Stacks and Queues Tips

Here are some fundamental tips and tricks to keep in mind before solving competitive problems on **stacks** and **queues**:

General Preparation

1. **Understand the Basics**
    - Know how stacks and queues work, their time complexities, and typical operations:
        - Stack: `push`, `pop`, `peek`, `isEmpty`.
        - Queue: `enqueue`, `dequeue`, `peek`, `isEmpty`.
2. **Familiarize with Variants**
    - Learn about common variants:
        - **Deque (Double-ended Queue)**: Supports insertion and deletion from both ends.
        - **Priority Queue**: Elements are accessed based on priority.
3. **Learn Standard Applications**
    - Stacks:
        - Parenthesis matching.
        - Next/previous greater element.
        - Backtracking algorithms.
    - Queues:
        - Breadth-First Search (BFS).
        - Sliding window problems.
        - Scheduling tasks.

Problem-Solving Tips

4. **Trace the Input and Output**
    - Always test your understanding of the problem by simulating the operations manually on sample inputs.
5. **Recognize Patterns**
    - Look for cues in the problem:

- "Last in, first out" ➡️ Stack.
- "First in, first out" ➡️ Queue.
- "Sliding window" or "fixed range processing" ➡️ Deque.

6. **Use Efficient Libraries**
   - Use built-in data structures for ease and performance:
     - Python: `collections.deque` (queue/deque), `queue.Queue`.
     - C++: `std::stack`, `std::queue`, `std::deque`.
7. **Be Careful with Edge Cases**
   - Empty stack/queue scenarios.
   - Input with one element.
   - Problems with large inputs and long operations (optimize!).

Optimization Tricks

8. **Min/Max Stack and Queue**
   - Use auxiliary stacks or queues to keep track of the minimum or maximum for constant-time queries:
     - Min stack: Store both the element and the current minimum.
     - Sliding window max: Use a deque.
9. **Two-Stack Queue (Queue using Stacks)**
   - For implementing a queue using stacks:
     - One stack for `enqueue`.
     - Another for `dequeue` (pop from one stack and push to the other when needed).
10. **Use Monotonic Stack/Queue for Range Queries**

- Monotonic Stack: Useful for "next greater element" or "largest rectangle" problems.
- Monotonic Queue: Used in sliding window problems to maintain order of elements efficiently.

Practice Problems

11. **Start Small**

- Begin with straightforward problems like:
  - Balancing parentheses.
  - Implementing stack/queue from scratch.
- Gradually move to intermediate problems like:
  - Next greater element.
  - Sliding window maximum.

12. **Review and Learn**

- After solving a problem, understand the solution thoroughly, especially for optimized approaches.

Debugging Tips

13. **Print Intermediate States**

- Print the stack/queue at key steps to debug issues.
- Use diagrams to visualize how elements flow through.

14. **Dry Run for Edge Cases**

- Walk through your code with edge cases to ensure correctness.

By mastering these tips, you'll be well-prepared to tackle a variety of competitive programming challenges involving stacks and queues.