Exercise walk-through

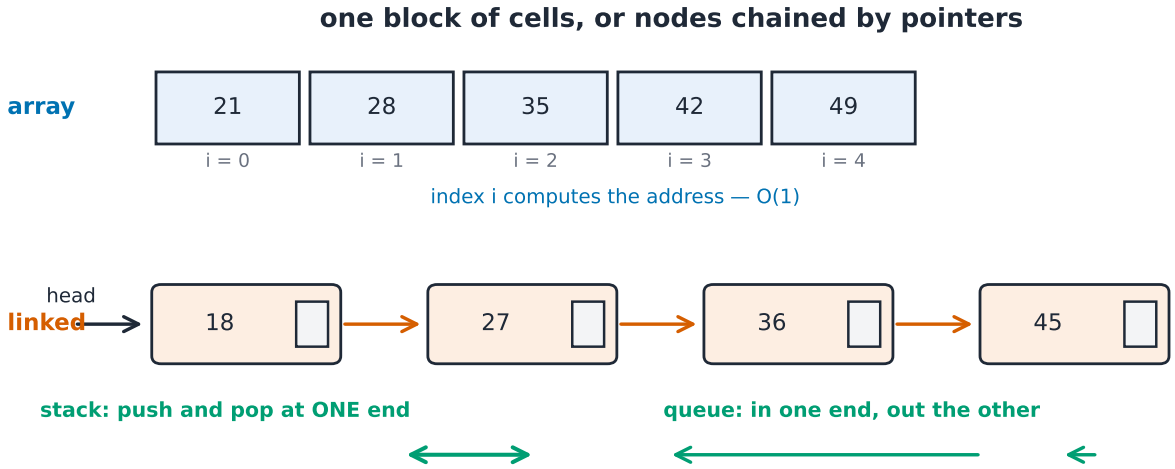
5.2
Lists, stacks and queues

University Computer Science

Lists, stacks and queues: which end is cheap

1. **Layout** A Python list is a contiguous **array** - index i computes the address; a **linked list** chains nodes, splicing cheaply but walking to index.
2. **Amortise** Doubling regrows make append amortized O(1); n front inserts move Theta(n squared) - the count quadruples when n doubles.
3. **Trace** push 3, 7, 1 then pop, pop returns 1 then 7 (last in, first out); enqueue a, b, c then two dequeues return a then b.

Array and linked layouts



One contiguous block with computed addresses versus nodes chained by pointers; the **stack** and **queue** differ only in which ends they use.

Exercises 2.1·2.2

2.1 `push 4, push 9, push 2, pop, push 7` – give the two pops that a further `pop, pop` returns. [3]

2.2 State the number of element moves for n front-inserts into an initially empty array, as a formula and at $n = 200$. [3]

Exercises 2.1·2.2

2.1 `push 4, push 9, push 2, pop, push 7` – give the two pops that a further `pop, pop` returns. [3]

2.1

- The pop removed 2; the stack reads $4, 9, 7$ (bottom to top) ✓.
- The next pops return 7 then 9 ✓.

2.2 State the number of element moves for n front-inserts into an initially empty array, as a formula and at $n = 200$. [3]

2.2

- Moves $= n(n+1)/2$ ✓.
- At $n = 200$: $201 \times 100 = 20\,100$ ✓.

Exercises 2.3·3.1

2.3 Enqueue `x, y, z`, dequeue twice – name the front element. [3]

3.1 A queue is built on a Python list using `append` for enqueue and `pop(0)` for dequeue.

State the cost class of each operation and of n paired enqueue-dequeue rounds. Rebuild it on `collections.deque` and state both costs again. Give the total class ratio between the two designs at $n = 10^4$, with the reason the list version loses. [8]

Exercises 2.3·3.1

2.3 Enqueue `x, y, z`, dequeue twice – name the front element. [3]

2.3

- `z` ✓.
- Dequeues removed `x` then `y`, leaving `z` at the front ✓.

3.1 A queue is built on a Python list using `append` for enqueue and `pop(0)` for dequeue.

State the cost class of each operation and of n paired enqueue-dequeue rounds. Rebuild it on `collections.deque` and state both costs again. Give the total class ratio between the two designs at $n = 10^4$, with the reason the list version loses. [8]

3.1

- List: enqueue $O(1)$ amortized; `pop(0)` slides all residents, $O(n)$ ✓.
- n rounds on the list: $\Theta(n^2)$ ✓.
- Deque: both ends $O(1)$; n rounds $\Theta(n)$ ✓.
- Ratio at $n = 10^4$: quadratic versus linear – the list moves about $10^8/2$ elements where the deque moves none per pop ✓.

Exercise 3.2

3.2 The bracket validator pushes opening brackets and matches each closer against the top of the stack.

Trace the scan of `{ [ ] ( ) }` listing the stack after each character. Trace `( [ ) ]` and name the character and the mismatch that fails. State the empty-stack case and why it also fails. [7]

Exercise 3.2

3.2 The bracket validator pushes opening brackets and matches each closer against the top of the stack.

Trace the scan of `{ [ ] ( ) }` listing the stack after each character. Trace `( [ ) ]` and name the character and the mismatch that fails. State the empty-stack case and why it also fails. [7]

3.2

- `{ [ ] ( ) }`: after `{` → `{`; `[` → `{ [`; `]` pops `[`; `(` → `{ (`; `)` pops `(`; `}` pops `{` – empty at the end: balanced ✓.
- `( [ ) ]`: stack after `( [`; the `)` finds `[` on top – mismatch at `)` (expected `]`) ✓.
- A closer arriving at an empty stack fails: nothing to match ✓.