Faire Interview Questions
Backend Software Engineer

Compiled from 1point3acres.com

November 9, 2025

Overview

This document contains interview questions for Backend Software Engineer positions at Faire, compiled from multiple sources:

- 1point3acres.com - Interview experiences from 2022-2025
- Prepfully.com - Behavioral interview questions
- Web search results - CodeSignal OA format and general interview structure

The information represents publicly available interview experiences and may not reflect current or complete interview content. Questions span multiple positions: Backend Engineer, Full Stack Engineer, Data Engineer, and Frontend Engineer/Intern roles.

Interview Process

Complete Interview Timeline

1. **Application & Recruiter Screening**
   - Initial phone call with recruiter (15-20 minutes)
   - Discussion of background, role expectations, compensation range
   - Response time: Usually 1-3 days
2. **Online Assessment (OA)**
   - CodeSignal assessment (4 questions, 70 minutes)
   - Passing score: 750+/850
   - Must be completed within 3-5 days of invitation
3. **Technical Phone Screen**
   - Duration: 45-60 minutes
   - 1-2 coding questions (medium difficulty)
   - Live coding in shared editor (CoderPad or similar)
   - Focus on problem-solving, code quality, edge cases

4. **Virtual Onsite (3-4 rounds, 2.5-3 hours total)**
    - **Round 1:** Coding (45-60 min)
        - Medium to Hard difficulty
        - Strong emphasis on test cases and edge cases
        - May include debugging existing code
    - **Round 2:** Coding (45-60 min)
        - Similar format to Round 1
        - Different problem domain
    - **Round 3:** Behavioral/Cultural Fit (30-45 min)
        - STAR method questions
        - Team collaboration scenarios
        - Why Faire? Career goals
    - **Round 4 (Senior/Staff):** System Design or Pipeline Design (45-60 min)
        - Backend: System design (e.g., design a like functionality)
        - Data Engineering: Pipeline design
        - Focus on scalability, trade-offs, communication
5. **Final Decision**
    - Response time: Same day to 3 days
    - Offer includes: Base salary, equity, benefits discussion

Key Process Notes

- **Low Error Tolerance:** According to recent feedback (2024-2025), even mostly working code may not guarantee passing
- **Fast Process:** Usually hear back same day or next day after each round
- **Responsive HR:** Recruiters provide feedback and are generally helpful
- **Interview Difficulty:** Rated as "Hard" by most candidates
- **Test Case Emphasis:** Interviewers heavily focus on edge cases and testing

1 Coding Questions

1.1 String and Array Problems

1.1.1 HTML Format Validation (Tag Validator)

PROBLEM

Problem Statement:
Given a string with custom tag syntax, determine if all tags are properly matched and nested. This is NOT standard HTML - uses special {{ }} delimiters with # and / prefixes.
Tag Format (from 1point3acres - Asked 5+ times):

- Opening tag: {{ `#tagname` }} (note the # prefix and spaces)
- Closing tag: {{ `/tagname` }} (note the / prefix and spaces)
- Single braces { or } are treated as NORMAL TEXT (not tags)
- Tags must be properly nested (LIFO order)

A valid structure must satisfy:

- Every opening tag has a matching closing tag
- Tags are properly nested (no overlapping)
- Closing tags match the most recent unclosed opening tag
- Complete tags must have both {{ and }}

Example 1 (Valid - from 1point3acres):

```
Input: "{{ #abc }} {{ #cba }} hello world {{ /cba }} {{ /abc }}"
Output: true
Explanation: Properly nested - #abc opens, #cba opens, /cba closes, /abc closes
```

Example 2 (Valid - Single brace OK):

```
Input: "{{ #abc }} hello { world {{ /abc }}"
Output: true
Explanation: Single { is treated as normal text, not a tag
```

Example 3 (Invalid - Incomplete tag):

```
Input: "{{ #abc }} hello world {{ /abc"
Output: false
Explanation: Missing closing }} for /abc tag
```

Example 4 (Invalid - Missing closing):

```
Input: "{{ #abc }} {{ #cba }} hello {{ /cba }}"
Output: false
Explanation: #abc was opened but never closed
```

Example 5 (Invalid - Wrong order):

```
Input: "{{ #abc }} {{ #cba }} hello {{ /abc }} {{ /cba }}"
Output: false
Explanation: Must close #cba before closing #abc (LIFO order)
```

Solution Approach:
Use a stack to track opening tags:

- Parse the string character by character
- Look for {{ to start a tag (not single {)

- Find matching }} to complete the tag
- Extract tag content and check for # or / prefix
- Opening tag (#): push tag name to stack
- Closing tag (/): pop stack and verify match
- Single braces: ignore (treated as normal text)
- Final stack must be empty
- Time: $O(N)$ where N = length of string
- Space: $O(T)$ where T = number of tags

SOLUTION

```
def is_valid_tags(s):
    if not s:
        return True

    stack = []
    i = 0

    while i < len(s):
        # Look for opening {{ (need at least 2 characters)
        if i < len(s) - 1 and s[i:i+2] == '{{':
            # Find closing }}
            j = i + 2
            found_closing = False

            while j < len(s) - 1:
                if s[j:j+2] == '}}':
                    # Extract tag content
                    tag_content = s[i+2:j].strip()

                    # Check if valid tag (starts with # or /)
                    if tag_content and tag_content[0] in '#/':
                        if tag_content[0] == '#':
                            # Opening tag
                            tag_name = tag_content[1:].strip()
                            stack.append(tag_name)
                        else:  # tag_content[0] == '/'
                            # Closing tag
                            tag_name = tag_content[1:].strip()
                            if not stack or stack[-1] != tag_name:
                                return False
                            stack.pop()

                    i = j + 2
```

```
                    found_closing = True
                    break
                j += 1

            if not found_closing:
                return False  # Incomplete tag
        else:
            # Regular character or single {
            i += 1

    return len(stack) == 0

# Test cases (from 1point3acres)
print(is_valid_tags("{{ #abc }} {{ #cba }} hello {{ /cba }} {{ /abc }}"))
# True
print(is_valid_tags("{{ #abc }} hello { world {{ /abc }}"))
# True (single { is OK)
print(is_valid_tags("{{ #abc }} {{ /abc"))
# False (incomplete tag)
print(is_valid_tags("{{ #abc }} {{ #cba }} {{ /abc }} {{ /cba }}"))
# False (wrong order)
```

TEST CASES & EDGE CASES

Edge Cases (Critical for Interview - from 1point3acres):

- Empty string: `""` → `true`
- No tags, only content: `"Hello World"` → `true`
- Single braces: `"{ text }"` → `true` (NOT tags!)
- Incomplete tag: `"{{ #abc"` → `false` (missing }})
- Only opening: `"{{ #abc }}"` → `false`
- Only closing: `"{{ /abc }}"` → `false`
- Wrong order: `"{{ #abc }} {{ #def }} {{ /abc }} {{ /def }}"` → `false`
- Nested same tags: `"{{ #div }} {{ #div }} {{ /div }} {{ /div }}"` → `true`
- Extra closing: `"{{ #abc }} {{ /abc }} {{ /abc }}"` → `false`
- Double braces without space: `"{{#abc}}"` - depends on implementation
- Tag names with numbers: `"{{ #tag123 }} {{ /tag123 }}"` → `true`

Common Interview Question: "What other edge cases can you think of?"

- Be ready to discuss: incomplete tags, single braces, wrong nesting order

- Interviewers specifically test punctuation/formatting edge cases

Test Cases:

```
def test_html_validation():
    # Valid cases
    assert isValidHTML("") == True
    assert isValidHTML("Hello World") == True
    assert isValidHTML("{{div}} {{/div}}") == True
    assert isValidHTML("{{div}} {{p}} {{/p}} {{/div}}") == True

    # Invalid cases
    assert isValidHTML("{{div}}") == False
    assert isValidHTML("{{/div}}") == False
    assert isValidHTML("{{div}} {{p}} {{/div}} {{/p}}") == False
    assert isValidHTML("{{div}} {{p}}") == False

    # Edge cases
    assert isValidHTML("{{div}} {{div}} {{/div}} {{/div}}") == True
    assert isValidHTML("{{a}} {{b}} {{c}} {{/c}} {{/b}} {{/a}}")
           == True

test_html_validation()
```

Interview Tips:

- Clarify tag naming rules (alphanumeric? special chars?)
- Ask about self-closing tags (e.g., `<img />`)
- Discuss handling of attributes if applicable
- Mention similarity to valid parentheses problem
- For Faire: Test thoroughly with malformed input

Source: Full Stack New Grad (2023-2025)

1.1.2 Haiku Finder - Find First Haiku in Sentence

PROBLEM

Problem Statement:
Given a sentence (single string with mixed case and punctuation) and a syllable dictionary, find the FIRST haiku in the sentence. A haiku consists of 3 consecutive word sequences with syllable counts of 5-7-5.
Input:

- A sentence string with words separated by spaces
- Words may contain mixed case (e.g., "Simple", "HELLO")

- Words may have trailing punctuation (e.g., "world.", "don't", "Void.")
- A syllable dictionary mapping lowercase words to syllable counts

Output:

- List of 3 strings representing the haiku lines (preserving original case & punctuation)
- Return None if no valid haiku exists
- Return the FIRST (leftmost) haiku if multiple exist

Example from 1point3acres (Actual Faire Interview):

```
Input:
sentence = "Internationalization a Simple flower Petals shine
            Vibrant don't Pure Stares into Void. Return home"

syllable_dict = {
    "internationalization": 8, "a": 1, "simple": 2, "flower": 2,
    "petals": 2, "shine": 1, "vibrant": 2, "don't": 1, "pure": 1,
    "stares": 2, "into": 2, "void": 1, "return": 2, "home": 1
}

Output:
["a Simple flower",                    # 1 + 2 + 2 = 5 syllables
 "Petals shine Vibrant don't Pure",    # 2 + 1 + 2 + 1 + 1 = 7
 "Stares into Void."]                  # 2 + 2 + 1 = 5

Explanation:
- "Internationalization" (8 syllables) - too many for any line
- Start from "a" (1) + "Simple" (2) + "flower" (2) = 5 [OK]
- Continue with "Petals" (2) + "shine" (1) + "Vibrant" (2)
  + "don't" (1) + "Pure" (1) = 7 [OK]
- End with "Stares" (2) + "into" (2) + "Void." (1) = 5 [OK]
- Original case and punctuation preserved in output
```

Example 2 - No Valid Haiku:

```
Input:
sentence = "Hello world"
syllable_dict = {"hello": 2, "world": 1}

Output: None
Explanation: Only 3 syllables total, cannot form 5-7-5 pattern
```

Solution Approach:
Key Implementation Steps:

1. **Parse words:** Split sentence by spaces

2. **Look up syllables:** For each word:
    - Strip trailing punctuation (keep for output)
    - Convert to lowercase for dictionary lookup
    - Get syllable count (0 if not in dict)
3. **Build prefix sum array:** prefix[i] = total syllables of words[0:i]
4. **Use hash map:** Map each sum value to its index for O(1) lookup
5. **Find haiku pattern:** For each starting position i:
    - Find j where prefix[j] - prefix[i] == 5 (end of line 1)
    - Find k where prefix[k] - prefix[j] == 7 (end of line 2)
    - Find m where prefix[m] - prefix[k] == 5 (end of line 3)
    - If all found and in order, return first haiku
6. **Reconstruct output:** Join words preserving original formatting

Complexity:

- Time: O(n) with prefix sums and hash map
- Space: O(n) for words array, syllables, and prefix sums
- Brute force: $O(n^2)$ or $O(n^3)$ trying all combinations

SOLUTION

```
import string

def find_haiku(sentence: str, syllable_dict: dict):
    if not sentence:
        return None

    words = sentence.split()
    if len(words) < 3:
        return None

    # Build syllable counts for each word
    syllables = []
    for word in words:
        # Strip punctuation and convert to lowercase for lookup
        clean_word = word.strip(string.punctuation).lower()
        syl_count = syllable_dict.get(clean_word, 0)
        syllables.append(syl_count)

    # Build prefix sum array: prefix_sums[i] = sum of syllables[0:i]
    prefix_sums = [0]
    for syl in syllables:
```

```
        prefix_sums.append(prefix_sums[-1] + syl)

    # Create hash map: sum -> first index with that sum
    # Reverse iteration to get first occurrence for each sum
    sum_to_index = {}
    for i in range(len(prefix_sums) - 1, -1, -1):
        sum_to_index[prefix_sums[i]] = i

    # Try each starting position
    for i in range(len(prefix_sums)):
        start_sum = prefix_sums[i]

        # Look for 5-7-5 pattern
        target_sum1 = start_sum + 5
        target_sum2 = start_sum + 12  # 5 + 7
        target_sum3 = start_sum + 17  # 5 + 7 + 5

        end_index1 = sum_to_index.get(target_sum1)
        end_index2 = sum_to_index.get(target_sum2)
        end_index3 = sum_to_index.get(target_sum3)

        # Check if all exist and are in correct order
        if (end_index1 is not None and
            end_index2 is not None and
            end_index3 is not None and
            i < end_index1 <= end_index2 <= end_index3):

            # Found haiku! Reconstruct with original formatting
            line1 = " ".join(words[i:end_index1])
            line2 = " ".join(words[end_index1:end_index2])
            line3 = " ".join(words[end_index2:end_index3])

            return [line1, line2, line3]

    return None
```

TEST CASES & EDGE CASES

Critical Edge Cases (from 1point3acres reports):

- **Punctuation:** Must strip before lookup but preserve in output
- **Case sensitivity:** Convert to lowercase for lookup, preserve in output
- **Missing words:** Words not in dictionary - treat as 0 syllables
- **Empty/short input:** Empty string or ¡ 3 words
- **No valid haiku:** Total syllables ¡ 17 or impossible pattern

- **Multiple haikus:** Return FIRST (leftmost) one only
- **Contractions:** "don't", "can't" - must handle punctuation correctly

Test Cases:

```
def test_haiku_finder():
    # Test 1: Actual Faire example from 1point3acres
    sentence = "Internationalization a Simple flower Petals " + \
               "shine Vibrant don't Pure Stares into Void. Return home"
    syllable_dict = {
        "internationalization": 8, "a": 1, "simple": 2, "flower": 2,
        "petals": 2, "shine": 1, "vibrant": 2, "don't": 1,
        "pure": 1, "stares": 2, "into": 2, "void": 1,
        "return": 2, "home": 1
    }
    result = find_haiku(sentence, syllable_dict)
    expected = ["a Simple flower",
                "Petals shine Vibrant don't Pure",
                "Stares into Void."]
    assert result == expected

    # Test 2: No haiku possible
    sentence2 = "Hello world"
    dict2 = {"hello": 2, "world": 1}
    assert find_haiku(sentence2, dict2) is None

    # Test 3: Punctuation handling
    sentence3 = "Hello, world! Nice day. Very good, yes."
    dict3 = {"hello": 2, "world": 1, "nice": 1, "day": 1,
             "very": 2, "good": 1, "yes": 1}
    result3 = find_haiku(sentence3, dict3)
    # Should preserve punctuation if haiku found

    # Test 4: Empty input
    assert find_haiku("", {}) is None
    assert find_haiku("a b", {"a": 1, "b": 1}) is None
```

Interview Insights from 1point3acres:

- **High emphasis on test cases:** Multiple candidates mentioned interviewers using JUnit and asking "what other edge cases?"
- **Optimization matters:** Brute force may pass basic tests but optimal O(n) solution expected
- **Code quality critical:** Clean, readable code with proper error handling
- **Common failure:** Not handling punctuation correctly or missing edge cases
- **Verify correctness:** Run code with provided examples before submitting

Source: Multiple 1point3acres reports (2021-2025), confirmed by 5+ candidates across different interview rounds

1.1.3 Funnel Problem

PROBLEM

Problem Statement:
You are analyzing a conversion funnel for an e-commerce platform. Users go through the following stages in order:

1. **Browse**: User browses product listings
2. **View**: User views product details
3. **Cart**: User adds product to cart
4. **Checkout**: User initiates checkout
5. **Purchase**: User completes purchase

Given a list of user events where each event is a tuple `(user_id, stage, timestamp)`, implement the following:

1. `calculate_conversion_rates()`: Return conversion rate for each stage transition
2. `find_dropoff_stage()`: Return the stage with highest drop-off rate
3. `get_funnel_metrics()`: Return dictionary with:
   - Total users at each stage
   - Conversion rate from previous stage
   - Overall conversion rate (Browse to Purchase)
4. `get_user_journey(user_id)`: Return ordered list of stages user visited

Rules:

- Users must progress through stages in order (can skip stages)
- A user can only be counted once per stage
- Conversion rate = (users in next stage) / (users in current stage)
- Drop-off rate = 1 - conversion rate

Example:

```
events = [
    ('u1', 'Browse', 100),
    ('u1', 'View', 105),
    ('u1', 'Cart', 110),
    ('u1', 'Purchase', 120),
```

```
    ('u2', 'Browse', 100),
    ('u2', 'View', 105),
    ('u3', 'Browse', 101),
    ('u3', 'View', 106),
    ('u3', 'Cart', 111),
]

Funnel:
Browse: 3 users (u1, u2, u3)
View: 3 users (100% conversion from Browse)
Cart: 2 users (66.7% conversion from View)
Checkout: 0 users (0% conversion from Cart)
Purchase: 1 user (N/A conversion - no one at Checkout)

Drop-off rates:
Browse -> View: 0%
View -> Cart: 33.3%
Cart -> Checkout: 100%  (highest drop-off)
```

Constraints:

- $1 \leq$ number of events $\leq 10^5$
- $1 \leq$ number of unique users $\leq 10^4$
- Timestamps are in ascending order per user
- Stage names are from the predefined list

SOLUTION

Approach:
Use sets to track unique users at each stage. Build a mapping of user journeys to handle edge cases like skipped stages or repeated events.
Time Complexity: $O(N)$ where N = number of events
Space Complexity: $O(U \times S)$ where U = users, S = stages
Python Solution:

```
from collections import defaultdict

class FunnelAnalyzer:
    def __init__(self, events):
        """
        events: list of tuples (user_id, stage, timestamp)
        """
        self.events = sorted(events, key=lambda x: (x[0], x[2]))
        self.stages = ['Browse', 'View', 'Cart', 'Checkout', 'Purchase']
        self.stage_order = {stage: i for i, stage in enumerate(self.stages)}
```

```
        # Build user journeys and stage sets
        self.user_journeys = defaultdict(list)
        self.stage_users = {stage: set() for stage in self.stages}

        self._process_events()

    def _process_events(self):
        """Process events and build user journeys"""
        for user_id, stage, timestamp in self.events:
            if stage not in self.stage_order:
                continue  # Invalid stage

            # Add to user journey
            self.user_journeys[user_id].append((stage, timestamp))

            # Add user to stage (only count once per stage)
            self.stage_users[stage].add(user_id)

    def calculate_conversion_rates(self):
        """Return conversion rate for each stage transition"""
        conversion_rates = {}

        for i in range(len(self.stages) - 1):
            current_stage = self.stages[i]
            next_stage = self.stages[i + 1]

            current_count = len(self.stage_users[current_stage])
            next_count = len(self.stage_users[next_stage])

            if current_count == 0:
                conversion_rates[f"{current_stage} -> {next_stage}"] = 0.0
            else:
                rate = (next_count / current_count) * 100
                conversion_rates[f"{current_stage} -> {next_stage}"] = round(rate, 2)

        return conversion_rates

    def find_dropoff_stage(self):
        """Return the stage with highest drop-off rate"""
        conversion_rates = self.calculate_conversion_rates()

        max_dropoff = 0
        dropoff_stage = None

        for transition, rate in conversion_rates.items():
            dropoff_rate = 100 - rate
```

```
            if dropoff_rate > max_dropoff:
                max_dropoff = dropoff_rate
                dropoff_stage = transition

        return dropoff_stage, max_dropoff

    def get_funnel_metrics(self):
        """Return comprehensive funnel metrics"""
        metrics = {}

        for i, stage in enumerate(self.stages):
            stage_count = len(self.stage_users[stage])

            metric = {
                'total_users': stage_count,
                'conversion_from_previous': None,
                'overall_conversion': None
            }

            # Conversion from previous stage
            if i > 0:
                prev_stage = self.stages[i - 1]
                prev_count = len(self.stage_users[prev_stage])
                if prev_count > 0:
                    metric['conversion_from_previous'] = round(
                        (stage_count / prev_count) * 100, 2
                    )

            # Overall conversion (from Browse)
            browse_count = len(self.stage_users['Browse'])
            if browse_count > 0:
                metric['overall_conversion'] = round(
                    (stage_count / browse_count) * 100, 2
                )

            metrics[stage] = metric

        return metrics

    def get_user_journey(self, user_id):
        """Return ordered list of stages user visited"""
        if user_id not in self.user_journeys:
            return []

        # Return unique stages in order
        seen = set()
```

```
        journey = []
        for stage, _ in self.user_journeys[user_id]:
            if stage not in seen:
                journey.append(stage)
                seen.add(stage)

        return journey

    def validate_user_journey(self, user_id):
        """Check if user followed correct order"""
        journey = self.get_user_journey(user_id)

        for i in range(len(journey) - 1):
            current_idx = self.stage_order[journey[i]]
            next_idx = self.stage_order[journey[i + 1]]

            # Next stage should come after current stage
            if next_idx <= current_idx:
                return False, f"Invalid order: {journey[i]} -> {journey[i+1]}"

        return True, "Valid journey"


# Example usage
events = [
    ('u1', 'Browse', 100),
    ('u1', 'View', 105),
    ('u1', 'Cart', 110),
    ('u1', 'Purchase', 120),
    ('u2', 'Browse', 100),
    ('u2', 'View', 105),
    ('u3', 'Browse', 101),
    ('u3', 'View', 106),
    ('u3', 'Cart', 111),
]

analyzer = FunnelAnalyzer(events)

print("Conversion Rates:")
print(analyzer.calculate_conversion_rates())

print("\nHighest Drop-off:")
print(analyzer.find_dropoff_stage())

print("\nFunnel Metrics:")
for stage, metrics in analyzer.get_funnel_metrics().items():
```

```
    print(f"{stage}: {metrics}")

print("\nUser Journey (u1):")
print(analyzer.get_user_journey('u1'))
```

TEST CASES & EDGE CASES

Edge Cases & Testing:

- Empty events list: all metrics return 0
- Single user, single event: 100% drop-off after first stage
- User skips stages: e.g., Browse -¿ Purchase (validate journey)
- User repeats same stage: count only once
- User goes backwards: Browse -¿ View -¿ Browse (invalid order)
- All users complete funnel: 100% conversion at each stage
- No users reach final stage: Purchase conversion = 0
- Duplicate timestamps: handle tie-breaking
- Invalid stage names: filter out or raise error
- Division by zero: when no users at a stage

Test Cases:

```
def test_funnel_analyzer():
    # Test 1: Basic funnel
    events = [
        ('u1', 'Browse', 1), ('u1', 'View', 2),
        ('u2', 'Browse', 1)
    ]
    analyzer = FunnelAnalyzer(events)
    metrics = analyzer.get_funnel_metrics()
    assert metrics['Browse']['total_users'] == 2
    assert metrics['View']['total_users'] == 1
    assert metrics['View']['conversion_from_previous'] == 50.0

    # Test 2: Complete journey
    events = [
        ('u1', 'Browse', 1), ('u1', 'View', 2),
        ('u1', 'Cart', 3), ('u1', 'Checkout', 4),
        ('u1', 'Purchase', 5)
    ]
    analyzer = FunnelAnalyzer(events)
    journey = analyzer.get_user_journey('u1')
```

```
    assert len(journey) == 5
    assert journey[-1] == 'Purchase'

    # Test 3: Drop-off detection
    events = [
        ('u1', 'Browse', 1), ('u1', 'View', 2),
        ('u2', 'Browse', 1), ('u2', 'View', 2),
        ('u3', 'Browse', 1), ('u3', 'View', 2),
        ('u1', 'Cart', 3)  # Only u1 proceeds
    ]
    analyzer = FunnelAnalyzer(events)
    dropoff_stage, rate = analyzer.find_dropoff_stage()
    assert 'View -> Cart' in dropoff_stage
    assert rate > 60  # ~66.7% drop-off

    # Test 4: Duplicate events (same stage)
    events = [
        ('u1', 'Browse', 1),
        ('u1', 'Browse', 2),  # Duplicate
        ('u1', 'View', 3)
    ]
    analyzer = FunnelAnalyzer(events)
    assert len(analyzer.stage_users['Browse']) == 1

    # Test 5: Empty events
    analyzer = FunnelAnalyzer([])
    metrics = analyzer.get_funnel_metrics()
    assert all(m['total_users'] == 0 for m in metrics.values())

    # Test 6: Skipped stages
    events = [
        ('u1', 'Browse', 1),
        ('u1', 'Purchase', 5)  # Skips View, Cart, Checkout
    ]
    analyzer = FunnelAnalyzer(events)
    journey = analyzer.get_user_journey('u1')
    assert journey == ['Browse', 'Purchase']

    print("All funnel tests passed!")

test_funnel_analyzer()
```

Interview Tips:

- Clarify whether users can skip stages
- Ask about handling duplicate events for same stage
- Discuss how to handle out-of-order events

- Consider time windows for conversion (e.g., must complete within 24h)
- For Faire: **Focus heavily on edge cases and testing**
- Use proper testing framework (JUnit, pytest) if allowed
- Validate input data (null checks, invalid stages)
- Consider performance with large datasets

Note: Interviewer heavily focused on test cases and edge cases. Even after comprehensive testing, be prepared for more edge case questions. Source: Coding interview (2024-2025)

1.1.4 Group Anagrams

PROBLEM

Problem Statement:
Given an array of strings `strs`, group the anagrams together. You can return the answer in any order.
An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, using all the original letters exactly once.
Example 1:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

Example 2:

```
Input: strs = [""]
Output: [[""]]
```

Example 3:

```
Input: strs = ["a"]
Output: [["a"]]
```

Constraints:

- $1 \leq$ strs.length $\leq 10^4$
- $0 \leq$ strs[i].length ≤ 100
- strs[i] consists of lowercase English letters

Solution Approach:
Method 1: Sorting (Optimal)

- For each string, sort its characters to create a key
- Use a hash map where key = sorted string, value = list of original strings
- All anagrams will have the same sorted key

- Time: $O(N \cdot K \log K)$ where N = number of strings, K = max length
- Space: $O(N \cdot K)$

Method 2: Character Count (Alternative)

- Use character frequency as key (e.g., "a1b2c1" for "abc", "bac")
- Time: $O(N \cdot K)$ - better than sorting
- Space: $O(N \cdot K)$

SOLUTION

```
from collections import defaultdict

def groupAnagrams(strs):
    anagrams = defaultdict(list)

    for s in strs:
        # Use sorted string as key
        key = ''.join(sorted(s))
        anagrams[key].append(s)

    return list(anagrams.values())

# Alternative: Character count method
def groupAnagrams_v2(strs):
    anagrams = defaultdict(list)

    for s in strs:
        # Use character frequency tuple as key
        count = [0] * 26
        for c in s:
            count[ord(c) - ord('a')] += 1
        anagrams[tuple(count)].append(s)

    return list(anagrams.values())
```

TEST CASES & EDGE CASES

Edge Cases:

- Empty string: `[""]` → `[[""]]`
- Single character: `["a"]` → `[["a"]]`
- All anagrams: `["abc", "bca", "cab"]` → `[["abc", "bca", "cab"]]`
- No anagrams: `["a", "b", "c"]` → `[["a"], ["b"], ["c"]]`
- Duplicate strings: `["a", "a"]` → `[["a", "a"]]`

- Mixed lengths: `["a", "ab", "ba"]` → `[["a"], ["ab", "ba"]]`
- Case sensitivity: Problem specifies lowercase only

Test Cases:

```
# Test 1: Standard case
assert sorted([sorted(g) for g in groupAnagrams(
    ["eat","tea","tan","ate","nat","bat"])]
) == sorted([["bat"], ["nat","tan"], ["ate","eat","tea"]])

# Test 2: Empty strings
assert groupAnagrams([""]) == [[""]]

# Test 3: Single element
assert groupAnagrams(["a"]) == [["a"]]

# Test 4: No anagrams
result = groupAnagrams(["abc", "def", "ghi"])
assert len(result) == 3

# Test 5: All anagrams
result = groupAnagrams(["abc", "bca", "cab"])
assert len(result) == 1 and len(result[0]) == 3
```

Interview Tips:

- Clarify if input can have empty strings or duplicates
- Discuss both sorting and character count approaches
- Mention trade-offs: sorting is simpler, char count is faster
- For Faire: Emphasize thorough testing and edge cases

Source: Full Stack New Grad (2023-2025)

1.1.5 Number to Word Conversion

PROBLEM

Problem Statement:
Given two integers `start` and `end`, convert all numbers in the range $[start, end]$ (inclusive) to their English word representations and calculate the total length of all characters (excluding spaces).
For example: 1 = "one", 2 = "two", ..., 23 = "twenty three"
Return the total length of all word representations.
Example 1:

```
Input: start = 1, end = 3
Output: 11
```

```
Explanation:
1 -> "one" (3 chars)
2 -> "two" (3 chars)
3 -> "three" (5 chars)
Total: 3 + 3 + 5 = 11
```

Example 2:

```
Input: start = 10, end = 12
Output: 22

Explanation:
10 -> "ten" (3 chars)
11 -> "eleven" (6 chars)
12 -> "twelve" (6 chars)
Total: 3 + 6 + 6 = 15 (NOT 22, recalculating...)
Actually: 3 + 6 + 6 = 15

Or if counting spaces:
10 -> "ten" (3)
11 -> "eleven" (6)
12 -> "twelve" (6)
= 15 without spaces
```

Constraints:

- $1 \leq start \leq end \leq 1000$
- Count only letters, not spaces

SOLUTION

Solution Approach:
Build number-to-word converter:

- Create maps for 1-19, tens (20, 30,...), and hundreds
- For each number, convert to words recursively
- Sum up character counts
- Time: $O(N)$ where N = end - start + 1
- Space: $O(1)$ for conversion maps

Python Solution:

```
def number_to_words(num):
    """Convert number (1-1000) to English words"""
    if num == 0:
        return "zero"
```

```
    ones = ["", "one", "two", "three", "four", "five",
            "six", "seven", "eight", "nine"]
    teens = ["ten", "eleven", "twelve", "thirteen",
             "fourteen", "fifteen", "sixteen",
             "seventeen", "eighteen", "nineteen"]
    tens = ["", "", "twenty", "thirty", "forty", "fifty",
            "sixty", "seventy", "eighty", "ninety"]

    def helper(n):
        if n == 0:
            return ""
        elif n < 10:
            return ones[n]
        elif n < 20:
            return teens[n - 10]
        elif n < 100:
            return tens[n // 10] + \
                   (" " + ones[n % 10] if n % 10 != 0 else "")
        else:
            return ones[n // 100] + " hundred" + \
                   (" " + helper(n % 100) if n % 100 != 0
                    else "")

    if num == 1000:
        return "one thousand"
    return helper(num)

def total_word_length(start, end):
    total = 0
    for num in range(start, end + 1):
        words = number_to_words(num)
        # Remove spaces and count length
        length = len(words.replace(" ", ""))
        total += length
    return total

# Tests
print(total_word_length(1, 3))   # 11
print(total_word_length(10, 12)) # 15
print(total_word_length(1, 5))   # one+two+three+four+five
                                 # 3+3+5+4+4 = 19
```

TEST CASES & EDGE CASES

Edge Cases & Testing:

- Single number: start = end = 1 → 3

- Teens (11-19): special handling
- Exact tens (20, 30, ...): no "and"
- Hundreds: 100 = "one hundred" (10 chars)
- 1000: "one thousand" (special case)
- Crossing boundaries: [19, 21] includes "nineteen", "twenty", "twenty one"

Test Cases:

```
def test_number_words():
    # Test individual conversions
    assert number_to_words(1) == "one"
    assert number_to_words(11) == "eleven"
    assert number_to_words(20) == "twenty"
    assert number_to_words(100) == "one hundred"
    assert number_to_words(1000) == "one thousand"

    # Test total length
    assert total_word_length(1, 1) == 3  # "one"
    assert total_word_length(1, 3) == 11 # 3+3+5
    assert total_word_length(10, 10) == 3 # "ten"

test_number_words()
```

Interview Tips:

- Clarify if spaces count toward length
- Ask about range limits (problem says ≤ 1000)
- Discuss British vs American English ("and" placement)
- Mention potential optimization for repeated ranges
- For Faire: Test edge cases like 1000, teens, exact hundreds

Source: Backend/Full Stack - Canadian Office (2022)

1.2 Graph and Path Problems

1.2.1 Ads Assortment Problem

PROBLEM

Problem Statement:
You are given `n` advertisements, where each ad has a `value` (revenue generated) and a `cost` (budget required). You have a total budget of `B`.
Select a subset of ads to maximize total value while staying within budget.
Additionally, some ads may have **category constraints**: You can only select at most `k` ads

from each category.

Example 1:

```
Input:
ads = [
  {'value': 60, 'cost': 10, 'category': 'tech'},
  {'value': 100, 'cost': 20, 'category': 'fashion'},
  {'value': 120, 'cost': 30, 'category': 'tech'}
]
budget = 50
category_limit = 2  # Max 2 ads per category

Output: 220
Explanation: Select ads 2 and 3 (fashion + tech)
             Cost: 20 + 30 = 50, Value: 100 + 120 = 220
```

Example 2:

```
Input:
ads = [
  {'value': 60, 'cost': 10},
  {'value': 100, 'cost': 20},
  {'value': 120, 'cost': 30}
]
budget = 35

Output: 160
Explanation: Select ads 1 and 2.
             Cost: 10 + 20 = 30, Value: 60 + 100 = 160
```

Constraints:

- $1 \leq n \leq 100$
- $1 \leq$ cost[i], value[i] ≤ 1000
- $1 \leq B \leq 10000$

SOLUTION

Solution Approach:
Method 1: Dynamic Programming (0/1 Knapsack)

- Classic 0/1 knapsack problem
- dp[i][j] = max value using first i ads with budget j
- Time: $O(N \times B)$
- Space: $O(N \times B)$, optimizable to $O(B)$

Method 2: Greedy (with category constraints)

- Sort ads by value/cost ratio (efficiency)
- Greedily select highest efficiency ads
- Track category counts
- Time: $O(N \log N)$
- Note: May not give optimal solution, but good approximation

Python Solution:

```
def max_ad_value_knapsack(ads, budget):
    """0/1 Knapsack DP solution"""
    n = len(ads)
    # dp[i][b] = max value using first i ads with budget b
    dp = [[0] * (budget + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        cost = ads[i-1]['cost']
        value = ads[i-1]['value']

        for b in range(budget + 1):
            # Don't take current ad
            dp[i][b] = dp[i-1][b]

            # Take current ad if possible
            if b >= cost:
                dp[i][b] = max(dp[i][b], dp[i-1][b-cost] + value)

    return dp[n][budget]

# Space-optimized version
def max_ad_value_optimized(ads, budget):
    """Space-optimized O(B) space"""
    dp = [0] * (budget + 1)

    for ad in ads:
        cost = ad['cost']
        value = ad['value']

        # Traverse backwards to avoid using same item twice
        for b in range(budget, cost - 1, -1):
            dp[b] = max(dp[b], dp[b - cost] + value)

    return dp[budget]
```

```
# With category constraints
def max_ad_value_with_categories(ads, budget, category_limit):
    """DP with category constraints"""
    from collections import defaultdict

    # Group ads by category
    categories = defaultdict(list)
    for i, ad in enumerate(ads):
        cat = ad.get('category', 'default')
        categories[cat].append(i)

    # Generate valid ad combinations respecting category limits
    # For simplicity, use greedy approach
    ads_sorted = sorted(enumerate(ads),
                        key=lambda x: x[1]['value'] / x[1]['cost'],
                        reverse=True)

    category_count = defaultdict(int)
    selected = []
    total_cost = 0
    total_value = 0

    for idx, ad in ads_sorted:
        cat = ad.get('category', 'default')
        if (category_count[cat] < category_limit and
            total_cost + ad['cost'] <= budget):
            selected.append(idx)
            category_count[cat] += 1
            total_cost += ad['cost']
            total_value += ad['value']

    return total_value

# Tests
ads1 = [
    {'value': 60, 'cost': 10},
    {'value': 100, 'cost': 20},
    {'value': 120, 'cost': 30}
]
print(max_ad_value_knapsack(ads1, 35))  # 160

ads2 = [
    {'value': 60, 'cost': 10, 'category': 'tech'},
    {'value': 100, 'cost': 20, 'category': 'fashion'},
    {'value': 120, 'cost': 30, 'category': 'tech'}
]
```

```
print(max_ad_value_with_categories(ads2, 50, 2))  # 220
```

TEST CASES & EDGE CASES

Edge Cases & Testing:

- Budget = 0: return 0
- Single ad within budget: return its value
- Single ad exceeds budget: return 0
- All ads exceed budget: return 0
- Category limit = 0: cannot select any ads
- Multiple ads same efficiency: test tie-breaking
- Exact budget match: select ads that sum to exactly B

Test Cases:

```
def test_ads_assortment():
    # Test 1: Basic knapsack
    ads = [{'value': 60, 'cost': 10}, {'value': 100, 'cost': 20}]
    assert max_ad_value_knapsack(ads, 25) == 100

    # Test 2: All ads fit
    ads = [{'value': 10, 'cost': 5}, {'value': 20, 'cost': 10}]
    assert max_ad_value_knapsack(ads, 15) == 30

    # Test 3: No ads fit
    ads = [{'value': 100, 'cost': 50}]
    assert max_ad_value_knapsack(ads, 40) == 0

    # Test 4: Empty ads
    assert max_ad_value_knapsack([], 100) == 0

    # Test 5: Zero budget
    ads = [{'value': 100, 'cost': 10}]
    assert max_ad_value_knapsack(ads, 0) == 0

test_ads_assortment()
```

Interview Tips:

- Recognize as 0/1 knapsack variant
- Discuss trade-offs: DP vs greedy
- Ask about fractional ads (0/1 vs fractional knapsack)
- Clarify category constraint details
- For Faire: Test boundary conditions

Source: CodeSignal OA (2024)

1.2.2 Course Schedule with Minimum Time

PROBLEM

Problem Statement:
You are given:

- `n` courses labeled from 0 to n-1
- `prerequisites`: array where `prerequisites[i] = [a, b]` means course `b` must be completed before course `a`
- `time`: array where `time[i]` is the duration (in days/weeks) to complete course `i`

Find the minimum time required to complete all courses. You can take multiple courses simultaneously if their prerequisites are met.

Example 1:

```
n = 3
prerequisites = [[1, 0], [2, 0]]
time = [1, 2, 3]

Course 0: 1 day, no prerequisites
Course 1: 2 days, requires course 0
Course 2: 3 days, requires course 0

Timeline:
Day 0-1: Complete course 0 (1 day)
Day 1-3: Complete courses 1 and 2 in parallel (max 3 days)

Total: 1 + 3 = 4 days

Output: 4
```

Example 2:

```
n = 4
prerequisites = [[1, 0], [2, 1], [3, 2]]
time = [1, 1, 1, 1]

Linear dependency: 0 -> 1 -> 2 -> 3
Must complete sequentially: 1+1+1+1 = 4 days

Output: 4
```

Example 3:

```
n = 4
prerequisites = [[1, 0], [2, 0], [3, 1], [3, 2]]
```

```
time = [1, 2, 3, 1]

Course 0: 1 day
Courses 1, 2: after 0 (2, 3 days) - parallel
Course 3: after both 1 and 2 (1 day)

Timeline:
Day 0-1: Course 0
Day 1-4: Courses 1 (done day 3) and 2 (done day 4) parallel
Day 4-5: Course 3

Total: 5 days

Output: 5
```

SOLUTION

Solution Approach:
Topological Sort + Critical Path Method (CPM):

- Build graph from prerequisites
- Check for cycles (if cycle exists, return -1)
- Use topological sort with earliest start time tracking:
  - `earliest[i]` = earliest time course `i` can start
  - For each course: `earliest[i] = max(earliest[prereq] + time[prereq])` for all prereqs
- Answer = `max(earliest[i] + time[i]) for all i`
- Time: $O(V + E)$ where V = courses, E = prerequisites
- Space: $O(V + E)$

Python Solution:

```
from collections import defaultdict, deque

def minimum_time_courses(n, prerequisites, time):
    # Build graph and in-degree
    graph = defaultdict(list)
    in_degree = [0] * n

    for course, prereq in prerequisites:
        graph[prereq].append(course)
        in_degree[course] += 1

    # Initialize earliest start times
```

```
    earliest = [0] * n

    # Topological sort using Kahn's algorithm
    queue = deque()
    for i in range(n):
        if in_degree[i] == 0:
            queue.append(i)

    completed = 0

    while queue:
        course = queue.popleft()
        completed += 1

        # Process neighbors
        for next_course in graph[course]:
            # Update earliest start time for next_course
            earliest[next_course] = max(
                earliest[next_course],
                earliest[course] + time[course]
            )

            in_degree[next_course] -= 1
            if in_degree[next_course] == 0:
                queue.append(next_course)

    # Check for cycle
    if completed != n:
        return -1  # Impossible due to cycle

    # Calculate total time (max finish time)
    max_time = 0
    for i in range(n):
        finish_time = earliest[i] + time[i]
        max_time = max(max_time, finish_time)

    return max_time

# Tests
print(minimum_time_courses(
    3, [[1, 0], [2, 0]], [1, 2, 3]
))  # Output: 4

print(minimum_time_courses(
    4, [[1, 0], [2, 1], [3, 2]], [1, 1, 1, 1]
))  # Output: 4
```

```
print(minimum_time_courses(
    4, [[1, 0], [2, 0], [3, 1], [3, 2]], [1, 2, 3, 1]
))  # Output: 5
```

TEST CASES & EDGE CASES

Edge Cases & Testing:

- No prerequisites: `max(time)` (all parallel)
- Linear chain: `sum(time)` (all sequential)
- Cycle in prerequisites: return -1
- Single course: `time[0]`
- Zero time courses: valid, acts as dependency marker
- Disconnected components: multiple independent course chains
- Course with multiple prerequisites: must wait for ALL

Test Cases:

```
def test_course_schedule_time():
    # Test 1: Parallel courses
    assert minimum_time_courses(
        3, [[1, 0], [2, 0]], [1, 2, 3]
    ) == 4

    # Test 2: Linear sequence
    assert minimum_time_courses(
        3, [[1, 0], [2, 1]], [1, 2, 3]
    ) == 6

    # Test 3: No prerequisites (all parallel)
    assert minimum_time_courses(
        3, [], [1, 2, 3]
    ) == 3

    # Test 4: Cycle detection
    assert minimum_time_courses(
        2, [[0, 1], [1, 0]], [1, 1]
    ) == -1

    # Test 5: Diamond dependency
    assert minimum_time_courses(
        4, [[1, 0], [2, 0], [3, 1], [3, 2]],
        [1, 2, 3, 1]
    ) == 5
```

```
test_course_schedule_time()
```

Interview Tips:

- Recognize this as Critical Path Method (CPM) problem
- Discuss difference from standard Course Schedule
- Mention that parallel execution is key
- Explain why we take max of prerequisite finish times
- For Faire: Test cycle detection and edge cases

Source: Senior SWE Interview (2022)

1.3 LeetCode-style Problems

1.3.1 Maze with Portals

PROBLEM

Problem Statement:
You are given a 2D grid maze where:

- `0` = walkable cell
- `1` = wall (cannot pass)
- `S` = start position
- `E` = end position
- `P` = portal (teleporter)

Additionally, you have a dictionary mapping portal positions to their destinations. When you step on a portal, you are instantly teleported to its destination.
Find the shortest path from S to E. Return the minimum number of steps, or -1 if impossible.
Movement: You can move up, down, left, right (4 directions). Each move counts as 1 step. Portal teleportation does NOT count as an extra step.
Example 1:

```
maze = [
  ['S', '0', '1', '0'],
  ['0', '1', 'P', '0'],
  ['0', '0', '0', '1'],
  ['1', '0', 'E', '0']
]

portals = {(1, 2): (3, 1)}  # Portal at (1,2) goes to (3,1)

Shortest path:
```

```
S(0,0) -> (1,0) -> (2,0) -> (2,1) -> (2,2) -> Portal(1,2)
       -> Teleport to (3,1) -> E(3,2)

Without portal: 6 steps
With portal: 4 steps (reach portal) + 1 step (to E) = 5 steps

Output: 5
```

Example 2:

```
maze = [
  ['S', '1', 'E']
]
Output: -1 (impossible, wall blocks path)
```

SOLUTION

Solution Approach:
Use BFS (Breadth-First Search) with portal handling:

- Standard BFS tracks (row, col, distance)
- When visiting a cell:
  - If it's a portal, add BOTH normal neighbors AND portal destination to queue
  - Portal destination inherits same distance (instant teleport)
- Use visited set to avoid cycles
- Time: $O(R \times C)$ where R = rows, C = columns
- Space: $O(R \times C)$ for queue and visited set

Python Solution:

```
from collections import deque

def shortest_path_with_portals(maze, portals):
    if not maze or not maze[0]:
        return -1

    rows, cols = len(maze), len(maze[0])

    # Find start and end
    start, end = None, None
    for r in range(rows):
        for c in range(cols):
            if maze[r][c] == 'S':
                start = (r, c)
            elif maze[r][c] == 'E':
```

```
                end = (r, c)

    if not start or not end:
        return -1

    # BFS
    queue = deque([(start[0], start[1], 0)])  # (row, col, dist)
    visited = {start}
    directions = [(0,1), (1,0), (0,-1), (-1,0)]

    while queue:
        r, c, dist = queue.popleft()

        # Check if reached end
        if (r, c) == end:
            return dist

        # Check if current cell is a portal
        if (r, c) in portals:
            pr, pc = portals[(r, c)]
            if (pr, pc) not in visited and \
               0 <= pr < rows and 0 <= pc < cols and \
               maze[pr][pc] != '1':
                visited.add((pr, pc))
                queue.append((pr, pc, dist))

        # Explore normal neighbors
        for dr, dc in directions:
            nr, nc = r + dr, c + dc

            if (0 <= nr < rows and 0 <= nc < cols and
                (nr, nc) not in visited and
                maze[nr][nc] != '1'):

                visited.add((nr, nc))
                queue.append((nr, nc, dist + 1))

    return -1  # No path found

# Test
maze1 = [
    ['S', '0', '1', '0'],
    ['0', '1', 'P', '0'],
    ['0', '0', '0', '1'],
    ['1', '0', 'E', '0']
]
```

```
portals1 = {(1, 2): (3, 1)}
print(shortest_path_with_portals(maze1, portals1))
```

TEST CASES & EDGE CASES

Edge Cases & Testing:

- Empty maze: return -1
- No start or end: return -1
- Start = End: return 0
- No path exists (walls block): return -1
- Portal leads to wall: ignore portal
- Portal leads out of bounds: ignore portal
- Multiple portals: test chaining portals
- Portal at start position: can immediately teleport
- Portal at end position: doesn't affect result
- Bidirectional portals: clarify if portals work both ways
- Portal cycles: A → B, B → A (visited set handles this)

Test Cases:

```
def test_maze_portals():
    # Test 1: With portal shortcut
    maze1 = [['S', '0', 'P'], ['1', '1', '0'], ['E', '0', '0']]
    portals1 = {(0, 2): (2, 0)}
    assert shortest_path_with_portals(maze1, portals1) == 2

    # Test 2: No path
    maze2 = [['S', '1', 'E']]
    portals2 = {}
    assert shortest_path_with_portals(maze2, portals2) == -1

    # Test 3: Direct path better than portal
    maze3 = [['S', '0', 'E']]
    portals3 = {(0, 1): (0, 0)}  # Portal doesn't help
    assert shortest_path_with_portals(maze3, portals3) == 2

    # Test 4: Start = End
    maze4 = [['S']]
    portals4 = {}
    # Modify to handle S=E: return 0 if (r,c) == end initially

test_maze_portals()
```

Interview Tips:

- Clarify if portals are one-way or bidirectional
- Ask if portal teleportation counts as a step
- Discuss handling of portal cycles/loops
- Mention BFS is optimal for shortest path
- For Faire: Test portal edge cases thoroughly

Source: Backend Engineer Phone Screen (2024)

1.3.2 General OA Topics

- Array manipulation and subarrays
- String processing and pattern matching
- Hash maps and frequency counting
- Greedy algorithms
- Dynamic programming (basic)
- Math and number theory (modular arithmetic)

2 System Design Questions

2.1 Design a Post "Like" Functionality

- **Description:** Design a system for liking posts (similar to social media)
- **Focus:** Communication and thought process
- **Key aspects to consider:**
  - How to think about the problem
  - Different solution approaches
  - Scalability considerations
- **Note:** More emphasis on communication than specific implementation
- **Position:** Backend Engineer
- **Source:** System Design round (2024-2025)

3 Data Engineering Questions

3.1 Senior Data Engineer Interview

- **Interview Structure:**
  - 1 round phone screen: Coding
  - 3 rounds virtual onsite:
    1. Coding
    2. Behavioral Questions (BQ)
    3. Pipeline Design
- **Timeline:** August application, late August recruiter reach out
- **Result:** Rejection
- **Feedback:** Very difficult for job-hopping while employed; low error tolerance
- **Source:** Senior Data Engineer interview (2025)

4 Behavioral Interview Questions

Common behavioral questions asked at Faire (compiled from Prepfully and 1point3acres):

1. Tell me about yourself and your background
2. Why do you want to work at Faire? What interests you about the company?
3. Describe a time when you had to work with a difficult team member. How did you handle it?
4. Tell me about a challenging technical problem you solved. What was your approach?
5. How do you handle disagreements with your manager or team members?
6. Describe a project you're most proud of. What was your role?
7. Tell me about a time you failed. What did you learn?
8. How do you prioritize tasks when you have multiple deadlines?
9. Describe a situation where you had to learn a new technology quickly
10. Where do you see yourself in 3-5 years?

4.1 Tips for Behavioral Rounds

- Use STAR method (Situation, Task, Action, Result)
- Prepare stories that demonstrate:
  - Technical problem-solving
  - Teamwork and collaboration
  - Leadership and ownership

  - Adaptability and learning
  - Communication skills
- Research Faire's business model (wholesale marketplace connecting retailers and brands)
- Be prepared to discuss why you're interested in B2B marketplace space
- Demonstrate understanding of Faire's mission to empower small businesses

5 Key Interview Tips

5.1 Test Cases and Edge Cases

- Faire interviewers place **heavy emphasis** on test cases
- Be prepared to discuss many edge cases
- Consider using testing frameworks (e.g., JUnit) during the interview
- Even comprehensive testing may not be enough - be ready to think of more cases

5.2 Recent Trends (2024-2025)

- Low error tolerance - even mostly working code may result in rejection
- Difficult for candidates job-hopping while employed due to time constraints
- Senior level may not include BQ or system design in some cases (varies by position)

6 Additional Notes

- **Company:** Faire is a wholesale marketplace (Canadian company with US presence)
- **Interview Difficulty:** Generally rated as "Hard" by candidates
- **Response Time:** Fast - usually hear back same day or next day after each round
- **Process:** HR is responsive and provides feedback

Disclaimer

This information is compiled from multiple public sources including 1point3acres.com, Prepfully.com, and web search results. Actual interview content may vary. Some details on 1point3acres.com were hidden behind paywalls and may not be completely comprehensive. Use this as a reference for preparation, but be prepared for variations in actual interviews.

Sources:

- 1point3acres Faire tag (3 pages): `https://www.1point3acres.com/bbs/tag/faire-7100-1.html`
- Prepfully Faire Interviews: `https://prepfully.com/interview-guides/faire`

- Various web search results for CodeSignal OA format and interview structure

Data collected: November 2025
Document compiled from 10+ interview experiences across multiple sources