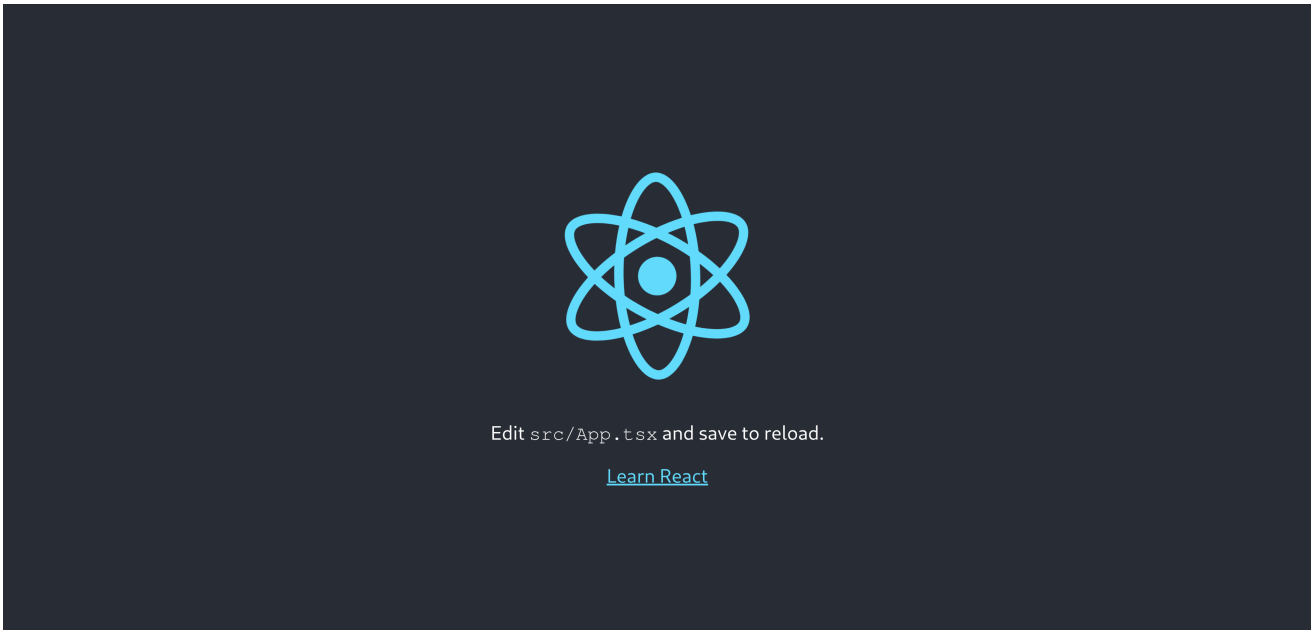
EPQ

To start off with, I create a typescript react app with the command `npx create-react-app artefact --template typescript`. This creates a folder called artefact containing all starting code for the project. This folder contains some boilerplate code, most of which will be removed. running the command `npm start` starts the boilerplate app, which looks like this:



The artefact folder looks like this:

```
artefact
├── node_modules
├── package.json
├── package-lock.json
├── public
├── README.md
├── src
└── tsconfig.json
```

`node_modules` contains the required modules for the project. `Public` contains the `index.html` document where the react app will be injected. And `src` is where the typescript files for the app will go.

I start off by removing all the boilerplate code from the src directory. This leaves me with four files,

```
artefact/src
├── App.css
├── App.tsx
├── index.css
└── index.tsx
```

`App.css` is empty and will be where all the component's CSS will go
`App.tsx` is the typescript file where the react components will go. Currently, it contains:

```
import React from "react";
import "./App.css";

function App() {
    return <div className="App"></div>;
}

export default App;
```

This code returns an empty div element to be injected into the app.

`index.css` contains some extra CSS for index.html, currently it just contains some font styling:

```
body {
    margin: 0;
    font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto",
        "Oxygen", "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans",
        "Helvetica Neue", sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}

code {
    font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",
        monospace;
}
```

`index.tsx` contains code that injects the empty app element created in `App.tsx` :

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";

ReactDOM.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>,
    document.getElementById("root")
);
```

Sorting

The sorting algorithm page will contain these elements:

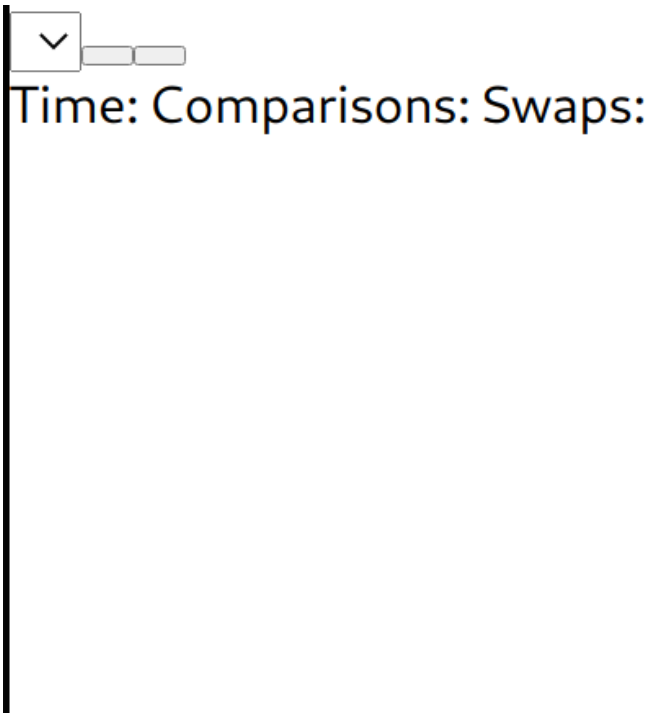
- The algorithm visuliser
- A select element to select the selected algorithm
- Buttons to start, stop and reset the visualizer
- Scrolling slide bars to select the number of bars to be sorted, and the sorting speed
- A panel containing metrics on the algorithm, which will display the time taken, number of comparisons and number of swaps
- A place to describe the selected algorithm

So to start off with, I need to create these components inside `App.tsx`. I have not decided on a final visual design for the application, so I will not style the components any more than required to get them working. Then, once I have decided on how the application will look, I will add the CSS to style the components. The file `App.css` now looks like this:

```
import React from "react";
import "./App.css";

// React element for the bar container
const BarContainer = () => {
    return <div className="barContainer"></div>;
};

// React element for the Controls
const Controls = () => {
    return (
        <div className="controls">
            <select className="algorithmSelect"></select>
            <button className="startstop"></button>
            <button className="reset"></button>
            <input type="range" />
            <input type="range" />
        </div>
    );
};

// React element for the Metrics
const Metrics = () => {
    return (
        <div className="metrics">
            <span>Time: </span>
            <br />
            <span>Comparisons: </span>
            <br />
            <span>Swaps: </span>
        </div>
    );
};

// react element for the Description
const Description = () => {
    return <div className="description"></div>;
};

// Base app
function App() {
    return (
        <div className="App">
            <BarContainer />
            <Controls />
            <Metrics />
            <Description />
        </div>
    );
}

export default App;
```

'App.tsx` now contains empty components for the elements of the page. I decided to put the speed, swaps, and comparisons in their own components to make things easier later when I have to update them with their values dynamically.

Now when I run the app it looks like this:



Because there's no styling at all, it is very hard to tell what is going on. Therefore I added some basic CSS styling for the bar container inside `App.css` :

```
.barContainer {
    width: 80vw;
    height: 20rem;
    background: black;
    margin: auto;
}
```

This, plus some added text inside the controls, makes the app now look like this:

Now to add some bars, I started by by creating an interface and functional component for the bar:

```
//interface for the props passed into the bar
interface barProps {
    size: number;
    maxSize: number;
    key: number;
}
// React element for the bars
const Bar = (props: barProps) => {
```

```
    // CSS styles for bar
    var style: CSSProperties = {
        height: ((props.size / props.maxSize) * 100).toString() + "%",
    };
    return <div className="bar" style={style}></div>;
};
```

The interface declares the type of props being passed into the bar. These props are then used to determine the height of the bar. The height of a bar is represented as $\left(\frac{BarSize}{MaxBarSize} \times 100\right)$% of the bar containers height.

To create the bars, I first needed a way of creating and storing the size of each bar. I achieved this by creating a `bars` array inside the program state. This array will contain the size of each bar. I also added a variable, `numOBars`, which contains the number of bars to be visualised.

I then created a method that adds the numbers 1 -> numOBars to the bars array. This method is then run when the page loads to create the bars.

```
// Base app
class App extends React.Component {
    componentDidMount() { // runs once the component has been loaded
        this.makeBars();
    }

    state = {
        numOBars: 50,
        bars: [],
    };

    // method to make the bars
    // if no parameters are entered, n is taken to be the number of bars
    makeBars = (n = this.state.numOBars) => {
        var b = [];
        for (let i = 0; i < n; i++) {
            // create an array b containing 1 -> n
            b.push(i + 1);
        }
        this.setState({ bars: b, numOBars: n }); // sets the state of bars to be
b
    };
```

These state values are then passed into the barContainer component, where they are used to create the bars.

```
//interface for the props passed into the bar container
interface barContainerProps {
    bars: Array<any>;
    maxSize: number;
}

// React element for the bar container
const BarContainer = (props: barContainerProps) => {
    return (
        <div className="barContainer">
            {props.bars.map(
                (
```

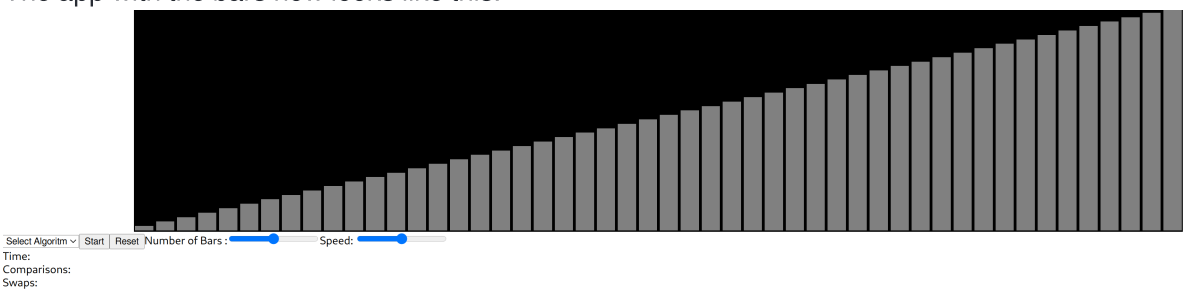
```
                    bar //loop through all the array, creating a bar for each
element
                ) => (
                    <Bar size={bar} maxSize={props.maxSize} key={bar} />
                )
            )}
        </div>
    );
};
```

To make the bars centred and visible, I added this CSS styling

```
.barContainer {
    width: 80vw;
    height: 20rem;
    background: black;
    margin: auto;
    display: flex;
    align-items: flex-end;
    justify-content: center;
}

.bar {
    background: gray;
    width: 5rem;
    margin: 2px;
}
```

I'm using a CSS flexbox for the bar container for two reasons. Firstly It will allow me centrally align the bars. Secondly, it will compress the elements inside of it so that they all fit. This means that all the bars will dynamically adjust their widths based on how much space is available.

The app with the bars now looks like this:



The next thing to do was to allow the number of bars to be edited by the range input created earlier. I moved the code for the `Controls` component into its own file named 'controls.tsx`. This created the codebase easier to deal with as the code is spread over multiple files.

Inside that file I added this code:

```
import React from "react";
import "./App.css";

interface controlProps {
    makeBars: any;
}

// react component for the control pannel
export default class Controls extends React.Component<controlProps> {
```

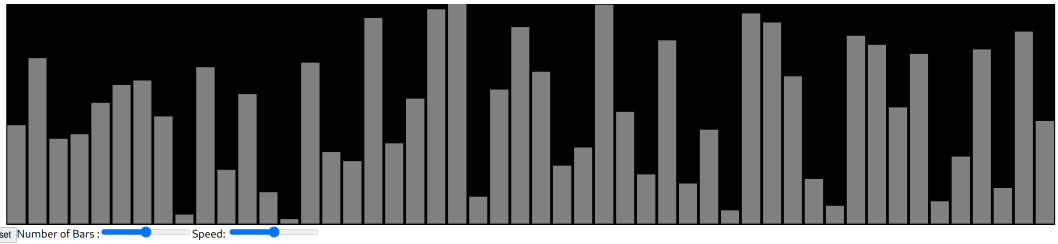
```
    barSelect: React.RefObject<HTMLInputElement>;
    constructor(props: any) {
        super(props);
        this.barSelect = React.createRef(); // creates a ref which will be
assigned to the bar select element
    }

    // calls the makeBars method from the app class and passes in the value of
the length range
    makeBars = () => {
        this.props.makeBars(this.barSelect.current?.value);
    };

    public render() {
        return (
            <div className="controls">
                <select className="algorithmSelect">
                    <option value="">Select Algoritm</option>
                </select>
                <button className="startstop">Start</button>
                <button className="reset">Reset</button>
                Number of Bars :
                <input
                    type="range"
                    ref={this.barSelect} // linking the barSelect ref to the
element
                    onChange={() => this.makeBars()} // call the `makeBars`
method whenever the value of the range is changed
                />
                Speed: <input type="range" />
            </div>
        );
    }
}
```

Whenever the value of the range representing the number of bars changes, the method `MakeBars` is called, and the bars are remade.

At the moment, the bars are always created in order, which makes sorting them pretty boring as they are already in order. Therefore inside the `MakeBars` method, I added some code to shuffle the array.

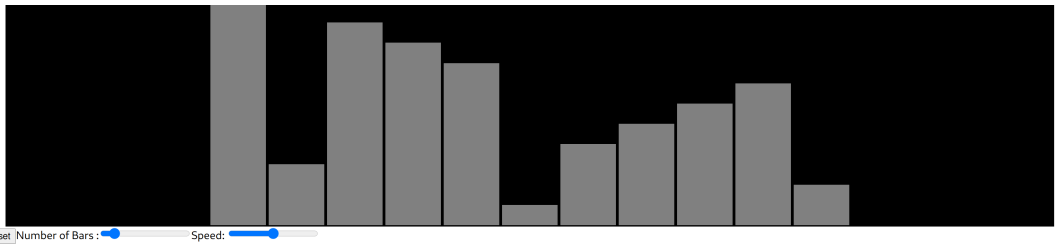
```
// shuffles array
for (let i = b.length - 1; i > 0; i--) {
    let j = Math.floor(Math.random() * (i + 1));
    [b[i], b[j]] = [b[j], b[i]];
}
```

Now, the bars look like this:

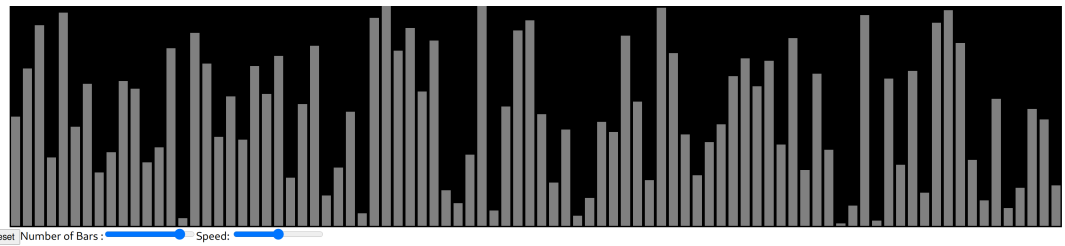


Select Algoritm Start Reset Number of Bars : Speed:
Time:
Comparisons:
Swaps:

And the range slider can be adjusted to increase or decrease the number of bars, which can be seen here:



Select Algoritm Start Reset Number of Bars : Speed:
Time:
Comparisons:
Swaps:



Select Algoritm Start Reset Number of Bars : Speed:
Time:
Comparisons:
Swaps: