Week 3 Study Summary: Insertion Sort, Stacks, and Expression Evaluation

Z5007 Programming and Data Structures · Oct 19–22, 2026 · Dr. Innocent Nyalala

1. Insertion Sort and Inversion Sensitivity

Insertion sort grows a sorted prefix $A[0..i-1]$ by shifting elements larger than `key = A[i]` one slot right. Each shift removes one inversion (j, i) with $j < i$ and $A[j] > A[i]$. Total shifts equal the exact inversion count I, running in $\Theta(n + I)$ time: $O(n)$ on sorted or nearly sorted arrays and $O(n^2)$ on reversed inputs. Using strict inequality (`A[j]` $>$ `key`) guarantees stability.

2. The LIFO Stack ADT: ArrayStack vs. LinkedStack

- **`ArrayStack`:** Stores items in `[0.._top]` with amortized $O(1)$ `push` (via doubling) and $O(1)$ `pop`/`peek`. Clear popped slots to `None` to allow garbage collection.
- **`LinkedStack`:** Places the stack top at the singly linked list `head`, guaranteeing $O(1)$ worst-case `push`, `pop`, and `peek` with zero resizing pauses. (Placing top at the tail makes `pop` $O(n)$ without a previous pointer.)

3. Classic Stack Applications

Bracket balancing pushes openers `(`, `[`, `{` and checks top matches on closers. Postfix (RPN) evaluation scans left-to-right, pushing operands and popping (b, a) to apply each operator in $O(n)$ time. An auxiliary monotonic minimum stack supports `get_min()` in $O(1)$.