# Common Java Data Structure Operations: Methods, Syntax, and Usage

## 1. Arrays

| Operation | Method | Syntax |
|---|---|---|
| Access element | arr[index] | int value = arr[2]; |
| Insert element (specific pos) | System.arraycopy() (or shift elements) | System.arraycopy(arr, 0, newArr, 0, index); |
| Remove element | Shift elements after removal index | arr[i] = arr[i + 1]; |
| Sort array | Arrays.sort(arr) | Arrays.sort(arr); |
| Search element | Linear Search or Binary Search | int index = Arrays.binarySearch(arr, target); |

## 2. Linked Lists

| Operation | Method | Syntax |
|---|---|---|
| Insert at head | newNode.next = head; head = newNode; | Node newNode = new Node(value); newNode.next = head; head = newNode; |
| Insert at tail | Traverse to the end, temp.next = newNode; | Node temp = head; while (temp.next != null) { temp = temp.next; } temp.next = newNode; |
| Delete node by value | Traverse and unlink node | temp.next = temp.next.next; |
| Reverse list | Iterate and reverse pointers | Node prev = null, curr = head; while (curr != null) { Node next = curr.next; curr.next = prev; prev = curr; curr = next; } head = prev; |
| Find middle node | Use slow and fast pointers | Node slow = head, fast = head; while (fast != null && fast.next != null) { slow = slow.next; fast = fast.next.next; } return slow; |

## 3. Stacks

| Operation | Method | Syntax |
|---|---|---|
| Push | stack.push(element) | stack.push(10); |
| Pop | stack.pop() | int element = stack.pop(); |
| Peek | stack.peek() | int top = stack.peek(); |
| Check if empty | stack.isEmpty() | boolean isEmpty = stack.isEmpty(); |
| Size | stack.size() | int size = stack.size(); |

## 4. Queues

| Operation | Method | Syntax |
|---|---|---|
| Enqueue | queue.add(element) or queue.offer(element) | queue.add(10); or queue.offer(10); |
| Dequeue | queue.poll() | int element = queue.poll(); |
| Peek | queue.peek() | int front = queue.peek(); |
| Check if empty | queue.isEmpty() | boolean isEmpty = queue.isEmpty(); |
| Size | queue.size() | int size = queue.size(); |

## 5. HashMap

| Operation | Method | Syntax |
|---|---|---|
| Insert key-value pair | map.put(key, value) | map.put("key", 10); |
| Get value by key | map.get(key) | int value = map.get("key"); |
| Remove key-value pair | map.remove(key) | map.remove("key"); |
| Check if key exists | map.containsKey(key) | boolean exists = map.containsKey("key"); |
| Check if value exists | map.containsValue(value) | boolean exists = map.containsValue(10); |
| Iterate through entries | for (Map.Entry<K, V> entry : map.entrySet()) | for (Map.Entry<String, Integer> entry : map.entrySet()) { K key = entry.getKey(); V value = entry.getValue(); } |

## 6. HashSet

| Operation | Method | Syntax |
|---|---|---|
| Insert element | set.add(element) | set.add(10); |
| Check if element exists | set.contains(element) | boolean exists = set.contains(10); |
| Remove element | set.remove(element) | set.remove(10); |
| Iterate through elements | for (T element : set) | for (int element : set) { } |

## 7. Trees (Binary Search Tree)

| Operation | Method | Syntax |
|---|---|---|
| Insert node | Traverse and insert recursively | if (root == null) { root = new Node(value); } else if (value < root.value) { insert(root.left, value); } else { insert(root.right, value); } |
| Search node | Traverse recursively for value | `if (root == null |
| In-order traversal | Traverse left -> root -> right | inorder(root.left); System.out.println(root.value); inorder(root.right); |
| Delete node | Recursively find node, handle children | if (root == null) return root; if (key < root.value) { root.left = deleteNode(root.left, key); } else if (key > root.value) { root.right = deleteNode(root.right, key); } |
| Find minimum (in BST) | Traverse leftmost child | while (root.left != null) { root = root.left; } return root.value; |

## 8. Graphs (Adjacency List)

| Operation | Method | Syntax |
|---|---|---|
| Add edge | adj[u].add(v); adj[v].add(u); | adj[u].add(v); adj[v].add(u); |
| BFS traversal | Use Queue, visit each node level by level | Queue<Integer> queue = new LinkedList<>(); visited[start] = true; queue.add(start); while (!queue.isEmpty()) { int node = queue.poll(); } |
| DFS traversal | Use Recursion, visit nodes in depth-first order | void dfs(int node, boolean[] visited) { visited[node] = true; for (int neighbor : adj[node]) { if (!visited[neighbor]) { dfs(neighbor, visited); } } } |