# Abstract data types

IB DP Computer Science B4 Abstract data types full package

# Why does the structure matter?

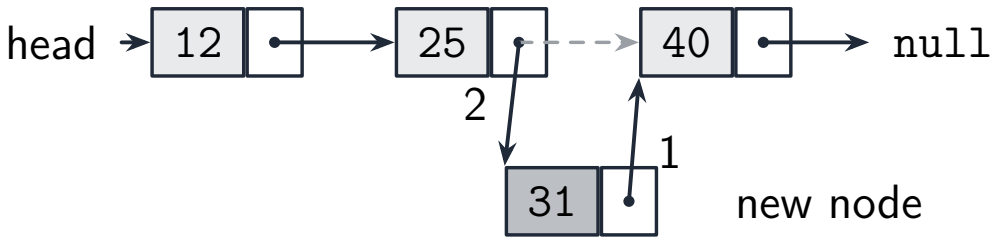
> The right structure makes both jobs fast.

- Theme B4 is Higher Level only.
- Prerequisites: B2 arrays, stacks, queues, recursion; B3 classes and references.

# Interface and implementation

| ADT | Key operations | Rule |
|---|---|---|
| stack | push, pop, peek | LIFO |
| queue | enqueue, dequeue | FIFO |
| **binary search tree** | insert, search | left smaller, right larger |
| **set** | add, contains | no duplicates |
| **hash table** | put, get by key | hash gives the slot |

# Nodes and pointers




No index: reading the 500th item takes 500 steps.

# Inserting 31 after 25

1. **Step 1** new.next = prev.next: 31 points to 40
2. **Step 2** prev.next = new: 25 points to 31

## Decision

Doing step 2 first would lose the rest of the list.

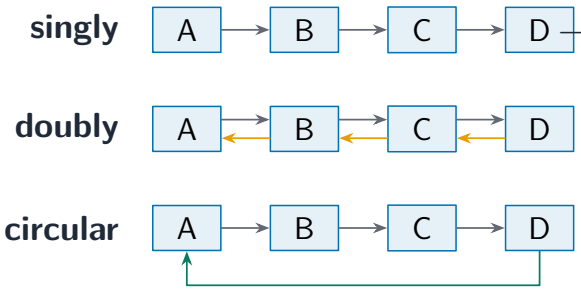
# Linked-list variants

| **Variant** | **Each node points to** | **Advantage** |
|---|---|---|
| singly | the next node | least memory |
| doubly | next and previous | **traverse** both ways |
| **circular** | next; last back to first | taking turns |

# Linked list or array?

| | **Array** | **Linked list** |
|---|---|---|
| access by index | O(1) | O(n) |
| insert in the middle | O(n): shift | O(1) at a known node |
| size | fixed or resized | grows a node at a time |
| memory | data only | data plus a **pointer** |

# Kinds of linked list



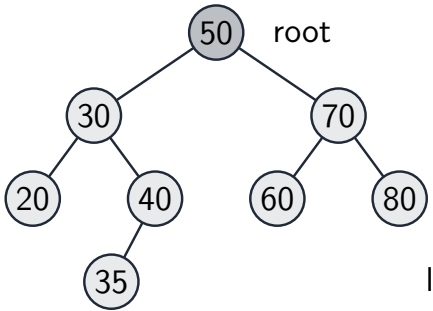

least memory

walk both ways; delete without the predecessor

last points to first: taking turns

**Circular**: the last points back to the first

# A binary search tree




leaves: 20, 35, 60, 80

> Insert 35: left of 50, right of 30, left of 40.

# Searching a BST

## Balanced tree

Each comparison drops one subtree.

Search is O(log n).

## Keys inserted sorted

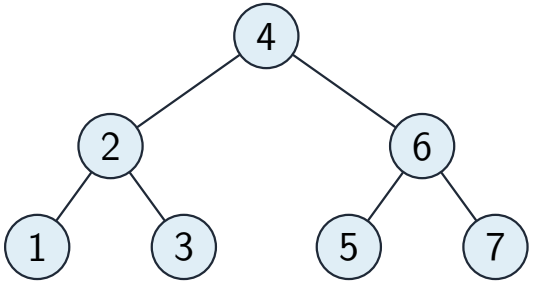
10, 20, 30... each goes right: one chain.

Search is O(n), no better than a list.

# Three traversals

| Traversal | Order | For the tree above |
|---|---|---|
| **pre-order** | node, left, right | 50, 30, 20, 40, 35, 70, 60, 80 |
| **in-order** | left, node, right | 20, 30, 35, 40, 50, 60, 70, 80 |
| **post-order** | left, right, node | 20, 35, 40, 30, 60, 80, 70, 50 |

# Traversing a binary search tree




pre-order: 4 2 1 3 6 5 7

in-order: 1 2 3 4 5 6 7

post-order: 1 3 2 5 7 6 4

**Post-order**: 1 3 2 5 7 6 4

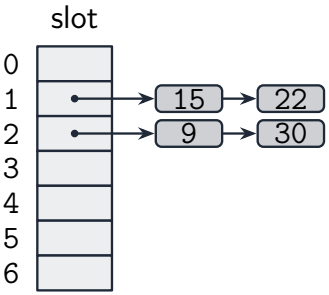
# Sets

1. **The sets** a holds maths, physics, cs; b holds cs, art
2. **Union: in either** maths, physics, cs, art
3. **Intersection: in both** cs
4. **Difference a — b** maths, physics

## Decision

Adding a value already present changes nothing.

# Hash tables and collisions




$h(k) = k \bmod 7$

15, 22 $\rightarrow$ slot 1

9, 30 $\rightarrow$ slot 2

collisions chained

15 and 22 both land in slot 1: a **collision**.

# Load factor

> Fuller table, more collisions: enlarge and rehash.

- **Load factor** = items / slots.
- A good **hash function** is fast and spreads keys evenly.

# Practice

## Your turn

Try these:

A. Insert 17 after 14 in $8 \to 14 \to 21$: the two statements.
B. Build the BST from 40, 20, 60, 10, 30, 50, 70, 25.
C. Give its three traversals.
D. Probe 12, 19, 5, 26 into 7 slots.

# Check yourself

