

ClickUp Backend Interview

Document Event Processing

Complete Preparation Guide - November 30, 2024 (v2)

November 30, 2025

Contents

1 Interview Overview	2
1.1 What to Expect	2
1.2 Success Criteria	2
2 CRITICAL: Questions to Ask First!	3
3 Core Problem: Document Event Processor	4
4 Examples with Step-by-Step Trace	6
4.1 Example 1: Multiple Appends	6
4.2 Example 2: Delete Event	6
5 Solution Strategy	8
5.1 Approach	8
5.2 Key Insights	8
5.3 Why List Over String Manipulation?	8
6 Complete Solution	10
7 Robust Version with Validation	12
8 Critical Edge Cases	14
9 Comprehensive Test Suite	16
10 Debugging Strategy	18
11 Follow-Up Questions & Variations	19
11.1 Expected Follow-Ups	19
11.2 Variation 1: Insert at Character Position	19
11.3 Variation 2: Delete Specific Lines	20
11.4 Variation 3: Replace Line Content	20
12 Alternative Implementations	22
12.1 JavaScript/TypeScript Version	22
12.2 Object-Oriented Approach	23

13 System Design Discussion	25
13.1 Real-World ClickUp Architecture	25
13.2 Scalability Challenges	25
13.3 Data Structures for Large Documents	26
14 Interview Execution Strategy	27
14.1 Time Allocation (60 minutes)	27
14.2 Before You Code	27
14.3 While Coding	27
14.4 After Coding	28
15 Communication Checklist	29
15.1 Before Interview	29
15.2 During Interview - Execution	29
15.3 During Interview - Communication	29
16 Quick Reference Card	30
16.1 Key Points to Remember	30
16.2 Common Mistakes to Avoid	30
16.3 Python Quick Reference	30
17 Final Preparation Checklist	31
17.1 Technical Readiness	31
17.2 Interview Skills	31
17.3 Day Before Interview	31
17.4 Interview Day	31

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 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!

3 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 \rightarrow 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!

4 Examples with Step-by-Step Trace

4.1 Example 1: Multiple Appends

Input:

```
1 events = [  
2   {"event_id": 1, "event_name": "append",  
3     "payload": {"newContent": "Line 1 ", "startLine": 1},  
4     "timestamp": 1641024000001},  
5   {"event_id": 2, "event_name": "APPEND",  
6     "payload": {"newContent": "Line 2 ", "startLine": 2},  
7     "timestamp": 1641024000002},  
8   {"event_id": 3, "event_name": "APPEND",  
9     "payload": {"newContent": "Line 3 ", "startLine": 3},  
10    "timestamp": 1641024000003}  
11 ]  
12  
13 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:

```
1 {  
2   "title": "Lorem Ipsum",  
3   "content": "Line 1 \nLine 2 \nLine 3 ",  
4   "lastUpdated": 1641024000003,  
5   "createdOn": 123456789  
6 }
```

4.2 Example 2: Delete Event

Input:

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

```
2
3 document = {
4     "title": "Lorem Ipsum",
5     "content": "This is Lorem ipsum",
6     "lastUpdated": 123456789,
7     "createdOn": 123456789
8 }
```

Step-by-Step:

Initial state:

```
lines = ["This is Lorem ipsum"]
```

After Event 1 (delete):

```
lines = []
content = ""
lastUpdated = 1641024000000
```

Expected Output:

```
1 {
2     "title": "Lorem Ipsum",
3     "content": "",
4     "lastUpdated": 1641024000000,
5     "createdOn": 123456789
6 }
```


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

```
1 def execute(events, document):
2     """
3     Process document events and return updated document.
4
5     This solution handles:
6     - Out-of-order events (sorts by timestamp)
7     - Case-insensitive event names
8     - Line gaps (fills with empty strings)
9     - Multiple appends to same line
10    - Document without initial content
11
12    Args:
13        events: List of event dictionaries
14        document: Document dictionary
15
16    Returns:
17        New document dictionary with processed changes
18    """
19    # Handle empty events - return unchanged
20    if not events:
21        return document
22
23    # Sort events by timestamp (handle out-of-order delivery)
24    sorted_events = sorted(events, key=lambda e: e["timestamp"])
25
26    # Initialize content as list of lines
27    content = document.get("content", "")
28    lines = content.split("\n") if content else []
29
30    # Track latest timestamp for lastUpdated
31    latest_timestamp = document.get("lastUpdated", 0)
32
33    # Process each event in chronological order
34    for event in sorted_events:
35        event_name = event["event_name"].lower()
36        timestamp = event["timestamp"]
37
38        if event_name == "append":
39            payload = event["payload"]
40            new_content = payload["newContent"]
41            start_line = payload["startLine"] # 1-indexed
42
43            # Convert to 0-indexed
44            line_idx = start_line - 1
45
46            # Extend lines list if necessary (fill gaps with empty
47            # strings)
48            while len(lines) <= line_idx:
49                lines.append("")
50
51            # Append to existing line content (don't replace!)
52            lines[line_idx] += new_content
53
54        elif event_name == "delete":
55            # Clear all content
56            lines = []
57
58    # Update to most recent timestamp
59    latest_timestamp = max(latest_timestamp, timestamp)
```


7 Robust Version with Validation

Solution

Enterprise Version with Error Handling

```
1 def execute_robust(events, document):
2     """
3     Robust version with comprehensive validation and error
4     handling.
5     Use this if interviewer asks about production considerations.
6     """
7     # Validate inputs
8     if not events:
9         return document.copy()
10
11     if not isinstance(events, list):
12         raise ValueError("events must be a list")
13
14     if not isinstance(document, dict):
15         raise ValueError("document must be a dictionary")
16
17     # Sort by timestamp (with fallback for missing timestamps)
18     sorted_events = sorted(events, key=lambda e: e.get("timestamp", 0))
19
20     # Initialize content
21     content = document.get("content", "")
22     lines = content.split("\n") if content else []
23
24     # Remove trailing empty line if present (from content ending
25     # in \n)
26     if lines and lines[-1] == "":
27         lines = lines[:-1]
28
29     latest_timestamp = document.get("lastUpdated", 0)
30
31     # Process each event with validation
32     for event in sorted_events:
33         # Validate event structure
34         if not isinstance(event, dict):
35             continue # Skip malformed events
36
37         event_name = event.get("event_name", "").lower()
38         timestamp = event.get("timestamp", 0)
39         payload = event.get("payload", {})
40
41         if event_name == "append":
42             # Validate payload
43             new_content = payload.get("newContent", "")
44             start_line = payload.get("startLine", 1)
45
46             # Validate line number (must be positive)
47             if start_line < 1:
48                 continue # Skip invalid line numbers
49
50             line_idx = start_line - 1
51
52             # Extend lines if needed
53             while len(lines) <= line_idx:
54                 lines.append("")
55
56             # Append content
57             lines[line_idx] += new_content
```


8 Critical Edge Cases

Important Edge Cases & Gotchas

YOU MUST HANDLE THESE:

1. Empty Events List

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

2. Out-of-Order Events - CRITICAL!

```
1 events = [
2     {"event_id": 2, ..., "timestamp": 102}, # Second
3     {"event_id": 1, ..., "timestamp": 101}  # First
4 ]
5 # Must sort by timestamp before processing!
```

3. Case-Insensitive Event Names

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

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

```
1 events = [
2     {"payload": {"newContent": "Hello ", "startLine": 1},
3     ...},
4     {"payload": {"newContent": "World", "startLine": 1}, ...}
5 ]
6 # Result: lines[0] = "Hello World" NOT "World"
```

5. Gap in Line Numbers

```
1 events = [{"payload": {"newContent": "Line5", "startLine": 5},
2     ...}]
3 # Result: lines = ["", "", "", "", "Line5"]
4 # Fill gaps with empty strings!
```

6. Delete Then Append

```
1 events = [
2     {"event_name": "delete", "timestamp": 100},
3     {"event_name": "append", "payload": {..., "startLine": 1},
4     "timestamp": 101}
5 ]
6 # Delete clears lines = [], then append starts fresh
```

7. Document Without Content Field

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

8. Empty Payload or Missing Fields

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

9. Line Number 0 or Negative

9 Comprehensive Test Suite

Test Cases

```
1 def test_document_processor():
2     """Complete test suite covering all edge cases."""
3
4     print("Running comprehensive test suite...")
5
6     # Test 1: Basic append to empty document
7     print("\n[Test 1] Basic append")
8     events = [{
9         "event_id": 1,
10        "event_name": "append",
11        "payload": {"newContent": "Hello", "startLine": 1},
12        "timestamp": 100
13    }]
14    doc = {"title": "Test", "lastUpdated": 0, "createdOn": 0}
15    result = execute(events, doc)
16    assert result["content"] == "Hello", f"Expected 'Hello', got {
17        'result['content']}"
18    assert result["lastUpdated"] == 100, f"Expected 100, got {
19        result['lastUpdated']}"
20    print("    PASSED")
21
22    # Test 2: Multiple appends to different lines
23    print("\n[Test 2] Multiple lines")
24    events = [
25        {"event_id": 1, "event_name": "append",
26         "payload": {"newContent": "Line1", "startLine": 1},
27         "timestamp": 100},
28        {"event_id": 2, "event_name": "append",
29         "payload": {"newContent": "Line2", "startLine": 2},
30         "timestamp": 101}
31    ]
32    doc = {"title": "Test", "lastUpdated": 0, "createdOn": 0}
33    result = execute(events, doc)
34    assert result["content"] == "Line1\nLine2"
35    print("    PASSED")
36
37    # Test 3: Multiple appends to SAME line (critical!)
38    print("\n[Test 3] Same line appends (CRITICAL)")
39    events = [
40        {"event_id": 1, "event_name": "append",
41         "payload": {"newContent": "Hello ", "startLine": 1},
42         "timestamp": 100},
43        {"event_id": 2, "event_name": "append",
44         "payload": {"newContent": "World", "startLine": 1},
45         "timestamp": 101}
46    ]
47    doc = {"title": "Test", "lastUpdated": 0, "createdOn": 0}
48    result = execute(events, doc)
49    assert result["content"] == "Hello World", \
50        f"Must APPEND not replace! Got: '{result['content']}'"
51    print("    PASSED")
52
53    # Test 4: Delete event
54    print("\n[Test 4] Delete clears all content")
55    events = [{"event_id": 1, "event_name": "delete", "timestamp":
56        100}]
57    doc = {"title": "Test", "content": "Some content",
58        "lastUpdated": 0, "createdOn": 0}
59    result = execute(events, doc)
60    assert result["content"] == ""
```


10 Debugging Strategy

Pro Tips

If Your Tests Are Failing:

1. Add Debug Prints

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

2. Check Event Sorting

```
1 print("Events before sorting:")
2 for e in events:
3     print(f"    {e['event_id']}: timestamp={e['timestamp']}")
4
5 sorted_events = sorted(events, key=lambda e: e["timestamp"])
6
7 print("\nEvents after sorting:")
8 for e in sorted_events:
9     print(f"    {e['event_id']}: timestamp={e['timestamp']}")
```

3. Verify Line Indexing

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

4. Check Append vs Replace

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

5. Verify Final Join

```
1 print(f"Lines array: {lines}")
2 content = "\n".join(lines)
3 print(f"Joined content: '{content}'")
4 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:

```
1 {
2   "event_name": "insert",
3   "payload": {
4     "newContent": "text",
5     "startLine": 2,
6     "position": 5 # Character position in line (0-indexed)
7   },
8   "timestamp": 100
9 }
```

Implementation:

```

1 elif event_name == "insert":
2     new_content = payload["newContent"]
3     start_line = payload["startLine"]
4     position = payload["position"]
5
6     line_idx = start_line - 1
7     while len(lines) <= line_idx:
8         lines.append("")
9
10    line = lines[line_idx]
11    # Insert at character position
12    lines[line_idx] = line[:position] + new_content + line[position:]

```

11.3 Variation 2: Delete Specific Lines

New Event Type:

```

1 {
2     "event_name": "delete_range",
3     "payload": {
4         "startLine": 2,
5         "endLine": 4      # Inclusive
6     },
7     "timestamp": 100
8 }

```

Implementation:

```

1 elif event_name == "delete_range":
2     start_line = payload["startLine"]
3     end_line = payload["endLine"]
4
5     start_idx = start_line - 1
6     end_idx = end_line - 1
7
8     # Delete lines in range
9     if start_idx < len(lines):
10        del lines[start_idx:min(end_idx + 1, len(lines))]

```

11.4 Variation 3: Replace Line Content

New Event Type:

```

1 {
2     "event_name": "replace",
3     "payload": {
4         "newContent": "completely new text",
5         "startLine": 3
6     },
7     "timestamp": 100
8 }

```

Implementation:

```

1 elif event_name == "replace":
2     new_content = payload["newContent"]
3     start_line = payload["startLine"]
4

```

```
5     line_idx = start_line - 1
6     while len(lines) <= line_idx:
7         lines.append("")
8
9     # Replace entire line (use = instead of +=)
10    lines[line_idx] = new_content
```

12 Alternative Implementations

12.1 JavaScript/TypeScript Version

```
1 function execute(events, document) {
2   // Handle empty events
3   if (!events || events.length === 0) {
4     return { ...document };
5   }
6
7   // Sort by timestamp
8   const sortedEvents = [...events].sort((a, b) =>
9     a.timestamp - b.timestamp
10  );
11
12  // Initialize content
13  let content = document.content || "";
14  let lines = content ? content.split("\n") : [];
15
16  let latestTimestamp = document.lastUpdated || 0;
17
18  // Process events
19  for (const event of sortedEvents) {
20    const eventName = event.event_name.toLowerCase();
21    const timestamp = event.timestamp;
22
23    if (eventName === "append") {
24      const { newContent, startLine } = event.payload;
25      const lineIdx = startLine - 1;
26
27      // Extend lines array if needed
28      while (lines.length <= lineIdx) {
29        lines.push("");
30      }
31
32      // Append to line
33      lines[lineIdx] += newContent;
34
35    } else if (eventName === "delete") {
36      lines = [];
37    }
38
39    latestTimestamp = Math.max(latestTimestamp, timestamp);
40  }
41
42  // Return new document (don't mutate input)
43  return {
44    ...document,
45    content: lines.join("\n"),
46    lastUpdated: latestTimestamp
47  };
48 }
49
50 // TypeScript version with types
51 interface Event {
52   event_id: number;
53   event_name: string;
54   payload: {
```

```

55         newContent?: string;
56         startLine?: number;
57     };
58     timestamp: number;
59 }
60
61 interface Document {
62     title: string;
63     content?: string;
64     lastUpdated: number;
65     createdOn: number;
66 }
67
68 function executeTyped(events: Event[], document: Document): Document {
69     // Same implementation as above
70     // TypeScript provides compile-time type safety
71 }

```

12.2 Object-Oriented Approach

```

1  class DocumentProcessor:
2      """OOP approach for document event processing."""
3
4      def __init__(self, document):
5          self.document = document.copy()
6          self.lines = self._init_lines()
7
8      def _init_lines(self):
9          """Initialize lines from document content."""
10         content = self.document.get("content", "")
11         return content.split("\n") if content else []
12
13     def process_events(self, events):
14         """Process all events and return updated document."""
15         if not events:
16             return self.document
17
18         sorted_events = sorted(events, key=lambda e: e["timestamp"])
19
20         for event in sorted_events:
21             self._process_event(event)
22
23         return self._finalize()
24
25     def _process_event(self, event):
26         """Process single event based on type."""
27         event_name = event["event_name"].lower()
28
29         if event_name == "append":
30             self._handle_append(event)
31         elif event_name == "delete":
32             self._handle_delete(event)
33         # Easy to add more event types here
34
35     def _handle_append(self, event):
36         """Handle append event."""
37         payload = event["payload"]

```



```

38     new_content = payload["newContent"]
39     start_line = payload["startLine"]
40
41     line_idx = start_line - 1
42
43     # Extend lines if needed
44     while len(self.lines) <= line_idx:
45         self.lines.append("")
46
47     self.lines[line_idx] += new_content
48     self.document["lastUpdated"] = event["timestamp"]
49
50     def _handle_delete(self, event):
51         """Handle delete event."""
52         self.lines = []
53         self.document["lastUpdated"] = event["timestamp"]
54
55     def _finalize(self):
56         """Finalize and return document."""
57         self.document["content"] = "\n".join(self.lines)
58         return self.document
59
60
61 # Usage
62 def execute(events, document):
63     processor = DocumentProcessor(document)
64     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

```
1 # Safe dictionary access
2 content = document.get("content", "") # Default empty string
3 payload = event.get("payload", {})    # Default empty dict
4
5 # Case-insensitive comparison
6 event_name = event["event_name"].lower()
7
8 # List operations
9 lines = []
10 lines.append("new")                # Add to end
11 lines[idx] += "text"               # Append to existing
12 while len(lines) <= idx:          # Extend with gaps
13     lines.append("")
14
15 # String operations
16 lines = content.split("\n")        # Split into list
17 content = "\n".join(lines)         # Join back to string
18
19 # Sorting
20 sorted_events = sorted(events, key=lambda e: e["timestamp"])
21
22 # List comprehension
23 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!