Trabalho de Análise de Algoritmos de Ordenação

Em duplas até dia 28/10 por email (joao.otavio@unesp.br)

Nesse trabalho cada dupla deve realizar a análise comparativa de desempenho entre diferentes algoritmos de ordenação, avaliando seus tempos de execução em cenários diferentes. A análise deve ser feita com a implementação dos algoritmos utilizando as linguagens C ou Python. Além disso, a análise deve considerar o comportamento dos algoritmos sobre vetores de tamanhos crescentes, com três diferentes configurações: vetores com elementos aleatórios, já ordenados e ordenados de forma inversa (decrescente).

Algoritmos a serem implementados

Os seguintes algoritmos de ordenação devem ser implementados:

- **Bubble Sort**
- **Bucket Sort**
- **Counting Sort**
- **Radix Sort**
- **Heap Sort**
- **Insertion Sort**
- **Merge Sort**
- **Quick Sort**
- **Selection Sort**
- **Shell Sort**

Requisitos do Trabalho

1. **Implementação dos Algoritmos**:
    - Todos os algoritmos de ordenação mencionados acima devem ser implementados na linguagem C ou Python.
    - Cada algoritmo deve ser encapsulado em uma função que recebe um vetor e seu tamanho como parâmetros, e retorna o vetor ordenado.
    - Para cada algoritmo, registre também o tempo de execução usando funções apropriadas para captura de tempo em sua linguagem de escolha.
2. **Conjuntos de Teste**:
    - **Vetores Aleatórios**: Gere vetores com elementos aleatórios (apenas números inteiros) utilizando um gerador de números aleatórios.
    - **Vetores Ordenados**: Gere vetores já ordenados de forma crescente.
    - **Vetores Inversamente Ordenados**: Gere vetores ordenados de forma decrescente.

- Para cada um desses cenários, realize testes com diferentes tamanhos de vetores. Você deve utilizar vetores com 10, 100, 1000, 10000, 100000 e 1000000 elementos.

3. **Medição de Tempo**:
    - Para cada execução de algoritmo em um determinado vetor com um arranjo de elementos específico, meça o tempo de execução.
    - Execute cada algoritmo várias vezes (pelo menos 3 execuções) para minimizar variações ocasionais, e calcule a média dos tempos de execução.
4. **Análise e Comparação**:
    - Compare os tempos de execução de todos os algoritmos em cada cenário de teste (vetores aleatórios, ordenados e inversamente ordenados).
    - Construa gráficos que mostrem a relação entre o tamanho dos vetores e o tempo de execução para cada algoritmo. O eixo X deve representar o tamanho do vetor, e o eixo Y o tempo de execução em segundos.
    - Apresente pelo menos três gráficos diferentes, um para cada tipo de vetor:
        - Vetores Aleatórios
        - Vetores Ordenados
        - Vetores Inversamente Ordenados
5. **Discussão dos Resultados**:
    - Explique o comportamento observado para cada algoritmo em cada tipo de vetor. Discuta a eficiência teórica (complexidade de tempo) dos algoritmos e compare-a com os resultados empíricos.
    - Discuta quais algoritmos são mais eficientes em cenários específicos, como em vetores já ordenados ou inversamente ordenados, e quais têm melhor desempenho em vetores aleatórios.
    - Explique também eventuais diferenças entre os tempos esperados e os tempos observados, levando em conta fatores como a arquitetura do computador e a implementação dos algoritmos.

Entrega

1. **Código-Fonte**:
    - Um único arquivo contendo a função principal e as demais funções dos algoritmos implementados.
    - O código da função principal (que chama as funções dos algoritmos de ordenação) deve estar documentada, com explicações claras sobre o funcionamento de cada algoritmo.
2. **Relatório**:

- O relatório deve conter os gráficos gerados, a análise dos resultados e uma explicação sobre o comportamento de cada algoritmo.
- Aponte observações sobre a complexidade de tempo de cada algoritmo, baseando-se na teoria e nos resultados obtidos.