PRD

Algorithmia: The Path of Logic - Unified Product Requirements Document (PRD v2.0)

Author: Senior Product Manager, Algorithmia Team
Date: [Insert Date]

==========

1. PRODUCT VISION
   ==========
   Algorithmia is an educational fantasy RPG that transforms learning Data Structures and Algorithms (DSA) into an immersive, emotionally engaging journey. Players explore a Pokémon-style world where logic is magic, and computational thinking drives progress through storytelling, puzzles, and real coding challenges.

Vision Statement:
"Pokémon meets Computer Science — you learn algorithms by living them."

Mission:
Make DSA learning intuitive, visual, and emotionally resonant, turning abstract theory into interactive adventure.

==========

2. OBJECTIVES AND SUCCESS METRICS

Primary Objectives:

- Teach DSA intuitively through narrative-driven gameplay.
- Integrate real coding challenges via the in-game Logic Forge.
- Visualize time and space complexity through Flow and Weight mechanics.
- Create a scalable, open-world system for future regions and community expansion.

Success Metrics:

- Learning Retention: ≥ 80% average quiz accuracy
- Average Playtime: ≥ 25 minutes per session
- Logic Forge Engagement: ≥ 70% attempt rate
- Completion Rate: ≥ 60% of players finish Array Plains
- Player Rating: ≥ 4.5 / 5 satisfaction score

==========

3. MVP SCOPE

Playable Regions (Initial Release):

- Prologue: Chamber of Flow (tutorial on time/space complexity)
- Array Plains: Arrays & Hashing
- Twin Rivers: Two Pointers / Sliding Window

Core Systems:

- Flow & Weight mechanics

- Dialogue, Quest, and Codex systems
- Logic Forge MVP (coding environment + sandbox judge)
- Auth, Cloud Save, and Inventory management

==========

4. CORE FEATURES

4.1 Flow & Weight System

- Flow (💫): Represents time complexity, drains faster with inefficiency.
- Weight (🪶): Represents space complexity, increases with data overhead.
- Both influence gameplay pacing, rewards, and efficiency ranking.

4.2 Logic Forge (Coding Practice Center)

- Monaco Editor integration for real code challenges.
- Node.js + Docker sandbox executes code safely.
- Challenge performance graded via FlowScore & WeightScore.
- Rewards include Flow Potions, Codex unlocks, and items.

4.3 Algorithmia Codex (DSA Pokédex)

- Stores learned concepts, analogies, and completed challenges.
- Visualizes mastery through Flow/Weight balance.
- Connects each Codex entry to related real-world coding problems.

4.4 Guardians (Boss Battles)

- Guardians embody algorithmic inefficiency.
- Defeating them requires solving logic puzzles or efficient traversal tasks.
- Example: The Shuffler (Array Plains) randomizes order; player must restore efficiency.

==========

5. WORLD STRUCTURE

Regions & Learning Mapping:

- Chamber of Flow: Time & Space (tutorial)
- Array Plains: Arrays & Hashing
- Twin Rivers: Two Pointers
- Stack Summit: Recursion, Stack
- Linkvale: Linked Lists
- Binary Ridge: Binary Search
- Arborium: Trees
- Heapspire: Heaps
- Cavern of Shadows: Backtracking

Epilogue: The Terminal of Trials — endgame tower featuring advanced algorithmic challenges (DP, Graphs).

==========

6. SYSTEM ARCHITECTURE

Frontend Stack:

- Next.js (React + TypeScript): Game shell and Logic Forge UI
- Phaser 3: Overworld engine for exploration and puzzles
- Zustand + Immer: Global game state management
- Zod: JSON schema validation for maps, quests, and NPC data
- Tiled → JSON: Map design pipeline

Backend Stack:

- Supabase (Postgres): Database, Auth, Cloud Save
- Node.js + Docker Sandbox: Safe code execution for challenges
- Redis (optional): Task queue for judge service
- PostHog / Sentry: Analytics and crash tracking

==========

7. API DESIGN

Base URL: https://api.algorithmia.app/v1/

Auth APIs:

- POST /auth/login — Log in or register user
- POST /auth/logout — End active session
- GET /auth/profile — Retrieve player profile and save data

Player APIs:

- POST /player/save — Save current progress and inventory
- GET /player/load — Load latest cloud save
- GET /player/inventory — Return player inventory
- GET /player/codex — Return Codex entries

Game Content APIs:

- GET /regions — List all accessible regions
- GET /region/:id — Load region details (tilemap, quests)
- GET /quest/:id — Get quest objective + reward info
- GET /codex/:id — Fetch Codex entry by ID

Logic Forge APIs:

- GET /forge/challenges — List region challenges
- GET /forge/challenge/:id — Return challenge metadata
- POST /forge/run — Execute player code and collect metrics
- GET /forge/result/:id — Retrieve challenge result

Example Request:
{
"language": "typescript",

"challengeId": "two-sum",
"sourceCode": "function twoSum(nums, target){...}"
}

Example Response:
{
"status": "success",
"metrics": {"timeMs": 1.8, "peakMemMB": 12.5},
"flowScore": 92,
"weightScore": 78,
"reward": ["item:flow-potion", "codex:two-sum"]
}

==========

8. DATABASE SCHEMA

Table: users

- id UUID PK
- email VARCHAR
- display_name VARCHAR
- created_at TIMESTAMP
- last_active TIMESTAMP

Table: saves

- id UUID PK
- user_id UUID FK users.id
- region_id VARCHAR
- quests_completed JSONB
- inventory JSONB
- flow INT
- weight INT
- codex_entries JSONB
- updated_at TIMESTAMP

Table: challenges

- id VARCHAR PK
- region_id VARCHAR
- title VARCHAR
- difficulty ENUM('easy','medium','hard')
- statement TEXT
- tests JSONB
- reference_flow VARCHAR
- reference_weight VARCHAR
- rewards JSONB

Table: forge_submissions

- id UUID PK
- user_id UUID
- challenge_id VARCHAR
- language VARCHAR
- time_ms FLOAT
- mem_mb FLOAT
- flow_score INT
- weight_score INT
- passed BOOLEAN
- created_at TIMESTAMP

==========

9. LOGIC FORGE SYSTEM FLOW

1. Player writes code in Monaco Editor.
2. /forge/run request sent → API → Judge Service.
3. Docker sandbox executes code.
4. Results (time, memory, pass/fail) collected.
5. Flow/Weight analyzer computes scores.
6. Submission saved and rewards granted.

Scoring Logic:
flowScore = 100 × (reference_time / player_time)
weightScore = 100 × (reference_mem / player_mem)

Efficiency Grade: S / A / B / C

==========

10. JUDGE SERVICE DESIGN

Architecture Layers:

- Runner: Node.js worker processing jobs.
- Sandbox: Docker container for isolated execution.
- Harness: Custom test runner for I/O validation.
- Metrics Collector: Records CPU/memory/time usage.
- Queue Broker: Redis (optional async jobs).

Security Constraints:

- CPU Time: 2s limit
- Memory: 256MB cap
- File I/O: Disabled
- Network: Disabled
- Output: 1MB max

Execution Sequence:

1. Receive request → validate input.
2. Mount container → run code in sandbox.
3. Collect metrics and compare outputs.
4. Return JSON with Flow/Weight results.

==========

11. CONTENT PIPELINE

Directory Structure:
/packages/content/
regions/
chamber-of-flow.json
array-plains.json
twin-rivers.json
codex/
arrays.json
two-pointers.json
challenges/
two-sum.json
container-with-most-water.json
npcs/
node.json
arria.json

CI Integration:

- All JSON validated via Zod schema.
- Automated checks: ESLint, content:validate, Playwright smoke tests.

==========

12. GAME STATE ARCHITECTURE

GameState Object:
region: string
flow: number
weight: number
quests: Record<string, boolean>
inventory: Item[]
codex: string[]
updateFlow(delta: number)
updateWeight(delta: number)
completeQuest(id: string)

Persistence:

- Local autosave every 30s.
- Cloud sync via /player/save after region completion or challenge.

==========

13. DEVELOPMENT PHASES

Phase 1 – Core Systems: Flow/Weight, Dialogue, Quest Engine
Phase 2 – Prologue + Array Plains: Fully playable base
Phase 3 – Twin Rivers + Logic Forge MVP
Phase 4 – Cloud Sync + Judge Service Deployment
Phase 5 – Polish, Codex UI, and Analytics Integration

==========

14. FUTURE EXPANSION

- Advanced regions: Trees, Graphs, Dynamic Programming.
- AI-powered mentor NPC for hints and adaptive guidance.
- Community region builder with JSON schema validator.
- Mobile-friendly visual learning port.
- Trial Tower endgame with multi-challenge progression.

==========

15. SUMMARY

Algorithmia fuses interactive storytelling, algorithmic mastery, and gamified coding practice.
This PRD defines both product and technical foundations for the first public deployment —
Prologue, Array Plains, and Twin Rivers — with scalable systems for future expansion.

End of Document