UNIVERSIDADE ESTADUAL DA PARAÍBA
Centro de Ciências e Tecnologia - CCT
Departamento de Computação

Comparativo dos Algoritmos Elementares de Ordenação

Disciplina: Laboratório de Estrutura de Dados

Aluna: Raquel Gomes de Vasconcelos da Silveira

Campina Grande - PB

Março - 2024.1

Sumário

1.0 Introdução

Os algoritmos de ordenação podem reduzir a complexidade de um problema. Além disso, serão categorizados neste documento usando a métrica de número de trocas de posição no array, número de comparações realizadas e tempo de execução.

No primeiro, é enumerada a quantidade de vezes que o algoritmo troca os elementos para ordenar uma entrada. No caso do Selection Sort requer o número mínimo de trocas. Em segundo, é registrado o número de vezes que o algoritmo compara elementos para ordenar uma entrada. E em terceiro, é definida a eficiência e o desempenho de cada algoritmo na ordenação de elementos.

Serão abordados neste relatório três algoritmos de ordenação comuns: Bubble Sort (ordenação por flutuação), Insertion Sort (ordenação por inserção) e Selection Sort (ordenação por seleção). Assim, usando a notação Big-O o método da bolha e o de inserção requerem no melhor caso a complexidade O(n) e o de seleção O(n^2). No entanto, no pior caso, todos possuem a comparação O(n^2) na maioria das saídas.

Insertion Sort é um algoritmo simples e eficiente quando aplicado em pequenas listas. Onde, a lista é percorrida da esquerda para a direita. Após cada comparação, os elementos da esquerda vão ficando ordenados.

Propriedades do Insertion Sort:

- **Complexidade Espacial**: O(1);
- **Complexidade Temporal:** O(n) para o melhor caso, O(n^2) para o médio e pior caso;
- **Melhor Caso:** o array já está ordenado;

Análise da Ordenação do Algoritmo Insertion Sort:

1 -) { 3, 5, 1, 2, 4}, 5 > 3 – > Não troca
2 -) {3, 5, 1, 2, 4}, 4 < 5 & 4 > 3 – > 4 Troca de posição com 5
3 -) {3, 4, 5, 2, 1}, 2 < 5 & 2 < 4 & 2 < 3 –> 2 Troca de posição com 4 liberando a posição do 3 e ocupando essa posição.
4 -) {2, 3, 4, 5, 1}, 1 < 5 & 1 < 4 & 1 < 3 & 1< 2 -> 1 Troca de posição com o 2, ordenando o vetor.
5 -) {1, 2, 3, 4, 5},

- No primeiro passo é verificado se o 5 é menor que o 3, não a troca por ser um número maior.
- Em seguida, verifica se o 4 é menor que o 5 e o 3, ele só é menor que, então o 5 e o 4 trocam de posição.
- Após a troca, é verificado se o 2 é menor que o 5, 4 e o 3, como ele é menor que 3, ocupa a posição do 4, assim a posição do 3 fica vazia e o 2 passa para essa posição.
- Por fim, é verificado se 1 é menor que o 5, 4 , 3 e o 2, como ele é menor que o 2, o 1 ocupa a primeira posição ordenando a lista.

Selection Sort seleciona o menor valor e troca para a primeira posição e após selecionar o segundo menor valor e troca para a segunda posição e assim sucessivamente.

Propriedades do Selection Sort:

- **Complexidade Espacial:** O(n);
- **Complexidade Temporal:** O(n^2) para o melhor, médio e pior caso;
- **Ordenação in loco:** Sim;
- **Estável:** Não;

Análise da Ordenação do Algoritmo Selection Sort:

- 1 -) {3, 5, 1, 2, 4}, 3 é escolhido, compara se é o menor número da sentença, o 1 é o menor, então eles trocam de posição.
- 2 -) {1, 5, 3, 2, 4}, 5 é escolhido, e o menor número da sentença é 2, então eles trocam de posição.
- 3 -) {1, 2, 3, 5 ,4}, 3 é escolhido, 5 e 4 não são menores que 3 e então o 3 permanece na mesma posição.
- 4 -) {1, 2, 3, 5, 4}, 5 é escolhido e o menor número encontrado é o 4, então eles trocam.
- 5-) {1, 2, 3, 4, 5}, vetor ordenado.

- Neste passo o primeiro número escolhido foi o 3, ele foi comparado com todos os números à sua direita e o menor número encontrado foi o 1, então os dois trocam de lugar.
- O mesmo processo do passo 1 acontece, o número escolhido foi o 5 e o menor número encontrado foi o 2.
- Não foi encontrado nenhum número menor que 3, então ele fica na mesma posição.
- O número 5 foi escolhido novamente e o único número menor que ele à sua direita é o 4, então eles trocam.
- Vetor já ordenado.

Bubble sort é o algoritmo mais simples de entender e implementar do zero e menos eficiente, inclusive sendo ineficiente para listas muito grandes. Neste algoritmo cada elemento da posição i será comparado com o elemento da posição 3. Devido a essa forma de execução, o vetor terá que ser percorrido quantas vezes for necessária, o bubble sort não percebe quando a lista está ordenada, necessitando concluir uma passagem inteira pela lista sem trocar nenhum valor para saber que a lista está ordenada. Com uma complexidade no pior caso $O(n^2)$, a ordenação por bubble sort é muito lenta em comparação com o quicksort.

Propriedades do Bubble Sort:

- **Complexidade espacial**: O(1);
- **Performance no melhor caso:** O(n);
- **Performance no caso médio:**
- **Performance no pior caso:** O (n*n)
- **Estável:** sim

Análise da Ordenação do Bubble Sort:

1-) {3, 5, 1, 2 ,4}, 3 > 5? –> Não Troca
2-) {3, 5, 1, 2 ,4}, 5 > 1? –> Troca
3-) {3, 1, 5, 2 ,4}, 5 > 2? –> Troca
4-) {3, 1, 2, 5 ,4}, 5 > 4? –> Troca
5-) {3, 1, 2, 4 ,5}, 3 > 1? –> Troca
6-) {1, 3, 2, 4 ,5}, 3 > 2? –> Troca
7-) {1, 2, 3, 4 ,5}, vetor ordenado

- É verificado se o 3 é maior que 5, por essa condição ser falsa, não há troca.
- É verificado se o 5 é maior que 1, por essa condição ser verdadeira, há uma troca.
- É verificado se o 5 é maior que 2, por essa condição ser verdadeira, há uma troca.
- É verificado se o 5 é maior que 4, por essa condição ser verdadeira, há uma troca.
- O método retorna ao início do vetor realizando os mesmos processos de comparações, isso é feito até que o vetor esteja ordenado.

2.0 Comparativos Adicionais

Na tabela abaixo, estão inseridos informativos adicionais acerca dos algoritmos estudados e analisado, bem como, sua complexidade, eficiência, estabilidade, espaço de memória, entre outros.

	Bubble Sort	Insertion Sort	Selection Sort
Descrição	Compara e troca elementos adjacentes até que a lista esteja ordenada.	Percorre a lista, inserindo cada elemento na posição correta em relação aos já ordenados.	Encontra e seleciona o menor elemento repetidamente e o posiciona corretamente.
Conjunto Totalmente Desordenado	Realiza um número considerável de trocas	Realiza um número variável de trocas, geralmente menor que o Bubble Sort	Realiza um número fixo de trocas
Conjunto Quase Ordenado	Realiza muitas trocas desnecessárias, menos eficiente	Mais eficiente, realiza menos trocas em conjuntos quase ordenados	Mais consistente, sempre realiza o mesmo número de trocas
Eficiência	Menos eficiente devido às trocas desnecessárias	Mais eficiente, especialmente em conjuntos quase ordenados	Mais consistente, realiza sempre o mesmo número de troca
Complexidade Temporal	O(n^2) no pior caso	O(n^2) no pior caso	O(n^2) no pior caso
Estabilidade	Estável	Estável	Instável
Espaço de Memória	In-place	In-place	In-place
Desempenho de Passadas	Percorre a lista, trocando elementos adjacentes	Percorre a lista, inserindo elementos na posição correta	Percorre a lista, selecionando o menor elemento restante
Desempenho de Trocas	Variável, geralmente alto	Variável, geralmente menor que o Bubble Sort	Fixo, sempre realiza um número fixo de trocas

2.0 Estudo de Caso

Neste tópico será abordado as definições dos algoritmos Insertion Sort, Selection Sort e Bubble Sort selecionados para análise.

A métrica para os testes foram: o número de trocas no array, número de comparações e tempo de execução, utilizando um vetor de tamanho 5 e a sequência {3, 5, 1, 2 ,4}.

O tipo de teste desenvolvido para encontrar o registro dessas 3 métricas foram: teste de ordem crescente, ordem decrescente e ordem aleatório. Onde, foram obtidos os resultados da tabela abaixo:

Comparativo de Desempenho do Algoritmos selecionados

Insertion Sort	Crescente	Decrescente	Aleatório
Número de Trocas	0	10	5
Número de Comparações	0	10	5
Tempo de Execução	4257 nanosegundos	3289 nanosegundos	3571 nanosegundos
Selection Sort	Crescente	Decrescente	Aleatório
Número de Trocas	5	5	5
Número de Comparações	10	10	10
Tempo de Execução	7751 nanosegundos	4520 nanosegundos	3908 nanosegundos
Bubble Sort	Crescente	Decrescente	Aleatório
Número de Trocas	0	10	5
Número de Comparações	10	10	10
Tempo de Execução	37071 nanosegundos	3660 nanosegundos	2558 nanosegundos

Tempo de Execução Médio:

Insertion Sort:

Média: (4257 + 3289 + 3571) / 3 = 3705,67 nanosegundos

Selection Sort:

Média: (7751 + 4520 + 3908) / 3 = 5393 nanosegundos

Bubble Sort:

Média: (37071 + 3660 + 2558) / 3 = 14429,67 nanosegundos

3.0 Conclusão

Por fim, é possível concluir que em termos de desempenho foi o Selection Sort, por conta de possuir a menor média de tempo de execução. O mais lento para todas 3 entradas: crescente, decrescente e aleatório foi o Bubble Sort, por por possuir a maior média em tempos de execução.
Além disso, o mais eficaz para a entrada crescente foi o Insertion Sort, pois não precisou de trocas adicionais ou comparações. Ele também é mais eficaz para quase ordenadas (crescentes), enquanto Bubble Sort é menos eficaz para entradas aleatórias.
O Bubble Sort foi o algoritmo menos eficaz para entradas aleatória, pois possui o maior número médio de comparações e trocas, tendo em vista os outros algoritmos de ordenação.