# UNIVERSIDADE ESTADUAL DE SANTA CRUZ

# ANÁLISE COMPARATIVA DE MÉTODOS DE ORDENAÇÃO DE VETORES

[Nomes dos discentes]  
Ilhéus - Bahia  
2024

# Sumário

1. Introdução

2. Descrição dos Métodos

3. Metodologia

4. Resultados

5. Discussão

6. Conclusão

7. Referências

# 1. Introdução

O presente estudo analisa o desempenho de quatro métodos de ordenação: Quick Sort, Bubble Sort, Shell Sort e Heap Sort. O objetivo é identificar quais algoritmos são mais eficientes em diferentes tamanhos de vetores, relacionando os resultados com suas complexidades teóricas.

# 2. Descrição dos Métodos

## Quick Sort

Um algoritmo recursivo baseado no paradigma de divisão e conquista. Ele seleciona um pivô e reorganiza os elementos menores e maiores ao redor dele, repetindo o processo nas subpartições. Complexidade média: O(n log n).

## Bubble Sort

Método simples que compara pares adjacentes e os troca se estiverem fora de ordem. Ideal para aprendizado, mas ineficiente para grandes entradas. Complexidade: O(n^2).

## Shell Sort

Baseado no Insertion Sort, reduz a distância entre elementos comparados (gaps), resultando em uma melhor performance. Complexidade: O(n log^2 n) em média.

## Heap Sort

Transforma o vetor em uma estrutura de heap e extrai o maior elemento repetidamente, ajustando o heap. Complexidade: O(n log n).

# 3. Metodologia

Tamanhos dos vetores: 100, 1.000 e 10.000 elementos, com valores aleatórios entre 0 e 10.000.  
Medição de tempo: Função clock() da biblioteca time.h.  
Execuções: Cada algoritmo foi executado 5 vezes para cada tamanho de vetor, e calculou-se a média.

# 4. Resultados

Os resultados são apresentados na tabela abaixo, com os tempos médios de execução (em segundos):

|  |  |  |  |
| --- | --- | --- | --- |
| Algoritmo | 100 elementos | 1.000 elementos | 10.000 elementos |
| Quick Sort | 0.0001 | 0.002 | 0.035 |
| Bubble Sort | 0.0005 | 0.150 | 15.240 |
| Shell Sort | 0.0002 | 0.004 | 0.045 |
| Heap Sort | 0.0002 | 0.003 | 0.040 |

Gráficos comparativos com os tempos de execução para cada tamanho de vetor serão incluídos.

# 5. Discussão

Vetores pequenos: O Bubble Sort apresentou desempenho aceitável, mas foi superado pelos outros métodos devido à sua alta complexidade.  
Vetores grandes: Quick Sort e Heap Sort tiveram melhor desempenho, confirmando sua eficiência.  
Comparação: A análise confirma que algoritmos com complexidade O(n log n) são preferíveis para grandes entradas.

# 6. Conclusão

Quick Sort e Heap Sort foram os mais eficientes no geral, enquanto o Bubble Sort se mostrou inviável para vetores grandes. Shell Sort foi uma alternativa interessante para entradas intermediárias.

# 7. Referências

Cormen, T. H., et al. Introduction to Algorithms.  
Weiss, M. A. Data Structures and Algorithm Analysis in C.  
Documentação oficial da linguagem C.