**VIVEKANAND EDUCATION SOCIETY’S**

**INSTITUTE OF TECHNOLOGY**

**Hashu Advani Memorial Complex, Collector’s Colony, R. C. Marg, Chembur, Mumbai – 400074.**



**A PROJECT REPORT**

**SORTING ALGO VISUAILIZATION**

**BY**

**24 - NARENDER KESWANI**

**10 – PRATHAMESH BHOSALE**

**62 – CHINMAY VYAPARI**

**Year 2021-22**

**Under the Guidance of**

**MRS. VAISHALI GATTY**

**CERTIFICATE**

This is to certify that the project entitled **SORTING ALGO VISUAILIZATION** of **NARENDER KESWANI [24], PRATHAMESH BHOSALE[10] & CHINMAY VYAPARI[62]** is submitted to satisfactorily complete a course of **MCAL11 - DATA STRUCTURES LAB WITH C AND / C++** under my supervision in the academic year **2021-2022**.

**Vaishali Gatty (Name and sign)**

**Supervisor/Guide Co-Supervisor/Guide**

**Dr. ShivKumar Goel Dr. Mrs. J.M. Nair**

**Head of Department Principal**

**ACKNOWLEDGEMENT**

With immense please we are presenting “**SORTING ALGO VISUAILIZATION**” Project report as part of the curriculum of ‘**Master of Computer Application**’. We wish to thank all the people who gave us unending support.

We, **NARENDER KESWANI [24], PRATHAMESH BHOSALE[10] & CHINMAY VYAPARI[62]**  express our profound thanks to **Mrs. Vaishali Gatty** and all those who have indirectly guided and helped us in preparation of this project.

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr No.** | **Content** | **Page No.** |
| **1** | **Introduction** | **2** |
| **2** | **Objectives & Purpose** | **4** |
| **3** | **Specifications** | **5** |
| **4** | **System Design** | **6** |
| **5** | **Code & Screen Shots** | **7** |
| **6** | **Conclusion** | **90** |
| **7** | **References** | **91** |

**Check Live Project at:** [**https://sorting.narenderkeswani.com/**](https://sorting.narenderkeswani.com/)

* 1. **INTRODUCTION:**

Algorithms and data structures as an essential part of knowledge in a framework of computer science1 have their stable position in computer science curriculam, since every computer scientist and every professional programmer should have the basic knowledge from the area. Our scope here is the higher education in the field of computer science. So within the paper, we discuss the extension of standard methods of teaching algorithms, using the whiteboard or slides, with the algorithm visualizations. According to they can be used to attract students’ attention during the lecture, explain concepts in visual terms, encourage a practical learning process, and facilitate better communication between students and instructors.

DS Algorithm visualization illustrates how data structure algorithms work in a graphical way. It mainly aims to simplify and deepen the understanding of algorithms operation. Within the paper we discuss the possibility of enriching the standard methods of teaching algorithms, with the algorithm visualizations. As a step in this direction, we introduce the DS Algorithm visualizer platform, present our practical experiences and describe possible future directions, based on our experiences and exploration performed by means of a simple questionnaire

How do you get something done? You don't have to be extremely complex in solving the problem, for example, if your car's headlight is broken (although nowadays, manufacturers are trying the patience of the community with their increasingly abstract, space-age designs). The main issue is figuring out the best way to go about it. To locate step-by-step directions in your car's handbook, you conduct research, or do you use instinct to find someone who knows how to do it? In short, my instinct tells me that I am a visual learner and hence more suited to acquire topics by watching them than by reading about them. In this case, I found that seeing the data move to its rightful spot as the result of an algorithm is MUCH easier to follow than looking at the source code and trying to figure out where the data was supposed to go. Our project was born out of our curiosity about sorting algorithms, Selection Sort, Bubble Sort, Insertion Sort,Heap Sort and Merge Sort are the four sorting algorithms. Let's imagine that you have printed each person's age on a separate index card. Bring the youngest card to the front and then sort the cards by age. To discover the next smallest item, identify the age that has already been ordered and position it behind the already ordered age. Index cards full of ages will be at the end of the pile. Selection Sort works in the same way as this. In this case, to sort a set of data, you select the smallest first, and then the next smallest, and so on until you've sorted all of the data. This technique is quite simple to explain to someone in conversation, but more advanced sorting algorithms, such as Quick Sort, which requires the data to be moved around a pivot point, are not easy to grasp using text alone. I wanted the animation to appeal to a wide spectrum of individuals utilizing various technology media, and so I had it made in a web-based format. Instead of requiring the user to install extra software or attempt to organize setups to use the tool, this helps to remove this source of anxiety. It uses HTML5 (Hypertext Markup Text Language) JavaScript, , CSS and React Framework for the website's layout (Cascading Style Sheets).

* 1. **OBJECTIVES & PURPOSE:**

**Objectives:**

The Objectives of Our Website are:

* Provide simple information and services for all faculty and students.
* Enhance the quality of learning of various Data Structures.
* Meet the learning style or understand the needs of students.
* Improve the efficiency and effectiveness.
* Improve user-accessibility and time flexibility to engage learners in the learning process.
* Expand and enhance technology support tools to meet the current needs and expectations.

**Purpose:**

* Enhance the quality of learning and teaching.
* Meet the learning style or needs of students.
* Improve the efficiency and effectiveness.
* Improve user-accessibility and time flexibility to engage learners in the learning process.
  1. **SPECIFICATIONS:**

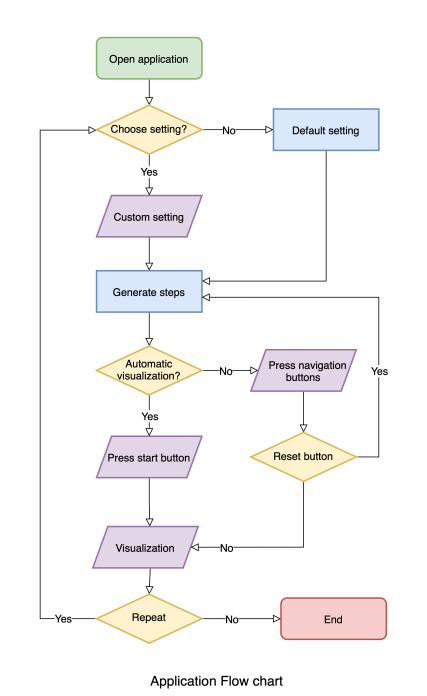
**Software Requirement**

* IDE(VS Code)
* Any Operating System
* Web Browser
* React Js
* NPM Package Manager

**Hardware Requirement**

* 4 Gb Ram
* 50 Gb Hard Disk
* i3 Intel Processor
  1. **SYSTEM DESIGN:**

**Flow Chart:**



* 1. **CODE & SCREEN SHOTS:**

**1) Bubble Sort**

**BubbleSort.js:**

export async function\* BubbleSort(array, swap, highlight, marksort) {

for (let i = 0; i < array.length; i++) {

for (var j = 0; j < array.length - i - 1; j++) {

yield await highlight([j, j + 1]);

if (array[j] > array[j + 1]) {

yield await swap(j, j + 1);

}

}

marksort(j);

yield;

}

}

**2) Selection Sort**

**SelectionSort.js**

export async function\* HeapSort(array, swap, highlight, markSort) {

let arrLength = array.length;

for (let i = Math.floor(arrLength / 2) - 1; i >= 0; i--) {

yield\* await maxHeap(i);

}

for (let i = array.length - 1; i > 0; i--) {

arrLength--;

markSort(arrLength);

yield await swap(0, i);

yield\* await maxHeap(0);

}

markSort(0);

async function\* maxHeap(i) {

const left = 2 \* i + 1;

const right = 2 \* i + 2;

let max = i;

const highlightArray = [];

if(left < arrLength)

highlightArray.push(left);

if(right < arrLength)

highlightArray.push(right);

yield await highlight(highlightArray, i);

if(left < arrLength){

if (array[left] > array[max]) {

max = left;

}

}

if(right < arrLength){

if (array[right] > array[max]) {

max = right;

}

}

if (max !== i) {

yield await swap(i, max);

yield\* await maxHeap(max);

}

}

}

**3) Insertion Sort**

**InsertionSort.js**

export async function\* InsertionSort(array, swap, highlight, marksort) {

for (let i = 0; i < array.length; i++) {

let keyIndex = i;

for (var j = i - 1; j >= 0; j--) {

yield await highlight([keyIndex, j]);

if (array[j] > array[keyIndex]) {

yield await swap(j, keyIndex);

keyIndex = j;

} else {

yield;

break;

}

}

marksort(i);

yield;

}

}

**4) Heap Sort**

**HeapSort.js**

export async function\* QuickSort(

array,

swap,

highlight,

markSort,

low = 0,

high = array.length - 1

) {

if (low <= high) {

let pivot = yield\* await partition(array, low, high);

yield\* await QuickSort(array, swap, highlight, markSort, low, pivot - 1);

yield\* await QuickSort(array, swap, highlight, markSort, pivot + 1, high);

}

async function\* partition(array, low, high) {

let pivot = low;

let i = low;

let j = high + 1;

while (i < j) {

while (--j > low) {

yield await highlight([i, j], pivot);

if (array[j] < array[pivot]) {

break;

}

}

while (i <= high && i < j) {

yield await highlight([i], pivot);

if (array[++i] > array[pivot]) {

break;

}

}

if (i < j) {

yield await swap(i, j);

}

}

if (pivot !== j) {

yield await swap(pivot, j);

}

markSort(j);

yield;

return j;

}

}

**5) Quick Sort**

**QuickSort.js**

export async function\* SelectionSort(array, swap, highlight, marksort) {

for (let i = 0; i < array.length; i++) {

let maxIndex = 0;

for (var j = 0; j < array.length - i; j++) {

yield await highlight([maxIndex, j]);

if (array[maxIndex] < array[j]) {

maxIndex = j;

}

}

j = j - 1;

if (maxIndex !== j && array[maxIndex] !== array[j]) {

yield await swap(maxIndex, j);

}

marksort(j);

yield;

}

}

**Web Application Files:**

**App.js:**

import React from "react";

import styled from "styled-components";

import { MainApp } from "./components/MainApp";

const Container = styled.div`

margin: 0 10px;

min-height: calc(100vh - 50px);

position: relative;

margin-bottom: 50px;

backgroundColor: black;

`;

export default function App() {

return (

<Container>

<MainApp />

</Container>

);

}

**ArrayContainer.jsx:**

import React from "react";

import styled from "styled-components";

import { useControls } from "../../common/store";

import {

ArrayHolder,

ArrayItem,

sourceAnimation,

destinationAnimation,

} from "../../common/styles";

export const comparisionColor = "pink";

export const swapColor = "yellow";

export const sortedColor = "springgreen";

export const pivotColor = "sandybrown";

let swapTime = useControls.getState().swapTime;

useControls.subscribe(

(time) => (swapTime = time),

(state) => state.swapTime

);

const Source = styled(ArrayItem)`

animation: ${(props) => destinationAnimation(props.distance, swapColor)}

${() => swapTime / 1000}s forwards;

`;

const Destination = styled(ArrayItem)`

animation: ${(props) => sourceAnimation(props.distance, swapColor)}

${() => swapTime / 1000}s forwards;

`;

export function ArrayContainer({

array,

source,

destination,

pivot = -1,

highlightIndices,

sortedIndices,

}) {

function getBackgroundColor(i) {

if (i === pivot) {

return pivotColor;

}

if (highlightIndices.includes(i)) {

return comparisionColor;

}

if (sortedIndices.includes(i)) {

return sortedColor;

}

return "";

}

return (

<ArrayHolder>

{array.map((value, i) => {

if (i === source) {

return (

<Source

key={i + ":" + source + ":" + destination + ":" + value}

distance={destination - source}

style={{

order: destination,

backgroundColor: getBackgroundColor(i),

}}

>

{value}

</Source>

);

}

if (i === destination) {

return (

<Destination

key={i + ":" + destination + ":" + source + ":" + value}

distance={destination - source}

style={{

order: source,

backgroundColor: getBackgroundColor(i),

}}

>

{value}

</Destination>

);

}

return (

<ArrayItem

key={i + ":" + destination + ":" + source + ":" + value}

style={{

order: i,

backgroundColor: getBackgroundColor(i),

}}

>

{value}

</ArrayItem>

);

})}

</ArrayHolder>

);

}

**SortManager.jsx:**

import React, { useEffect, useRef, useState } from "react";

import styled from "styled-components";

import { ArrayContainer } from "./ArrayContainer";

import { Timer } from "./Timer";

import Card from "@material-ui/core/Card";

import shallow from "zustand/shallow";

import { useControls, useData } from "../../common/store";

let compareTime = useControls.getState().compareTime;

let swapTime = useControls.getState().swapTime;

export const comparisionColor = "pink";

export const swapColor = "yellow";

export const sortedColor = "springgreen";

export const pivotColor = "sandybrown";

function delay(time) {

return new Promise((resolve) => setTimeout(resolve, time));

}

useControls.subscribe(

([cTime, sTime]) => {

compareTime = cTime;

swapTime = sTime;

},

(state) => [state.compareTime, state.swapTime],

shallow

);

const Container = styled(Card)`

padding: 10px;

border: 1px solid rgba(0, 0, 0, 0.15);

`;

const AlgoHeaderBar = styled.div`

display: flex;

justify-content: space-between;

align-items: center;

column-gap: 20px;

`;

const TimerDiv = styled.div`

display: flex;

column-gap: 5px;

min-width: 8rem;

justify-content: flex-end;

`;

const InfoFlex = styled.div`

display: flex;

justify-content: space-between;

`;

export const SortManager = React.memo(function ({

array,

sortFunction,

sortingAlgorithmName,

}) {

const [swapIndices, setSwapIndices] = useState([-1, -1]);

const [hightlightedIndices, setHightlightedIndices] = useState([-1, -1]);

const algoArray = useRef([]);

const sortedIndices = useRef([]);

const pivot = useRef(-1);

const swapCount = useRef(0);

const comparisionCount = useRef(0);

const isAlgoExecutionOver = useRef(false);

const isComponentUnMounted = useRef(false);

const markSortngDone = useControls((state) => state.markSortngDone);

const progress = useRef("");

const sortProgressIterator = useRef(null);

async function reset() {

algoArray.current = [...useData.getState().sortingArray];

sortedIndices.current = [];

pivot.current = -1;

swapCount.current = 0;

comparisionCount.current = 0;

isAlgoExecutionOver.current = false;

setSwapIndices([-1, -1]);

setHightlightedIndices([-1, -1]);

sortProgressIterator.current = await sortFunction(

algoArray.current,

swap,

highlight,

markSort

);

}

useEffect(() => {

progress.current = useControls.getState().progress;

useControls.subscribe(

(value) => {

progress.current = value;

if (progress.current === "start")

{

runAlgo();

}

if (progress.current === "reset")

{

reset();

}

},

(state) => state.progress,

);

return () => {

isComponentUnMounted.current = true;

};

}, []);

useEffect(() => {

reset();

}, [array]);

async function runAlgo() {

let completion = { done: false };

while (

!completion?.done &&

progress.current === "start" &&

!isComponentUnMounted.current

) {

completion = await sortProgressIterator.current?.next();

}

if (isComponentUnMounted.current) {

return;

}

if (!isAlgoExecutionOver.current && completion?.done) {

isAlgoExecutionOver.current = true;

pivot.current = -1;

setSwapIndices([-1, -1]);

setHightlightedIndices([-1, -1]);

markSortngDone();

}

}

async function swap(i, j) {

let tmp = algoArray.current[i];

algoArray.current[i] = algoArray.current[j];

algoArray.current[j] = tmp;

setSwapIndices([i, j]);

pivot.current = -1;

swapCount.current += 1;

await delay(swapTime);

}

async function highlight(indices, p) {

setSwapIndices([-1, -1]);

comparisionCount.current += 1;

pivot.current = p;

setHightlightedIndices(indices);

await delay(compareTime);

}

function markSort(...indices) {

sortedIndices.current.push(...indices);

}

const arrayContainer = (

<ArrayContainer

array={algoArray.current}

source={swapIndices[0]}

destination={swapIndices[1]}

pivot={pivot.current}

highlightIndices={hightlightedIndices}

sortedIndices={sortedIndices.current}

/>

);

return (

<Container>

<AlgoHeaderBar>

<strong>{sortingAlgorithmName}</strong>

<TimerDiv>

<span>Time:</span>

<strong>

<Timer isAlgoExecutionOver={isAlgoExecutionOver.current} />

</strong>

</TimerDiv>

</AlgoHeaderBar>

{(sortingAlgorithmName = arrayContainer)}

<InfoFlex>

<div>

Number of Swaps: <strong>{swapCount.current}</strong>

</div>

<div>

Number of Comparisions: <strong>{comparisionCount.current}</strong>

</div>

</InfoFlex>

</Container>

);

});

**Timer.jsx:**

import { useEffect, useState } from "react";

import { useControls } from "../../common/store";

export function Timer({ isAlgoExecutionOver }) {

const [minutes, setMinutes] = useState(0);

const [seconds, setSeconds] = useState(0);

const [milliSeconds, setMilliSeconds] = useState(0);

const progress = useControls((state) => state.progress);

function resetTimer() {

setMilliSeconds(0);

setSeconds(0);

setMinutes(0);

}

useEffect(() => {

if (isAlgoExecutionOver) return;

if (progress === "start")

var intervalId = setInterval(() => setMilliSeconds((ml) => ml + 1), 100);

else if (progress === "reset") resetTimer();

return () => clearInterval(intervalId);

}, [progress, isAlgoExecutionOver]);

useEffect(() => {

if (milliSeconds === 10) {

setSeconds((seconds) => seconds + 1);

setMilliSeconds(0);

}

}, [milliSeconds]);

useEffect(() => {

if (seconds === 60) {

setMinutes((minutes) => minutes + 1);

setSeconds(0);

}

}, [seconds]);

return `${minutes.toString().padStart(2, 0)}:${seconds

.toString()

.padStart(2, 0)}:${milliSeconds} s`;

}

**Controller.jsx:**

import React, { useState } from "react";

import styled from "styled-components";

import { VscDebugStart } from "react-icons/vsc";

import { VscDebugRestart } from "react-icons/vsc";

import { ImPause } from "react-icons/im";

import TextField from "@material-ui/core/TextField";

import shallow from "zustand/shallow";

import Card from "@material-ui/core/Card";

import { useControls, useData } from "../common/store";

const Container = styled(Card)`

padding: 10px;

border: 1px solid rgba(0, 0, 0, 0.15);

`;

const TabArrayBar = styled.div`

font-size: 2rem;

display: flex;

align-items: center;

margin: 15px 0;

flex-wrap: wrap;

`;

const ArrayBar = styled.div`

display: flex;

align-items: center;

flex-basis: 40%;

min-width: 500px;

`;

const ExecutionBar = styled.div`

display: flex;

align-items: center;

flex-basis: 15%;

`;

function convertInputToArrayString(string) {

string = string.replaceAll(/\s/g, "");

string = string.replaceAll(/\d{4}/g, "");

string = string.replaceAll(/\s\s/g, " ");

string = string.replaceAll(/\s,/g, ",");

string = string.replaceAll(/,,/g, ",");

string = string.replaceAll(/[^0-9,\s]/g, "");

return string;

}

function convertArrayStringToArray(string) {

return string

.split(",")

.filter((v) => v !== "")

.map((v) => +v);

}

function delay(time) {

return new Promise((resolve) => setTimeout(resolve, time));

}

export function Controller() {

const [isPausing, setIsPausing] = useState(false);

const [progress] = useControls(

(state) => [state.progress],

shallow

);

const [sortingArray, setSortingArray] = useData(

(state) => [state.sortingArray, state.setSortingArray],

shallow

);

const [startSorting, pauseSorting, resetSorting, setSpeed] = useControls(

(state) => [

state.startSorting,

state.pauseSorting,

state.resetSorting,

state.setSpeed,

],

shallow

);

setSpeed(1);

const [arrayInput, setArrayInput] = useState(sortingArray);

const startElement = <VscDebugStart onClick={startSorting} />;

const pauseElement = <ImPause onClick={pauseAndDelaySorting} />;

const resetElement = <VscDebugRestart onClick={resetSorting} />;

const disabledPauseElement = <ImPause style={{ color: "#e5e5e5" }} />;

async function pauseAndDelaySorting(){

pauseSorting();

setIsPausing(true);

await delay(useControls.getState().swapTime);

setIsPausing(false);

}

function arrayDataChangeHandler(value) {

const arrayString = convertInputToArrayString(value);

setArrayInput(arrayString);

const array = convertArrayStringToArray(arrayString);

setSortingArray(array);

resetSorting();

}

function getProgressButton() {

if(isPausing)

return disabledPauseElement;

switch (progress) {

case "reset":

return startElement;

case "start":

return pauseElement;

case "pause":

return startElement;

case "done":

return disabledPauseElement;

default:

return "Not Found";

}

}

return (

<Container>

<TabArrayBar>

<ArrayBar>

<TextField

id="outlined-basic"

label="Input"

variant="outlined"

onChange={(event) => arrayDataChangeHandler(event.target.value)}

value={arrayInput}

size="small"

style={{ flexGrow: 1, margin: "0 10px" }}

/>

</ArrayBar>

<ExecutionBar>

<div

style={{ display: "flex", marginLeft: "10px", columnGap: "5px" }}

>

{getProgressButton()}

{resetElement}

</div>

</ExecutionBar>

</TabArrayBar>

</Container>

);

}

**MainApp.jsx:**

import React from "react";

import { makeStyles } from "@material-ui/core/styles";

import { Tab, Tabs, TabList, TabPanel } from "react-tabs";

import "react-tabs/style/react-tabs.css";

import { useControls, useData } from "../common/store";

import shallow from "zustand/shallow";

import styled from "styled-components";

import Card from "@material-ui/core/Card";

import { BubbleSort } from "../sortFunctions/BubbleSort";

import { SelectionSort } from "../sortFunctions/SelectionSort";

import { InsertionSort } from "../sortFunctions/InsertionSort";

import { QuickSort } from "../sortFunctions/QuickSort";

import { HeapSort } from "../sortFunctions/HeapSort.js";

import { BubbleSortTheory } from "./BubbleSortTheory";

import { Controller } from "./Controller";

import { SortManager } from "./visualizer/SortManager";

import { SelectionSortTheory } from "./SelectionSortTheory";

import { InsertionSortTheory } from "./InsertionSortTheory";

import { HeapSortTheory } from "./HeapSortTheory";

import { QuickSortTheory } from "./QuickSortTheory";

const useStyles = makeStyles((theme) => ({

root: {

flexGrow: 1,

width: "100%",

backgroundColor: theme.palette.background.paper,

},

}));

const FooterDiv = styled(Card)`

display: flex;

padding: 10px;

border: 1px solid rgba(0, 0, 0, 0.15);

justify-content: center;

align-items: center;

position: absolute;

width: 100%;

`;

export function MainApp() {

const classes = useStyles();

const [sortingArray] = useData(

(state) => [state.sortingArray],

shallow

);

const [resetSorting] = useControls(

(state) => [

state.resetSorting,

],

shallow

);

return (

<div className={classes.root}>

<div

style={{

display: "flex",

justifyContent: "space-between",

alignItems: "center",

}}

></div>

<Tabs>

<TabList>

<Tab onClick={resetSorting}>BubbleSort</Tab>

<Tab onClick={resetSorting}>SelectionSort</Tab>

<Tab onClick={resetSorting}>InsertionSort</Tab>

<Tab onClick={resetSorting}>HeapSort</Tab>

<Tab onClick={resetSorting}>QuickSort</Tab>

</TabList>

<Controller />

<br />

<TabPanel>

<SortManager

array={sortingArray}

sortFunction={BubbleSort}

sortingAlgorithmName={"BubbleSort"}

key={"BubbleSort"}

/>

<br />

<BubbleSortTheory />

</TabPanel>

<TabPanel>

<SortManager

array={sortingArray}

sortFunction={SelectionSort}

sortingAlgorithmName={"SelectionSort"}

key={"SelectionSort"}

/>

<br />

<SelectionSortTheory />

</TabPanel>

<TabPanel>

<SortManager

array={sortingArray}

sortFunction={InsertionSort}

sortingAlgorithmName={"InsertionSort"}

key={"InsertionSort"}

/>

<br />

<InsertionSortTheory />

</TabPanel>

<TabPanel>

<SortManager

array={sortingArray}

sortFunction={HeapSort}

sortingAlgorithmName={"HeapSort"}

key={"HeapSort"}

/>

<br />

<HeapSortTheory />

</TabPanel>

<TabPanel>

<SortManager

array={sortingArray}

sortFunction={QuickSort}

sortingAlgorithmName={"QuickSort"}

key={"QuickSort"}

/>

<br />

<QuickSortTheory />

</TabPanel>

</Tabs>

<br />

<FooterDiv>

<h4>

<span>

Made with ♥ by &nbsp;

<a href="https://www.narenderkeswani.com/">

Narender Keswani, Prathamesh Bhosale & Chinmay Vyapari

</a>

</span>

</h4>

</FooterDiv>

</div>

);

}

**BubbleSortTheory.jsx:**

import React from "react";

import CodeEditor from "@uiw/react-textarea-code-editor";

import { Tab, Tabs, TabList, TabPanel } from "react-tabs";

import "react-tabs/style/react-tabs.css";

import Card from "@material-ui/core/Card";

import styled from "styled-components";

const Container = styled(Card)`

padding: 10px;

border: 1px solid rgba(0, 0, 0, 0.15);

`;

export function BubbleSortTheory() {

const [cppcode, setcppCode] = React.useState(

`// C++ program for implementation of Bubble sort

#include <bits/stdc++.h>

using namespace std;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void bubbleSort(int arr[], int n)

{

int i, j;

for (i = 0; i < n-1; i++)

// Last i elements are already in place

for (j = 0; j < n-i-1; j++)

if (arr[j] > arr[j+1])

swap(&arr[j], &arr[j+1]);

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i = 0; i < size; i++)

cout << arr[i] << " ";

cout << endl;

}

// Driver code

int main()

{

int arr[] = {64, 34, 25, 12, 22, 11, 90};

int n = sizeof(arr)/sizeof(arr[0]);

bubbleSort(arr, n);

cout<<"Sorted array: \n";

printArray(arr, n);

return 0;

}

`

);

const [ccode, setcCode] = React.useState(

`// C program for implementation of Bubble sort

#include <stdio.h>

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void bubbleSort(int arr[], int n)

{

int i, j;

for (i = 0; i < n-1; i++)

// Last i elements are already in place

for (j = 0; j < n-i-1; j++)

if (arr[j] > arr[j+1])

swap(&arr[j], &arr[j+1]);

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

printf("\n");

}

// Driver program to test above functions

int main()

{

int arr[] = {64, 34, 25, 12, 22, 11, 90};

int n = sizeof(arr)/sizeof(arr[0]);

bubbleSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

`

);

const [javacode, setjavaCode] = React.useState(

`

// Java program for implementation of Bubble Sort

class BubbleSort

{

void bubbleSort(int arr[])

{

int n = arr.length;

for (int i = 0; i < n-1; i++)

for (int j = 0; j < n-i-1; j++)

if (arr[j] > arr[j+1])

{

// swap arr[j+1] and arr[j]

int temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

/\* Prints the array \*/

void printArray(int arr[])

{

int n = arr.length;

for (int i=0; i<n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver method to test above

public static void main(String args[])

{

BubbleSort ob = new BubbleSort();

int arr[] = {64, 34, 25, 12, 22, 11, 90};

ob.bubbleSort(arr);

System.out.println("Sorted array");

ob.printArray(arr);

}

}

`

);

const [pythoncode, setpythonCode] = React.useState(

`# Python program for implementation of Bubble Sort

def bubbleSort(arr):

n = len(arr)

# Traverse through all array elements

for i in range(n):

# Last i elements are already in place

for j in range(0, n-i-1):

# traverse the array from 0 to n-i-1

# Swap if the element found is greater

# than the next element

if arr[j] > arr[j+1] :

arr[j], arr[j+1] = arr[j+1], arr[j]

# Driver code to test above

arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array is:")

for i in range(len(arr)):

print ("%d" %arr[i]),

`

);

const [cscode, setcsCode] = React.useState(

`

// C# program for implementation

// of Bubble Sort

using System;

class GFG

{

static void bubbleSort(int []arr)

{

int n = arr.Length;

for (int i = 0; i < n - 1; i++)

for (int j = 0; j < n - i - 1; j++)

if (arr[j] > arr[j + 1])

{

// swap temp and arr[i]

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

/\* Prints the array \*/

static void printArray(int []arr)

{

int n = arr.Length;

for (int i = 0; i < n; ++i)

Console.Write(arr[i] + " ");

Console.WriteLine();

}

// Driver method

public static void Main()

{

int []arr = {64, 34, 25, 12, 22, 11, 90};

bubbleSort(arr);

Console.WriteLine("Sorted array");

printArray(arr);

}

}

`

);

const [phpcode, setphpCode] = React.useState(

`

<?php

// PHP program for implementation

// of Bubble Sort

function bubbleSort(&$arr)

{

$n = sizeof($arr);

// Traverse through all array elements

for($i = 0; $i < $n; $i++)

{

// Last i elements are already in place

for ($j = 0; $j < $n - $i - 1; $j++)

{

// traverse the array from 0 to n-i-1

// Swap if the element found is greater

// than the next element

if ($arr[$j] > $arr[$j+1])

{

$t = $arr[$j];

$arr[$j] = $arr[$j+1];

$arr[$j+1] = $t;

}

}

}

}

// Driver code to test above

$arr = array(64, 34, 25, 12, 22, 11, 90);

$len = sizeof($arr);

bubbleSort($arr);

echo "Sorted array : \n";

for ($i = 0; $i < $len; $i++)

echo $arr[$i]." ";

// This code is contributed by ChitraNayal.

?>

`

);

return (

<Container>

<h4>

Bubble sort is a sorting algorithm that compares two adjacent elements

and swaps them until they are not in the intended order. Just like the

movement of air bubbles in the water that rise up to the surface, each

element of the array move to the end in each iteration. Therefore, it is

called a bubble sort.

</h4>

<Tabs>

<TabList>

<Tab>C++</Tab>

<Tab>C</Tab>

<Tab>Java</Tab>

<Tab>Python</Tab>

<Tab>C#</Tab>

<Tab>PHP</Tab>

</TabList>

<TabPanel>

<CodeEditor

value={cppcode}

language="cpp"

onChange={(evn) => setcppCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={ccode}

language="c"

onChange={(evn) => setcCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={javacode}

language="java"

onChange={(evn) => setjavaCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={pythoncode}

language="py"

onChange={(evn) => setpythonCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={cscode}

language="csharp"

onChange={(evn) => setcsCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={phpcode}

language="php"

onChange={(evn) => setphpCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

</Tabs>

</Container>

);

}

**HeapSortTheory.jsx:**

import React from "react";

import CodeEditor from "@uiw/react-textarea-code-editor";

import { Tab, Tabs, TabList, TabPanel } from "react-tabs";

import "react-tabs/style/react-tabs.css";

import Card from "@material-ui/core/Card";

import styled from "styled-components";

const Container = styled(Card)`

padding: 10px;

border: 1px solid rgba(0, 0, 0, 0.15);

`;

export function HeapSortTheory() {

const [cppcode, setcppCode] = React.useState(

`

// C++ program for implementation of Heap Sort

#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

// If left child is larger than root

if (l < n && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest so far

if (r < n && arr[r] > arr[largest])

largest = r;

// If largest is not root

if (largest != i) {

swap(arr[i], arr[largest]);

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

// main function to do heap sort

void heapSort(int arr[], int n)

{

// Build heap (rearrange array)

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// One by one extract an element from heap

for (int i = n - 1; i > 0; i--) {

// Move current root to end

swap(arr[0], arr[i]);

// call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

/\* A utility function to print array of size n \*/

void printArray(int arr[], int n)

{

for (int i = 0; i < n; ++i)

cout << arr[i] << " ";

cout << "\n";

}

// Driver code

int main()

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

int n = sizeof(arr) / sizeof(arr[0]);

heapSort(arr, n);

cout << "Sorted array is \n";

printArray(arr, n);

}

`

);

const [javacode, setjavaCode] = React.useState(

`

// Java program for implementation of Heap Sort

public class HeapSort {

public void sort(int arr[])

{

int n = arr.length;

// Build heap (rearrange array)

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// One by one extract an element from heap

for (int i = n - 1; i > 0; i--) {

// Move current root to end

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

// call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

// If left child is larger than root

if (l < n && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest so far

if (r < n && arr[r] > arr[largest])

largest = r;

// If largest is not root

if (largest != i) {

int swap = arr[i];

arr[i] = arr[largest];

arr[largest] = swap;

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

/\* A utility function to print array of size n \*/

static void printArray(int arr[])

{

int n = arr.length;

for (int i = 0; i < n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver code

public static void main(String args[])

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

int n = arr.length;

HeapSort ob = new HeapSort();

ob.sort(arr);

System.out.println("Sorted array is");

printArray(arr);

}

}

`

);

const [pythoncode, setpythonCode] = React.useState(

`

# Python program for implementation of heap Sort

# To heapify subtree rooted at index i.

# n is size of heap

def heapify(arr, n, i):

largest = i # Initialize largest as root

l = 2 \* i + 1 # left = 2\*i + 1

r = 2 \* i + 2 # right = 2\*i + 2

# See if left child of root exists and is

# greater than root

if l < n and arr[largest] < arr[l]:

largest = l

# See if right child of root exists and is

# greater than root

if r < n and arr[largest] < arr[r]:

largest = r

# Change root, if needed

if largest != i:

arr[i], arr[largest] = arr[largest], arr[i] # swap

# Heapify the root.

heapify(arr, n, largest)

# The main function to sort an array of given size

def heapSort(arr):

n = len(arr)

# Build a maxheap.

for i in range(n//2 - 1, -1, -1):

heapify(arr, n, i)

# One by one extract elements

for i in range(n-1, 0, -1):

arr[i], arr[0] = arr[0], arr[i] # swap

heapify(arr, i, 0)

# Driver code

arr = [12, 11, 13, 5, 6, 7]

heapSort(arr)

n = len(arr)

print("Sorted array is")

for i in range(n):

print("%d" % arr[i],end=" ")

`

);

const [cscode, setcsCode] = React.useState(

`

// C# program for implementation of Heap Sort

using System;

public class HeapSort {

public void sort(int[] arr)

{

int n = arr.Length;

// Build heap (rearrange array)

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// One by one extract an element from heap

for (int i = n - 1; i > 0; i--) {

// Move current root to end

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

// call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

void heapify(int[] arr, int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

// If left child is larger than root

if (l < n && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest so far

if (r < n && arr[r] > arr[largest])

largest = r;

// If largest is not root

if (largest != i) {

int swap = arr[i];

arr[i] = arr[largest];

arr[largest] = swap;

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

/\* A utility function to print array of size n \*/

static void printArray(int[] arr)

{

int n = arr.Length;

for (int i = 0; i < n; ++i)

Console.Write(arr[i] + " ");

Console.Read();

}

// Driver code

public static void Main()

{

int[] arr = { 12, 11, 13, 5, 6, 7 };

int n = arr.Length;

HeapSort ob = new HeapSort();

ob.sort(arr);

Console.WriteLine("Sorted array is");

printArray(arr);

}

}

`

);

const [phpcode, setphpCode] = React.useState(

`

<?php

// Php program for implementation of Heap Sort

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

function heapify(&$arr, $n, $i)

{

$largest = $i; // Initialize largest as root

$l = 2\*$i + 1; // left = 2\*i + 1

$r = 2\*$i + 2; // right = 2\*i + 2

// If left child is larger than root

if ($l < $n && $arr[$l] > $arr[$largest])

$largest = $l;

// If right child is larger than largest so far

if ($r < $n && $arr[$r] > $arr[$largest])

$largest = $r;

// If largest is not root

if ($largest != $i)

{

$swap = $arr[$i];

$arr[$i] = $arr[$largest];

$arr[$largest] = $swap;

// Recursively heapify the affected sub-tree

heapify($arr, $n, $largest);

}

}

// main function to do heap sort

function heapSort(&$arr, $n)

{

// Build heap (rearrange array)

for ($i = $n / 2 - 1; $i >= 0; $i--)

heapify($arr, $n, $i);

// One by one extract an element from heap

for ($i = $n-1; $i > 0; $i--)

{

// Move current root to end

$temp = $arr[0];

$arr[0] = $arr[$i];

$arr[$i] = $temp;

// call max heapify on the reduced heap

heapify($arr, $i, 0);

}

}

/\* A utility function to print array of size n \*/

function printArray(&$arr, $n)

{

for ($i = 0; $i < $n; ++$i)

echo ($arr[$i]." ") ;

}

// Driver program

$arr = array(12, 11, 13, 5, 6, 7);

$n = sizeof($arr)/sizeof($arr[0]);

heapSort($arr, $n);

echo 'Sorted array is ' . "\n";

printArray($arr , $n);

?>

`

);

return (

<Container>

<h4>

Heap sort is one of the sorting algorithms used to arrange a list of

elements in order. Heapsort algorithm uses one of the tree concepts

called Heap Tree. In this sorting algorithm, we use Max Heap to arrange

list of elements in Descending order and Min Heap to arrange list

elements in Ascending order.

</h4>

<Tabs>

<TabList>

<Tab>C++</Tab>

<Tab>C</Tab>

<Tab>Java</Tab>

<Tab>Python</Tab>

<Tab>C#</Tab>

<Tab>PHP</Tab>

</TabList>

<TabPanel>

<CodeEditor

value={cppcode}

language="cpp"

onChange={(evn) => setcppCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={javacode}

language="java"

onChange={(evn) => setjavaCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={pythoncode}

language="py"

onChange={(evn) => setpythonCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={cscode}

language="csharp"

onChange={(evn) => setcsCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={phpcode}

language="php"

onChange={(evn) => setphpCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

</Tabs>

</Container>

);

}

**InsertionsSortTheory.jsx:**

import React from "react";

import CodeEditor from "@uiw/react-textarea-code-editor";

import { Tab, Tabs, TabList, TabPanel } from "react-tabs";

import "react-tabs/style/react-tabs.css";

import Card from "@material-ui/core/Card";

import styled from "styled-components";

const Container = styled(Card)`

padding: 10px;

border: 1px solid rgba(0, 0, 0, 0.15);

`;

export function InsertionSortTheory() {

const [cppcode, setcppCode] = React.useState(

`// C++ program for insertion sort

#include <bits/stdc++.h>

using namespace std;

/\* Function to sort an array using insertion sort\*/

void insertionSort(int arr[], int n)

{

int i, key, j;

for (i = 1; i < n; i++)

{

key = arr[i];

j = i - 1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

// A utility function to print an array of size n

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < n; i++)

cout << arr[i] << " ";

cout << endl;

}

/\* Driver code \*/

int main()

{

int arr[] = { 12, 11, 13, 5, 6 };

int n = sizeof(arr) / sizeof(arr[0]);

insertionSort(arr, n);

printArray(arr, n);

return 0;

}

`

);

const [ccode, setcCode] = React.useState(

`

// C program for insertion sort

#include <math.h>

#include <stdio.h>

/\* Function to sort an array using insertion sort\*/

void insertionSort(int arr[], int n)

{

int i, key, j;

for (i = 1; i < n; i++) {

key = arr[i];

j = i - 1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

// A utility function to print an array of size n

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < n; i++)

printf("%d ", arr[i]);

printf("\n");

}

/\* Driver program to test insertion sort \*/

int main()

{

int arr[] = { 12, 11, 13, 5, 6 };

int n = sizeof(arr) / sizeof(arr[0]);

insertionSort(arr, n);

printArray(arr, n);

return 0;

}

`

);

const [javacode, setjavaCode] = React.useState(

`

// Java program for implementation of Insertion Sort

class InsertionSort {

/\*Function to sort array using insertion sort\*/

void sort(int arr[])

{

int n = arr.length;

for (int i = 1; i < n; ++i) {

int key = arr[i];

int j = i - 1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

/\* A utility function to print array of size n\*/

static void printArray(int arr[])

{

int n = arr.length;

for (int i = 0; i < n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver method

public static void main(String args[])

{

int arr[] = { 12, 11, 13, 5, 6 };

InsertionSort ob = new InsertionSort();

ob.sort(arr);

printArray(arr);

}

}

`

);

const [pythoncode, setpythonCode] = React.useState(

`

# Python program for implementation of Insertion Sort

# Function to do insertion sort

def insertionSort(arr):

# Traverse through 1 to len(arr)

for i in range(1, len(arr)):

key = arr[i]

# Move elements of arr[0..i-1], that are

# greater than key, to one position ahead

# of their current position

j = i-1

while j >= 0 and key < arr[j] :

arr[j + 1] = arr[j]

j -= 1

arr[j + 1] = key

# Driver code to test above

arr = [12, 11, 13, 5, 6]

insertionSort(arr)

for i in range(len(arr)):

print ("% d" % arr[i])

`

);

const [cscode, setcsCode] = React.useState(

`

// C# program for implementation of Insertion Sort

using System;

class InsertionSort {

// Function to sort array

// using insertion sort

void sort(int[] arr)

{

int n = arr.Length;

for (int i = 1; i < n; ++i) {

int key = arr[i];

int j = i - 1;

// Move elements of arr[0..i-1],

// that are greater than key,

// to one position ahead of

// their current position

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

// A utility function to print

// array of size n

static void printArray(int[] arr)

{

int n = arr.Length;

for (int i = 0; i < n; ++i)

Console.Write(arr[i] + " ");

Console.Write("\n");

}

// Driver Code

public static void Main()

{

int[] arr = { 12, 11, 13, 5, 6 };

InsertionSort ob = new InsertionSort();

ob.sort(arr);

printArray(arr);

}

}

`

);

const [phpcode, setphpCode] = React.useState(

`

<?php

// PHP program for insertion sort

// Function to sort an array

// using insertion sort

function insertionSort(&$arr, $n)

{

for ($i = 1; $i < $n; $i++)

{

$key = $arr[$i];

$j = $i-1;

// Move elements of arr[0..i-1],

// that are greater than key, to

// one position ahead of their

// current position

while ($j >= 0 && $arr[$j] > $key)

{

$arr[$j + 1] = $arr[$j];

$j = $j - 1;

}

$arr[$j + 1] = $key;

}

}

// A utility function to

// print an array of size n

function printArray(&$arr, $n)

{

for ($i = 0; $i < $n; $i++)

echo $arr[$i]." ";

echo "\n";

}

// Driver Code

$arr = array(12, 11, 13, 5, 6);

$n = sizeof($arr);

insertionSort($arr, $n);

printArray($arr, $n);

?>

`

);

return (

<Container>

<h4>

Insertion sort is a simple sorting algorithm that works similar to the

way you sort playing cards in your hands. The array is virtually split

into a sorted and an unsorted part. Values from the unsorted part are

picked and placed at the correct position in the sorted part.

</h4>

<Tabs>

<TabList>

<Tab>C++</Tab>

<Tab>C</Tab>

<Tab>Java</Tab>

<Tab>Python</Tab>

<Tab>C#</Tab>

<Tab>PHP</Tab>

</TabList>

<TabPanel>

<CodeEditor

value={cppcode}

language="cpp"

onChange={(evn) => setcppCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={ccode}

language="c"

onChange={(evn) => setcCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={javacode}

language="java"

onChange={(evn) => setjavaCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={pythoncode}

language="py"

onChange={(evn) => setpythonCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={cscode}

language="csharp"

onChange={(evn) => setcsCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={phpcode}

language="php"

onChange={(evn) => setphpCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

</Tabs>

</Container>

);

}

**QuickSortTheory.jsx:**

import React from "react";

import CodeEditor from "@uiw/react-textarea-code-editor";

import { Tab, Tabs, TabList, TabPanel } from "react-tabs";

import "react-tabs/style/react-tabs.css";

import Card from "@material-ui/core/Card";

import styled from "styled-components";

const Container = styled(Card)`

padding: 10px;

border: 1px solid rgba(0, 0, 0, 0.15);

`;

export function QuickSortTheory() {

const [cppcode, setcppCode] = React.useState(

`

/\* C++ implementation of QuickSort \*/

#include <bits/stdc++.h>

using namespace std;

// A utility function to swap two elements

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

/\* This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot \*/

int partition (int arr[], int low, int high)

{

int pivot = arr[high]; // pivot

int i = (low - 1); // Index of smaller element and indicates the right position of pivot found so far

for (int j = low; j <= high - 1; j++)

{

// If current element is smaller than the pivot

if (arr[j] < pivot)

{

i++; // increment index of smaller element

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index \*/

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

int pi = partition(arr, low, high);

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i = 0; i < size; i++)

cout << arr[i] << " ";

cout << endl;

}

// Driver Code

int main()

{

int arr[] = {10, 7, 8, 9, 1, 5};

int n = sizeof(arr) / sizeof(arr[0]);

quickSort(arr, 0, n - 1);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

`

);

const [javacode, setjavaCode] = React.useState(

`

// Java implementation of QuickSort

import java.io.\*;

class GFG{

// A utility function to swap two elements

static void swap(int[] arr, int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

/\* This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot \*/

static int partition(int[] arr, int low, int high)

{

// pivot

int pivot = arr[high];

// Index of smaller element and

// indicates the right position

// of pivot found so far

int i = (low - 1);

for(int j = low; j <= high - 1; j++)

{

// If current element is smaller

// than the pivot

if (arr[j] < pivot)

{

// Increment index of

// smaller element

i++;

swap(arr, i, j);

}

}

swap(arr, i + 1, high);

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index

\*/

static void quickSort(int[] arr, int low, int high)

{

if (low < high)

{

// pi is partitioning index, arr[p]

// is now at right place

int pi = partition(arr, low, high);

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Function to print an array

static void printArray(int[] arr, int size)

{

for(int i = 0; i < size; i++)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver Code

public static void main(String[] args)

{

int[] arr = { 10, 7, 8, 9, 1, 5 };

int n = arr.length;

quickSort(arr, 0, n - 1);

System.out.println("Sorted array: ");

printArray(arr, n);

}

}

// This code is contributed by Ayush Choudhary

`

);

const [pythoncode, setpythonCode] = React.useState(

`

# Python3 implementation of QuickSort

# This Function handles sorting part of quick sort

# start and end points to first and last element of

# an array respectively

def partition(start, end, array):

# Initializing pivot's index to start

pivot\_index = start

pivot = array[pivot\_index]

# This loop runs till start pointer crosses

# end pointer, and when it does we swap the

# pivot with element on end pointer

while start < end:

# Increment the start pointer till it finds an

# element greater than pivot

while start < len(array) and array[start] <= pivot:

start += 1

# Decrement the end pointer till it finds an

# element less than pivot

while array[end] > pivot:

end -= 1

# If start and end have not crossed each other,

# swap the numbers on start and end

if(start < end):

array[start], array[end] = array[end], array[start]

# Swap pivot element with element on end pointer.

# This puts pivot on its correct sorted place.

array[end], array[pivot\_index] = array[pivot\_index], array[end]

# Returning end pointer to divide the array into 2

return end

# The main function that implements QuickSort

def quick\_sort(start, end, array):

if (start < end):

# p is partitioning index, array[p]

# is at right place

p = partition(start, end, array)

# Sort elements before partition

# and after partition

quick\_sort(start, p - 1, array)

quick\_sort(p + 1, end, array)

# Driver code

array = [ 10, 7, 8, 9, 1, 5 ]

quick\_sort(0, len(array) - 1, array)

print(f'Sorted array: {array}')

`

);

return (

<Container>

<h4>

QuickSort is a Divide and Conquer algorithm. It picks an element as

pivot and partitions the given array around the picked pivot. There are

many different versions of quickSort that pick pivot in different ways.

<span>

<ol>

<li>Always pick first element as pivot.</li>

<li>Always pick last element as pivot</li>

<li>Pick a random element as pivot. </li>

<li>Pick median as pivot. </li>

</ol>

</span>

The key process in quickSort is partition(). Target of partitions is,

given an array and an element x of array as pivot, put x at its correct

position in sorted array and put all smaller elements (smaller than x)

before x, and put all greater elements (greater than x) after x.

</h4>

<Tabs>

<TabList>

<Tab>C++</Tab>

<Tab>C</Tab>

<Tab>Java</Tab>

<Tab>Python</Tab>

</TabList>

<TabPanel>

<CodeEditor

value={cppcode}

language="cpp"

onChange={(evn) => setcppCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={javacode}

language="java"

onChange={(evn) => setjavaCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={pythoncode}

language="py"

onChange={(evn) => setpythonCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

</Tabs>

</Container>

);

}

**SelectionSortTheory.jsx:**

import React from "react";

import CodeEditor from "@uiw/react-textarea-code-editor";

import { Tab, Tabs, TabList, TabPanel } from "react-tabs";

import "react-tabs/style/react-tabs.css";

import Card from "@material-ui/core/Card";

import styled from "styled-components";

const Container = styled(Card)`

padding: 10px;

border: 1px solid rgba(0, 0, 0, 0.15);

`;

export function SelectionSortTheory() {

const [cppcode, setcppCode] = React.useState(

`// C++ program for implementation of selection sort

#include <bits/stdc++.h>

using namespace std;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// One by one move boundary of unsorted subarray

for (i = 0; i < n-1; i++)

{

// Find the minimum element in unsorted array

min\_idx = i;

for (j = i+1; j < n; j++)

if (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first element

swap(&arr[min\_idx], &arr[i]);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

cout << arr[i] << " ";

cout << endl;

}

// Driver program to test above functions

int main()

{

int arr[] = {64, 25, 12, 22, 11};

int n = sizeof(arr)/sizeof(arr[0]);

selectionSort(arr, n);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

`

);

const [ccode, setcCode] = React.useState(

`

// C program for implementation of selection sort

#include <stdio.h>

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// One by one move boundary of unsorted subarray

for (i = 0; i < n-1; i++)

{

// Find the minimum element in unsorted array

min\_idx = i;

for (j = i+1; j < n; j++)

if (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first element

swap(&arr[min\_idx], &arr[i]);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

printf("\n");

}

// Driver program to test above functions

int main()

{

int arr[] = {64, 25, 12, 22, 11};

int n = sizeof(arr)/sizeof(arr[0]);

selectionSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

`

);

const [javacode, setjavaCode] = React.useState(

`

// Java program for implementation of Selection Sort

class SelectionSort

{

void sort(int arr[])

{

int n = arr.length;

// One by one move boundary of unsorted subarray

for (int i = 0; i < n-1; i++)

{

// Find the minimum element in unsorted array

int min\_idx = i;

for (int j = i+1; j < n; j++)

if (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first

// element

int temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

// Prints the array

void printArray(int arr[])

{

int n = arr.length;

for (int i=0; i<n; ++i)

System.out.print(arr[i]+" ");

System.out.println();

}

// Driver code to test above

public static void main(String args[])

{

SelectionSort ob = new SelectionSort();

int arr[] = {64,25,12,22,11};

ob.sort(arr);

System.out.println("Sorted array");

ob.printArray(arr);

}

}

`

);

const [pythoncode, setpythonCode] = React.useState(

`# Python program for implementation of Selection

# Sort

import sys

A = [64, 25, 12, 22, 11]

# Traverse through all array elements

for i in range(len(A)):

# Find the minimum element in remaining

# unsorted array

min\_idx = i

for j in range(i+1, len(A)):

if A[min\_idx] > A[j]:

min\_idx = j

# Swap the found minimum element with

# the first element

A[i], A[min\_idx] = A[min\_idx], A[i]

# Driver code to test above

print ("Sorted array")

for i in range(len(A)):

print("%d" %A[i],end=" ")

`

);

const [cscode, setcsCode] = React.useState(

`

// C# program for implementation

// of Selection Sort

using System;

class GFG

{

static void sort(int []arr)

{

int n = arr.Length;

// One by one move boundary of unsorted subarray

for (int i = 0; i < n - 1; i++)

{

// Find the minimum element in unsorted array

int min\_idx = i;

for (int j = i + 1; j < n; j++)

if (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first

// element

int temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

// Prints the array

static void printArray(int []arr)

{

int n = arr.Length;

for (int i=0; i<n; ++i)

Console.Write(arr[i]+" ");

Console.WriteLine();

}

// Driver code

public static void Main()

{

int []arr = {64,25,12,22,11};

sort(arr);

Console.WriteLine("Sorted array");

printArray(arr);

}

}

`

);

const [phpcode, setphpCode] = React.useState(

`

<?php

// PHP program for implementation

// of selection sort

function selection\_sort(&$arr, $n)

{

for($i = 0; $i < $n ; $i++)

{

$low = $i;

for($j = $i + 1; $j < $n ; $j++)

{

if ($arr[$j] < $arr[$low])

{

$low = $j;

}

}

// swap the minimum value to $ith node

if ($arr[$i] > $arr[$low])

{

$tmp = $arr[$i];

$arr[$i] = $arr[$low];

$arr[$low] = $tmp;

}

}

}

// Driver Code

$arr = array(64, 25, 12, 22, 11);

$len = count($arr);

selection\_sort($arr, $len);

echo "Sorted array : \n";

for ($i = 0; $i < $len; $i++)

echo $arr[$i] . " ";

?>

`

);

return (

<Container>

<h4>

Selection sort is a sorting algorithm that selects the smallest element

from an unsorted list in each iteration and places that element at the

beginning of the unsorted list.

</h4>

<Tabs>

<TabList>

<Tab>C++</Tab>

<Tab>C</Tab>

<Tab>Java</Tab>

<Tab>Python</Tab>

<Tab>C#</Tab>

<Tab>PHP</Tab>

</TabList>

<TabPanel>

<CodeEditor

value={cppcode}

language="cpp"

onChange={(evn) => setcppCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={ccode}

language="c"

onChange={(evn) => setcCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={javacode}

language="java"

onChange={(evn) => setjavaCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={pythoncode}

language="py"

onChange={(evn) => setpythonCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={cscode}

language="csharp"

onChange={(evn) => setcsCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

<TabPanel>

<CodeEditor

value={phpcode}

language="php"

onChange={(evn) => setphpCode(evn.target.value)}

padding={15}

style={{

fontSize: 12,

backgroundColor: "#f5f5f5",

fontFamily:

"ui-monospace,SFMono-Regular,SF Mono,Consolas,Liberation Mono,Menlo,monospace",

}}

/>

</TabPanel>

</Tabs>

</Container>

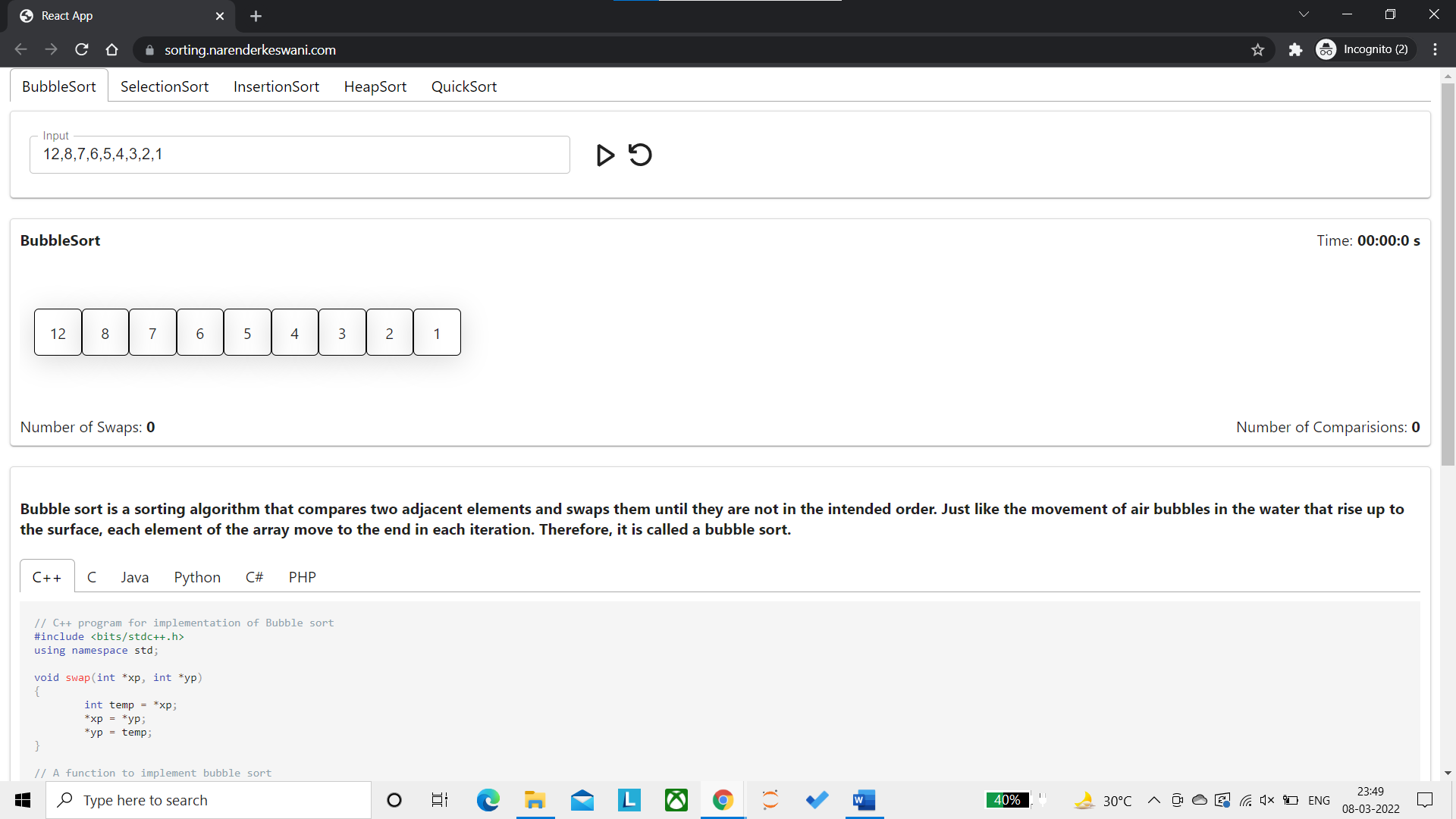
);

}

**ScreenShots:**

**Basic Layout:**

We can see a panel with bubble sort selected as the method, it has provided us with an input box(for array), visualization panel, algorithm description panel and code panel.

****

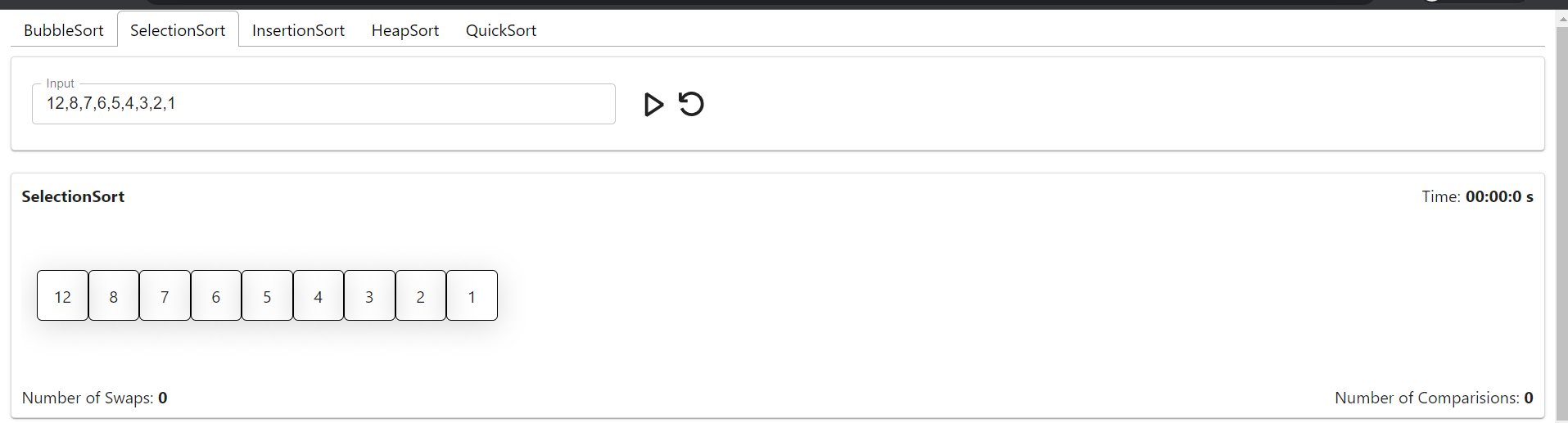
**Functioning Visualization Panel:**

We can see the input box where we can put our array to be sorted with separation by comma. We can see a run button to start visualization and a restart button to restart it.

We can also see a timer at the right corner that shows the total time taken by the algorithm to sort the array.

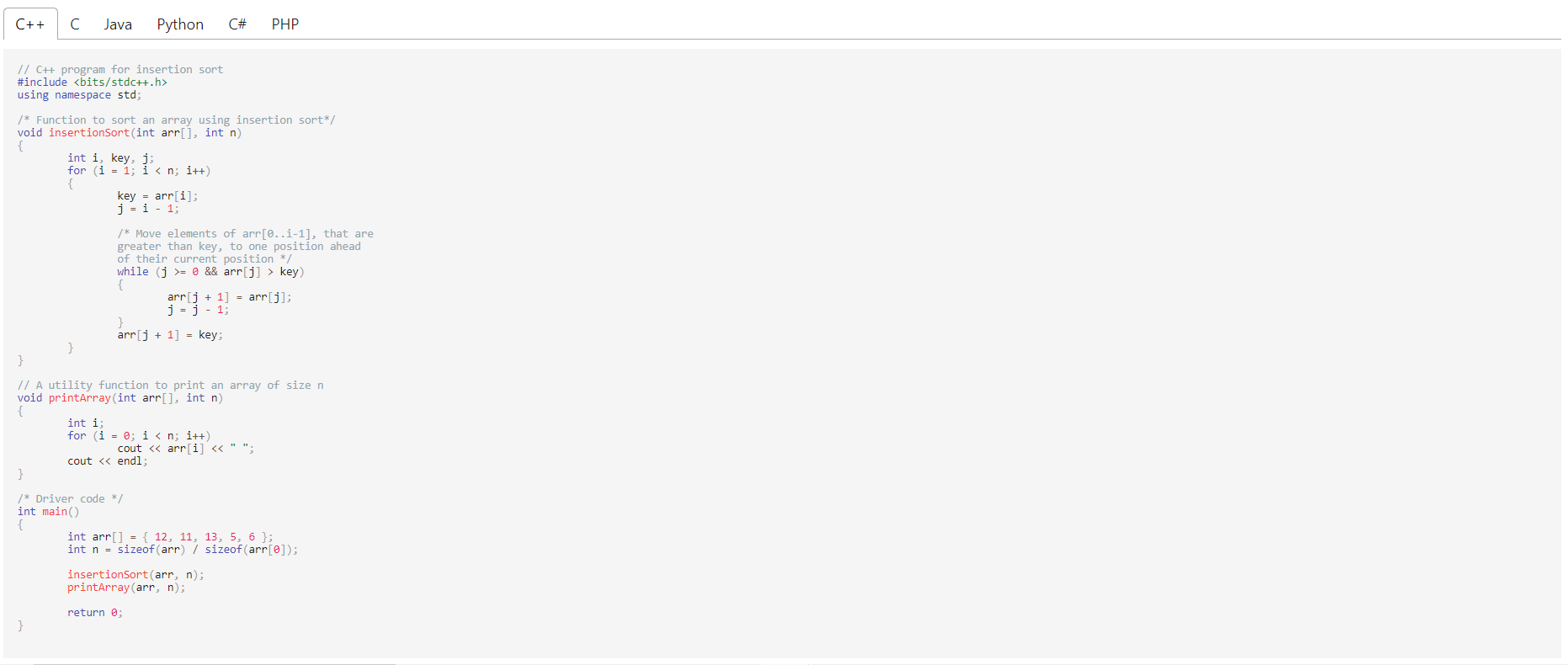
We also have a count of the number of comparisons the algorithm does throughout the sorting process.

In the bottom left, we have the number of swaps the algorithm will perform throughout the process.

****

**Code Panel:**

We can see that we get the code of the selected algorithm in selected programming languages.

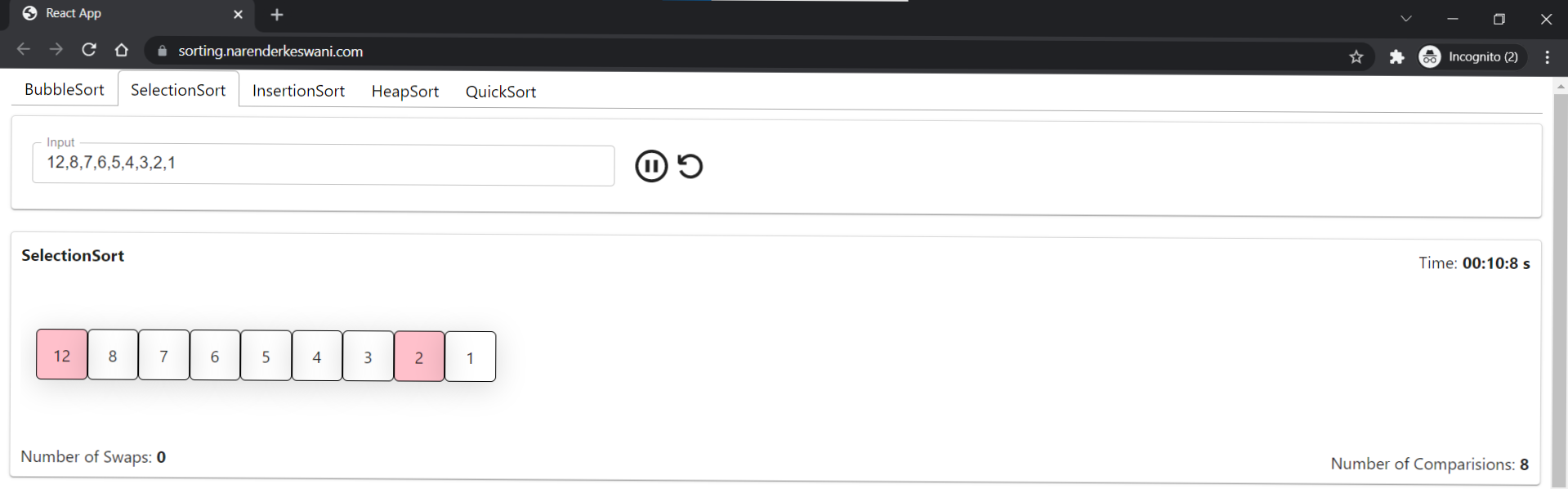
****

****

**Visualization Process:**

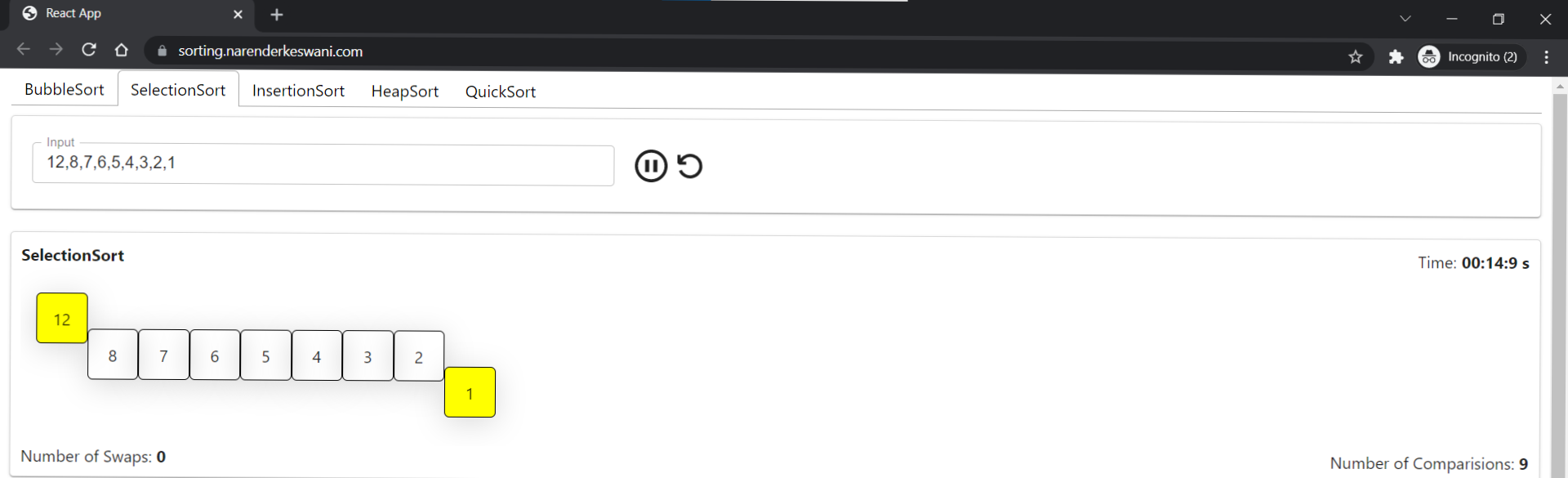
**Comparison**

The numbers in red indicate that they are being compared in that iteration.

****

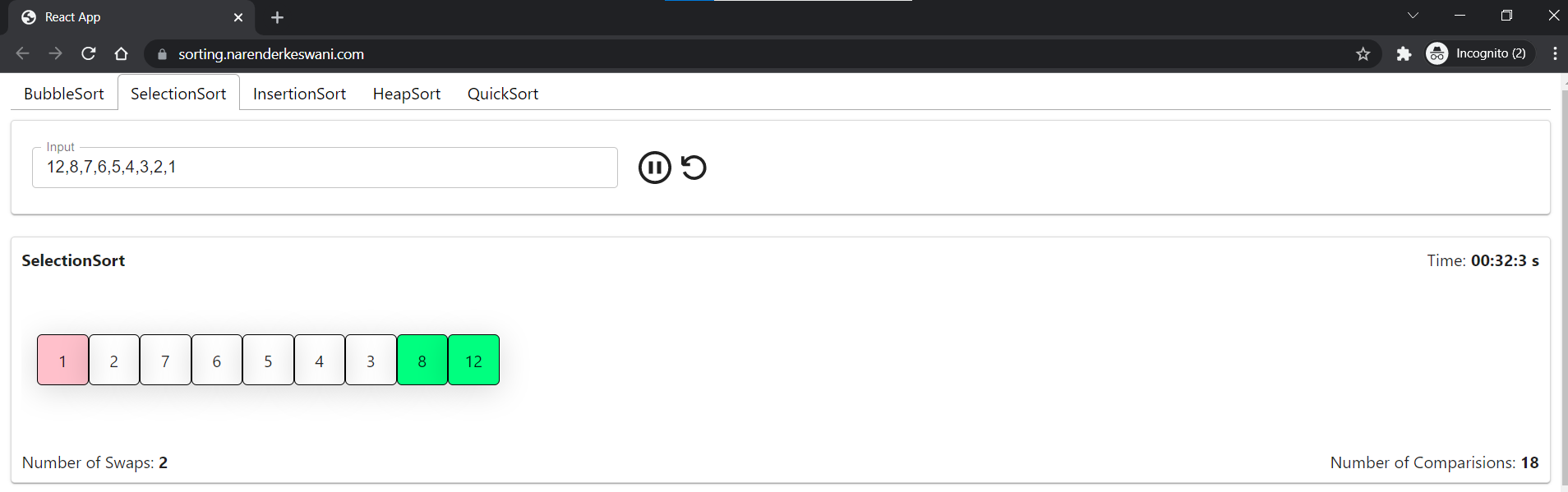
**Swap**

Numbers in yellow indicate that the condition is satisfied and hence those two numbers will now be swapped. We can observe the number of swaps, comparisons and time till current iteration.

****

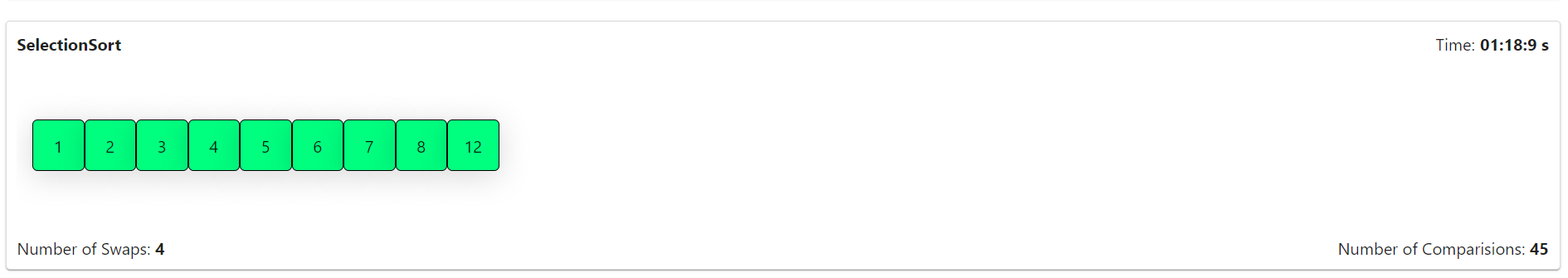
**Position Finalized**

The numbers which now have a fixed position in the array are indicated in green box.

****

**Whole array sorted**

When all the boxes are green, the algorithm stops as the array is sorted. Now the number of swaps, comparisons and time is final.

****

* 1. **CONCLUSION:**

Online education is here and is highly likely to stay and grow. The review of its history clearly shows online education has developed rapidly, fueled by Internet connectivity, advanced technology, and a massive market. The Internet has made online learning possible, and many researchers and educators are interested in online learning to enhance and improve student learning outcomes. This website Gives Overview and Visualization of a Each and every Data Structure which we will including in that Website Online Education Brought a Positive Impact in the Lives of Student and Working Professionals. In future this website can get updated as per the requirement of the Student with new ideas and also different Programming concepts and all other languages. We can Also Include compile, video lecture, coding projects, coding challenges. We will include more features, function For Online Learning Like code editor, We can Also Include compile, video lecture, coding projects, coding challenges.

This system is implemented for visualizing of algorithms. This is a helpful tool for all kinds of learners/scholars to easily understand the implicit sequences of algorithm . Here the users are allowed to select the options, either searching or sorting and graph path finding . Then they are allowed to give input and they can select the algorithms from the list and the algorithm is explained visually. In future to enhance and continue this project, the system may include greedy algorithm and dynamic progrmming . Voice can further be included to the system, to give more interaction for the end users. It helps to improve the quality of education in the field

* 1. **REFERENCES:**

[1] K. Mehlhorn, P. Sanders, Algorithms and Data Structures (Springer-Verlag, Berlin Heidelberg, 2008)

[2] GeeksForGeeks: https://www.geeksforgeeks.org/sorting-algorithms/

[3] S. Khuri, Designing Effective Algorithm Visualizations, In proceedings of: First International Program Visualization Workshop, ITiCSE, Porvoo, Finland, July 7 - 8, 2000, Available: <http://www.cs.sjsu.edu/~khuri/invited.html>

[4] C.D. Hundhausen, S.A. Douglas, J.T. Stasko, A Meta-Study of Algorithm Visualization Effectiveness, J. Visual Lang. Comput. 13, 259–290, 2002

[5] ACM/IEEE-CS Joint Task Force on Computing Curricula, Computer Science Curricula 2013, ACM Press and IEEE Computer Society Press, December 2013.

**Check Live Project at:** [**https://sorting.narenderkeswani.com/**](https://sorting.narenderkeswani.com/)