combinações possíveis

Análise de alguns algoritmos

Comb Sort

- ✓ É uma melhoria do Bubble Sort, com a diferença de usar um gap que diminui progressivamente

- ✓ Utiliza um fator de encolhimento (geralmente 1.3) para comparar elementos distantes e, progressivamente, encurta o gap entre eles

- ✓ **Melhor caso –** O (n Log n)
- ✓ **Pior Caso –** O (n^2)
- ✓ **Caso Médio –** O (n^2)

- ✓ **Exemplo de Aplicação -** Útil em situações onde o Bubble Sort pode ser aplicado, mas com listas um pouco maiores e quando busca-se uma solução simples, porém mais eficiente.

UNIFESP

Análise de alguns algoritmos

Cocktail Sort

- ✓ Variante do Bubble Sort que faz a varredura bidirecional (da esquerda para a direita e de volta). Isso melhora a eficiência em listas que podem ter muitos elementos mal posicionados tanto no início quanto no final.
- ✓ Usar a análise iterativa onde cada iteração tem dois ciclos de comparação, o que leva ao comportamento quadrático no pior caso

- ✓ **Melhor caso –** O (n)
- ✓ **Pior Caso –** O (n^2)
- ✓ **Caso Médio –** O (n^2)

- ✓ **Exemplo de Aplicação -** Quando a lista possui elementos desordenados de forma dispersa, podendo melhorar a performance em relação ao Bubble Sort comum.

Análise de alguns algoritmos

Tim Sort (Nativo do Python)

- ✓ Combina o Merge Sort e o Insertion Sort. Funciona dividindo a lista em segmentos, aplicando o Merge Sort para combinar essas sublistas.

- ✓ É uma abordagem baseada em divisão e conquista

- ✓ **Melhor caso –** O (n)
- ✓ **Pior Caso –** O (n Log n)
- ✓ **Caso Médio –** O (n Log n)

- ✓ **Exemplo de Aplicação -** Ordenação de listas grandes e quase ordenadas.

UNIFESP

Análise de alguns algoritmos

Tim Sort (Implementado Manualmente)

- ✓ Para uma implementação manual do Tim Sort, as mesmas técnicas aplicadas à versão nativa podem ser usadas, considerando as otimizações específicas implementadas.

- ✓ **Melhor caso –** O (n Log n)
- ✓ **Pior Caso –** O (n)
- ✓ **Caso Médio –** O (n)

- ✓ **Exemplo de Aplicação -** Utilizado quando há necessidade de controle personalizado do Tim Sort, geralmente em sistemas que não suportam a versão nativa.

UNIFESP

Análise de alguns algoritmos

Twist Sort

- ✓ Algoritmo experimental que tenta combinar a eficiência de ordenações rápidas, como Quick Sort, com o processamento bidirecional de Cocktail Sort.

- ✓ A ideia principal é manter os benefícios de uma boa complexidade média.

- ✓ **Melhor caso –** O (n Log n)
- ✓ **Pior Caso –** O (n Log n)
- ✓ **Caso Médio –** O (n Log n)

- ✓ **Exemplo de Aplicação -** Pouco utilizado na prática, geralmente em experimentos acadêmicos sobre ordenações híbridas.

UNIFESP

Análise de alguns algoritmos

Smooth Sort

- ✓ Um algoritmo de ordenação comparativa derivado do Heap Sort, mas que usa uma estrutura de árvores.

- ✓ Tem a vantagem de ser quase tão eficiente quanto o Heap Sort, mas otimizado para listas quase ordenadas.

- ✓ **Melhor caso –** O (n)
- ✓ **Pior Caso –** O (n Log n)
- ✓ **Caso Médio –** O (n Log n)

- ✓ **Exemplo de Aplicação -** Ideal para ordenar listas que já estão quase ordenadas, semelhante ao Tim Sort.

Análise de alguns algoritmos

Cartesian Tree Sort

- ✓ Usa uma árvore cartesiana (uma árvore binária de busca que mantém a propriedade de heap) para realizar a ordenação. O tempo de inserção de cada elemento na árvore dita a complexidade do algoritmo.

- ✓ A análise do tempo de construção e a análise da profundidade da árvore podem ser resolvidas com substituição de recorrência

- ✓ **Melhor caso –** O (n Log n)
- ✓ **Pior Caso –** O (n Log n)
- ✓ **Caso Médio –** O (n Log n)

- ✓ **Exemplo de Aplicação -** Para cenários onde a estrutura de árvore binária é preferida, como em cálculos geométricos ou problemas de otimização.

Análise de alguns algoritmos

Tournament Sort

- ✓ Algoritmo de ordenação baseado em torneio, em que os elementos competem dois a dois, formando uma árvore de vencedores e perdedores. É eficiente para manter o menor elemento no topo.

- ✓ É um método de árvores de recorrência é adequado para capturar o comportamento da comparação entre pares e a subsequente atualização da árvore.

- ✓ **Melhor caso –** O (n Log n)
- ✓ **Pior Caso –** O (n Log n)
- ✓ **Caso Médio –** O (n Log n)

- ✓ **Exemplo de Aplicação -** Aplicável em sistemas de competição ou jogos, onde o vencedor de cada comparação precisa ser determinado de forma eficiente.

UNIFESP

Análise de alguns algoritmos

Pancake Sort

- ✓ Ordena a lista usando operações de "flip" (inversão de sub-listas). A ideia é levar o maior elemento para o início da lista e depois realizar um flip para posicioná-lo no final.

- ✓ A análise do tempo de construção e a análise da profundidade da árvore podem ser resolvidas com substituição de recorrência

- ✓ **Melhor caso –** O (n)
- ✓ **Pior Caso –** O (n^2)
- ✓ **Caso Médio –** O (n^2)

- ✓ **Exemplo de Aplicação -** Para cenários fazendo uso de conceitos envolvendo organização de pilhas ou filas.

UNIFESP

Resultados

Comparação Com Algoritmos Clássicos

✓ Comb Sort

Tabela de Resultados:

Algoritmo	Complexidade Assintótica	Tempo Crescente (s)	Memória Crescente (bytes)	Tempo Decrescente (s)	Memória Decrescente (bytes)
Bubble Sort	O(n^2)	11.3979	0	15.2149	0
Selection Sort	O(n^2)	5.6764	0	4.6656	0
Comb Sort	O(n log n)	0.0497	0	0.0392	0

✓ Cocktal Sort

Tabela de Resultados:

Algoritmo	Complexidade Assintótica	Tempo Crescente (s)	Memória Crescente (bytes)	Tempo Decrescente (s)	Memória Decrescente (bytes)
Bubble Sort	O(n^2)	11.4522	0	15.1466	0
Insertion Sort	O(n^2)	4.5859	0	10.4734	0
Cocktail Sort	O(n)	9.868	0	15.2323	0

UNIFESP

Resultados

Comparação Com Algoritmos Clássicos

- ✓ **Tim Sort (Nativo do Python)**

Tabela de Resultados:

Algoritmo	Complexidade Assintótica	Tempo Crescente (s)	Memória Crescente (bytes)	Tempo Decrescente (s)	Memória Decrescente (bytes)
Tim Sort (Nativo)	O(n log n)	0.0044	0	0.0021	0
Merge Sort	O(n log n)	0.0468	0	0.0407	0
Heap Sort	O(n log n)	0.007	0	0.0065	0

- ✓ **Tim Sort (Implementado Manualmente)**

Tabela de Resultados:

Algoritmo	Complexidade Assintótica	Tempo Crescente (s)	Memória Crescente (bytes)	Tempo Decrescente (s)	Memória Decrescente (bytes)
Tim Sort (Implementado Manualmente)	O(n log n)	0.0512	0	0.0494	0
Merge Sort	O(n log n)	0.0456	0	0.0394	0
Heap Sort	O(n log n)	0.0053	0	0.0048	0

UNIFESP

Resultados

Comparação Com Algoritmos Clássicos

- ✓ **Cartesian Tree Sort**

Tabela de Resultados:

Algoritmo	Complexidade Assintótica	Tempo Crescente (s)	Memória Crescente (bytes)	Tempo Decrescente (s)	Memória Decrescente (bytes)
Cartesian Tree Sort	O(n log n)	0.1533	0	0.0779	0
Binary Search Tree Sort	O(n log n)	0.064	0	5.475	0

- ✓ **Pancake Sort**

Tabela de Resultados:

Algoritmo	Complexidade Assintótica	Tempo Crescente (s)	Memória Crescente (bytes)	Tempo Decrescente (s)	Memória Decrescente (bytes)
Pancake Sort	$O(n^2)$	2.8504	0	2.3725	0
Selection Sort	$O(n^2)$	4.5192	0	5.9743	0
Bubble Sort	$O(n^2)$	11.3247	0	14.915	0

Resultado Geral

Tabela Resumo:

Método	Tempo Crescente	Memória Crescente	Tempo Decrescente	Memória Decrescente
Comb Sort	0.388532 s	0 bytes	0.712541 s	811008 bytes
Cocktail Sort	260.438305 s	0 bytes	258.852205 s	540672 bytes
Tim Sort Nativo	0.020411 s	270336 bytes	0.018245 s	270336 bytes
Tim Sort Manual	0.586876 s	0 bytes	0.579912 s	0 bytes
Twist Sort	0.304899 s	0 bytes	0.308404 s	0 bytes
Smooth Sort	0.356037 s	0 bytes	0.355154 s	0 bytes
Cartesian Tree Sort	0.502270 s	6758400 bytes	0.512758 s	540672 bytes
Tournament Sort	1063.671742 s	1622016 bytes	1059.243228 s	0 bytes
Topological Sort	32.356951 s	319488 bytes	31.948269 s	589824 bytes
Sorting Network	0.898903 s	0 bytes	0.887440 s	0 bytes
Batcher Odd-Even Merge Sort	0.278069 s	0 bytes	0.279546 s	0 bytes

Conclusões

- A análise assintótica Big O é uma ferramenta crucial para avaliar e otimizar o desempenho de algoritmos. Através da análise dos limites superiores da complexidade de tempo e espaço, é possível identificar algoritmos que oferecem melhor eficiência e escalabilidade.

  - ✓ **Ganho de Desempenho**: A aplicação da análise assintótica permite a seleção de algoritmos que minimizam o tempo de processamento.

  - ✓ **Otimização de Recursos**: A análise não apenas melhora a eficiência temporal, mas também otimiza o uso de memória.

  - ✓ **Impacto no Sistema**: A escolha de algoritmos eficientes melhora a escalabilidade e o tempo de resposta dos sistemas. Isso é crucial para aplicações em tempo real e sistemas críticos, onde o desempenho pode impactar significativamente a operação geral.

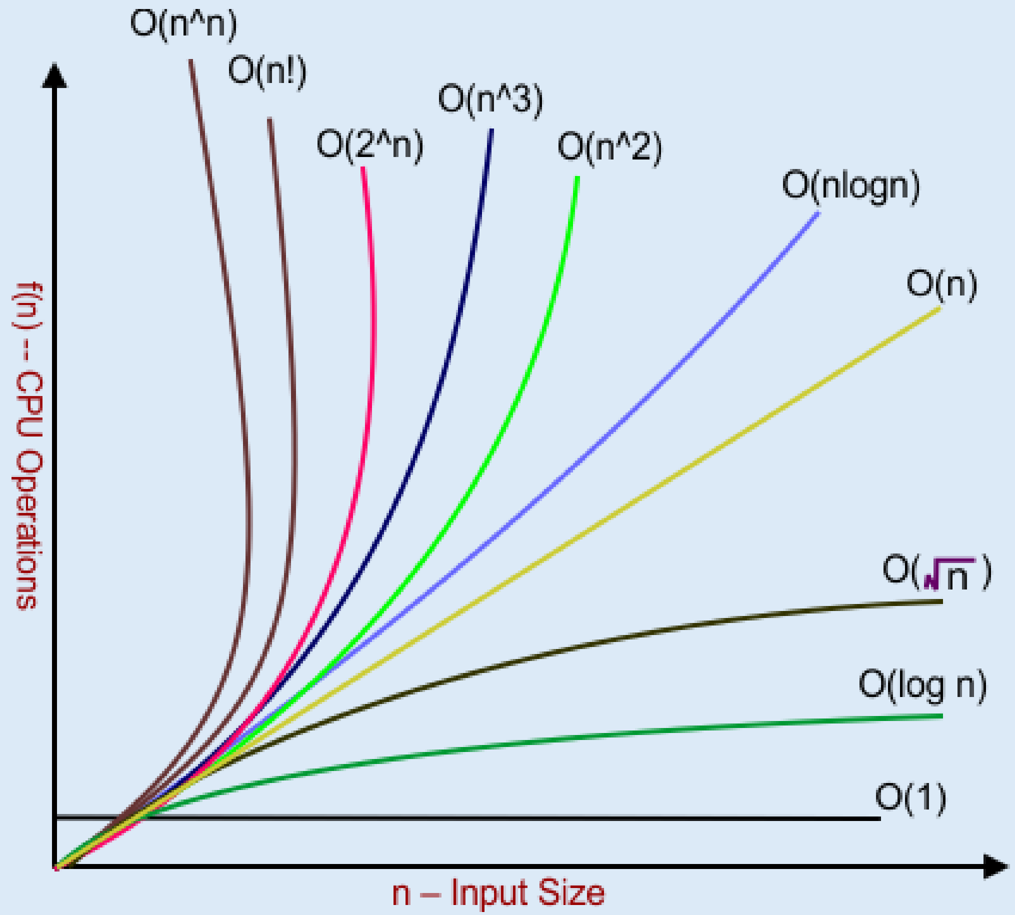
UNIFESP

Conclusões

- Portanto, a análise assintótica Big O não apenas fornece uma compreensão teórica do desempenho dos algoritmos;

- Mas também orienta a prática de desenvolvimento de software;

- Levando a sistemas mais rápidos e com uso mais eficiente dos recursos.



Fonte Fig.: CS Taleem, https://cstaleem.com/time-complexity-of-algorithms

UNIFESP

Referências

- CORMEM, T. H., LEISERSON C. E., RIVEST R. L., STEIN C., **Algoritmos Teoria e Prática** – Gen LTC, 3ª Edição, 2012, ISBN-13: 978-8535236996

- ZIVIANI, N., **Projetos de Algoritmos com Implementação e Java e C++,** Cengage Learning, 2006
- Espaço do Conhecimento UFMG**, Ada Lovelace: A Primeira Programadaora da História**, 11 julho 2023, https://www.ufmg.br/espacodoconhecimento/ada-lovelace-a-primeira-programadora-da-historia/
- TAPES G., **Por que a matemática é essencial para a programação?,** https://www.tabnews.com.br/gabrielTapes/por-que-a-matematica-e-essencial-para-a-programacao
- FEOFILOFF, P. **Minicurso de Análise de Algoritmos**, 2010, http://www.ime.usp.br/~pf/livrinho-AA/
- GOODRICH M. T., TAMASSIA R., GOLDWASSER M. H., **Data Structures and Algorithms in Python**, 2013

UNIFESP

Referências

- Growing with the web, **Comb Sort**, https://www.growingwiththeweb.com/2016/09/comb-sort.html

- Growing with the web, **Cocktail sort**, https://www.growingwiththeweb.com/2016/04/cocktail-sort.html

- Geeksforgeeks, **Tim Sort**, https://www.geeksforgeeks.org/timsort/

- Técnicas de Ordenação, **Tim Sort Nativo Python**, https://docs.python.org/pt-br/dev/howto/sorting.html

- Geeksforgeeks, **Smooth Sort**, https://www.geeksforgeeks.org/introduction-to-smooth-sort/?ref=header_outind

- Geeksforgeeks, **Cartesian Tree Sort**, https://www.geeksforgeeks.org/cartesian-tree-sorting/

- OI Wiki competitive programming, **Tournament Sort**, https://en.oi-wiki.org/basic/tournament-sort/

- Geeks for geeks, **Pancake Sort**, https://www.geeksforgeeks.org/pancake-sorting-in-python/

Muito Obrigado!!!

Duvidas, Perguntas ou Questionamentos

E-mail: wagner.cardozo72@gmail.com

LinkedIn: www.linkedin.com/in/wagner-lopes-cardozo-8b4a031ab