



30 Algoritmos Fundamentais em Java: Conceitos, Implementações e Aplicações Didáticas

Exploraremos 30 algoritmos fundamentais em Java, abordando desde operações básicas até algoritmos mais avançados.

Cada exemplo inclui uma explicação e um código didático para ajudar na compreensão.

1. Soma Simples

- **Descrição:** Calcula a soma de dois números.

```
public class SomaSimples {  
    public static void main(String[] args) {  
        int a = 5, b = 10;  
        System.out.println("A soma é: " + (a + b));  
    }  
}
```

2. Fatorial

- **Descrição:** Calcula o fatorial de um número usando recursão.

```
public class Fatorial {  
    public static int calcularFatorial(int n) {  
        return (n <= 1) ? 1 : n * calcularFatorial(n - 1);  
    }  
  
    public static void main(String[] args) {  
        int n = 5;  
        System.out.println("O fatorial de " + n + " é: " + calcularFatorial(n));  
    }  
}
```

3. Fibonacci Recursivo

- **Descrição:** Calcula a sequência de Fibonacci até o enésimo termo.

```
public class Fibonacci {  
    public static int calcularFibonacci(int n) {  
        if (n <= 1) return n;  
        return calcularFibonacci(n - 1) + calcularFibonacci(n - 2);  
    }  
}
```



```
public static void main(String[] args) {  
    int n = 10;  
    for (int i = 0; i < n; i++) System.out.print(calcularFibonacci(i) + " ");  
}
```

4. Bubble Sort

- **Descrição:** Ordena uma lista de números em ordem crescente.

```
public class BubbleSort {  
    public static void bubbleSort(int[] arr) {  
        int n = arr.length;  
        for (int i = 0; i < n - 1; i++) {  
            for (int j = 0; j < n - i - 1; j++) {  
                if (arr[j] > arr[j + 1]) {  
                    int temp = arr[j];  
                    arr[j] = arr[j + 1];  
                    arr[j + 1] = temp;  
                }  
            }  
        }  
    }  
  
    public static void main(String[] args) {  
        int[] arr = {64, 34, 25, 12, 22, 11, 90};  
        bubbleSort(arr);  
        for (int num : arr) System.out.print(num + " ");  
    }  
}
```

5. QuickSort

- **Descrição:** Algoritmo de ordenação rápida que utiliza o método divisão e conquista.

```
public class QuickSort {  
    public static void quickSort(int[] arr, int low, int high) {  
        if (low < high) {  
            int pi = partition(arr, low, high);  
            quickSort(arr, low, pi - 1);  
            quickSort(arr, pi + 1, high);  
        }  
    }  
  
    private static int partition(int[] arr, int low, int high) {  
        int pivot = arr[high];  
        int i = (low - 1);  
        for (int j = low; j < high; j++) {  
            if (arr[j] < pivot) {  
                i++;  
                int temp = arr[i];  
                arr[i] = arr[j];  
                arr[j] = temp;  
            }  
        }  
        int temp = arr[i + 1];
```



```
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        quickSort(arr, 0, arr.length - 1);
        for (int num : arr) System.out.print(num + " ");
    }
}
```

6. Busca Binária

- **Descrição:** Localiza um elemento em uma lista ordenada, dividindo-a repetidamente ao meio.

```
public class BuscaBinaria {
    public static int buscaBinaria(int[] arr, int x) {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == x) return mid;
            if (arr[mid] < x) low = mid + 1;
            else high = mid - 1;
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {2, 3, 4, 10, 40};
        int result = buscaBinaria(arr, 10);
        System.out.println("Elemento encontrado no índice: " + result);
    }
}
```

7. Contagem de Caracteres em String

- **Descrição:** Conta a quantidade de ocorrências de cada caractere em uma string.

```
import java.util.HashMap;

public class ContagemCaracteres {
    public static void main(String[] args) {
        String str = "programacao";
        HashMap<Character, Integer> charCount = new HashMap<>();
        for (char c : str.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }
        System.out.println(charCount);
    }
}
```

8. Verificação de Palíndromo

- **Descrição:** Verifica se uma palavra ou frase é um palíndromo



```
public class Palindromo {
    public static boolean isPalindromo(String str) {
        int i = 0, j = str.length() - 1;
        while (i < j) {
            if (str.charAt(i) != str.charAt(j)) return false;
            i++;
            j--;
        }
        return true;
    }

    public static void main(String[] args) {
        String str = "radar";
        System.out.println("É palíndromo? " + isPalindromo(str));
    }
}
```

9. Máximo Comum Divisor (MDC) - Algoritmo de Euclides

- **Descrição:** Calcula o maior divisor comum entre dois números.

```
public class MDC {
    public static int calcularMDC(int a, int b) {
        if (b == 0) return a;
        return calcularMDC(b, a % b);
    }

    public static void main(String[] args) {
        int a = 56, b = 98;
        System.out.println("MDC: " + calcularMDC(a, b));
    }
}
```

10. Verificação de Número Primo

- **Descrição:** Verifica se um número é primo.

```
public class NumeroPrimo {
    public static boolean isPrimo(int num) {
        if (num <= 1) return false;
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) return false;
        }
        return true;
    }

    public static void main(String[] args) {
        int num = 29;
        System.out.println("É primo? " + isPrimo(num));
    }
}
```



11. Ordenação por Inserção (Insertion Sort)

- **Descrição:** Ordena uma lista de números inserindo cada elemento na posição correta.

```
public class InsertionSort {
    public static void insertionSort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }

    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6};
        insertionSort(arr);
        for (int num : arr) System.out.print(num + " ");
    }
}
```

12. Ordenação por Seleção (Selection Sort)

- **Descrição:** Ordena uma lista encontrando o menor elemento e movendo-o para a posição correta.

```
public class SelectionSort {
    public static void selectionSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[minIndex]) minIndex = j;
            }
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }

    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        selectionSort(arr);
        for (int num : arr) System.out.print(num + " ");
    }
}
```

13. Contagem de Vogais em uma String

- **Descrição:** Conta quantas vogais há em uma string.

```
public class ContagemVogais {
    public static int contarVogais(String str) {
        int count = 0;
```



```
for (char c : str.toLowerCase().toCharArray()) {
    if ("aeiou".indexOf(c) != -1) count++;
}
return count;
}

public static void main(String[] args) {
    String str = "programacao";
    System.out.println("Quantidade de vogais: " + contarVogais(str));
}
}
```

14. Contagem de Palavras em uma String

- **Descrição:** Conta o número de palavras em uma string, considerando espaços como separadores.

```
public class ContagemPalavras {
    public static int contarPalavras(String str) {
        if (str == null || str.isEmpty()) return 0;
        String[] words = str.trim().split("\\s+");
        return words.length;
    }

    public static void main(String[] args) {
        String str = "Java é uma linguagem poderosa";
        System.out.println("Quantidade de palavras: " + contarPalavras(str));
    }
}
```

15. Ordenação por Contagem (Counting Sort)

- **Descrição:** Algoritmo de ordenação que conta a ocorrência de cada valor.

```
public class CountingSort {
    public static void countingSort(int[] arr, int max) {
        int[] count = new int[max + 1];
        for (int num : arr) count[num]++;
        int index = 0;
        for (int i = 0; i <= max; i++) {
            while (count[i]-- > 0) arr[index++] = i;
        }
    }

    public static void main(String[] args) {
        int[] arr = {4, 2, 2, 8, 3, 3, 1};
        countingSort(arr, 8);
        for (int num : arr) System.out.print(num + " ");
    }
}
```

16. Potenciação Recursiva

- **Descrição:** Calcula a potência de um número utilizando recursão.

```
public class Potenciacao {
```



```
public static int potencia(int base, int expoente) {
    if (expoente == 0) return 1;
    return base * potencia(base, expoente - 1);
}

public static void main(String[] args) {
    int base = 3, expoente = 4;
    System.out.println("Potência: " + potencia(base, expoente));
}
}
```

17. Contagem de Dígitos

- **Descrição:** Conta a quantidade de dígitos em um número inteiro.

```
public class ContagemDigitos {
    public static int contarDigitos(int num) {
        return String.valueOf(num).length();
    }

    public static void main(String[] args) {
        int num = 12345;
        System.out.println("Quantidade de dígitos: " + contarDigitos(num));
    }
}
```

18. Ordenação por Intercalação (Merge Sort)

- **Descrição:** Algoritmo de ordenação que utiliza o método de dividir e conquistar.

```
public class MergeSort {
    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
    }

    public static void merge(int[] arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;
        int[] L = new int[n1];
        int[] R = new int[n2];
        for (int i = 0; i < n1; i++) L[i] = arr[left + i];
        for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];
        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static void main(String[] args) {
```



```
int[] arr = {12, 11, 13, 5, 6, 7};
mergeSort(arr, 0, arr.length - 1);
for (int num : arr) System.out.print(num + " ");
}
}
```

19. Busca Linear

- **Descrição:** Realiza a busca de um elemento em uma lista de maneira sequencial.

```
public class BuscaLinear {
    public static int buscaLinear(int[] arr, int x) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == x) return i;
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {2, 3, 4, 10, 40};
        int x = 10;
        System.out.println("Elemento encontrado no índice: " + buscaLinear(arr, x));
    }
}
```

20. Inversão de String

- **Descrição:** Inverte os caracteres de uma string.

```
public class InversaoString {
    public static String inverterString(String str) {
        return new StringBuilder(str).reverse().toString();
    }

    public static void main(String[] args) {
        String str = "programacao";
        System.out.println("String invertida: " + inverterString(str));
    }
}
```

21. Máximo de um Array

- **Descrição:** Encontra o valor máximo em um array.

```
public class MaximoArray {
    public static int encontrarMaximo(int[] arr) {
        int max = arr[0];
        for (int num : arr) {
            if (num > max) max = num;
        }
        return max;
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.println("Valor máximo: " + encontrarMaximo(arr));
    }
}
```




```
}  
}
```

22. Mínimo de um Array

- **Descrição:** Encontra o valor mínimo em um array.

```
public class MinimoArray {  
    public static int encontrarMinimo(int[] arr) {  
        int min = arr[0];  
        for (int num : arr) {  
            if (num < min) min = num;  
        }  
        return min;  
    }  
  
    public static void main(String[] args) {  
        int[] arr = {5, 4, 3, 2, 1};  
        System.out.println("Valor mínimo: " + encontrarMinimo(arr));  
    }  
}
```

23. Produto de Elementos de um Array

- **Descrição:** Calcula o produto de todos os elementos de um array.

```
public class ProdutoArray {  
    public static int calcularProduto(int[] arr) {  
        int produto = 1;  
        for (int num : arr) produto *= num;  
        return produto;  
    }  
  
    public static void main(String[] args) {  
        int[] arr = {1, 2, 3, 4};  
        System.out.println("Produto dos elementos: " + calcularProduto(arr));  
    }  
}
```

24. Verificação de Números Duplicados

- **Descrição:** Verifica se há números duplicados em um array.

```
import java.util.HashSet;  
  
public class DuplicadosArray {  
    public static boolean verificarDuplicados(int[] arr) {  
        HashSet<Integer> set = new HashSet<>();  
        for (int num : arr) {  
            if (!set.add(num)) return true;  
        }  
        return false;  
    }  
  
    public static void main(String[] args) {  
        int[] arr = {1, 2, 3, 4, 5, 2};  
        System.out.println("Tem duplicados? " + verificarDuplicados(arr));  
    }  
}
```



```
}  
}
```

25. Soma de Dígitos de um Número

- **Descrição:** Calcula a soma dos dígitos de um número inteiro.

```
public class SomaDigitos {  
    public static int somarDigitos(int num) {  
        int soma = 0;  
        while (num != 0) {  
            soma += num % 10;  
            num /= 10;  
        }  
        return soma;  
    }  
  
    public static void main(String[] args) {  
        int num = 12345;  
        System.out.println("Soma dos dígitos: " + somarDigitos(num));  
    }  
}
```

26. Ordenação por Shell Sort

- **Descrição:** Algoritmo de ordenação que compara elementos a uma certa "distância".

```
public class ShellSort {  
    public static void shellSort(int[] arr) {  
        int n = arr.length;  
        for (int gap = n / 2; gap > 0; gap /= 2) {  
            for (int i = gap; i < n; i++) {  
                int temp = arr[i];  
                int j;  
                for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {  
                    arr[j] = arr[j - gap];  
                }  
                arr[j] = temp;  
            }  
        }  
    }  
  
    public static void main(String[] args) {  
        int[] arr = {12, 34, 54, 2, 3};  
        shellSort(arr);  
        for (int num : arr) System.out.print(num + " ");  
    }  
}
```

27. Conversão de Número Decimal para Binário

- **Descrição:** Converte um número decimal para seu equivalente binário.

```
public class DecimalParaBinario {  
    public static String converterParaBinario(int num) {  
        return Integer.toBinaryString(num);  
    }  
}
```



```
}  
  
public static void main(String[] args) {  
    int num = 10;  
    System.out.println("Binário: " + converterParaBinario(num));  
}  
}
```

28. Cálculo de Média de um Array

- **Descrição:** Calcula a média dos valores de um array.

```
public class MediaArray {  
    public static double calcularMedia(int[] arr) {  
        int soma = 0;  
        for (int num : arr) soma += num;  
        return (double) soma / arr.length;  
    }  
  
    public static void main(String[] args) {  
        int[] arr = {1, 2, 3, 4, 5};  
        System.out.println("Média: " + calcularMedia(arr));  
    }  
}
```

29. Ordenação por Heap Sort

- **Descrição:** Algoritmo de ordenação que utiliza a estrutura de heap.

```
public class HeapSort {  
    public static void heapSort(int[] arr) {  
        int n = arr.length;  
        for (int i = n / 2 - 1; i >= 0; i--) heapify(arr, n, i);  
        for (int i = n - 1; i >= 0; i--) {  
            int temp = arr[0];  
            arr[0] = arr[i];  
            arr[i] = temp;  
            heapify(arr, i, 0);  
        }  
    }  
  
    public static void heapify(int[] arr, int n, int i) {  
        int largest = i;  
        int left = 2 * i + 1;  
        int right = 2 * i + 2;  
        if (left < n && arr[left] > arr[largest]) largest = left;  
        if (right < n && arr[right] > arr[largest]) largest = right;  
        if (largest != i) {  
            int swap = arr[i];  
            arr[i] = arr[largest];  
            arr[largest] = swap;  
            heapify(arr, n, largest);  
        }  
    }  
  
    public static void main(String[] args) {  
        int[] arr = {12, 11, 13, 5, 6, 7};  
        heapSort(arr);  
    }  
}
```



```
        for (int num : arr) System.out.print(num + " ");  
    }  
}
```

30. Ordenação por Radix Sort

- **Descrição:** Algoritmo de ordenação que classifica números inteiros de maneira eficiente.

```
import java.util.Arrays;  
  
public class RadixSort {  
    public static void radixSort(int[] arr) {  
        int max = Arrays.stream(arr).max().getAsInt();  
        for (int exp = 1; max / exp > 0; exp *= 10) countSort(arr, exp);  
    }  
  
    public static void countSort(int[] arr, int exp) {  
        int n = arr.length;  
        int[] output = new int[n];  
        int[] count = new int[10];  
        Arrays.fill(count, 0);  
        for (int i = 0; i < n; i++) count[(arr[i] / exp) % 10]++;  
        for (int i = 1; i < 10; i++) count[i] += count[i - 1];  
        for (int i = n - 1; i >= 0; i--) {  
            output[count[(arr[i] / exp) % 10] - 1] = arr[i];  
            count[(arr[i] / exp) % 10]--;  
        }  
        System.arraycopy(output, 0, arr, 0, n);  
    }  
  
    public static void main(String[] args) {  
        int[] arr = {170, 45, 75, 90, 802, 24, 2, 66};  
        radixSort(arr);  
        for (int num : arr) System.out.print(num + " ");  
    }  
}
```



A Importância dos Algoritmos em Java e Suas Aplicações

Os algoritmos desempenham um papel fundamental em todos os sistemas computacionais. Em Java, a compreensão e o domínio desses algoritmos permitem desenvolver aplicações robustas, eficientes e escaláveis.

Desde operações simples, como ordenação e busca, até algoritmos avançados de cálculo, cada um serve para resolver um problema específico, seja em desenvolvimento de sistemas, otimização de recursos ou criação de inteligências artificiais.

Algoritmos de ordenação e busca, por exemplo, são fundamentais para manipulação e recuperação de dados em bancos de dados e aplicações de alto desempenho.

Algoritmos de criptografia e hashing tornam-se indispensáveis em áreas de segurança da informação e proteção de dados, especialmente na era digital.

Com a crescente demanda por soluções que lidem com grandes volumes de dados e precisem de respostas rápidas, conhecer e aplicar algoritmos eficientes é um diferencial essencial para qualquer programador e engenheiro de software.

A prática com esses algoritmos em Java ajuda o desenvolvedor a entender o comportamento de diferentes abordagens, compreender a complexidade de tempo e espaço, e optar pela solução mais adequada para cada situação. No contexto profissional, esses conhecimentos são aplicáveis em diversas áreas, como:

- **Desenvolvimento de aplicações web e mobile**, onde eficiência e escalabilidade são indispensáveis.
- **Sistemas de recomendação e busca**, que exigem algoritmos rápidos e eficientes.
- **Inteligência Artificial e Machine Learning**, onde algoritmos complexos são utilizados para processar e analisar grandes volumes de dados.

Dominar algoritmos é fundamental para enfrentar desafios no mundo real e entregar soluções tecnológicas que realmente façam a diferença. Com os exemplos práticos deste artigo, esperamos que você possa aplicar esses conhecimentos em seus projetos e desenvolver uma base sólida em algoritmos que irá se expandir e evoluir com suas experiências e estudos.