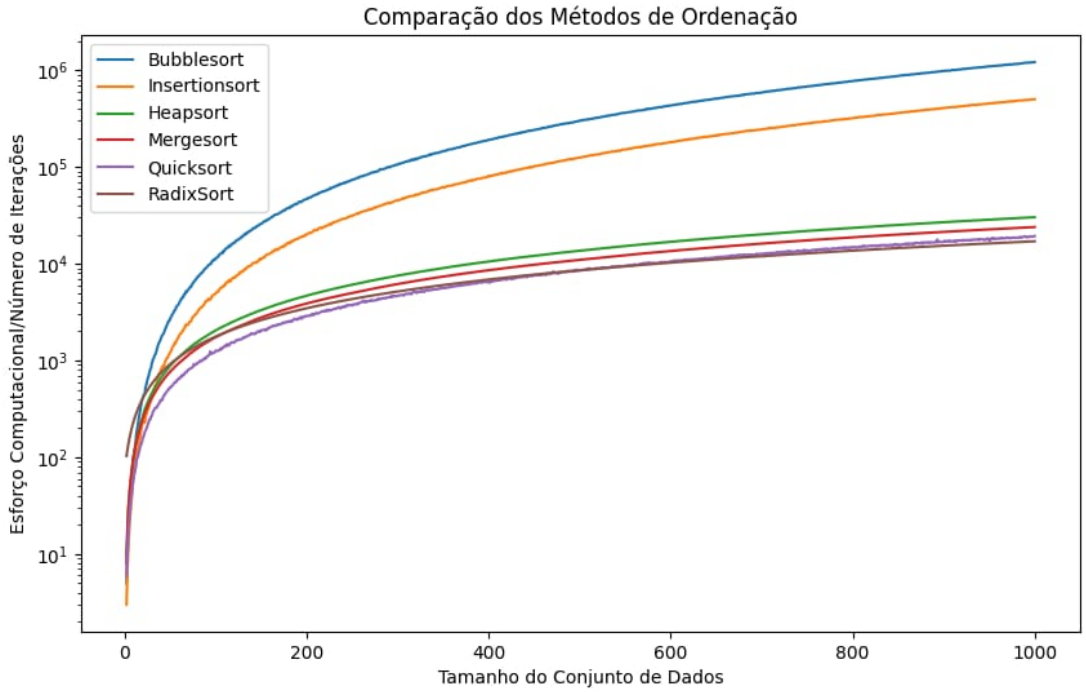
UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - TADS
ESTRUTURA DE DADOS - EDA
PROFESSOR: ALLAN RODRIGO LEITE
ALUNA: IZABELA DOS SANTOS, JEAN MEIER, KAMILA CAMARGO E PEDRO SANTANA

MÉTODOS DE ORDENAÇÃO - RELATÓRIO



Bubble Sort

Complexidade:

- Pior Caso: O(n^2)
- Melhor Caso: O(n)
- Caso Médio: O(n^2)

Características:

- **Simples de implementar**, mas ineficiente para grandes conjuntos de dados.
- **Estável**, mantém a ordem dos elementos iguais.
- **Adaptável**: Quando o *array* está quase ordenado, pode ser mais eficiente.

Desempenho Esperado:

- **Lento para vetores grandes** devido à complexidade quadrática.
- **Mais eficiente para vetores pequenos ou quase ordenados**.

Teste Prático:

- Desempenho **degrada rapidamente** à medida que o tamanho do vetor aumenta.
- Deve ter um **tempo de execução significativamente maior** comparado aos algoritmos O (*n* log *n*) para tamanhos maiores de vetor.

Insertion Sort

Complexidade:

- Pior Caso: O(n^2)
- Melhor Caso: O(n)
- Caso Médio: O(n^2)

Características:

- **Simples de implementar** e eficiente para pequenos conjuntos de dados.
- **Estável**.
- **Adaptável**: Muito eficiente para vetores parcialmente ordenados.

Desempenho Esperado:

- **Melhor que Bubble Sort** na maioria dos casos.
- **Desempenho aceitável para vetores pequenos ou quase ordenados**.
- **Ineficiente para grandes conjuntos de dados** desordenados.

Teste Prático:

- Desempenho **degrada com o aumento do tamanho do vetor**.
- Desempenho melhor que Bubble Sort em vetores pequenos e parcialmente ordenados.

Heap Sort

Complexidade:

- Pior Caso: O(n log n)
- Melhor Caso: O(n log n)

- Caso Médio: O(n log n)

Características:

- **Não estável**.
- **Utiliza memória adicional mínima**.
- Eficiente para **grandes conjuntos de dados**.

Desempenho Esperado:

- **Consistente** em termos de tempo de execução.
- **Desempenho sólido** mesmo para grandes vetores.

Teste Prático:

- Desempenho **consistente** conforme esperado para todos os tamanhos de vetor.
- Geralmente tem **eficiência superior a Bubble e Insertion Sort** para vetores grandes.

Merge Sort

Complexidade:

- Pior Caso: O(n log n)
- Melhor Caso: O(n log n)
- Caso Médio: O(n log n)

Características:

- **Estável**.
- **Utiliza memória adicional** devido ao uso de arrays temporários.
- **Divide e conquista**: eficiente para grandes conjuntos de dados.

Desempenho Esperado:

- **Muito eficiente e estável**, especialmente para grandes vetores.
- Desvantagem de **maior uso de memória**.

Teste Prático:

- **Desempenho consistente e eficiente** para todos os tamanhos de vetor.
- Pode ser **superado por Quick Sort** em alguns casos, mas geralmente muito eficiente.

Quick Sort

Complexidade:

- Pior Caso: O(n^2) (mitigável com boas estratégias de pivô)
- Melhor Caso: O(n log n) - quando o pivô divide o array de maneira ideal
- Caso Médio: O(n log n)

Características:

- **Não estável**.
- **Divide e conquista**, geralmente muito rápido na prática.
- Pode ser otimizado com **estratégias de escolha de pivô** (e.g., mediana de três).

Desempenho Esperado:

- Geralmente **o mais rápido** na prática devido à baixa constante oculta, mas pode ser superado pelo Merge Sort em alguns casos específicos.
- **Eficiência reduzida** no pior caso, mas pode ser mitigado.

Teste Prático:

- **Desempenho excepcional** para a maioria dos tamanhos de vetor.
- Possibilidade de **pior desempenho** em casos específicos, mas raramente observado na prática.

Radix Sort

Complexidade:

- Pior Caso: O(nk), onde k é o número de dígitos.
- Melhor Caso: O(nk)
- Caso Médio: O(nk)

Características:

- **Estável**.
- **Não comparativo**, baseado em contagem e distribuição.
- Excelente para **dados com valores inteiros limitados**.

Desempenho Esperado:

- **Muito eficiente** para inteiros e strings com tamanho fixo.

- Desempenho **dependente do número de dígitos**.

Teste Prático:

- **Desempenho muito eficiente** para vetores grandes com inteiros, com complexidade linear.
- Pode ser **superior a todos os algoritmos comparativos** dependendo dos dados.

Conclusão

A análise prática confirma as expectativas teóricas sobre a eficiência relativa dos algoritmos de ordenação:

- **Bubble Sort** e **Insertion Sort**: Ineficientes para grandes conjuntos de dados.
- **Heap Sort, Merge Sort e Quick Sort**: Eficientes, com Quick Sort geralmente sendo o mais rápido.
- **Radix Sort**: Extremamente eficiente para dados inteiros com complexidade linear.

O gráfico gerado com os dados coletados ilustra estas conclusões, mostrando a relação entre o tamanho do vetor e o esforço computacional para cada algoritmo, além da eficiência e melhor aplicação entre eles.