# Data Structures & Algorithms Cheatsheet

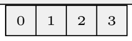
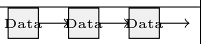
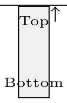
*A Quick Reference Guide*

## Complexity Reference

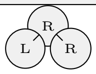
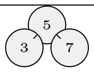
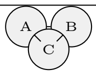
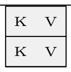
| | |
|---|---|
| **O(1)** | Constant time |
| **O(log n)** | Logarithmic time |
| **O(n)** | Linear time |
| **O(n log n)** | Linearithmic time |
| **O($n^2$)** | Quadratic time |

## Data Structures

### Linear Data Structures

| Name | Description | Operations (Time) | Diagram |
|---|---|---|---|
| **Array** | Contiguous memory block | Access: O(1)<br>Insert/Delete: O(n) |  |
| **Linked List** | Nodes with pointers | Access: O(n)<br>Insert/Delete: O(1) |  |
| **Stack** | LIFO structure | Push/Pop: O(1) |  |
| **Queue** | FIFO structure | Enqueue/Dequeue: O(1) | |

### Non-linear Data Structures

| Name | Description | Operations (Time) | Diagram |
|---|---|---|---|
| **Binary Tree** | Hierarchical nodes | Traverse: O(n) |  |
| **BST** | Ordered binary tree | Search/Insert: O(h) |  |
| **Graph** | Nodes and edges | BFS/DFS: O(V+E) |  |
| **Hash Table** | Key-value pairs | Access: O(1) avg |  |

## Algorithms

### Sorting Algorithms

| Name | Description | Complexity | Pseudocode |
|---|---|---|---|
| **Bubble Sort** | Repeated swaps | Time: O($n^2$)<br>Space: O(1) | bubbleSort(arr):<br>for i = 0 to n-1:<br>for j = 0 to n-i-1:<br>if arr[j] > arr[j+1]:<br>swap(arr[j], arr[j+1]) |
| **Merge Sort** | Divide and merge | Time: O(n log n)<br>Space: O(n) | mergeSort(arr):<br>if len(arr) > 1:<br>mid = len(arr) / 2<br>L = arr[:mid]<br>R = arr[mid:]<br>mergeSort(L)<br>mergeSort(R)<br>merge(L, R, arr) |
| **Quick Sort** | Partition around pivot | Time: O(n log n) avg<br>Space: O(log n) | quickSort(arr, low, high):<br>if low < high:<br>pi = partition(arr, low, high)<br>quickSort(arr, low, pi-1)<br>quickSort(arr, pi+1, high) |

## Searching & Graph Algorithms

| **Name** | Description | Complexity | Pseudocode |
|---|---|---|---|
| **Binary Search** | Search in sorted array | Time: O(log n)<br>Space: O(1) | binarySearch(arr, target):<br>  left = 0, right = n-1<br>  while left <= right:<br>    mid = (left + right) / 2<br>    if arr[mid] == target:<br>      return mid<br>    elif arr[mid] < target:<br>      left = mid + 1<br>    else:<br>      right = mid - 1 |
| **BFS** | Breadth-first traversal | Time: O(V+E)<br>Space: O(V) | bfs(graph, start):<br>  queue = [start]<br>  visited = {start}<br>  while queue:<br>    node = queue.pop(0)<br>    for neighbor in graph[node]:<br>      if neighbor not in visited:<br>        visited.add(neighbor)<br>        queue.append(neighbor) |
| **DFS** | Depth-first traversal | Time: O(V+E)<br>Space: O(V) | dfs(graph, node, visited):<br>  visited = visited or set()<br>  visited.add(node)<br>  for neighbor in graph[node]:<br>    if neighbor not in visited:<br>      dfs(graph, neighbor, visited) |