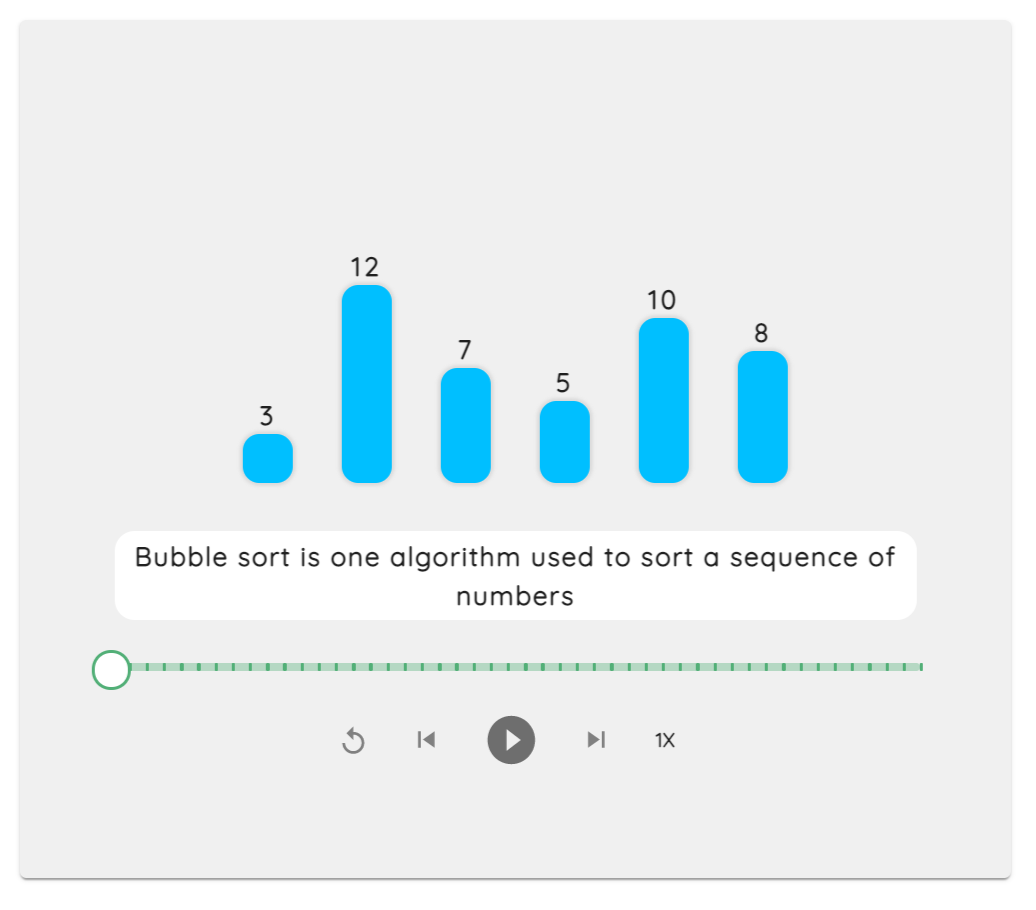
SwapAnimation

The animation of sorting algorithms is based on a fundamental component called SwapAnimation. It is used to animate an array of number as bars with different height, which represents the value of each number. It provides a smooth animation in swapping elements. In order to visualise the comparing, swapping, insertion and merging process in an intuitive way, colours and y-axis value of bars can also be changed smoothly by this component. SwapAnimation takes a property called trace which is an array of bar states of a sequence of time frames. Each element in the trace array contains the values, colours of bars and an explanation sentence which will be displayed in the explanation box. After certain time interval, this component will visualise series of bars as is shown in figure X to illustrate the process of sorting.



The animation is realised by a component “motion” from a third-party library Framer Motion. Transition property is the animation effect and animate property could adjust background colour and x, y position of a bar.

<motion.li

    key={bar.key} // each bar's identification

    layout

    transition={spring}

    style={bar}

    className={classes.bar}

    animate={{

    backgroundColor: bar. backgroundColor,

    y: bar.y,

    x: bar.x,

    }}

>

Code X

The timeline of animation is defined by a function called setTimeout. We use the order of each trace to set different timeout period for traces so that bars would be modified in a certain time interval.

subTrace.forEach((item, i) => {

    let timeoutId = setTimeout(

        (item) => {

            // update the current step

            setCurrentStep((prevStep) =>

                i === trace.length - 1 ? prevStep : prevStep + 1

            );

            // update bars to be animated

            setBars(item);

            i === subTrace.length - 1

                ? setIsPlaying(false)

                : setIsPlaying(true);

        },

        i \* timer, //time interval

        item

    );

    timeoutIds.push(timeoutId);

});

Code Y

To increase the degree of interaction with users and interest, this component consists of three subparts, AnimationControl, AnimationSlider and InputBar, allowing users to control the animation and define the array of numbers.

AnimationControl

This component contains five buttons which are reset, step backward, play/pause, step forward and speed.

Pause is realised by clear the timeout arguments set in the Code Y.

// It is used to clean timeouts to pause the animation

const clearTimeouts = () => {

    timeOutIds.forEach((timeoutId) => clearTimeout(timeoutId));

    setTimeOutIds([]);

};

// To pause the animation

const pause = () => {

    setIsPlaying(false);

    clearTimeouts();

};

Code Z

Play is realised by taking the incomplete traces as new trace for the component to play.

// To resume the animation

const resume = () => {

    setIsPlaying(true);

    const newtrace = trace.slice(currentStep);

    run(newtrace);

};

Code A

Reset is realised by pause the animation and set the trace to the initial state.

const handleResetClick = () => {

    pause();

    setCurrentStep(0);

    setBars(trace[0]);

};

Code B

Step forward and backward are realised by increasing or decreasing the trace index by one and display the updated state.

// Go to next step and pause

const stepForward = () => {

    if (currentStep < trace.length - 1) {

        pause();

        const item = trace[currentStep + 1];

        setCurrentStep((prevStep) => prevStep + 1);

        setBars(item);

    }

};

// Go to the previous step and pause

const stepBackward = () => {

    if (currentStep > 0) {

        pause();

        const item = trace[currentStep - 1];

        setCurrentStep((prevStep) => prevStep - 1);

        setBars(item);

    }

};

Code C

AnimationSlider

AnimationSlider is a progress bar with scales on it. Users can drag the thumb on it to go to a specific frame of the whole animation.

const handleSliderChange = (event, newValue) => {

    if (isPlaying) {

        pause();

    }

    const item = trace[newValue];

    setCurrentStep(newValue);

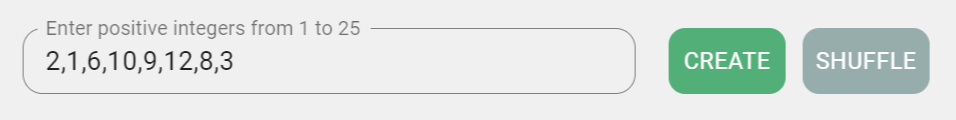
    setBars(item);

};

Code D

InputBar

InputBar allows user to type in numbers to create a self-defined array of bars for visualisation. This component has a functionality to restrict the user input. An error message will be shown if users tried to visualise an array with large number or size or characters other than integers. A shuffle button is provided to shuffle the bars with random size and numbers.



This is realised by modifying an third-party input component from Material-UI library.