Sorting Algorithm Visualization - Project Documentation

This project is a visualization of popular sorting algorithms using the SDL library in C++. The program allows the user to visualize the sorting process of different algorithms like Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort in real-time.

1. Initializing SDL

SDL (Simple DirectMedia Layer) is a cross-platform library used for multimedia applications, including game development. In this project, SDL is used to render and display the sorting process visually.

Function: init()

This function initializes SDL by calling SDL_Init(SDL_INIT_VIDEO). It also creates a window and renderer using SDL_CreateWindow and SDL_CreateRenderer respectively. If any step fails, an error message is displayed, and the program stops.

Code Explanation:

SDL_Init(SDL_INIT_VIDEO) - Initializes the SDL video subsystem.

SDL_CreateWindow(...) - Creates a window for rendering.

SDL_CreateRenderer(...) - Creates a renderer to render graphics to the window.

2. Closing SDL

The close() function is responsible for cleaning up and freeing the resources used by SDL, such as the renderer and the window. After destroying these, it calls SDL_Quit() to shut down all SDL subsystems.

Code Explanation:

SDL_DestroyRenderer(renderer) - Destroys the renderer created.

SDL_DestroyWindow(window) - Destroys the window created.

SDL_Quit() - Cleans up all initialized SDL subsystems.

3. Drawing the Current State of the Array

The drawState() function visually represents the state of the array at each step of the sorting process. It draws colored vertical lines where each line's height represents an element of the array.

Code Explanation:

SDL_SetRenderDrawColor(...) - Sets the drawing color (RGB) for rendering.

SDL_RenderClear(renderer) - Clears the screen to prepare for new drawing.

SDL_RenderDrawLine(renderer, ...) - Draws a vertical line corresponding to an array element.

SDL_RenderPresent(renderer) - Updates the window with the drawn lines.

4. Sorting Algorithms

4.1 Bubble Sort

Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

Key steps:

swap(v[i], v[j]) - Swaps elements if they are in the wrong order.

drawState(v, renderer, ...) - Visualizes the array state after each comparison.

4.2 Selection Sort

Selection Sort is an in-place comparison sorting algorithm. It has $O(n^2)$ time complexity, which makes it inefficient on large lists. During each iteration, it selects the smallest (or largest, depending on the sorting order) element from the unsorted part and swaps it with the first unsorted element.

Key steps:

swap(v[i], v[sortedElem]) - Swaps the smallest element with the first unsorted element.

drawState(v, renderer, ...) - Visualizes the array after each iteration.

4.3 Insertion Sort

Insertion Sort is another simple sorting algorithm that builds the sorted array one item at a time. It iterates through the array and, at each step, inserts the current element into its correct position in the sorted part of the array.

Key steps:

While (j >= 0 && v[j] > key) - Compares the key element with previous elements and shifts larger elements to the right.

drawState(v, renderer, ...) - Shows the visual update during sorting.

4.4 Merge Sort

Merge Sort is a divide-and-conquer algorithm that splits the array into smaller subarrays, sorts them, and then merges them back together. It's known for its O(n log n) time complexity.

Key steps:

merge(...) - Merges two sorted subarrays into one sorted array.

drawState(v, renderer, ...) - Visualizes the merging process.

4.5 Quick Sort

Quick Sort is an efficient divide-and-conquer algorithm. It picks an element as a pivot and partitions the array around the pivot such that elements smaller than the pivot come before it, and larger elements come after it.

Key steps:

partition(...) - Partitions the array and places the pivot in its correct position.

drawState(v, renderer, ...) - Shows the sorting in action after partitioning.

5. Conclusion

In this project, we explored how to visualize various sorting algorithms using SDL. By creating visual representations of the sorting process, we can understand the inner workings of each algorithm. SDL functions like SDL_Renderer, SDL_RenderDrawLine, and SDL_SetRenderDrawColor allow us to render graphics easily, making SDL an excellent tool for such educational projects.