# Elevation Sort Visualizer – Final Report

**Team Information:**
**Team Name:** Elevation Sort Visualizer
**Team Members:**
Arion Stern (GitHub: arionstern)
Peter Walsh (GitHub: peterwalsh3)
**GitHub Repository:** https://github.com/arionstern/DSA_P3
**Demo Video:** https://youtu.be/1cRVrCb5_Lg?si=8P1aUyK4040bAnMq

**Extended & Refined Proposal:**

**Problem:**
Sorting algorithms are abstract and often difficult to visualize for learners. Likewise, interpreting real-world elevation data in raw numerical form can be unintuitive. Our project bridges both challenges by offering an interactive visual tool that animates sorting algorithms applied to elevation data.

**Motivation:**
Elevation data is spatial and continuous, making it an ideal candidate for sorting-based visual learning. This project leverages real elevation measurements across random geographic coordinates and allows users to sort them using multiple algorithms while watching the data transform live. This bridges the gap between abstract algorithms and tangible understanding. It also helps interpret large raw datasets in a digestible format.

**Features Implemented:**

The program fetches real elevation data from the NOAA ETOPO1 dataset using the BRIDGES API. It includes interactive buttons for selecting among six sorting algorithms—Quick Sort, Merge Sort, Insertion Sort, Selection Sort, Heap Sort, and Bubble Sort—within the visualizer. Sorting steps are animated using Pygame, with a display showing sorting time, comparison count, and swap count. A color-coded bar visualization represents elevation values, with dynamic themes including terrain, grayscale, and heatmap. Users can view a side-by-side elevation heatmap comparison before and after sorting using Matplotlib. A hover tooltip displays coordinates and elevation for each data bar, while keyboard controls allow resetting the dataset, shuffling the current values, and cycling through themes. The interface also shows a dynamic summary of the lowest, median, and highest elevation values during interaction.

**Data Used:**
We use a rectangular grid of elevation data pulled randomly using the BRIDGES API. The API gives access to a random portion of NOAA's ETOPO1 elevation map. The number of rows and columns can be configured by the user at runtime (e.g., a 10x10 grid = 100 values). Each data point consists of latitude, longitude, and elevation in meters.

**Tools / Languages / Libraries**
**Language: Python 3.12**
**Libraries:**

- pygame for visualization and GUI
- matplotlib for heatmaps
- bridges for data access
- numpy for data reshaping

**Algorithms Implemented:**

Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort

**Each was animated frame-by-frame with comparisons and swaps tracked.**

**Additional Structures**

- Tuples with elevation + index used for stability in sorting
- Custom rendering system for scaling and drawing visual bars
- Sorting metrics dictionary (comparisons, swaps) to track steps
- Optional hover and theme parameters passed to rendering functions

**Division of Work**
**Arion Stern:** Sorting algorithm logic, metric tracking, sorting animations, and overall debugging.
**Peter Walsh:** Heatmap plotting, color theme switching, elevation data integration, and user interaction features.
**Both:** Collaborated on UI design, testing, and presentation planning.

**Analysis:**

**Post-Proposal Changes**
**Several enhancements were made beyond the original scope of our proposal:**

- GUI-Based Sort Selection: Originally, sorting was to be selected via the command line. We transitioned to interactive GUI buttons inside the Pygame window for smoother user experience.

- Sorting Metrics Display: We added real-time metrics tracking, including comparison count, swap count, and total sort time, to help users understand algorithm efficiency.

- Color Themes and Visualization Options: In addition to the default terrain colors, we introduced dynamic color themes (terrain, grayscale, and heatmap), which can be toggled with a key press. This makes the elevation trends more visually flexible.

- Heatmap Enhancements: The basic elevation heatmap was upgraded to a side-by-side “before and after” comparison, allowing users to visually evaluate how sorting affects spatial elevation distribution.

- Hover Tooltips: Users can hover over any bar to see the associated latitude, longitude, and elevation, making the visualization more informative and interactive.

- Keyboard Shortcuts: We added several keyboard controls (e.g., R to reset, S to shuffle, C to change theme, ESC to quit) to improve usability without relying solely on the GUI.

- Shuffle Without Reset: Users can now shuffle the current data without regenerating new coordinates, preserving the original dataset while testing different sorting behaviors.

**Time Complexity (Worst Case):**

**Variable Declarations:**
Let n be the total number of elevation data points (n = rows × cols)
Let rows be the number of latitude grid points
Let cols be the number of longitude grid points

**Sorting Algorithms:**

| Algorithm | Time Complexity | Why |
|---|---|---|
| Bubble Sort | $O(n^2)$ | Two nested loops are used: the outer loop runs n times, and the inner loop iterates up to n - i - 1 times. Each comparison and potential swap is done inside the nested structure, resulting in quadratic operations. |
| Insertion Sort | $O(n^2)$ | The outer loop runs from index 1 to n, and the inner while loop may shift all previous elements in the worst case. This results in up to n shifts per element. |
| Selection Sort | $O(n^2)$ | For each index i, the algorithm compares it to all remaining elements j > i using a nested loop. The number of comparisons is constant per element and grows quadratically overall. |
| Merge Sort | O(n log n) | The merge_sort() function splits the list in half recursively, and merge() combines sublists using linear-time logic. There are about log n levels of recursion, each processing n elements. |
| Quick Sort | $O(n^2)$ | In the worst case (such as sorted input), the partition() function splits the list into a single element and the rest. The recursion becomes unbalanced, leading to n levels with up to n comparisons each. |
| Heap Sort | O(n log n) | The heap is built using a bottom-up approach in linear time. During sorting, heapify() is called n times, and each call may traverse to the bottom of the heap (up to log n steps). |

**Rendering + Visual Elements:**

| Function | Time Complexity | Why |
|---|---|---|
| draw_bars() | O(n) | The function loops through all n elevation data points using enumerate(data) and draws a bar for each value. Each drawing operation is constant time. |
| draw_color_legend() | O(1) | The color legend is drawn using a loop with a fixed width of 200 pixels. The number of iterations does not depend on input size. |
| Hover tooltip lookup | O(1) | The index of the hovered bar is computed using integer division: mouse_x // bar_width, which is a single arithmetic operation independent of data size. |
| get_summary_text() | O(n log n) | The function sorts the data by elevation using Python's built-in sorted(), which takes O(n log n), and then accesses three indices. |

**Main Visualizer Loop:**

| Part of Loop | Time Complexity | Why |
|---|---|---|
| Event handling | O(1) | Events such as keypresses and mouse clicks are handled using if and elif statements, which are constant time operations. |

| | | |
|---|---|---|
| Sorting after click | $O(n^2)$ or O(n log n) | When a sort button is clicked, one of the sorting algorithms is run once on the list. The complexity depends on the selected algorithm. |
| Redraw frame | O(n) | Each frame calls draw_bars() once to redraw all n elevation bars. The loop inside draw_bars() scales linearly. |

**Elevation Data:**

| **Function** | **Time Complexity** | **Why** |
|---|---|---|
| get_elevation_grid() | O(rows × cols) | The function uses two nested loops: the outer loop runs over rows and the inner over cols. Each iteration appends one elevation point to the result list. |
| API call itself | O(1) | The get_elevation_data([...]) function is a single external call that returns a fixed-size region. Its duration does not scale with the number of requested points. |

**Heatmap Plotting:**

| **Function** | **Time Complexity** | **Why** |
|---|---|---|
| show_elevation_heatmap() | O(n) | The function converts a flat list of n elevation values into a NumPy array and reshapes it into a rows × cols grid. Each value is processed once. |
| show_comparison_heatmap() | O(n) | Two elevation lists of size n (original and sorted) are reshaped and visualized side by side. All processing is done linearly. |

**Reflection:**

**Overall Experience**
This project was an engaging way to connect abstract algorithm logic to meaningful real-world data. By combining geographic elevation with sorting algorithms, we built an interactive educational tool that bridges computer science and Earth data.

**Challenges Faced**

- Implementing stable sorting behavior with visual continuity required careful tracking of original indexes.
- Rescaling and managing frame rates for large datasets (ex: 50x50 grids) introduced complexity.
- Keeping the Pygame interface responsive while handling user interaction and drawing logic was also difficult.
- Debugging color mapping inconsistencies and aligning them with the legend was harder than expected.

**Improvements if Starting Again**

- We would modularize the visualizer earlier to make code easier to extend and test.
- Might incorporate a scrollable or zoomable view for large data input grids.
- Could explore labeling the heatmap with real location names via an API like OpenStreetMap.

**Individual Learning**

- **Arion:** Learned how to build real-time visual interfaces with Pygame and integrate API-based data into algorithm visualizations.
- **Peter:** Gained experience with applying color theory to represent data visually, working with event-driven GUI programming, and understanding how different sorting algorithms behave in practice.

**References**


- NOAA ETOPO1 Elevation Dataset via BRIDGES: https://bridgesuncc.github.io/tutorials/Data_Elevation.html
- BRIDGES Python API Documentation
- Tech With Tim – Sorting Visualizer with Pygame (YouTube & GitHub)
- Matplotlib: https://matplotlib.org/
- Pygame Documentation: https://www.pygame.org/docs/