Algoritmos de Ordenação

Bubble Sort

Conceito • Implementação • Análise de Complexidade

Estrutura de Dados

3º Semestre - ADS

Bubble Sort: A Ideia Principal

Conceito Intuitivo

O Bubble Sort funciona como **bolhas de ar na água**: os elementos maiores "borbulham" para o topo (final do array) através de comparações e trocas sucessivas.

Como Funciona

1. Compara elementos adjacentes
2. Troca se estiverem na ordem errada
3. Repete até que não haja mais trocas

Analogia: Bolhas na Água

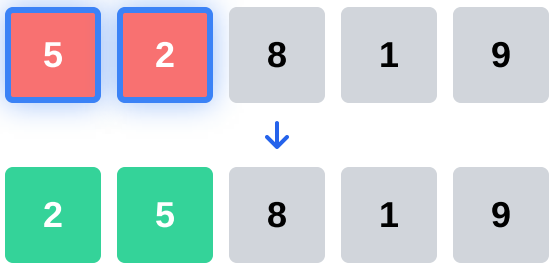
Como Funciona? (Passo a Passo)

Exemplo: Ordenando [5, 2, 8, 1, 9]

Estado Inicial:

1ª Passada:

Compara 5 > 2? Sim → Troca



Compara 5 > 8? Não → Não troca

Após 1ª Passada:

O maior elemento (9) chegou ao final!

O processo continua até que nenhuma troca seja necessária

Pseudocódigo do Bubble Sort

Algoritmo

```
ALGORITMO BubbleSort
ENTRADA: array[0..n-1], n
SAÍDA: array ordenado

PARA i = 0 ATÉ n-2 FAÇA
  PARA j = 0 ATÉ n-2-i FAÇA
    SE array[j] > array[j+1] ENTÃO
      // Troca os elementos
      temp = array[j]
      array[j] = array[j+1]
      array[j+1] = temp
    FIM SE
  FIM PARA
FIM PARA
```

Explicação das Variáveis

i — **Laço externo**
Controla o número de passadas (0 a n-2)

j — **Laço interno**
Percorre o array fazendo comparações (0 a n-2-i)

t — **Variável temporária**
Armazena valor durante a troca

Funcionamento dos Laços

Laço externo (i): Cada iteração garante que um elemento chegue à sua posição final

Laço interno (j): Compara elementos adjacentes e os troca se necessário

Otimização: A cada passada, o limite do laço interno diminui (n-2-i)

Análise de Complexidade do Bubble Sort

Análise dos Casos

Pior Caso: O(n²)

Array em ordem decrescente

```
[9, 8, 7, 6, 5] → Máximo de comparações e trocas
```

Comparações: n(n-1)/2 ≈ n²/2

Caso Médio: O(n²)

Array em ordem aleatória

```
[5, 2, 8, 1, 9] → Algumas comparações e trocas
```

Em média: n²/4 comparações

Melhor Caso: O(n)

Array já ordenado (versão otimizada)

```
[1, 2, 3, 4, 5] → Apenas comparações, sem trocas
```

Uma passada: n-1 comparações

Cálculo da Complexidade

Número de Comparações

1ª passada: (n-1) comparações

2ª passada: (n-2) comparações

3ª passada: (n-3) comparações

...

Última passada: 1 comparação

```
Total = (n-1) + (n-2) + ... + 1
      = n(n-1)/2 = O(n²)
```

Complexidade de Espaço

O(1) - Constante

Usa apenas uma variável temporária (`temp`) para as trocas, independentemente do tamanho do array.

Otimização Possível

Early Stop: Parar se nenhuma troca for feita em uma passada

```
bool swapped = false; // Flag para detectar trocas
```

Melhora o melhor caso para O(n)

Comparação e Conclusão

Comparação com Outros Algoritmos

Algoritmo	Melhor	Médio	Pior
Bubble Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)
Merge Sort	O(n log n)	O(n log n)	O(n log n)

Vantagens do Bubble Sort

- Simples de entender e implementar
- Não requer memória adicional (in-place)
- Estável (mantém ordem de elementos iguais)
- Detecta se o array já está ordenado

Desvantagens do Bubble Sort

- Muito lento para arrays grandes (O(n^2))
- Mais comparações que outros algoritmos O(n^2)
- Não é prático para uso em produção

Quando Usar?

✅ Recomendado para:

- Aprendizado e ensino de algoritmos
- Arrays muito pequenos (n < 10)
- Quando simplicidade é mais importante que eficiência
- Verificar se um array está quase ordenado

❌ Não recomendado para:

- Arrays grandes (n > 100)
- Aplicações que precisam de performance
- Sistemas em produção
- Quando há alternativas mais eficientes disponíveis

Valor Educacional

O Bubble Sort é excelente para entender conceitos fundamentais: comparações, trocas, laços aninhados e análise de complexidade.

Exercícios Propostos

Exercícios Práticos

1. Implementação Básica

Implemente o Bubble Sort e teste com diferentes arrays. Adicione prints para visualizar cada passada.

2. Versão Otimizada

Adicione uma flag para detectar quando não há mais trocas e pare a execução antecipadamente.

3. Contador de Operações

Modifique o código para contar o número de comparações e trocas realizadas.

4. Ordenação Decrescente

Adapte o algoritmo para ordenar em ordem decrescente.

Exercícios de Análise

5. Análise Experimental

Teste o algoritmo com arrays de diferentes tamanhos e meça o tempo de execução.

6. Comparação de Casos

Compare o desempenho em arrays ordenados, desordenados e inversamente ordenados.

7. Visualização

Crie uma função que imprima o estado do array após cada troca para visualizar o processo.

Próximos Passos

- Selection Sort - Algoritmo de seleção
- Insertion Sort - Algoritmo de inserção
- Merge Sort - Algoritmo de intercalação
- Quick Sort - Algoritmo de particionamento

Dica Final

Pratique implementando o algoritmo do zero várias vezes. A repetição ajuda a fixar os conceitos fundamentais!

Obrigado!

Dúvidas?