DSA ALGORITHM VISUALIZER - TECHNICAL DOCUMENTATION

==========

PROJECT OVERVIEW

Project Name: DSA Algorithm Visualizer

Type: Interactive Web Application

Purpose: Educational platform for learning Data Structures and Algorithms

Framework: React 18 + TypeScript

Build Tool: Vite

TECHNOLOGY STACK

Frontend Framework:

- React 18.3.1 - Modern UI framework with hooks

- TypeScript 5.8.3 - Type-safe development

- Vite 5.4.19 - Fast development server and build tool

Styling & UI:

- Tailwind CSS 3.4.17 - Utility-first CSS framework

- Radix UI - Accessible component library

- Lucide React 0.462.0 - Icon library

- PostCSS - CSS processing and optimization

- Autoprefixer - Cross-browser compatibility

Data Visualization:

- Recharts 2.15.4 - Chart library for React

  - Bar charts for complexity comparison

  - Line charts for performance metrics

  - Custom tooltips and legends

Development Tools:

- ESLint 9.32.0 - Code linting and formatting

- React Hook Form 7.61.1 - Form management

- Zod 3.25.76 - Schema validation

- Sonner 1.7.4 - Toast notifications

- React Router DOM 6.30.1 - Client-side routing

CORE ALGORITHM IMPLEMENTATION

Data Structures Used:

1. FIFO Queue - FCFS algorithm implementation (O(n) time complexity)

2. Min-Heap (Priority Queue) - SRTF and Priority scheduling (O(n log n))

3. Circular Queue - Round Robin algorithm (O(n × q) where q is time quantum)

4. Arrays - Process storage and sorting operations

Algorithm Complexity Analysis:

```
interface AlgorithmComplexity {
  timeComplexity: {
    best: string;     // O(n), O(log n), etc.
    average: string;   // Average case complexity
    worst: string;     // Worst case complexity
  };
  spaceComplexity: string;    // Space requirements
  dataStructures: string[];   // Data structures used
}
```

Algorithm Implementations:

- FCFS: O(n) time complexity, FIFO queue

- SJF: O(n log n) average, $O(n^2)$ worst, array sorting

- SRTF: O(n log n) average, $O(n^2)$ worst, min-heap priority queue

- Priority: O(n log n) average, $O(n^2)$ worst, min-heap priority queue

- Round Robin: O(n × q) where q is time quantum, circular queue

DESIGN SYSTEM

Color Palette:

- Primary: Purple gradient (#8b5cf6, #a855f7)
- Secondary: Blue (#5b9cff, #3b82f6)
- Accent: Cyan (#06b6d4, #0891b2)
- Success: Green (#22c55e, #16a34a)
- Warning: Yellow/Orange (#eab308, #f97316)
- Error: Red (#ef4444, #dc2626)

Typography:

- Font Family: Inter (Google Fonts)
- Weights: 400 (Regular), 500 (Medium), 600 (Semibold), 700 (Bold), 800 (Extra Bold)
- Sizes: Responsive scaling from xs (12px) to 3xl (30px)

UI Components:

- Glass Cards: Frosted glass effect with backdrop blur
- Buttons: Gradient backgrounds with hover states
- Forms: Floating labels and validation states
- Charts: Custom styled with consistent color scheme

DATA FLOW ARCHITECTURE

State Management:

```
const [processes, setProcesses] = useState<Process[]>([]);
const [timeline, setTimeline] = useState<TimelineItem[]>([]);
const [metrics, setMetrics] = useState<Metrics | null>(null);
```

const [algorithm, setAlgorithm] = useState<AlgorithmType>('FCFS');

Data Processing Pipeline:

1. Input → Process creation (manual or CSV)

2. Validation → Form validation and type checking

3. Algorithm Selection → Choose scheduling algorithm

4. Execution → Run algorithm with process data

5. Visualization → Generate timeline and metrics

6. Analysis → Complexity comparison and detailed breakdown

Component Communication:

- Props drilling for data flow

- Callback functions for state updates

- Custom hooks for shared logic

- Context not used (simple state management)

BUILD & DEPLOYMENT

Development Commands:

npm run dev # Start development server (localhost:8080)

npm run build # Build for production

npm run preview # Preview production build

npm run lint # Run ESLint

Build Configuration:

- Vite Config: TypeScript support, React plugin

- Output: Single-page application with optimized assets

- Bundle Size: ~500KB (gzipped)

- Browser Support: Modern browsers

DEPENDENCIES

Production Dependencies:

- React ecosystem (React, TypeScript, Tailwind)
- UI libraries (Radix UI, Lucide, Recharts)
- Utilities (React Hook Form, Zod, Sonner)

Development Dependencies:

- Vite (build tool)
- ESLint (linting)
- PostCSS (CSS processing)
- TypeScript compiler

KEY FEATURES

- Interactive Visualizations: Real-time algorithm execution with animations
- Complexity Analysis: Big O notation breakdowns and comparisons
- Data Structure Demos: Queues, heaps, and arrays in action
- Educational Tools: Step-by-step explanations and info modals
- Modern UI: Glassmorphism design with smooth animations
- Sound Effects: Audio feedback for process events
- CSV Import: Bulk process data upload
- Responsive Design: Works on desktop and mobile

PERFORMANCE METRICS

- Bundle size: ~500KB (gzipped)
- First Contentful Paint: <1s
- Time to Interactive: <2s
- Lighthouse Score: 95+ (Performance, Accessibility, Best Practices)

BROWSER SUPPORT

- Chrome 90+
- Firefox 88+
- Safari 14+
- Edge 90+

ALGORITHM COMPLEXITY REFERENCE

Time Complexity Levels (from best to worst):

O(1) - Constant time

O(log n) - Logarithmic time

O(n) - Linear time

O(n log n) - Linearithmic time

$O(n^2)$ - Quadratic time

$O(n^3)$ - Cubic time

O(2^n) - Exponential time

O(n!) - Factorial time

Space Complexity:

All algorithms use O(n) space for process storage and timeline data.

EDUCATIONAL VALUE

DSA Concepts Demonstrated:

- Queue Operations: Enqueue, dequeue, circular queue behavior
- Heap Operations: Insert, extract-min, heapify
- Sorting Algorithms: Comparison of different approaches
- Complexity Analysis: Big O notation in practice
- Data Structure Selection: Choosing the right structure
- Algorithm Trade-offs: Time vs space, simplicity vs efficiency

Target Audience:

- Computer Science students
- DSA learners
- Algorithm enthusiasts
- Educators and teachers

Learning Outcomes:

- Understanding algorithmic complexity
- Visualizing data structure operations
- Comparing algorithm efficiency
- Applying theoretical concepts to practical problems

==========

Generated: February 2026

Project Repository: DSA Algorithm Visualizer