**SORTING VISUALIZER DOCUMENTATION ([LINK](https://github.com/kumaripriyanshukp/shorting-visualizer))**

**OVERVIEW**

The sortingVisualizer class is a Java Swing-based graphical application that visually demonstrates different sorting algorithms on an integer array. The array is displayed as a series of vertical bars, where each bar’s height corresponds to the value of the element in the array.

This visualizer supports five sorting algorithms:

* Bubble Sort
* Selection Sort
* Insertion Sort
* Merge Sort
* Heap Sort

Users can interact with the application through buttons to trigger any sorting algorithm or reset the array to a new random configuration.

**FEATURES**

* **Visual representation:** Displays an array as bars with heights proportional to the element values.
* **Multiple sorting algorithms:** Implements five popular sorting methods with step-by-step animation.
* **Multithreaded animation:** Sorting runs in a separate thread to maintain UI responsiveness.
* **Array reset:** Users can reset the array to a fresh set of random values.
* **Interactive GUI:** Buttons to select and run sorting algorithms or reset the array.

**CLASS: SORTINGVISUALIZER**

**Fields**

* private int[] array  
  Holds the integer array currently being visualized and sorted.
* private final int SIZE  
  Defines the size of the array (20 elements).
* private final int BAR\_WIDTH  
  Width in pixels of each bar representing an array element.
* private final int BAR\_HEIGHT\_MULTIPLIER  
  Multiplier to scale the bar height visually according to the element’s value.
* private boolean isSorting  
  A flag indicating if a sorting operation is currently running (prevents multiple simultaneous sorts).

**Constructor**

* public sortingVisualizer()  
  Initializes the panel with a random integer array and sets the preferred size of the drawing area.

**Methods**

**private int[] generateRandomArray()**

Generates a new array filled with random integers between 1 and 30.

**Returns:**  
A new integer array of length SIZE containing random values.

**protected void paintComponent(Graphics g)**

Overrides JPanel’s paint method to draw the array bars and their corresponding values.

**Parameters:**  
g – The Graphics object used for drawing.

**private void swap(int i, int j)**

Swaps the elements at indices i and j in the array, repaints the panel, and pauses briefly to animate the swap.

**Parameters:**

* i – First index
* j – Second index

**Sorting Algorithms (each runs on a new thread)**

* public void bubbleSort()  
  Implements the bubble sort algorithm with animation.
* public void selectionSort()  
  Implements the selection sort algorithm with animation.
* public void insertionSort()  
  Implements the insertion sort algorithm with animation.
* public void mergeSort()  
  Initiates merge sort with animation.
* public void heapSort()  
  Implements heap sort with animation.

**Helper methods for merge sort and heap sort**

* private void mergeSortHelper(int left, int right)  
  Recursive method for merge sort that divides the array and calls the merge function.
* private void merge(int left, int mid, int right)  
  Merges two sorted subarrays into one sorted subarray, with animation.
* private void heapify(int n, int i)  
  Maintains the heap property for heap sort.

**public void resetArray()**

Generates a new random array and repaints the panel to display it.

**Main Method**

* public static void main(String[] args)

Creates the main window frame, adds the sortingVisualizer panel and a button panel containing buttons for each sort and the reset action. Each button is linked to the respective sorting or reset method.

**Usage Instructions**

1. Run the program. A window titled "Sorting Visualizer" appears, showing a random array as vertical bars.
2. Click any of the sorting algorithm buttons (Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Heap Sort) to start the animated sorting.
3. Click "Reset" to generate a new random array for visualization.
4. Sorting animations will show how the array elements get sorted step-by-step.
5. The interface disables starting another sort while a sort is already running.