Advanced Data Structures

The heap, deque, tree, and graph operations you actually write in an interview, with the time complexity that explains why.

1 HEAP *The smallest (or largest) item, ready in O(1), whenever you do not need the rest sorted.*

OPERATION	SYNTAX	TIME
Build	`heapq.heapify(heap)`	O(n)
Push	`heapq.heappush(heap, x)`	O(log n)
Pop	`heapq.heappop(heap)`	O(log n)
Peek	`heap[0]`	O(1)
Max-heap push	`heappush(heap, (-x, item))`	O(log n)

Tuples compare left to right, so two equal priorities fall through to comparing the second value, and a TypeError follows if that value cannot be ordered, like a dict. Push a counter in the middle, (priority, count, item), so the tie never reaches it.

2 DEQUE *A queue or a stack, whichever end you need O(1) at, without two different classes to remember.*

OPERATION	SYNTAX	TIME
Append (right)	`dq.append(x)`	O(1)
Append (left)	`dq.appendleft(x)`	O(1)
Pop (right)	`dq.pop()`	O(1)
Pop (left)	`dq.popleft()`	O(1)
Peek	`dq[0] / dq[-1]`	O(1)

A list only tracks one end efficiently, so popping from the front means sliding every remaining item over, an O(n) cost. A deque tracks both ends directly, so popleft() finishes in O(1) no matter how long the queue is.

3 TREE *Parent and child links you walk top-down, one level or one branch at a time.*

OPERATION	SYNTAX	TIME
Node	`TreeNode(value)`	O(1)
Search (any tree)	`search_value(root, x)`	O(n)
Search (BST, avg)	`search_value(root, x)`	O(log n)
BFS (level order)	`queue = deque([root])`	O(n)
DFS (recursive)	`dfs(node.left); dfs(node.right)`	O(n)

BFS holds a whole level in its queue at once, O(w) space for the widest level. DFS only holds the current branch, O(h) space for the height. A wide, shallow tree favors DFS; a deep, narrow one favors BFS.

4 GRAPH *Nodes and edges you look up by key, not by index, an adjacency list first.*

OPERATION	SYNTAX	TIME
Build	`adj_list = defaultdict(list)`	O(1)
Add edge	`adj_list[u].append(v)`	O(1)
Neighbors	`adj_list[u]`	O(1)
Visited check	`u in visited`	O(1)
Traversal	`bfs(adj_list, start)`	O(n + e)

A tree only has one path in; a graph can loop back on itself. Skip the visited set and a cycle spins the traversal forever, so mark a node visited the moment you enqueue or recurse into it, not once you get around to processing it.