Task 6.1p

Data structures and algorithms

Part A: Stack with O(1) Push, Pop, and Min ()

To implement a Stack that supports Push, Pop, and Min operations all in **constant time O(1)**, we can use **two stacks**:

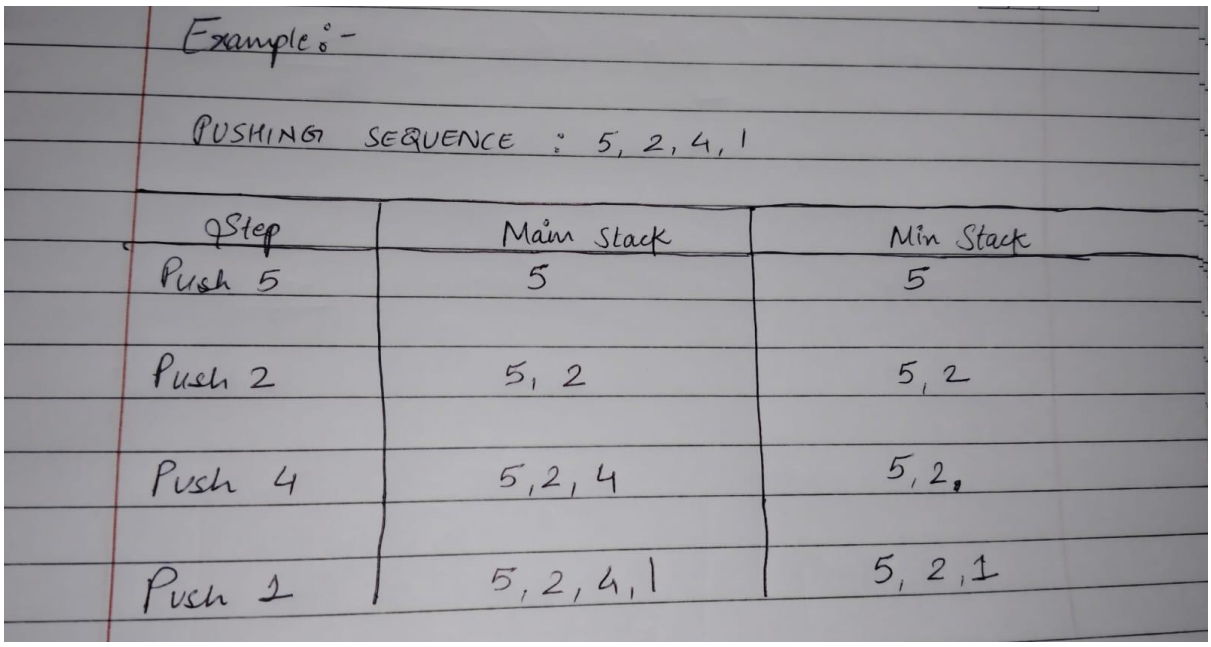
1. **Main Stack:**
   Stores all the elements in the stack as usual.
2. **Min Stack:**
   Tracks the minimum element at each level of the Main Stack.

Explanation:

- **Push(x):**
  - Push x onto the Main Stack.
  - If Min Stack is empty or x ≤ MinStack.top(), push x onto Min Stack.

    In other words , when we push a new value, we check if it is **smaller than or equal** to the current minimum, if yes , we also push it onto the min stack
- **Pop():**
  - Pop from the Main Stack.
  - If the popped value equals MinStack.top(), also pop from Min Stack.

    In other words , when we pop, we remove from both stacks if the popped value is the current minimum.
- **Min():**
  - Return MinStack.top() (this is always the current minimum).



How do we know that the time complexity is O (1)?

- Both Push and Pop affect **only the top element** of the stacks → O(1).
- We never need to scan through the stack to find the minimum.
- There is **no iteration or recursion** in any of these operations —
  Each action involves either:

  Direct access to the top of a stack (array index or linked list head),

  A simple comparison,

  Or one push/pop action.

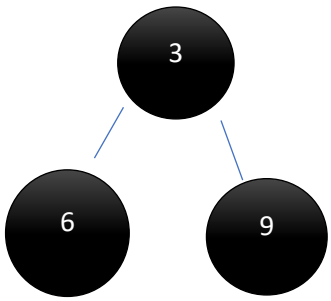
Part B : Incorrect Max-Heap DownHeap Implementation Example

Problem with the given pseudocode : In a Max-Heap, the parent should always **swap with the largest of both children**.
The algorithm should **compare left and right children together**, then decide.

But here, the pseudocode:

- **Ignores the right child completely if the left is bigger** (even if the right child is larger than both).

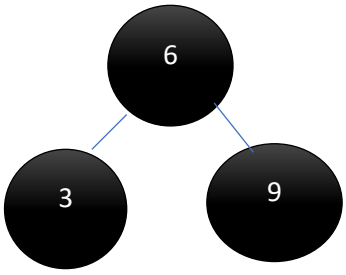
  Example heap



LEFT CHILD = 6

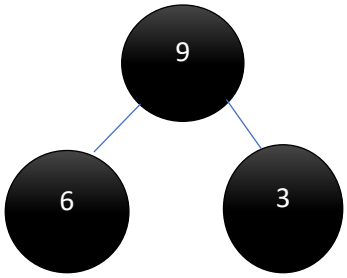
RIGHT CHILD = 9 LARGEST

Incorrect DownHeap swaps with **6** (since it checks left first), resulting in:



Invalid, Root should be the largest.

CORRECTED HEAP ,



Part C: DeleteElement(index) in Max-Heap

Procedure RemoveElement(index):

If index < 1 or index > heapSize:

Report "Invalid index"

Exit

If index == heapSize:

Remove the last element

Decrease heapSize by 1

Exit

Replace H[index] with the last element H[heapSize]

Remove last element

Decrease heapSize by 1

If index == 1 or H[index] ≤ H[Parent(index)]:

Call RestoreDown(index)   // Push down if needed

Else:

Call RestoreUp(index)   // Bubble up if needed