

# RESUMO RÁPIDO - Algoritmos e Complexidade

Guia de Consulta Instantânea

# ⚡ COMPLEXIDADES - COLA RÁPIDA

## 📊 Hierarquia de Complexidades

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n!)$

Notação	Nome	n=100	n=1000	Exemplo
$O(1)$	Constante	1	1	Array[index]
$O(\log n)$	Logarítmica	7	10	Busca binária
$O(n)$	Linear	100	1.000	Busca linear
$O(n \log n)$	Linearítmica	700	10.000	Merge sort
$O(n^2)$	Quadrática	10.000	1.000.000	Bubble sort
$O(2^n)$	Exponencial	$10^{30}$	$10^{301}$	Força bruta

# ESTRUTURAS DE DADOS - GUIA RÁPIDO

## Comparação Direta

Estrutura	Acesso	Busca	Inserção	Remoção	Uso Principal
Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$	Acesso por índice
Lista Ligada	$O(n)$	$O(n)$	$O(1)^*$	$O(1)^*$	Inserção frequente
Pilha	-	-	$O(1)$	$O(1)$	LIFO (undo, histórico)
Fila	-	-	$O(1)$	$O(1)$	FIFO (processos)
Hash Table	-	$O(1)$	$O(1)$	$O(1)$	Chave-valor rápido
BST	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	Dados ordenados
Heap	-	-	$O(\log n)$	$O(\log n)$	Prioridades

\*Com ponteiro para posição

# ALGORITMOS DE ORDENAÇÃO - GUIA RÁPIDO

## Escolha Rápida

```
void ordenar_inteligente(int arr[], int n) {  
    if (n < 20) {  
        insertion_sort(arr, n);    // Simples, rápido para n pequeno  
    } else if (quase_ordenado(arr, n)) {  
        insertion_sort(arr, n);    // O(n) para quase ordenado  
    } else if (precisa_estabilidade) {  
        merge_sort(arr, 0, n-1);  // Sempre O(n log n), estável  
    } else {  
        quick_sort(arr, 0, n-1);  // Mais rápido na prática  
    }  
}
```

## Comparação de Algoritmos

Algoritmo	Melhor	Médio	Pior	Estável	In-place	Quando Usar
-----------	--------	-------	------	---------	----------	-------------

# ALGORITMOS DE BUSCA - GUIA RÁPIDO

## Escolha por Situação

### Dados Não Ordenados:

```
// Única opção: Busca Linear  $O(n)$ 
for (int i = 0; i < n; i++) {
    if (arr[i] == valor) return i;
}
```

### Dados Ordenados:

```
// Busca Binária  $O(\log n)$ 
int busca_binaria(int arr[], int n, int valor) {
    int esq = 0, dir = n - 1;
    while (esq <= dir) {
        int meio = esq + (dir - esq) / 2;
        if (arr[meio] == valor) return meio;
        if (arr[meio] < valor) esq = meio + 1;
    }
}
```

# DICAS PRÁTICAS - COLA

## Debugging Rápido

### 1. Contar Operações:

```
int comparacoes = 0;
for (int i = 0; i < n; i++) {
    comparacoes++; // Contar para análise
    if (arr[i] == valor) return i;
}
printf("Comparações: %d\n", comparacoes);
```

### 2. Medir Tempo:

```
clock_t inicio = clock();
algoritmo();
clock_t fim = clock();
double tempo = ((double)(fim - inicio)) / CLOCKS_PER_SEC;
printf("Tempo: %.6f segundos\n", tempo);
```

# PROBLEMAS CLÁSSICOS - SOLUÇÕES RÁPIDAS

## 1. Buscar Dois Números que Somam Target

```
// Hash Table approach - O(n)
int* dois_soma(int nums[], int n, int target) {
    HashTable ht;
    for (int i = 0; i < n; i++) {
        int complemento = target - nums[i];
        if (existe(&ht, complemento)) {
            int* resultado = malloc(2 * sizeof(int));
            resultado[0] = get_indice(&ht, complemento);
            resultado[1] = i;
            return resultado;
        }
        insert(&ht, nums[i], i);
    }
    return NULL;
}
```

## 2. Validar Parênteses Balanceados

# OTIMIZAÇÕES INSTANTÂNEAS

## Code Patterns

### 1. Early Break:

```
// Parar assim que encontrar
for (int i = 0; i < n; i++) {
    if (condicao) {
        return resultado; // Sair imediatamente
    }
}
```

### 2. Cache Dados Frequentes:

```
// Calcular uma vez, reusar
int tamanho = strlen(string); // Fora do loop
for (int i = 0; i < tamanho; i++) {
    // usar i
}
```



# CHECKLIST PRÉ-ENTREGA

## Validações Essenciais

- ☐ Ponteiros NULL verificados
- ☐ Bounds de arrays verificados
- ☐ Casos extremos testados ( $n=0$ ,  $n=1$ )
- ☐ Memory leaks verificados
- ☐ Overflow prevenido

## Análise de Complexidade

- ☐ Melhor caso analisado
- ☐ Pior caso analisado
- ☐ Caso médio considerado
- ☐ Complexidade de espaço verificada

# DICAS PARA ENTREVISTAS

## Abordagem Estruturada

### 1. Entender o Problema:

- Clarificar entrada e saída
- Perguntar sobre restrições
- Dar exemplos pequenos

### 2. Pensar em Voz Alta:

- "Posso usar estrutura auxiliar?"
- "Qual a complexidade esperada?"
- "Há casos especiais?"

### 3. Começar Simples:

# RESUMO DOS RESUMOS

## Regra de Ouro


"Correto primeiro, depois optimize"

## Complexidades Mais Importantes

- $O(1)$ : Hash table access, array index
- $O(\log n)$ : Binary search, balanced tree
- $O(n)$ : Linear scan, simple loop
- $O(n \log n)$ : Optimal sorting, merge
- $O(n^2)$ : Nested loops, naive algorithms

## Estruturas Mais Usadas

1. **Array**: Quando precisar de acesso por índice

 Este resumo rápido serve como consulta instantânea para os conceitos mais importantes de algoritmos e complexidade.

*Versão: Express - Última atualização: 27 de agosto de 2025*