ClickUp Backend Interview
Document Event Processing

Complete Preparation Guide - November 30, 2024 (v2)

November 30, 2025

Contents

1 Interview Overview

1.1 What to Expect

Interview Type: Backend Live Coding (60 minutes)

Format:

- CodeSignal platform (browser-based IDE)
- No pre-written test suite
- You must test your own code
- Recommended languages: Python, JavaScript, TypeScript
- Focus on correctness, edge cases, and code quality

Problem Domain:

- Event processing in memory
- Document state management (similar to Google Docs)
- Real-time collaborative editing
- ClickUp's core functionality

1.2 Success Criteria

1. **Correctness**: Handle all test cases including edge cases
2. **Code Quality**: Clean, readable, well-structured code
3. **Communication**: Think out loud, explain your approach
4. **Testing**: Demonstrate testing strategy
5. **Problem-Solving**: Handle follow-up questions and variations

2 Core Problem: Document Event Processor

Problem

Context: ClickUp has a feature that allows users to create documents similar to Google Docs. Changes are tracked through batches of events.

Task: Process events on a document and return the correctly updated document.

Data Structures:
Document:

```
{
    title: string,
    content: string,          // Lines separated by \n
    lastUpdated: timestamp,
    createdOn: timestamp
}
```

Event:

```
{
    event_id: number,
    event_name: string,       // Case-insensitive: "append", "APPEND", "Append"
    payload: object,
    timestamp: timestamp
}
```

Event Types:
1. **append** - Add content to a specific line

```
payload: {
    newContent: string,
    startLine: number        // 1-indexed (line 1 = first line)
}
```

Important: If content exists at that line, **append** newContent to the end of that line (don't replace!).
2. **delete** - Delete all content from document

```
payload: {}  // Empty
```

Requirements:

- Process events in timestamp order (may arrive out-of-order)
- Update document.lastUpdated to latest event timestamp
- Handle case-insensitive event names
- Return updated document object

Important Edge Cases & Gotchas

IMPORTANT NOTE: The provided examples show lastUpdated incrementing by 1 (123456789 → 123456790), but event timestamps are in the billions (1641024000001). This appears to be an inconsistency in the problem statement.

Most logical interpretation: Set lastUpdated to the timestamp of the most recent event.

Action: Ask the interviewer to clarify this behavior before coding!

3 CRITICAL: Questions to Ask First!

Pro Tips

Before you start coding, ask these clarifying questions:

1. **lastUpdated behavior:** "Should lastUpdated be set to the timestamp of the latest event, or should it be incremented by 1?"
   - The examples show inconsistent behavior - clarify this!
   - Most logical: use the latest event's timestamp
2. **Invalid line numbers:** "What should happen if startLine is 0 or negative?"
   - Skip the event? Return error? Assume line 1?
3. **Event validation:** "Should I validate event structure and handle malformed events?"
   - Missing required fields?
   - Unknown event types?
4. **Same timestamp:** "If multiple events have the same timestamp, what determines their order?"
   - Use event_id as tiebreaker?
   - Stable sort (preserve original order)?
5. **Document mutation:** "Should I modify the input document in place or return a new copy?"
   - Safer to return a copy
   - But in-place is more efficient
6. **Content initialization:** "If document doesn't have a content field initially, should I initialize it as empty string?"
7. **Trailing newlines:** "Should the final content have a trailing newline character?"

These questions show you think critically and catch ambiguities!

4 Examples with Step-by-Step Trace

4.1 Example 1: Multiple Appends

Input:

```
events = [
    {"event_id": 1, "event_name": "append",
     "payload": {"newContent": "Line 1 ", "startLine": 1},
     "timestamp": 1641024000001},
    {"event_id": 2, "event_name": "APPEND",
     "payload": {"newContent": "Line 2 ", "startLine": 2},
     "timestamp": 1641024000002},
    {"event_id": 3, "event_name": "APPEND",
     "payload": {"newContent": "Line 3 ", "startLine": 3},
     "timestamp": 1641024000003}
]

document = {"title": "Lorem Ipsum", "lastUpdated": 123456789, "
    createdOn": 123456789}
```

Step-by-Step Execution:

```
Initial state:
  lines = []
  lastUpdated = 123456789

After Event 1 (append to line 1):
  lines = ["Line 1 "]
  lastUpdated = 1641024000001

After Event 2 (append to line 2):
  lines = ["Line 1 ", "Line 2 "]
  lastUpdated = 1641024000002

After Event 3 (append to line 3):
  lines = ["Line 1 ", "Line 2 ", "Line 3 "]
  lastUpdated = 1641024000003

Final: Join lines with "\n"
  content = "Line 1 \nLine 2 \nLine 3 "
```

Expected Output:

```
{
    "title": "Lorem Ipsum",
    "content": "Line 1 \nLine 2 \nLine 3 ",
    "lastUpdated": 1641024000003,
    "createdOn": 123456789
}
```

4.2 Example 2: Delete Event

Input:

```
events = [{"event_id": 1, "event_name": "delete", "timestamp":
    1641024000000}]
```

```

document = {
    "title": "Lorem Ipsum",
    "content": "This is Lorem ipsum",
    "lastUpdated": 123456789,
    "createdOn": 123456789
}
```

Step-by-Step:

```
Initial state:
  lines = ["This is Lorem ipsum"]

After Event 1 (delete):
  lines = []
  content = ""
  lastUpdated = 1641024000000
```

Expected Output:

```
{
    "title": "Lorem Ipsum",
    "content": "",
    "lastUpdated": 1641024000000,
    "createdOn": 123456789
}
```

5 Solution Strategy

5.1 Approach

1. **Sort events by timestamp** (handle out-of-order delivery)
2. **Initialize content** as list of lines
3. **Process each event** in order:
   - For `append`: Update/create line at specified position
   - For `delete`: Clear all content
4. **Join lines** back into string with \n
5. **Update lastUpdated** to latest event timestamp
6. **Return updated document**

5.2 Key Insights

- Use **list for lines** - O(1) indexing, easy modification
- Handle **1-indexed lines** (convert to 0-indexed: `line_idx = start_line - 1`)
- **Extend list** with empty strings if startLine exceeds current length
- Event names are **case-insensitive** (use `.lower()`)
- **Append, don't replace** - use `lines[idx] += new_content`
- Return a **copy** to avoid modifying input (safer)

5.3 Why List Over String Manipulation?

- **Strings are immutable** in Python - expensive to modify
- **List operations** are O(1) for append and indexed assignment
- **Only convert** to string once at the end
- Much more efficient for multiple operations

6 Complete Solution

Solution

Clean Production-Ready Implementation

```python
def execute(events, document):
    """
    Process document events and return updated document.

    This solution handles:
    - Out-of-order events (sorts by timestamp)
    - Case-insensitive event names
    - Line gaps (fills with empty strings)
    - Multiple appends to same line
    - Document without initial content

    Args:
        events: List of event dictionaries
        document: Document dictionary

    Returns:
        New document dictionary with processed changes
    """
    # Handle empty events - return unchanged
    if not events:
        return document

    # Sort events by timestamp (handle out-of-order delivery)
    sorted_events = sorted(events, key=lambda e: e["timestamp"])

    # Initialize content as list of lines
    content = document.get("content", "")
    lines = content.split("\n") if content else []

    # Track latest timestamp for lastUpdated
    latest_timestamp = document.get("lastUpdated", 0)

    # Process each event in chronological order
    for event in sorted_events:
        event_name = event["event_name"].lower()
        timestamp = event["timestamp"]

        if event_name == "append":
            payload = event["payload"]
            new_content = payload["newContent"]
            start_line = payload["startLine"]  # 1-indexed

            # Convert to 0-indexed
            line_idx = start_line - 1

            # Extend lines list if necessary (fill gaps with empty
                strings)
            while len(lines) <= line_idx:
                lines.append("")

            # Append to existing line content (don't replace!)
            lines[line_idx] += new_content

        elif event_name == "delete":
            # Clear all content
            lines = []

        # Update to most recent timestamp
        latest_timestamp = max(latest_timestamp, timestamp)
```

7 Robust Version with Validation

Solution

Enterprise Version with Error Handling

```python
def execute_robust(events, document):
    """
    Robust version with comprehensive validation and error handling.
    Use this if interviewer asks about production considerations.
    """
    # Validate inputs
    if not events:
        return document.copy()

    if not isinstance(events, list):
        raise ValueError("events must be a list")

    if not isinstance(document, dict):
        raise ValueError("document must be a dictionary")

    # Sort by timestamp (with fallback for missing timestamps)
    sorted_events = sorted(events, key=lambda e: e.get("timestamp", 0))

    # Initialize content
    content = document.get("content", "")
    lines = content.split("\n") if content else []

    # Remove trailing empty line if present (from content ending in \n)
    if lines and lines[-1] == "":
        lines = lines[:-1]

    latest_timestamp = document.get("lastUpdated", 0)

    # Process each event with validation
    for event in sorted_events:
        # Validate event structure
        if not isinstance(event, dict):
            continue  # Skip malformed events

        event_name = event.get("event_name", "").lower()
        timestamp = event.get("timestamp", 0)
        payload = event.get("payload", {})

        if event_name == "append":
            # Validate payload
            new_content = payload.get("newContent", "")
            start_line = payload.get("startLine", 1)

            # Validate line number (must be positive)
            if start_line < 1:
                continue  # Skip invalid line numbers

            line_idx = start_line - 1

            # Extend lines if needed
            while len(lines) <= line_idx:
                lines.append("")

            # Append content
            lines[line_idx] += new_content
```

8 Critical Edge Cases

Important Edge Cases & Gotchas

YOU MUST HANDLE THESE:

1. **Empty Events List**

```
events = []
# Should return document unchanged
```

2. **Out-of-Order Events** - CRITICAL!

```
events = [
    {"event_id": 2, ..., "timestamp": 102},  # Second
    {"event_id": 1, ..., "timestamp": 101}   # First
]
# Must sort by timestamp before processing!
```

3. **Case-Insensitive Event Names**

```
"append", "APPEND", "Append", "aPpEnD"  # All valid
# Use event_name.lower() for comparison
```

4. **Multiple Appends to Same Line** - MUST APPEND, NOT REPLACE!

```
events = [
    {"payload": {"newContent": "Hello ", "startLine": 1},
        ...},
    {"payload": {"newContent": "World", "startLine": 1}, ...}
]
# Result: lines[0] = "Hello World" NOT "World"
```

5. **Gap in Line Numbers**

```
events = [{"payload": {"newContent": "Line5", "startLine": 5},
    ...}]
# Result: lines = ["", "", "", "", "Line5"]
# Fill gaps with empty strings!
```

6. **Delete Then Append**

```
events = [
    {"event_name": "delete", "timestamp": 100},
    {"event_name": "append", "payload": {..., "startLine": 1},
        "timestamp": 101}
]
# Delete clears lines = [], then append starts fresh
```

7. **Document Without Content Field**

```
document = {"title": "Test", "lastUpdated": 0, "createdOn": 0}
# No "content" key initially
# Use document.get("content", "") to handle safely
```

8. **Empty Payload or Missing Fields**

```
{"event_name": "delete", "timestamp": 100}
# Delete has no payload - that's valid
# Use payload.get("key", default) for safety
```

9. **Line Number 0 or Negative**

9 Comprehensive Test Suite

Test Cases

```python
def test_document_processor():
    """Complete test suite covering all edge cases."""

    print("Running comprehensive test suite...")

    # Test 1: Basic append to empty document
    print("\n[Test 1] Basic append")
    events = [{
        "event_id": 1,
        "event_name": "append",
        "payload": {"newContent": "Hello", "startLine": 1},
        "timestamp": 100
    }]
    doc = {"title": "Test", "lastUpdated": 0, "createdOn": 0}
    result = execute(events, doc)
    assert result["content"] == "Hello", f"Expected 'Hello', got
        '{result['content']}'"
    assert result["lastUpdated"] == 100, f"Expected 100, got {
        result['lastUpdated']}"
    print("  PASSED")

    # Test 2: Multiple appends to different lines
    print("\n[Test 2] Multiple lines")
    events = [
        {"event_id": 1, "event_name": "append",
         "payload": {"newContent": "Line1", "startLine": 1},
         "timestamp": 100},
        {"event_id": 2, "event_name": "append",
         "payload": {"newContent": "Line2", "startLine": 2},
         "timestamp": 101}
    ]
    doc = {"title": "Test", "lastUpdated": 0, "createdOn": 0}
    result = execute(events, doc)
    assert result["content"] == "Line1\nLine2"
    print("  PASSED")

    # Test 3: Multiple appends to SAME line (critical!)
    print("\n[Test 3] Same line appends (CRITICAL)")
    events = [
        {"event_id": 1, "event_name": "append",
         "payload": {"newContent": "Hello ", "startLine": 1},
         "timestamp": 100},
        {"event_id": 2, "event_name": "append",
         "payload": {"newContent": "World", "startLine": 1},
         "timestamp": 101}
    ]
    doc = {"title": "Test", "lastUpdated": 0, "createdOn": 0}
    result = execute(events, doc)
    assert result["content"] == "Hello World", \
        f"Must APPEND not replace! Got: '{result['content']}'"
    print("  PASSED")

    # Test 4: Delete event
    print("\n[Test 4] Delete clears all content")
    events = [{"event_id": 1, "event_name": "delete", "timestamp":
        100}]
    doc = {"title": "Test", "content": "Some content",
           "lastUpdated": 0, "createdOn": 0}
    result = execute(events, doc)
    assert result["content"] == ""
```

10 Debugging Strategy

Pro Tips

If Your Tests Are Failing:

1. **Add Debug Prints**

```
for event in sorted_events:
    print(f"Processing: {event['event_name']} at line {event.
        get('payload', {}).get('startLine')}")
    print(f"Lines before: {lines}")
    # ... process event ...
    print(f"Lines after: {lines}")
    print()
```

2. **Check Event Sorting**

```
print("Events before sorting:")
for e in events:
    print(f"  {e['event_id']}: timestamp={e['timestamp']}")

sorted_events = sorted(events, key=lambda e: e["timestamp"])

print("\nEvents after sorting:")
for e in sorted_events:
    print(f"  {e['event_id']}: timestamp={e['timestamp']}")
```

3. **Verify Line Indexing**

```
print(f"startLine={start_line} (1-indexed)")
print(f"line_idx={line_idx} (0-indexed)")
print(f"lines length before: {len(lines)}")
# ... extend lines ...
print(f"lines length after: {len(lines)}")
```

4. **Check Append vs Replace**

```
print(f"Line {line_idx} before: '{lines[line_idx]}'")
lines[line_idx] += new_content  # Should use +=, not =
print(f"Line {line_idx} after: '{lines[line_idx]}'")
```

5. **Verify Final Join**

```
print(f"Lines array: {lines}")
content = "\n".join(lines)
print(f"Joined content: '{content}'")
print(f"Content length: {len(content)}")
```

Common Mistakes:

- Forgetting to sort events by timestamp
- Using `=` instead of `+=` for append
- Off-by-one error with 1-indexed vs 0-indexed
- Not handling case-insensitive event names
- Creating extra empty lines with improper split/join

11 Follow-Up Questions & Variations

11.1 Expected Follow-Ups

1. **What if events can arrive significantly out of order?**
   - Current solution handles this with sorting
   - For streaming: use priority queue or buffering window
   - Trade-off: latency vs correctness
2. **How would you handle millions of events?**
   - Batch processing
   - Periodic snapshots + incremental updates
   - Event sourcing pattern
   - Compaction/aggregation of old events
3. **What about concurrent editing by multiple users?**
   - Operational Transform (OT)
   - Conflict-free Replicated Data Types (CRDTs)
   - Last-write-wins with timestamps
   - Use event_id as tiebreaker
4. **How would you optimize for very large documents?**
   - Rope data structure instead of list
   - Lazy loading of content
   - Chunk-based storage
   - Only process visible viewport
5. **What if we add more event types?**
   - `insert`: Insert at character position within line
   - `delete_range`: Delete specific line range
   - `replace`: Replace content at line
   - `format`: Apply formatting (bold, italic)

11.2 Variation 1: Insert at Character Position

New Event Type:

```
{
    "event_name": "insert",
    "payload": {
        "newContent": "text",
        "startLine": 2,
        "position": 5  # Character position in line (0-indexed)
    },
    "timestamp": 100
}
```

Implementation:

```python
elif event_name == "insert":
    new_content = payload["newContent"]
    start_line = payload["startLine"]
    position = payload["position"]

    line_idx = start_line - 1
    while len(lines) <= line_idx:
        lines.append("")

    line = lines[line_idx]
    # Insert at character position
    lines[line_idx] = line[:position] + new_content + line[position:]
```

11.3 Variation 2: Delete Specific Lines

New Event Type:

```python
{
    "event_name": "delete_range",
    "payload": {
        "startLine": 2,
        "endLine": 4    # Inclusive
    },
    "timestamp": 100
}
```

Implementation:

```python
elif event_name == "delete_range":
    start_line = payload["startLine"]
    end_line = payload["endLine"]

    start_idx = start_line - 1
    end_idx = end_line - 1

    # Delete lines in range
    if start_idx < len(lines):
        del lines[start_idx:min(end_idx + 1, len(lines))]
```

11.4 Variation 3: Replace Line Content

New Event Type:

```python
{
    "event_name": "replace",
    "payload": {
        "newContent": "completely new text",
        "startLine": 3
    },
    "timestamp": 100
}
```

Implementation:

```python
elif event_name == "replace":
    new_content = payload["newContent"]
    start_line = payload["startLine"]

```

```
    line_idx = start_line - 1
    while len(lines) <= line_idx:
        lines.append("")

    # Replace entire line (use = instead of +=)
    lines[line_idx] = new_content
```

12 Alternative Implementations

12.1 JavaScript/TypeScript Version

```javascript
function execute(events, document) {
    // Handle empty events
    if (!events || events.length === 0) {
        return { ...document };
    }

    // Sort by timestamp
    const sortedEvents = [...events].sort((a, b) =>
        a.timestamp - b.timestamp
    );

    // Initialize content
    let content = document.content || "";
    let lines = content ? content.split("\n") : [];

    let latestTimestamp = document.lastUpdated || 0;

    // Process events
    for (const event of sortedEvents) {
        const eventName = event.event_name.toLowerCase();
        const timestamp = event.timestamp;

        if (eventName === "append") {
            const { newContent, startLine } = event.payload;
            const lineIdx = startLine - 1;

            // Extend lines array if needed
            while (lines.length <= lineIdx) {
                lines.push("");
            }

            // Append to line
            lines[lineIdx] += newContent;

        } else if (eventName === "delete") {
            lines = [];
        }

        latestTimestamp = Math.max(latestTimestamp, timestamp);
    }

    // Return new document (don't mutate input)
    return {
        ...document,
        content: lines.join("\n"),
        lastUpdated: latestTimestamp
    };
}

// TypeScript version with types
interface Event {
    event_id: number;
    event_name: string;
    payload: {
```

```
        newContent?: string;
        startLine?: number;
    };
    timestamp: number;
}

interface Document {
    title: string;
    content?: string;
    lastUpdated: number;
    createdOn: number;
}

function executeTyped(events: Event[], document: Document): Document {
    // Same implementation as above
    // TypeScript provides compile-time type safety
}
```

12.2 Object-Oriented Approach

```
class DocumentProcessor:
    """OOP approach for document event processing."""

    def __init__(self, document):
        self.document = document.copy()
        self.lines = self._init_lines()

    def _init_lines(self):
        """Initialize lines from document content."""
        content = self.document.get("content", "")
        return content.split("\n") if content else []

    def process_events(self, events):
        """Process all events and return updated document."""
        if not events:
            return self.document

        sorted_events = sorted(events, key=lambda e: e["timestamp"])

        for event in sorted_events:
            self._process_event(event)

        return self._finalize()

    def _process_event(self, event):
        """Process single event based on type."""
        event_name = event["event_name"].lower()

        if event_name == "append":
            self._handle_append(event)
        elif event_name == "delete":
            self._handle_delete(event)
        # Easy to add more event types here

    def _handle_append(self, event):
        """Handle append event."""
        payload = event["payload"]
```

```
        new_content = payload["newContent"]
        start_line = payload["startLine"]

        line_idx = start_line - 1

        # Extend lines if needed
        while len(self.lines) <= line_idx:
            self.lines.append("")

        self.lines[line_idx] += new_content
        self.document["lastUpdated"] = event["timestamp"]

    def _handle_delete(self, event):
        """Handle delete event."""
        self.lines = []
        self.document["lastUpdated"] = event["timestamp"]

    def _finalize(self):
        """Finalize and return document."""
        self.document["content"] = "\n".join(self.lines)
        return self.document


# Usage
def execute(events, document):
    processor = DocumentProcessor(document)
    return processor.process_events(events)
```

13 System Design Discussion

13.1 Real-World ClickUp Architecture

In production, ClickUp likely uses:

1. **Event Store / Event Sourcing**
    - Kafka or similar for event streaming
    - Permanent event log (source of truth)
    - Can replay events to rebuild state
    - Enables time-travel debugging
2. **CRDT (Conflict-free Replicated Data Types)**
    - Handle concurrent edits from multiple users
    - Eventual consistency without conflicts
    - Examples: Yjs, Automerge
    - Used by: Figma, Notion, Google Docs
3. **Operational Transform (OT)**
    - Transform conflicting operations
    - Maintain causal ordering
    - More complex but deterministic
    - Used by: Google Docs originally
4. **Snapshot + Delta Pattern**
    - Store periodic snapshots
    - Apply only recent events
    - Faster recovery and queries
    - Reduce memory usage
5. **WebSocket for Real-time Sync**
    - Push events to all connected clients
    - Low latency updates
    - Handle reconnection gracefully

13.2 Scalability Challenges

- **Large Documents**: Millions of characters, thousands of lines
- **High Event Rate**: Hundreds of events per second per document
- **Many Concurrent Users**: 10+ people editing simultaneously
- **Offline Support**: Sync when reconnected, handle conflicts
- **Undo/Redo**: Maintain operation history efficiently
- **Performance**: Sub-second response time even for large docs

13.3 Data Structures for Large Documents

- **Rope**: Tree-based string for efficient insertions/deletions
- **Gap Buffer**: Used by Emacs, good for cursor-based editing
- **Piece Table**: Used by VS Code, great for undo/redo
- **CRDT Text**: Yjs uses linked list with tombstones

14 Interview Execution Strategy

14.1 Time Allocation (60 minutes)

- **5 min**: Clarify requirements, ask questions, confirm understanding
- **3 min**: Discuss approach, mention data structures, complexity
- **2 min**: Write function signature and comments
- **25 min**: Implement core solution (clean, working code)
- **10 min**: Write and run tests (catch bugs early!)
- **5 min**: Add error handling and edge cases
- **10 min**: Follow-up questions, optimizations, discussion

14.2 Before You Code

Pro Tips

The First 5 Minutes Are Critical!

1. **Clarify Requirements**
   - "Should lastUpdated use the event timestamp or increment?"
   - "What happens with invalid line numbers?"
   - "Should I validate event structure?"
2. **Confirm Examples**
   - Walk through Example 1 verbally
   - Confirm expected output matches your understanding
   - Note any inconsistencies in examples
3. **Discuss Approach**
   - "I'll sort events by timestamp first..."
   - "I'll use a list for lines for O(1) indexing..."
   - "Time complexity will be O(n log n) for sorting..."
4. **Mention Edge Cases**
   - "I'll need to handle out-of-order events..."
   - "Case-insensitive event names..."
   - "Multiple appends to same line..."

This shows you're thinking critically and builds trust!

14.3 While Coding

1. **Think Out Loud**
   - "Now I'll sort the events by timestamp..."

- "Converting to 0-indexed here because Python lists..."
- "Using += here to append, not replace..."

2. **Write Clean Code**
   - Descriptive variable names: `line_idx`, not `i`
   - Add comments for non-obvious logic
   - Consistent spacing and formatting
3. **Handle Errors Gracefully**
   - Use `.get()` for optional dictionary keys
   - Check for None/empty before processing
   - Validate inputs if time permits
4. **Ask If Stuck**
   - "I'm debating between X and Y, which would you prefer?"
   - "Should I prioritize robustness or simplicity here?"
   - Don't sit in silence - communicate!

14.4 After Coding

1. **Test Immediately**
   - Run provided examples first
   - Test edge cases (empty, out-of-order, same line)
   - Fix any bugs found
2. **Walk Through Code**
   - Explain your solution at high level
   - Point out key design decisions
   - Mention trade-offs considered
3. **Discuss Improvements**
   - "For production, I'd add validation..."
   - "Could optimize with rope data structure..."
   - "Would need CRDT for real-time collaboration..."
4. **Handle Follow-Ups**
   - Be ready for variations (insert, delete range)
   - Explain how to extend solution
   - Discuss system design implications

15 Communication Checklist

15.1 Before Interview

- ☐ Practiced core solution multiple times (can code in 20-25 min)
- ☐ Memorized critical edge cases
- ☐ Tested with all provided examples
- ☐ Understand time/space complexity
- ☐ Reviewed follow-up variations
- ☐ Comfortable with both Python and JavaScript
- ☐ Practiced explaining thought process out loud
- ☐ Prepared clarifying questions to ask

15.2 During Interview - Execution

- ☐ Asked clarifying questions upfront
- ☐ Confirmed understanding of examples
- ☐ Explained approach before coding
- ☐ Thought out loud while implementing
- ☐ Handled edge cases explicitly
- ☐ Wrote clean, readable code
- ☐ Tested code with examples
- ☐ Discussed complexity analysis
- ☐ Proposed optimizations
- ☐ Asked intelligent follow-up questions

15.3 During Interview - Communication

- ☐ Maintained conversational tone
- ☐ Explained reasoning for decisions
- ☐ Asked for feedback/hints when stuck
- ☐ Admitted when unsure (don't fake it)
- ☐ Showed enthusiasm and engagement
- ☐ Treated interviewer as collaborator

16 Quick Reference Card

16.1 Key Points to Remember

1. **ALWAYS sort events by timestamp first!**
2. **Use list for lines** (not string manipulation)
3. **Lines are 1-indexed** in problem, 0-indexed in Python
4. **Case-insensitive event names** - use `.lower()`
5. **APPEND to line** with `+=`, DON'T REPLACE with `=`
6. **Fill gaps** with empty strings when extending
7. **Update lastUpdated** to latest event timestamp
8. **Return a copy** - don't mutate input document

16.2 Common Mistakes to Avoid

- X Forgetting to sort events
- X Not handling case-insensitive event names
- X Using `=` instead of `+=` for append (replaces instead of appends!)
- X Off-by-one errors with 1-indexed vs 0-indexed
- X Not handling empty events or missing content
- X Modifying input document directly
- X Not testing edge cases

16.3 Python Quick Reference

```
# Safe dictionary access
content = document.get("content", "")  # Default empty string
payload = event.get("payload", {})     # Default empty dict

# Case-insensitive comparison
event_name = event["event_name"].lower()

# List operations
lines = []
lines.append("new")            # Add to end
lines[idx] += "text"           # Append to existing
while len(lines) <= idx:       # Extend with gaps
    lines.append("")

# String operations
lines = content.split("\n")    # Split into list
content = "\n".join(lines)     # Join back to string

# Sorting
sorted_events = sorted(events, key=lambda e: e["timestamp"])

# List comprehension
valid = [e for e in events if e.get("timestamp") is not None]
```

17 Final Preparation Checklist

17.1 Technical Readiness

- ☐ Can implement solution in under 25 minutes
- ☐ All 12+ test cases pass without bugs
- ☐ Understand why each edge case matters
- ☐ Can explain time/space complexity
- ☐ Know how to extend for new event types
- ☐ Comfortable with alternative approaches (OOP, JS)

17.2 Interview Skills

- ☐ Practiced asking clarifying questions
- ☐ Can explain approach before coding
- ☐ Comfortable thinking out loud
- ☐ Know how to debug when tests fail
- ☐ Can discuss system design implications
- ☐ Ready for follow-up variations

17.3 Day Before Interview

- ☐ Do one final practice run (timed, 30 min)
- ☐ Review edge cases one more time
- ☐ Read through this guide's key sections
- ☐ Prepare 2-3 questions to ask interviewer
- ☐ Get good sleep - fresh mind is critical!

17.4 Interview Day

- ☐ Test internet connection and mic/camera
- ☐ Have this guide open for reference (if allowed)
- ☐ Water nearby, comfortable environment
- ☐ Positive mindset - you've got this!

You Are Ready!

You've prepared thoroughly. You know the solution inside-out. You understand the edge cases. You can handle follow-ups. Now trust your preparation and show them what you can do!

Remember:

- The interviewer wants you to succeed
- They're evaluating how you think, not just coding speed
- Communication matters as much as the solution
- It's okay to ask questions - that's smart!
- One bug doesn't fail you - recovery matters

Good luck with your ClickUp interview!