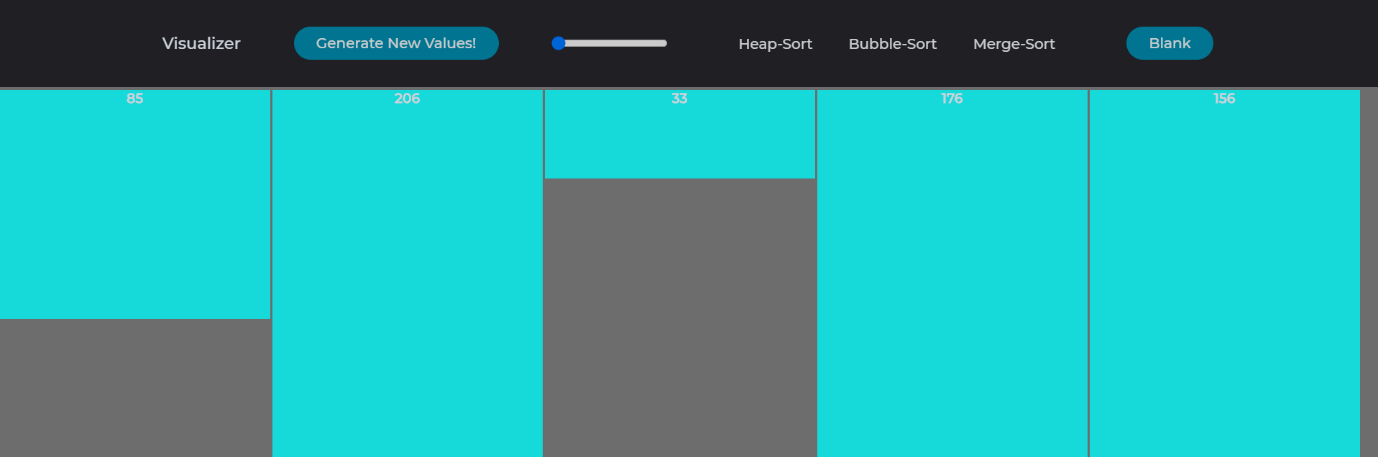
**Goal:** Build a Visualizer Tool for well-known sorting algorithms.

**Tech-Stack**: ReactJS, CSS, HTML, Bootstrap.

**UI-Layout**:

Sample:



**Source-Code**: Uploaded to GitHub(<https://github.com/AkhilReddy1998/Sorting-algos-visualizer>)

**Components**:

This Project is ReactJS heavy and also a lot of CSS nuances must be known in prior

**Source Code:**

Base Class: Visualizer extends Component (React)

**Class**: Visualizer

Constructor: Calls constructor in super class.

State: Very Important, this is called state variable, responsible for displaying/altering our page view

1. In this state container, we have included items named list & NUMBER\_OF\_ARRAY\_BARS.
2. list: This contains all the numbers which are displayed on screen
3. NUMBER\_OF\_ARRAY\_BARS: Initialized as 5, but it stores the length of list array.

**render()**: (Program Executions starts from this function)

1. widhtOfBars(var): We invoke a method(getWidthOfBars) to find the perfect width size for our bars displayed on screen.
2. displayNumberOnBar(var): We invoke a method (getNumberOnBar) to fins if we can display the number on the Bars displaying on screen.
3. return(method): All the HTML is enclosed inside a <div> tag
   1. <header> tag: Contains all the header elements i.e. Title, Buttons
   2. <div className="container"> tag: This occupies the rest of Web page and displays all the bars.
4. <header> Tag:
   1. <a onClick={this.generateNewValues}> Generate New Values: Clicking this will generate new random value into our list array inside state, invokes method(**generateNewValues**) for computation.
   2. <input id=”slider”>: This allows users to change the number of elements displayed on the screen, min value is 3 and max is 400, invokes method(**reSizeArray**) for computation.
   3. Remaining are Buttons which invoke sorting methods respectively.
5. <div className="container"> tag: This renders all the bars.
   1. this.state.list.map: This loop us through every element inside list array in the state. If the list has 5 elements, the loop runs 5 times and displays 5 bars.
   2. <div className="normalBar">: class(normalBar) makes sure that the bar colour is aqua-blue, **id,keY** are responsible for uniquely identifying this bar when we sort elements, **style** this inline style is responsible for height & width of the bar, height=3\* number & width is computed from above point(1)
   3. <p> Tag: This is responsible for displaying the number on the bar, if bar count increases the number disappears to avoid UI collisions, computed from above point (2)

**getWidthOfBars()**: Responsible for calculating width of the bars displaying. Takes two parameters bar\_count, screen\_width.

1. bar\_count: This represents number of bars displaying on screen
2. screen\_width: We can get the current devices screen\_width from JS variable window, i.e. window.innerWidth

Now we have to decrease screen\_width by 20px (Because right-scroll bar occupies 20px in browsers).

For every bar we have a margin-left of 2px in CSS (This gives us a gap between bars), so we take this under consideration and subtract 2px\*number\_of\_bars from our screen\_width.

Now divide screen\_width by the bar count to get the width for each bar.

**getNumberOnBar()**: To check if we can display a number on the bar

If the bar count is less than 65, we could display the number because of the width allocated to the bars else return transparent to make the number disappear.

**reSizeArray():** When user uses the slider, we call this method to increase/decrease the number of bars.

1. We use setState method to change the NUMBER\_OF\_ARRAY\_BARS, event.target.value is responsible for fetching the number on slider.
2. Now the list array should be changed in parallel, so we write a call back method in setState to generate new values (method mentioned below)

**generateNewValues()**: This is invoked from reSizeArray method, so we have to increase/decrease the number of elements inside list array based on slider’s current value.

1. Use setState to make the list array empty
2. Invoke generateValuesInArray with parameter of number of array bars to compute the list array.

**generateValuesInArray()**: Responsible for generating list array with random numbers.

1. We set min, max and loop to get random numbers inserted into our array
2. After generating the list, we use setState to modify our display with new values in list array.

**sleep()**: Responsible for generating a pause when required in our program. Takes milliseconds as input.

**changeColorOnNodes()**: Responsible for changing the colour of an element, taken colour, elementID as inputs.

**async()**: This is not a method but it should be used if we want to use Timeouts, above sleep method inside our method.

**getTimeToPause()**: This is responsible for finding the number of milliseconds to pause while sorting, more elements means less time to pause in between the sorting.