SRI KRISHNA COLLEGE OF TECHNOLOGY
(An Autonomous Institution | Affiliated to Anna University Chennai| Accredited by NAAC with A Grade)
KOVAIPUDUR, COIMBATORE 641042

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

A Project Report on

Visualization of Bubble Sort

by

18TUCS248 VIGNESH R.

18TUCS214 SANTHOSH R.

18TUCS249 VIGNESHWAR M.

18TUCS209 SABARI BALAMURGAN

18CS405 – DESIGN ANALYSIS OF ALGORITHMS

MINI PROJECT

MARCH 2020

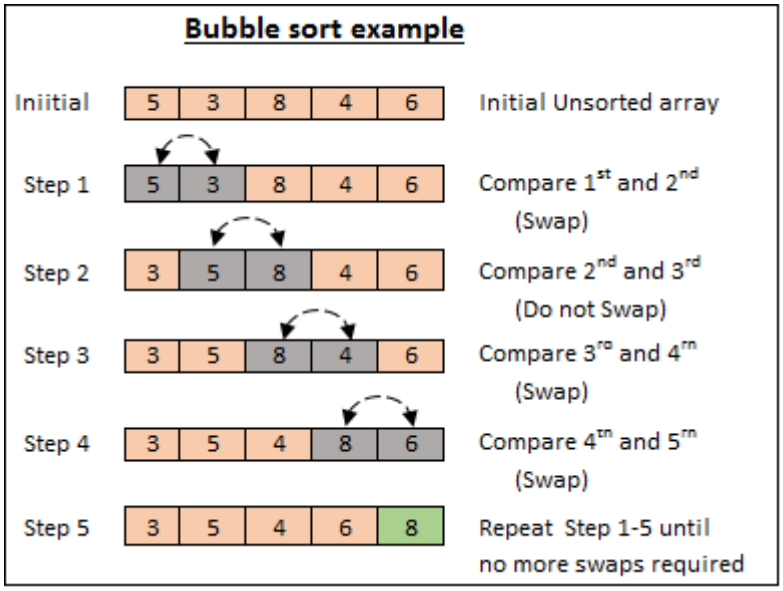
INDEX

Abstract:

There are many efficient algorithms to perform different tasks. The bubble sort is an elementary sorting algorithm; it is the oldest and simplest sort in use that does not require extra memory. Unfortunately, it is also one of the slowest. It works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted [1].

Project description:

Computer scientists always strive to find better and faster algorithms for computational problems. It is usually true that programmers and/or users come across a plethora of different algorithms when looking to solve a particular problem efficiently. Each one of these algorithms might offer different guarantees and properties, but it is unlikely that a single one of them is the best (in terms of speed) in all possible cases [1]. Sorting algorithms are an important part of managing data. Each algorithm has particular strengths and weaknesses. For instance, in the bubble sort algorithm, the number of comparisons is irrespective of the input data set, as it does not require extra memory, which makes it optimum in terms of memory requirements. In fact, bubble sort performs exceptionally well for some input cases [2,3]. On the other hand, many algorithms that have the same time complexity do not have the same speed on the same input. Therefore, algorithms are better judged based on their average case, best case, and worst case efficiencies[4,5]. Most of the sorting algorithms in use have an algorithmic efficiency of either $O(n^2)$ or $O(n*\log(n))$. Quicksort, for instance, typically has a time complexity of $O(n\log(n))$ which is optimal in general cases, though its worst case efficiency is similar to bubble sort which has a time complexity of $O(n^2)$ [6,7]. The size of the sorting algorithm itself is not much of an issue today, as there are considerable variations of shorter and simpler versions of the algorithms that are easier to maintain [8]. This paper introduces an improvement to the bubble Sort algorithm. The improvement uses dynamic programming (through the use of a stack) to eliminate unnecessary passes by detecting the number of comparisons. A stack is used to store the location of previous Max (largest value) elements found, instead of starting from the beginning each time the largest element is found and placed at the end of the array; as is the case in the classical algorithm.



Modules Used:

1. **Shuffle-array:**

   Shuffle-array is javascript module used to shuffle various elements.

IMPLEMENTATION:

Genetic Algorithm Phases:

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

import shuffle from "shuffle-array";

const WIDTH = 1200;

const HEIGHT = 500;

function Bar(props) {

  let height = props.height * (HEIGHT / props.count);

  return (
```

```
    <div
      style={{
        display: "inline-block",
        width: `${props.width}px`,
        height: `${height}px`,
        backgroundColor: `${props.color}`,
        marginLeft: "2px"
      }}
    ></div>
  );
}

function App() {
  const [count, changeCount] = useState(0);
  const [bars, changeBars] = useState([]);
  const [current, changeCurrent] = useState([0, 0]);
  const [running, changeRunning] = useState(false);

  let width = WIDTH / count - 2;

  useEffect(() => {
    let temp = [];
    for (let i = 1; i <= count; i++) temp.push(i);
    shuffle(temp);
    changeRunning(false);
```

```
  changeCurrent([0, 0]);
  changeBars(temp);
}, [count]);

useEffect(() => {
  setTimeout(() => {
    if (running) {
      if (current[0] < count) {
        if (current[1] < count - 1) {
          if (bars[current[1]] > bars[current[1] + 1]) {
            [bars[current[1]], bars[current[1] + 1]] = [
              bars[current[1] + 1],
              bars[current[1]]
            ];
            changeBars(bars);
          }
          changeCurrent([current[0], current[1] + 1]);
        } else changeCurrent([current[0] + 1, 0]);
      } else changeRunning(false);
    }
  });
}, [running, count, current, bars]);

return (
  <>
```

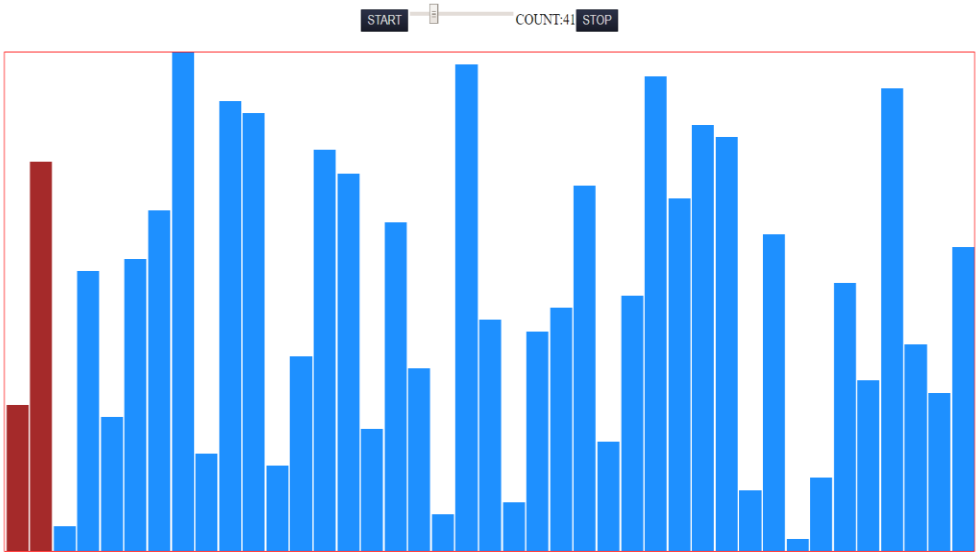
```
<header style={{ textAlign: "center" }}>
<button onClick={() => changeRunning(true)}>START</button>
{running ? (
<input
type="range"
min="0"
max="200"
step="1"
value={count}
onChange={e => changeCount(e.target.value)}
disabled
/>
) : (
<input
type="range"
min="0"
max="200"
step="1"
value={count}
onChange={e => changeCount(e.target.value)}
/>
)}
<span>COUNT:{count}</span>
<button onClick={() => changeRunning(false)}>STOP</button>
```

```
</header>
<main
  style={{
    width: `${WIDTH}px`,
    height: `${HEIGHT}px`,
    border: "1px solid red",
    margin: "20px auto"
  }}
>
  {bars.map((height, index) => {
    return (
      <Bar
        height={height}
        width={width}
        count={count}
        color={
          index === current[1] || index === current[1] + 1
            ? "brown"
            : "DodgerBlue"
        }
        key={height}
      />
    );
  })}
</main>
```

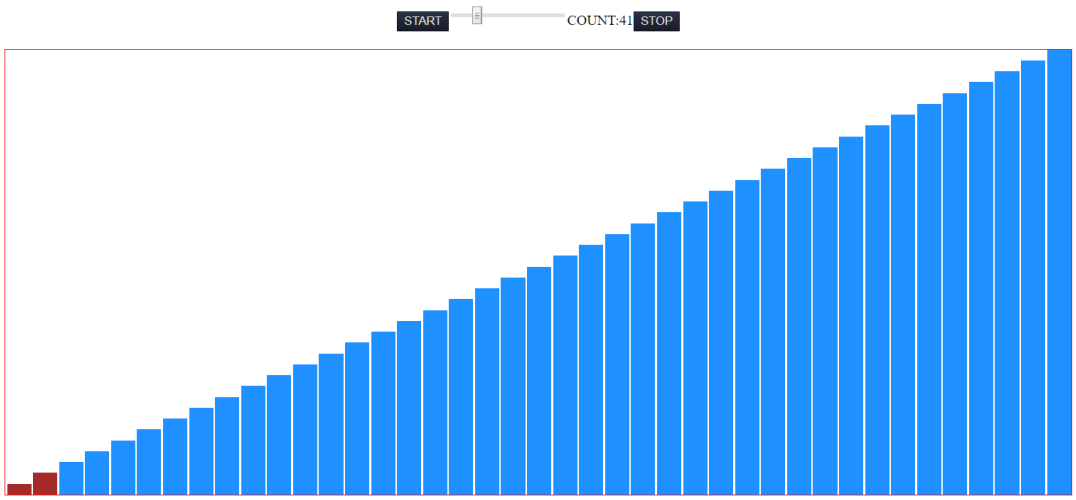
```
                    </>
                  );
                }

                export default App;bestEver = population[i];
            }
            if (d < currentRecord) {
                currentRecord = d;
                currentBest = population[i];
            }
            fitness[i] = 1 / (pow(d, 8) + 1);
        }
}
```

OUTPUT:

SELECTING COUNT AND BEFORE SORTING:



AFTER SORTING:



CONCLUSION:

Thus the implementation of Visualization of Bubble Sort has been established with a simple application and at the same the development at the future includes many features and so on…