**5.n Implementing Necessary Components of Interface**

**5.n.m SwapAnimation**

**Animation and timeline**

The animation is realised by a component “motion” from a third-party library, Framer Motion [1]. Transition property produces the animation effect. In the implementation of this project, a spring transition type is used to simulate the motion of a spring. Animate property can adjust background colour and x, y position of a bar. In the animation of sorting process, a bar could change its colour and x, y axis value. This motion components will identify each bar by their keys and move them smoothly.

<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 1

The timeline of animation is defined by a function called setTimeout. According to each trace's order different timeout periods will be set for traces so that bars would be modified after each specific time interval. At each specific time frame, variable currentStep and bars will be set for motion component to visualise each step.

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 2

**Animation control and auxiliaries**

This component consists of four subparts, AnimationControl, AnimationSlider, ExplanationBox and InputBar, allowing users to control the animation and define the array of numbers.

**AnimationControl**

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

Pause is realised by clear the timeout arguments set in Code 2.

// 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 3

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

// To resume the animation

const resume = () => {

    setIsPlaying(true);

    const newtrace = trace.slice(currentStep);

    run(newtrace);

};

Code 4

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

const handleResetClick = () => {

    pause();

    setCurrentStep(0);

    setBars(trace[0]);

};

Code 5

Step forward and backwards are realised by increasing or decreasing the trace index by one and displaying 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 6

**AnimationSlider**

AnimationSlider is a progress bar with scales and a thumb on it. Users can drag the thumb to go to a specific time frame of the whole animation. Each time the value of the slider changes, it will update the state of bars.

const handleSliderChange = (event, newValue) => {

    if (isPlaying) {

        pause();

    }

    const item = trace[newValue];

    setCurrentStep(newValue);

    setBars(item);

};

Code 7