B4 ABSTRACT DATA TYPES · B4.1 · HL ONLY · FINAL SUBTOPIC

Fundamentals of ADTs

Structures defined by their **operations**, not their implementation: **linked lists** (singly, doubly, circular), **binary search trees** and **sets**. HL only.

01 ADT principles

ADT	Defined by its operations.
Hidden	Implementation is invisible.
Abstraction	What it does, not how.
Encapsulation	Data + operations bundled.
Modularity	Swap implementations freely.

02 Linked lists

Node	Data + pointer to next.
Singly	One way; tail → null.
Doubly	Pointers both directions.
Circular	Tail points back to head.
Costs	Insert O(1); search O(n).

03 The three ADTs at a glance

1

Linked list

A chain of nodes; flexible size, sequential access.

2

Binary search tree

Left < parent < right; O(log n) search when balanced.

3

Set

Unordered, no duplicates; union, intersection, difference.

04 Binary search trees

Rule	Left < parent; right > parent.
Root / leaf	Top node / no children.
Balanced	Search/insert/delete O(log n).
Skewed	A chain; degrades to O(n).
Dynamic	Nodes added/removed freely.

05 Sets

Properties	Unordered; no duplicates.
Membership	Is x in the set?
Union	$\{1,2,3\} \cup \{3,4,5\} = \{1,2,3,4,5\}$.
Intersection	$\{1,2,3\} \cap \{3,4,5\} = \{3\}$.
Difference	$\{1,2,3\} - \{3,4,5\} = \{1,2\}$.

06 Know the difference

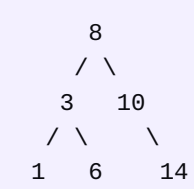
ADT vs implementation	The contract of operations versus the hidden code (array, list, tree) that fulfils it.	CORE
Array vs linked list	Fast random access but fixed size versus flexible size but sequential access.	STORAGE
Balanced vs skewed BST	Minimal depth giving O(log n) versus a linear chain giving O(n).	TREES
Set vs list	Unordered and duplicate-free versus ordered and allowing repeats.	COLLECTIONS

07 Insert into a BST · go left if smaller, right if larger

Insert order · 8, 3, 10, 1, 6, 14 into an empty tree

Insert	Comparisons	Placed as
8	tree is empty	the root
3	3 < 8	left child of 8
10	10 > 8	right child of 8
1	1 < 8, 1 < 3	left child of 3
6	6 < 8, 6 > 3	right child of 3
14	14 > 8, 14 > 10	right child of 10

The finished tree



In-order traversal (left, node, right) reads the keys sorted: **1, 3, 6, 8, 10, 14**. That always holds for a BST.

Your turn · insert 5 into the finished tree

5 < 8 (left to 3), 5 > 3 (right to 6), 5 < 6 (left, empty) → 5 becomes the **left child of 6**.

FINAL PASS BEFORE THE EXAM

Rapid exam tips

Eight things that lose marks if you slip on them. B4.1 is HL only. Skim before you walk in.

01

An **ADT is defined by its operations**; the implementation stays hidden.

02

Singly: tail → null. **Doubly**: both ways. **Circular**: tail → head.

03

Linked list insert/delete at a known node is **O(1)**; search is **O(n)**.

04

BST rule: **left < parent < right**, and it applies to whole subtrees.

05

BST O(log n) needs a **balanced** tree; a skewed chain is O(n).

06

Sets are **unordered, no duplicates**; adding an existing element does nothing.

07

Know **union, intersection, difference, membership** with small examples.

08

"Which ADT and why?" Match the **defining property to the scenario** to earn the marks.