JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

NOIDA - 128

DATA STRUCTURES LAB PROJECT

Algo - Visualizer

SUBMITTED BY :

NITIN CHAUDHARY (9921103163)

SHIVANSH PANDEY(9921103152)

SANSKAR GUPTA (9921103153)

ARYAN PATEL (9921103156)

SUBMITTED TO :

DR. VARSHA GARG

DR. SHIKHA MEHTA

DR. KRISHNA ASAWA

INDEX

1. PROBLEM STATEMENT

The problem that our project will solve is the learning and visualization issue faced by the students who are new to data structures and algorithms and can not intuitively visualize all the algorithms and the things that happen behind the scenes when these algos are executed on a given data structure. Our project will help all such students and make learning data structures and algorithms easier for them.

The project will be helpful not only for the students who are trying to learn the topic by themselves but also for the teacher so that we can facilitate them and make it easier for the teacher so that they have to spend less time on a given topic. They can very easily use this project of ours and explain the sorting techniques that we have visualized to explain it to their respective students.

2. SOLUTION APPROACH

The way our team approached this problem was by thinking of the issues that we ourselves faced when we were trying to learn sorting techniques in the 3rd semester of our college and we also thought of any such issues that we will face later on. We wanted to visualize all the sorting techniques so we had to find an external library that will help us in doing the same. We decided that we will be using the Raylib C++ library to add the visualization to our project. We will be using C++ and we will be visualizing sorting techniques like Bubble Sort, Selection Sort, Insertion Sort, Merge Sort..

3. IMPLEMENTATION

While making this algorithm visualizer project in C++ we have used the **Raylib** library for adding graphics and 2-D animations. We have used various predefined functions and data-types provided in the above library and various user-defined functions in this application to provide the effective **Graphical User Interface.**

We have also added a feature where the teacher or anybody using the software can give their personalized inputs for the sizes of bars that are going to be sorted. This makes sure that they are able to relate to the program and see their own inputs getting sorted using the requested sorting technique. They will also be able to check on each step if their implementation matches with the software.

a. Some Built-in Functions used :

- **void DrawText(const char *text, int posX, int posY, int fontSize, Color color):**
  Draw text (using default font)
- **void InitWindow(int width, int height, const char *title);**
  Initialize window and OpenGL context
- **void ClearBackground(Color color);**
  Set background color (framebuffer clear color)
- **void BeginDrawing(void);**
  Setup canvas (framebuffer) to start drawing
- **int MeasureText(const char *text, int fontSize);**
  Measure string width for default font
- **void DrawRectangleV(Vector2 position, Vector2 size, Color color);**
  Draw a color-filled rectangle (Vector version)

Some User-Defined Functions used :

- **void ShowStartingScreen();**
  First screen the user sees
- **void ShowOptions();**
  Screen shown on selecting a sorting technique
- **void Button(float x, float y, char *Text, Color color, bool & state);**
  Position of button, text in the button and the state

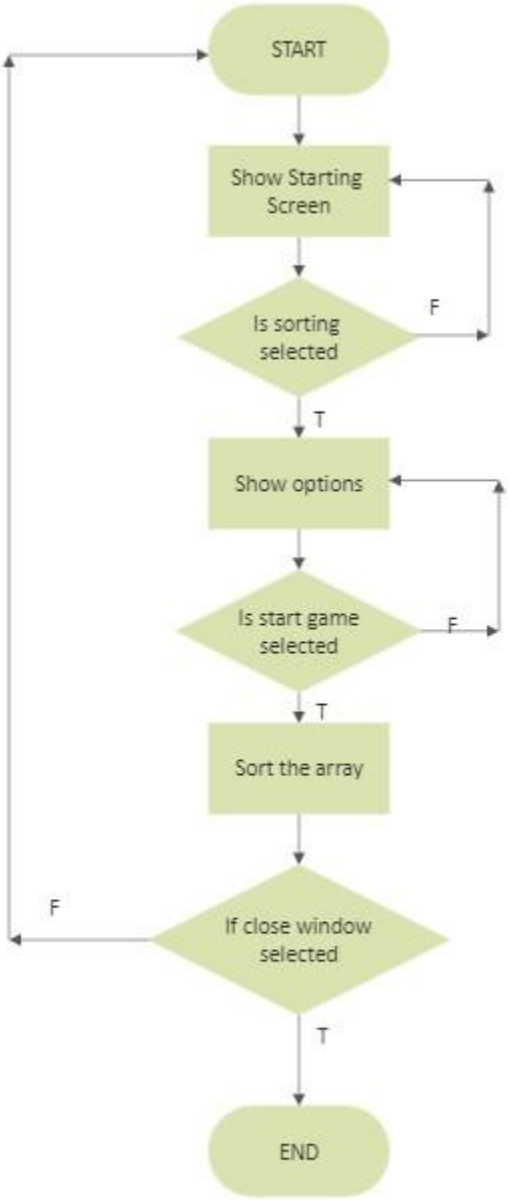
Some Data-Types or Class used :

- **Rectangle(float x, float y, int width, int height) :**
  Make a rectangle box with coordinates (x,y), width and height given in the argument.
- **Color :**
  This is a class used to declare an object which can store color type.
  We use this to color our bars in visualization.

Data Structures Used-

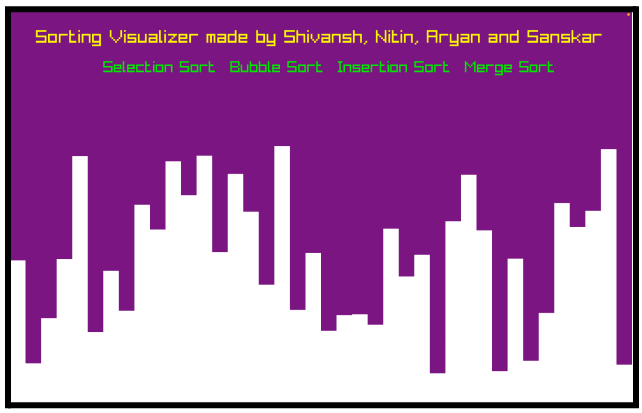
We have used **Arrays** and **Vectors** that are the basic and simple Data structures and can be used to easily learn sorting algorithms and their visualization.

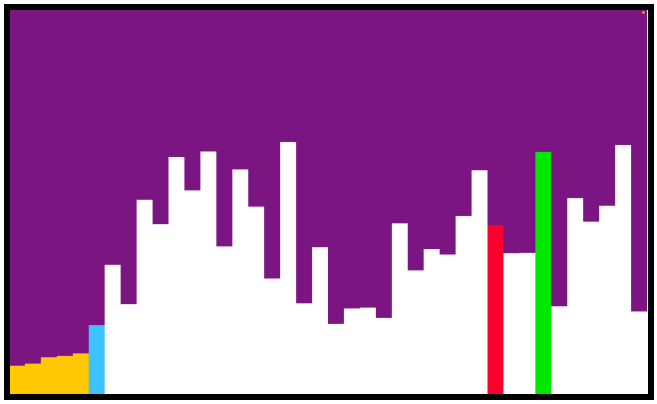
b.Flow chart

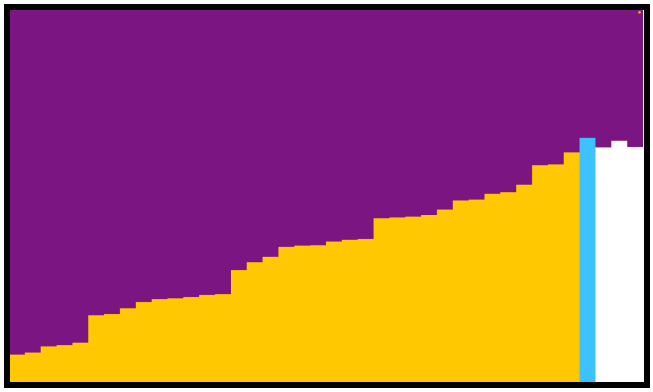

START
Show Starting Screen
Is sorting selected
F
T
Show options
Is start game selected
F
T
Sort the array
If close window selected
F
T
END

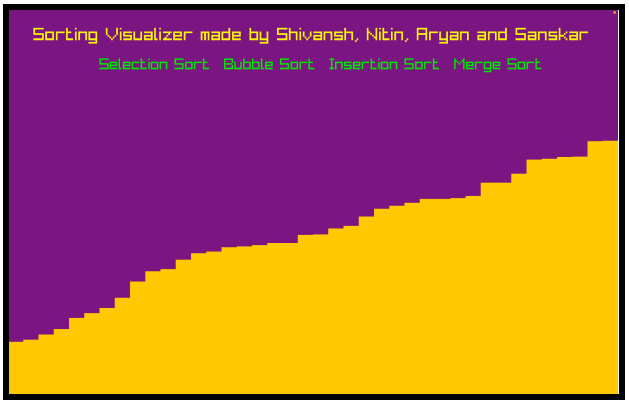
c.Output Screen shots-

- **Selection sort visualization**

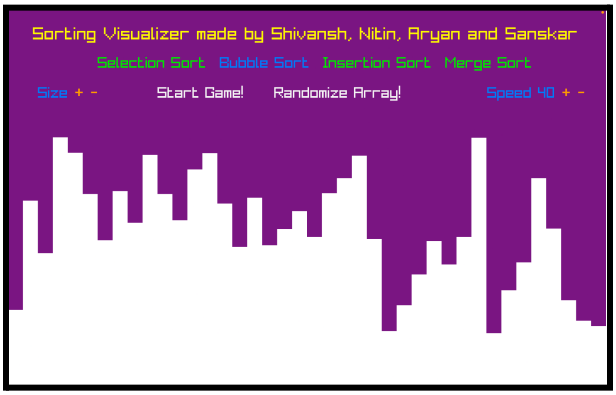


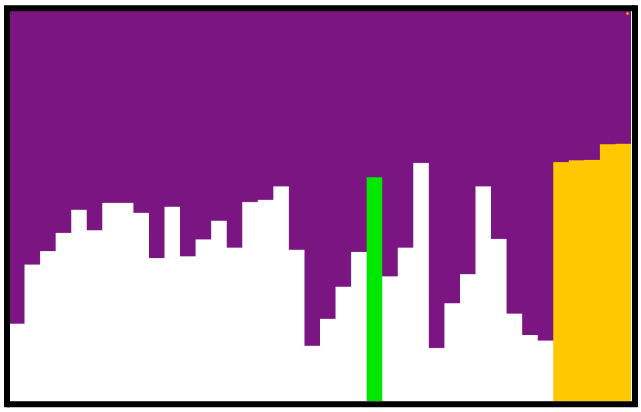


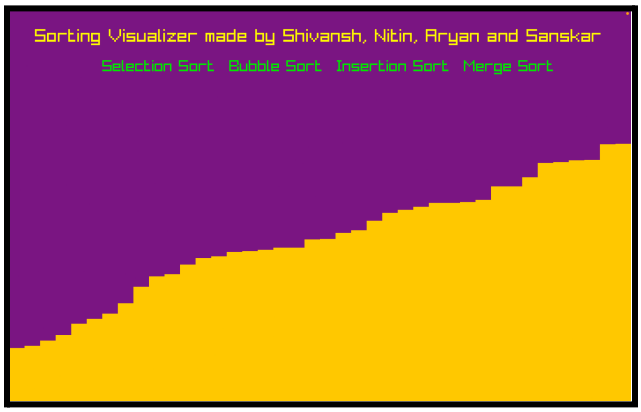




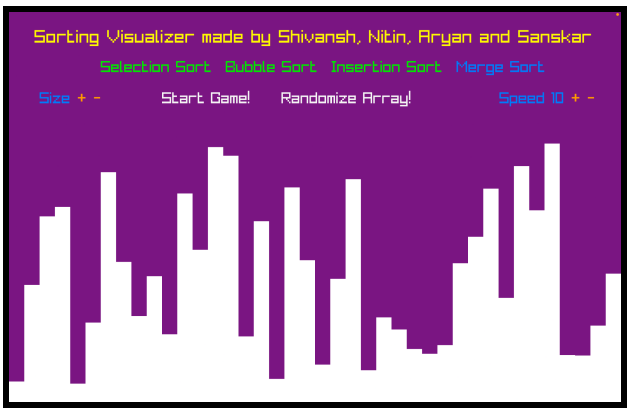
- **Bubble sort visualization**

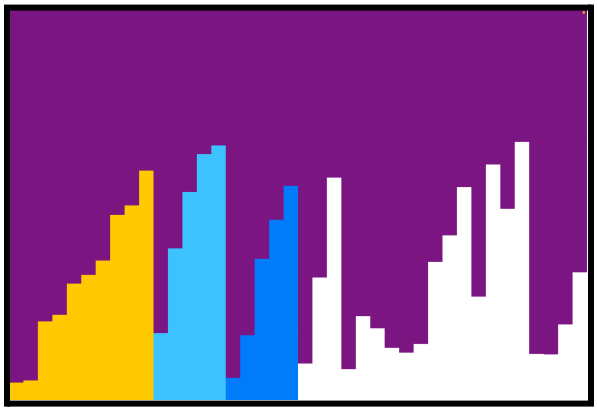


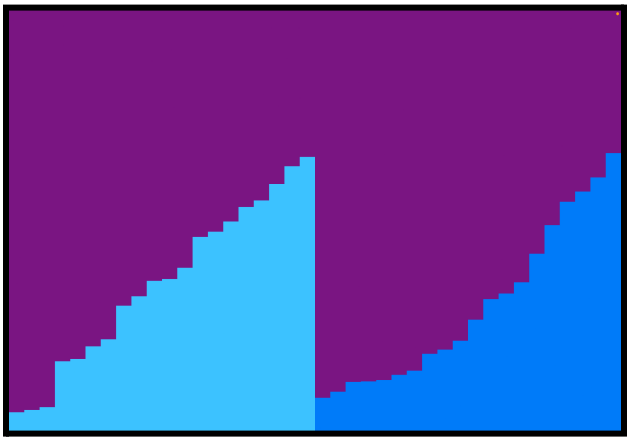


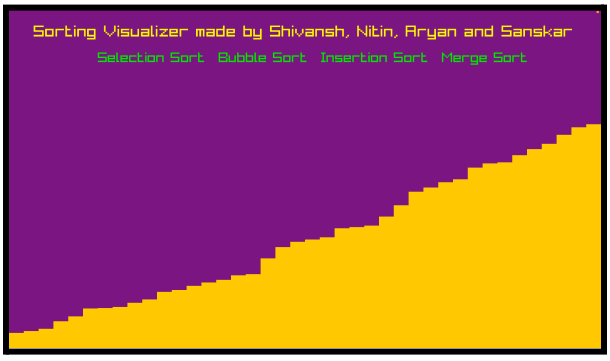


- **Merge sort visualization**

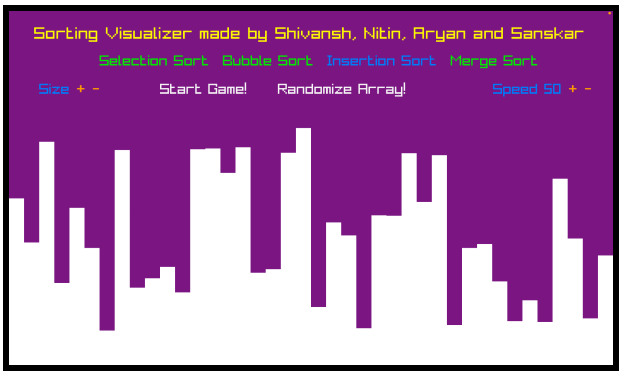


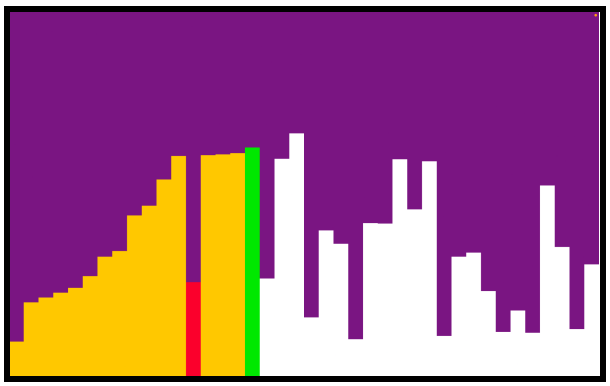


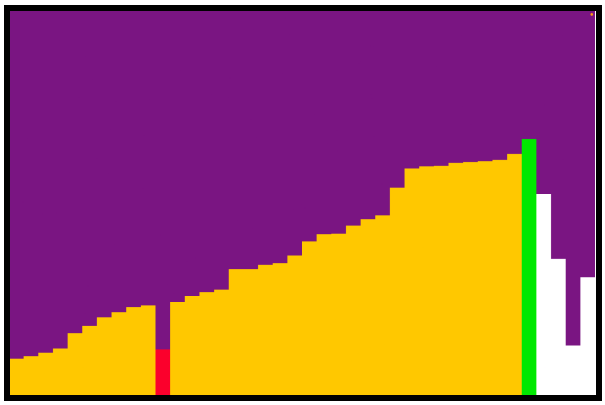


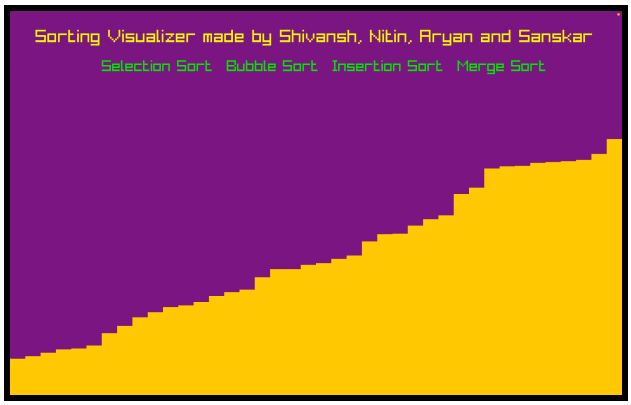


- **Insertion sort visualization**









4. UTILITY OF PROJECT

One of the essential steps in the education field is to understand concepts which is easily possible by visualization. Visualization takes the raw data, models it, and delivers the data so that easy understanding can be obtained.

Data visualization is very important for faster and better understanding of data for faster, better decision-making. Understanding of greater volumes of data in less time is possible by visualization only.

By using this project of ours , teachers can teach the students very easily with help of visualization which will make all the concepts easier to understand for students and memorize for a long time.

Our project is the implementation of this algorithm's visualization which has a huge utility and future scope.

5. COMPLEXITY ANALYSIS

Selection Sort :-

Time Complexities:

- **Worst Case Complexity: O(n2)**

  If we want to sort in ascending order and the array is in descending order then, the worst case occurs.

- **Best Case Complexity: O(n2)**

  It occurs when the array is already sorted

- **Average Case Complexity: O(n2)**

  It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

  The time complexity of the selection sort is the same in all cases. At every step, you have to find the minimum element and put it in the right place. The minimum element is not known until the end of the array is not reached.

Space Complexity:

Space complexity is O(1) because an extra variable temp is used.

Insertion Sort :-

Time Complexities

- **Worst Case Complexity: O(n2)**

  Suppose, an array is in ascending order, and you want to sort it in descending order. In this case, worst case complexity occurs.

  Each element has to be compared with each of the other elements so, for every nth element, (n-1) number of comparisons are made.

  Thus, the total number of comparisons = n*(n-1) ~ n2

- **Best Case Complexity: O(n)**

  When the array is already sorted, the outer loop runs for n number of times whereas the inner loop does not run at all. So, there are only n number of comparisons. Thus, complexity is linear.

- **Average Case Complexity: O(n2)**

  It occurs when the elements of an array are in jumbled order (neither ascending nor descending).

  Space Complexity

  Space complexity is O(1) because an extra variable key is used.

Bubble Sort :-

Time Complexities

- **Worst Case Complexity: O(n2)**

  If we want to sort in ascending order and the array is in descending order then the worst case occurs.

  Best Case Complexity: O(n)

  If the array is already sorted, then there is no need for sorting.

- **Average Case Complexity: O(n2)**

  It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

- **Space Complexity :**

  Space complexity is O(1) because an extra variable is used for swapping.

  In the optimized bubble sort algorithm, two extra variables are used. Hence, the space complexity will be O(2).

Merge Sort :-

Time Complexity :

- **Best Case Complexity: O(n*log n)**

  If the array is already sorted, then there is no need for sorting.

- **Worst Case Complexity: O(n*log n)**

  The worst-case time complexity is also O(n*log n), which occurs when we sort the descending order of an array into the ascending order.

- **Average Case Complexity: O(n*log n)**

  The average-case time complexity for the merge sort algorithm is O(n*log n), which happens when 2 or more elements are jumbled, i.e., neither in the ascending order nor in the descending order.

Space Complexity

The space complexity of merge sort is O(n) as the extra space is used due to recursion which we have used to perform the merge sort.

6. REFERENCES

- https://www.raylib.com/cheatsheet/cheatsheet.html
- https://www.youtube.com/watch?v=HPDLTQ4J_zQ
- https://www.youtube.com/watch?v=E2Brnev0Olc
- https://www.raylib.com/
- https://www.raylib.com/examples.html
- https://robloach.github.io/raylib-cpp/classraylib_1_1_rectangle.html
- https://www.raylib.com/cheatsheet/cheatsheet.html