University of Delhi

SOFTWARE ENGINEERING PROJECT REPORT

BSC (Hons.) COMPUTER SCIENCE
(2021 – 24)

iSort - The Sorting Visualizer

Submitted By -
Rishabh Jain
Rishabh Chopra
Deepanshu Punj
Priyanshu Mishra

RAM LAL ANAND COLLEGE
(University Of Delhi)

iSort - The Sorting Visualizer
(Software Engineering Project Report)

BSC (Hons.) COMPUTER SCIENCE
(2021 – 24)

Submitted By -
Rishabh Jain (21058670042)
Rishabh Chopra (21058570041)
Deepanshu Punj (21058570013)
Priyanshu Mishra (21058570037)

Under Supervision of :
Dr. Vandana Gandotra

Certificate

This is to certify that the Software Engineering project report entitled "**iSort**" is carried out by **Rishabh Jain**, **Rishabh Chopra**, **Deepanshu Punj,** and **Priyanshu Mishra**, students of BSc (Hons) Computer Science 4th Semester, Ram Lal Anand College, the University of Delhi under the supervision of **Dr. Vandana Gandotra**. This report is not submitted to any other organization/institution to award any other degree/diploma.

Dr. Vandana Gandotra
(Supervisor)

Dr. Rakesh Kumar Gupta
(Principal)

ACKNOWLEDGEMENT

The success of any project depends largely on the encouragement and guidelines of many other people. We take this opportunity to express our gratitude to the people who have been instrumental in completing this project.

We want to express our sincere gratitude to our project supervisor **Dr. Vandana Gandotra** for guiding us. We are highly indebted for her guidance and constant supervision, for providing necessary information regarding the project, and for her support in completing the project. The guidance and support received from all staff members were vital for the project's success.

We are grateful for their constant support and help. We would take this opportunity to express our gratitude toward principal **Dr. Rakesh Kumar Gupta**, who was always a source of encouragement for us. We have made efforts in this project. However, it would not have been possible without many individuals' kind support and help. We want to extend our sincere thanks to all of them.

ABSTRACT

The Sorting Visualizer project is a software application that aims to provide a comprehensive understanding of various sorting algorithms through a visual representation. The project has been developed using various programming languages, including HTML, CSS, and JavaScript. The Sorting Visualizer offers a user-friendly interface that allows users to select and compare different sorting algorithms and understand how they operate.

The main objective of this project is to provide a visual representation of the sorting algorithms, which would make it easier for individuals to understand and analyze the sorting process. The sorting algorithms implemented in this project include Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort, and many more. Users can select one or multiple sorting algorithms and visualize how each algorithm performs the sorting process. The application offers customization options that allow users to adjust the size of the array and delay time.

The Sorting Visualizer project has several benefits. Firstly, it provides a visual representation of the sorting process, which makes it easier for users to understand the sorting algorithms. Secondly, the project offers a comparison feature that allows users to compare the performance of different sorting algorithms by showing their time complexity. This feature helps users to determine which sorting algorithm is most efficient for a particular dataset. Thirdly, the project offers customization options that allow users to adjust the size of the array and delay time. This feature enhances the user experience and allows users to personalize the application according to their preferences.

In conclusion, the Sorting Visualizer project is an excellent educational tool for anyone interested in learning about sorting algorithms. The project offers a comprehensive understanding of various sorting algorithms through a visual representation. The application is user-friendly, easy to navigate, and offers customization options that enhance the user experience. Overall, the Sorting Visualizer project is a valuable resource for anyone looking to improve their understanding of sorting algorithms.

Project Logo

Project Website Link

Contents

Introduction

The Sorting Visualizer project is a software application that aims to provide a visual representation of various sorting algorithms. This project has been developed using programming languages such as HTML, CSS, and JavaScript. The application provides a user-friendly interface that allows users to select and compare different sorting algorithms and understand how they operate. The main objective of this project is to help individuals gain a better understanding of sorting algorithms by providing a visual representation of their operation. This report will discuss the development, features, and benefits of the Sorting Visualizer project.

1.1 Background

Sorting algorithms are an essential aspect of computer science and play a crucial role in data processing and analysis. The visualization of sorting algorithms is an effective way of understanding how these algorithms work. The Sorting Visualizer project aims to provide a visual representation of various sorting algorithms to enhance understanding and analysis. This project uses web technologies, including HTML, CSS, and JavaScript, to create an interactive user interface that enables users to observe and compare sorting algorithms.

Existing Systems

The existing system for sorting visualizers is manual and lacks a visual representation of sorting algorithms. They are however not user-friendly as most of the functionality is hidden in sidebars, etc.

Problem with Existing System

The existing system for sorting visualizers is mostly limited to text-based outputs, which can be difficult to understand for individuals without programming experience. While some sorting algorithms can be visually demonstrated through diagrams and graphs, these methods are not interactive and do not offer customization options. The Sorting Visualizer project aims to fill this gap by providing a comprehensive visual representation of various sorting algorithms through a user-friendly interface with customization options.

1.2 Problem Statement

The main objective of this project is to create a web application as a visualization tool. A web application built using modern JavaScript technology that will visualize the flow and logic of various sorting algorithms.

The UI will contain options to select one of the sorting algorithms which were implemented and several items or elements in the data array, a control button to adjust sorting speed along with a dry run with the inputted array of the running algorithm. The data array of the selected size will be filled in with randomly generated unique values. The data set is represented as a vertical bar with the height of their respective values. After the sorting is started, the
stepwise arrangement of data in ascending order based on their value/height will be visualized in the UI.

1.3 Process Model

A process model for a software engineering project is chosen based on the nature of the project and application, the methods and tools to be used, and the required controls and deliverables.
The Model used to build **"iSort"** is **Incremental Process Model**.

The Incremental Process Model is a software development methodology that involves developing a system through a series of small iterations or increments.

Each iteration consists of the requirements gathering, design, implementation, testing, and deployment of a small section of the system. In the context of the Sorting Visualizer project, the Incremental Process Model can be applied in the following manner:

• **Requirements Gathering**:
In the first iteration, the basic requirements of the Sorting Visualizer project were identified. These requirements included the selection of sorting algorithms, the size of the array, and the delay time between sorting steps. The requirements were further refined in subsequent iterations based on user feedback and testing.

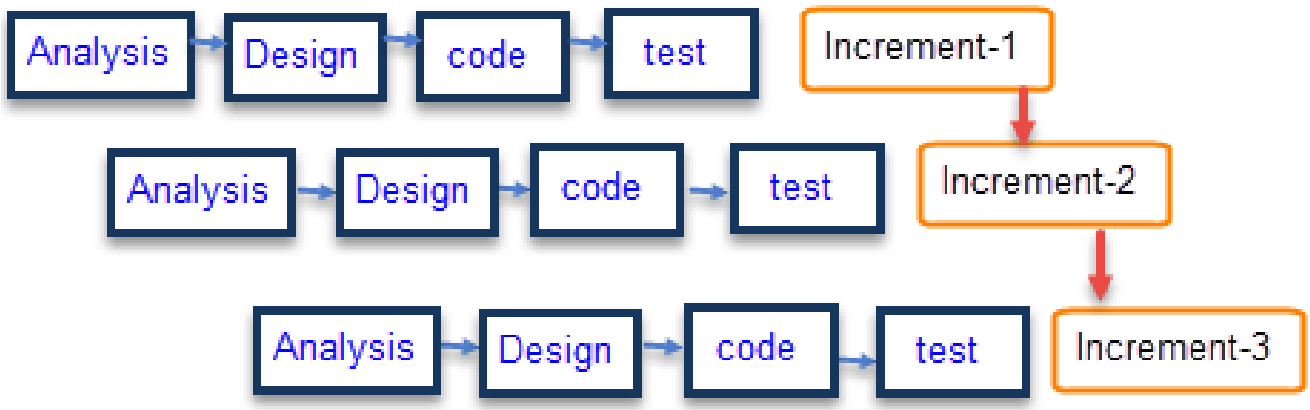
• **System Design**:
In the second iteration, the initial design of the Sorting Visualizer application was created. The design included the user interface, the selection of sorting algorithms, and the visualization of the sorting process. The design was further improved in subsequent iterations based on user feedback and testing.

• **Implementation**:
In the third iteration, the initial implementation of the Sorting Visualizer application was created. This involved writing the code for the user interface, sorting algorithms, and visualization of the sorting process. The implementation was further improved in subsequent iterations based on user feedback and testing.

• **Testing**:
In the fourth iteration, the initial testing of the Sorting Visualizer application was performed. This involved testing the application for errors, bugs, and user interface issues. The testing was further improved in subsequent iterations based on user feedback and testing.

• **Deployment**:
In the fifth iteration, the initial deployment of the Sorting Visualizer application was performed. The application was made available to users for testing and feedback. The deployment was further improved in subsequent iterations based on user feedback and testing.

Using the Incremental Process Model in the development of the Sorting Visualizer project allowed for continuous feedback and improvement throughout the development process. Each iteration allowed for the refinement of the requirements, design, implementation, testing, and deployment of the application. This approach ensured that the final product was of high quality and met the needs of its users. The Incremental Process Model is a valuable methodology for developing software applications that require continuous improvement and feedback.



Incremental Model

Requirement and Analysis

The primary function of requirement analysis is that it translates the ideas in the mind of the clients into a formal document. Thus, the output of this phase is a set of precisely specified requirements that are complete and consistent. This document is called Software Requirement Specification.

2.1 Software Requirement Specification

A Software Requirement Specification (SRS) is an extensive document that defines the requirements of a software system to be developed.

This document describes all the software requirements for "**ISORT: The Sorting Visualizer**".It describes what functions the software has to perform and provides a complete picture of the project. It acts as a medium of communication between clients and the development team, clearly defines the scope of the project, serves as a baseline for quality assurance, and identifies potential risks associated with the project.

2.1.1 Purpose

This Sorting Visualizer aims to make it easier to understand and allow them to Visualize and compare different sorting algorithms.
The aim is to assist users in understanding the efficiency of different sorting algorithms via a visual representation of how they work. It allows users to choose from a pool of various sorting algorithms, such as bubble sort, insertion sort, quick sort, and heap sort, and see how differently these algorithms sort an array of data.
Overall, iSORT aims to provide users with an easy and effective way to understand the behavior and versatility of different sorting algorithms and choose them accordingly in different situations for different types of data provided. It strives to be a practical and beneficial tool for learning.

2.1.2 Scope

ISORT will be a website that will be implemented and will allow users to choose from a collection of different sorting algorithms and visualize their work in a facile and operative manner.

The website will be easy to use with an intuitive user interface that will efficiently show the sorting process in different sorting algorithms.
It will not store any user data and will only show the sorting process.
The website will be accessible on any device with a steady internet connection.
The website should be efficient and smooth running and perform visualization effectively for users to understand algorithms better.

2.1.3 Overall Description

Users can search for the website on the search engine which will lead them to the home page of the website by clicking the website link where:

- Users will find several sorting algorithms such as Merge sort, bubble sort, quick sort, etc. to choose from the sidebar menu.
- Users can generate random numbers or can enter input values manually to be sorted and can adjust the speed of visualization to observe and understand the procedure of sorting even better.

- There are bars of different lengths shown on the screen representing the set of values provided which will be sorted when the ‘sort’ button will be clicked. The bars are provided with functionality that the selected bars (i.e. the bars getting compared) are of different colours and of different colours after getting sorted from the original colour of bars to make the visualization process easy to observe and understand.
- Users can see the line of code in the process of sorting on the right side of the same page to help visualize the sorting behaviour and understand which line of code is working at one particular time which makes it easy to understand the algorithm along with its sorting behaviour.
- The website is developed using modern web technologies, including HTML, CSS and JavaScript, it is compatible with a wide range of desktop and mobile devices. It is easy to use and access.
- Users can provide feedback on their experience by clicking on the feedback button present at the footer of the page which can then be used to improve or add new functionalities and effectiveness of the visualization process.

2.1.4 User Interface

- ·The first is the home screen where the user can read about sorting algorithms.
- ·Option for choosing sorting algorithm from the sidebar menu
- ·Bars for visualization and code on the right side on the same page.
- ·About the chosen algorithm (concept, complexity, stability, etc.) on the same page under the visualization area.

2.1.5 Environment Used

VS CODE – IDE used for making the website.
Programming languages used -

- Hyper Text Markup Language (**HTML**) for creating user interface objects, and is the basic building block of the web.
- Cascading Style Sheets (**CSS**) for styling and beautifying the HTML document.
- **JavaScript** for user interaction such as taking user input, buttons, speed control, etc. It is a text-based programming language and makes our web pages interactive.

2.2 Use Case Approach

A use case in software and systems engineering is a list of actions or event steps, typically defining the interactions between a role (known in the Unified Modelling Language as an actor) and a system to achieve a goal. The actor can be a human, an external system, or a time. In systems engineering, use cases are used higher than in software engineering, often representing missions or stakeholder goals. Another use case that describes how a real-world actor interacts with the system is another way to look at it. In a system use case, we include high-level implementation decisions.

Planning of Use Case

- Brief - This Software has been developed to give users a real-time experience, to visualize the sorting of numbers with the help of mobile bars, colors, and effects by using different kinds of sorting algorithms.
- Actors -
    1. User or Customer.
    2. Coder or Manager or Engineer.

- The flow of Events -
    1. Basic Flow - Users will choose a sorting algorithm of their choice and a random array of integers will be generated by the system and then will get sorted and output will be returned to the user.
    2. Alternative Flow - Users can also input a random array of numbers of their choice, can also see the intermediate steps like swapping, choosing, and fading colors of bars.

- Special Requirements - Users can use any device like a smartphone, laptop, computer, etc. but should have a lag-free environment or stable internet/network strength to operate the website.

- Preconditions -
    1. Users/Customer/Students.
    2. Selection of Sorting Algorithms of their choice.

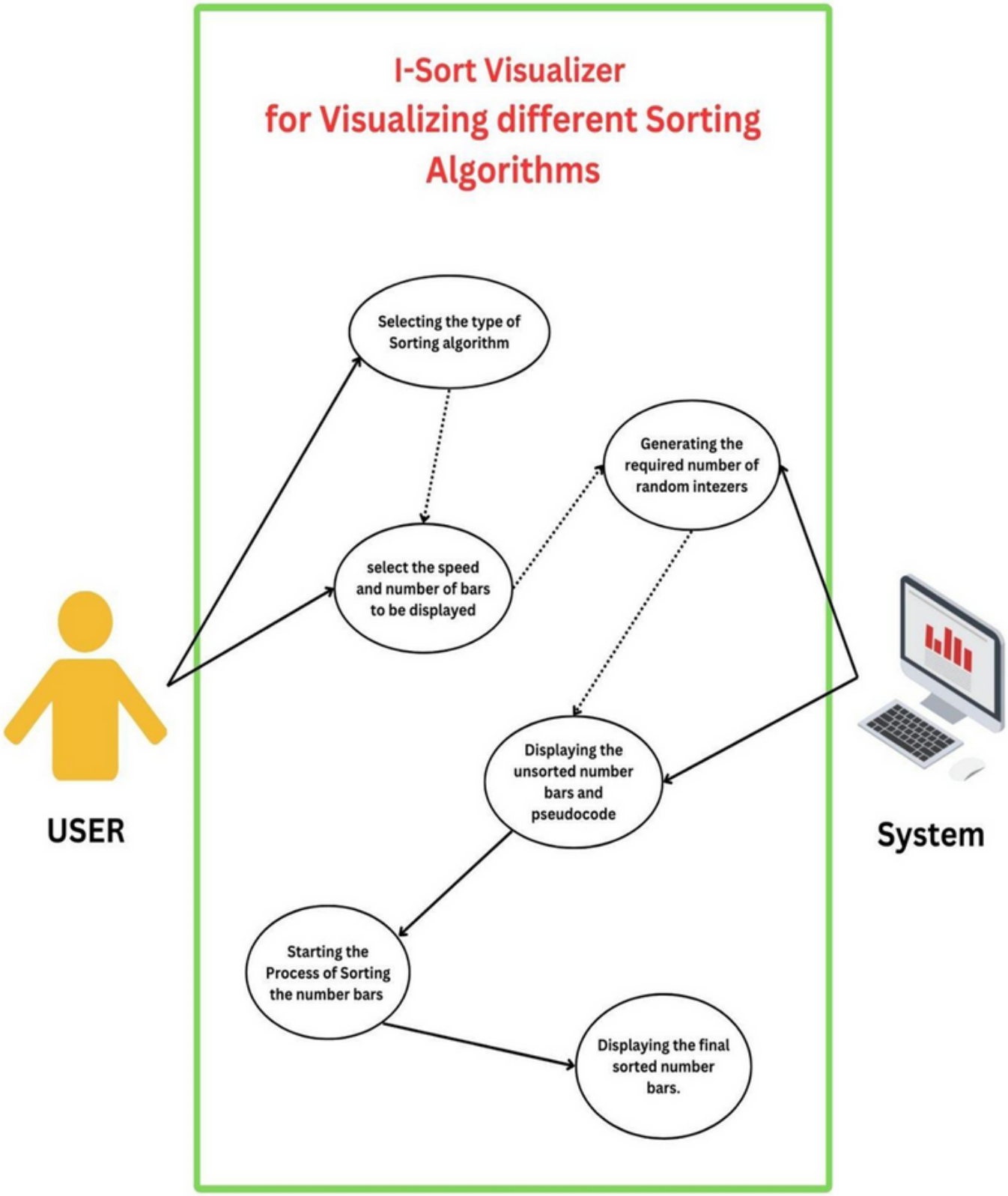
- Post Conditions - End the current running or executed sorting algorithm/process before starting a new one.

- Extension Points -
    1.1. Users will also be provided with the dry run of the currently running sorting algorithm.
    1.2. Users can change the speed of the sorting algorithm process(swapping) to fast , moderate and slow as per their requirements/choice.

2.3 Software and Hardware Requirements

Hardware Requirements - A laptop/Pc with a stable internet connection.

Software used while developing the application:
OS - Windows 11
Platform - VS Code
Programming Language - Javascript
User Interface - HTML, CSS, Javascript



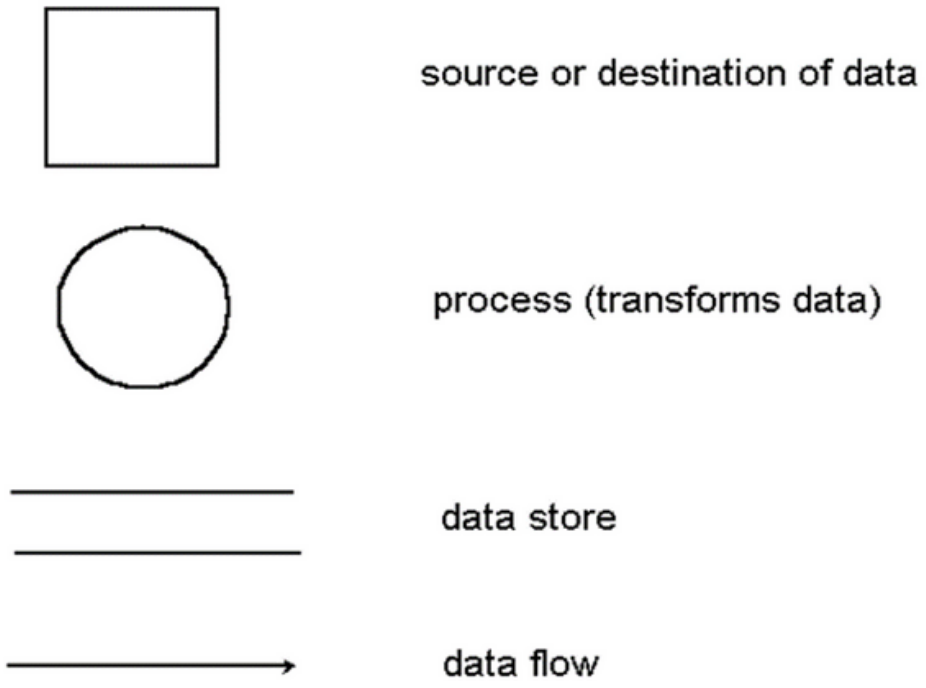
Use Case Diagram

2.4 Data Flow Diagram DFD

The data flow diagram shows the data flow from external entities into the system and from one process to another. It is a graphical representation of the flow of data through a system.

There are four symbols for drawing a DFD:

- Rectangles represent external entities, sources, or destinations of data.
- Ellipses represent processes, which take data input, validate and process it and output it.
- Arrows represent the data flow, either electronic data or physical items.
- Open-ended rectangles represent data stores, including electronic stores such as databases.



DFD Notations

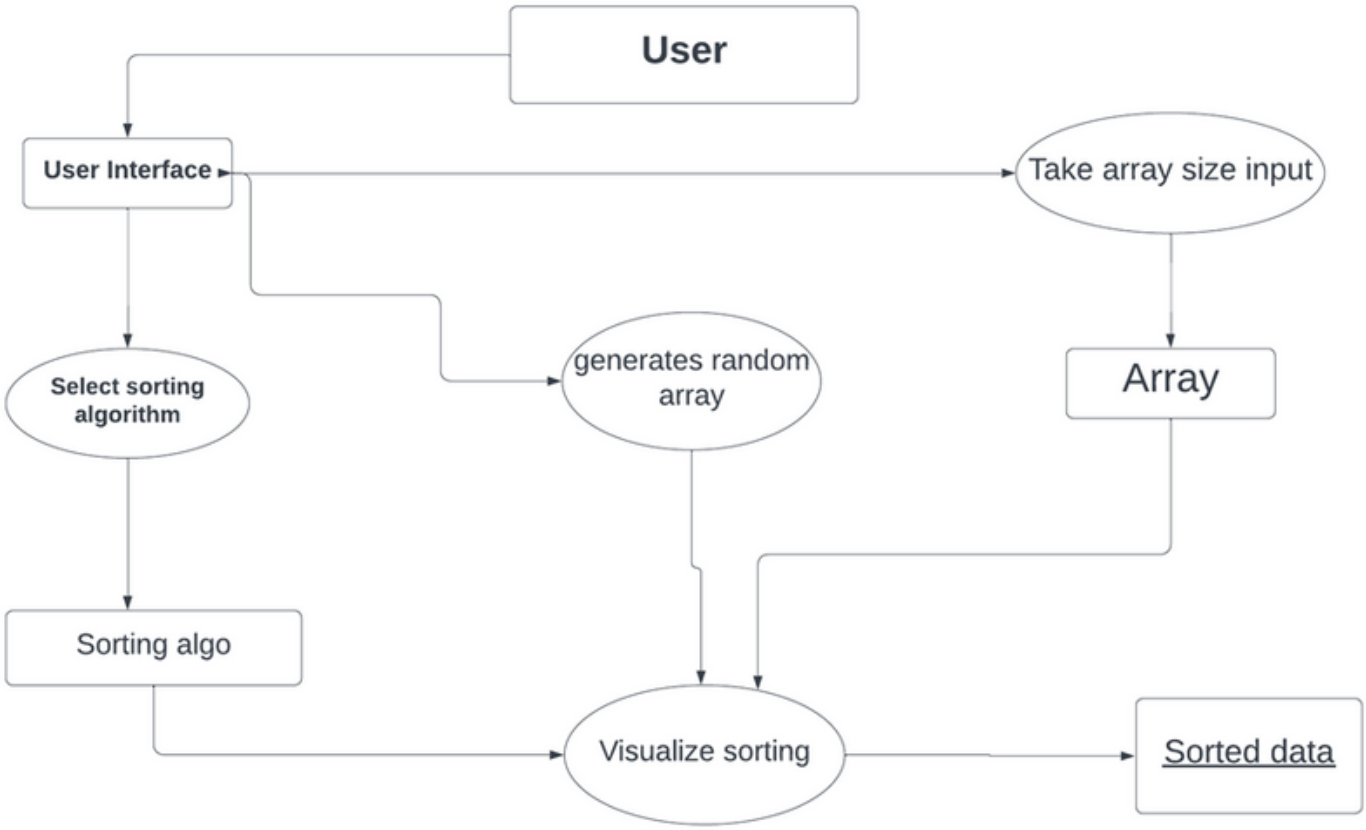
CONTEXT LEVEL DIAGRAM (LEVEL 0)

It is also known as the fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows.

Level-0 DFD

1-Level DFD

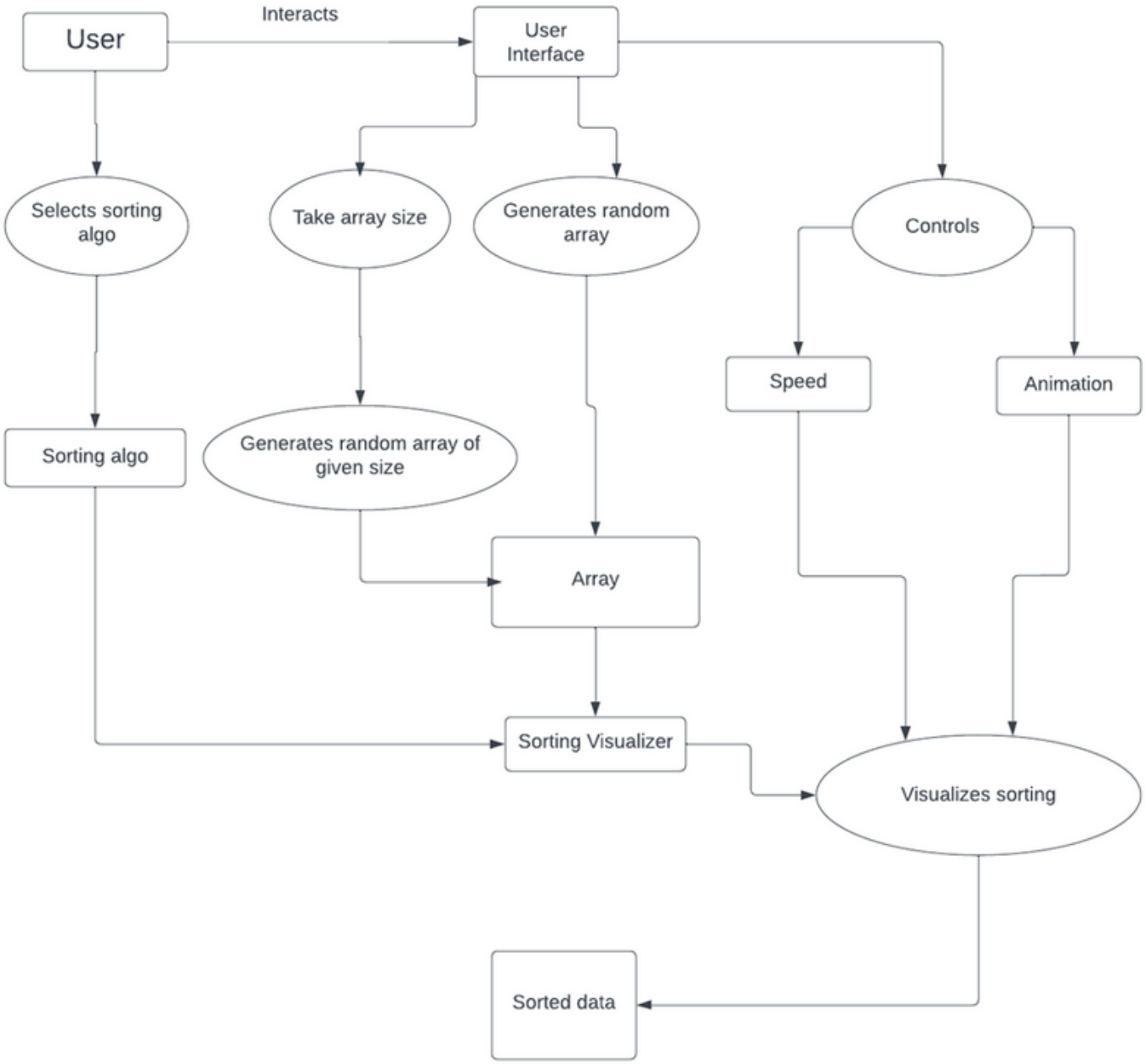
In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. At this level, we highlight the system's main objectives and break down the high-level process of 0-level DFD into subprocesses.



Level-1 DFD

2-Level DFD

2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary details about the system's functioning.



Level-2 DFD

2.5 Data Dictionary

A data dictionary collects the names, definitions, and attributes of data elements and models. The data in a data dictionary is the metadata about the database. These elements are then used as a database, research project, or information system.

The data dictionary contains information about the following –

- Names of all the database tables and their schemas.
- Details about all the tables in the database, such as their owners, their security constraints, and when they were created.
- Physical information about the tables, such as where they are stored and how.
- Table constraints such as primary key attributes, foreign key information, etc.
- Information about the visible database views.

Field name	Data type	Field length	constraint	Description
Sorting algo	String	Not defined	Primary key	Select sorting algoritm
speed	int	10	not null	Defines visualization speed
n	int	50	can be null	Defines user array size input
Array	int[]	50	not null	Array which is to be sorted and visualized

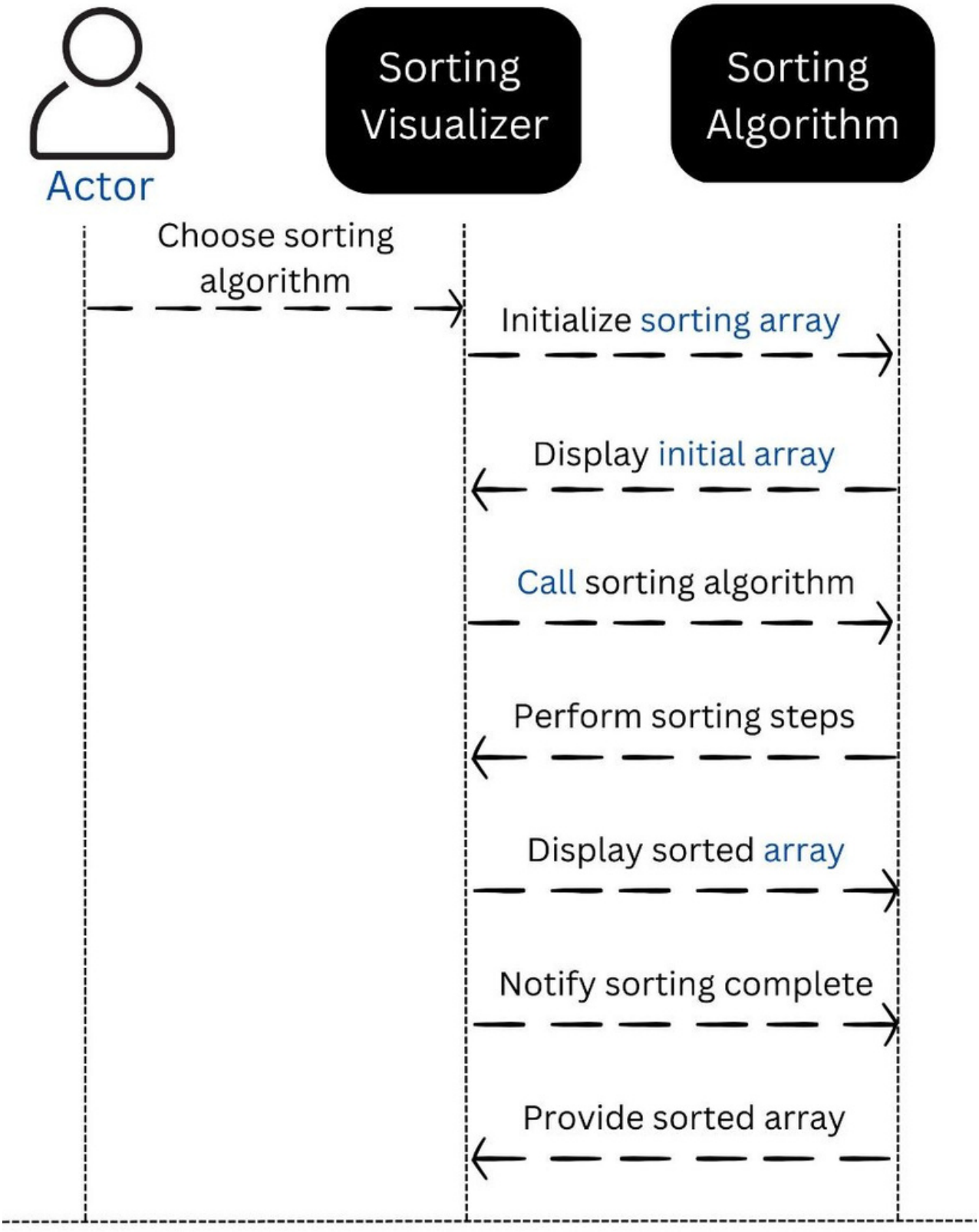
Data Dictionary

2.6 Sequence Diagram

The sequence diagram represents the flow of messages in the system and is also termed an event diagram.
It helps in envisioning several dynamic scenarios.
It portrays the communication between any two lifelines as time-ordered sequences of events, such that these lifelines took part in the run time.

Purpose of a Sequence Diagram

- To model high-level interaction among active objects within a system.
- To model the interaction among objects inside a collaboration realizing a use case.
- It either models generic interactions or specific instances of interaction.



Sequence Diagram

Project Management

Software Project Management (SPM) involves the planning, monitoring, and control of the people, processes, and events that occur as software evolves from a preliminary concept to an operational implementation.

It is necessary for an organization to deliver quality products. therefore, software project management is needed to incorporate user requirements along with budget and time constraints.

3.1 Functional Point Estimation

Questions	**VAFS**
Does the system require reliable backup and recovery?	1
Are specialized data communications required to transfer information to or from the application?	0
Are there distributed processing functions?	0

Is performance critical?	5
Will the system run in an existing, heavily utilized operational environment?	2
Does the system require on-line data entry?	0
Does the on-line data entry require the input transaction to be built over multiple screens or operations?	0
Are the ILFs updated online?	0
Are the inputs, outputs, files, or inquiries complex?	3
Is the internal processing complex?	3
Is the code designed to be reusable?	4
Are conversions and installations included in the design?	0

Is the system designed for multiple installations in different organizations?	0
Is the system designed for multiple installations in different organizations?	5

Total Degree of Influence

Value Adjustment Factor = ∑fi = 23
Now, the Complexity Adjustment Factor
(CAF) = 0.65 + (0.01 * ∑fi)
= 0.65 + (0.01 * 23)
= 0.65 + 0.23
= 0.88

Functional Units		Weighing Factor		
	Count	Simple	Average	Complex
External Input	10	3	4	6
External Output	3	4	5	7
External Enquiries	0	3	4	6
Internal Logical Files	0	7	10	15
External Interface Files	30	5	7	10

Unadjusted Function Point
(UFP) = 10*4 + 3*5 + 0 + 0 + 30*5
= 40 + 15 + 150
= 205

Function Point = UFP * CAF
= 205 * 0.88
= 180.4 ~ 180

3.2 COCOMO II Model

Barry Boehm introduced a hierarchy of software estimation models bearing the name COCOMO, for COnstructive COst MOdel. The original COCOMO model became one of the most widely used and discussed software cost estimation models in history. It has evolved into a more comprehensive estimation model, called COCOMO II.

Like all estimation models for software, the COCOMO II models require sizing information. The COCOMO II application composition model uses object points.

Once complexity is determined, the number of screens, reports, mathematical computations, and statistical computations are weighted. The object point count is then determined by multiplying the original number of object instances by the weighting factor and summing to obtain a total object point count. When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the point count is adjusted.

$$\boldsymbol{NOP = (object\ points) \times [\frac{100 - \%reuse}{100}]}$$

Where, NOP = New Object Points

3.3 Effort Estimation

Object Type	No. Of Objects	Complexity Weight			Count
		Simple	Medium	Difficult	
Screen	12	2(1)	10(2)	0	22
Report	1	1(5)	0	0	5
3GL Component	1	0	0	1(10)	10
			Total Object Points:		**37**

Data used in estimating effort are:
(1)Object point is 37 (taken from the above table)
(2)Estimated reuse is 55%.
(3)Prod is 13 (average value is taken)

NOP = Object points *[(100-reuse%)/100]
= 37 * [(100-55)/100]
= 16.65
= 16
Estimated Effort = NOP/PROD
= 16/13
= 1.23
≈ 1 person/month

Hence the estimated effort of the project
is 1 person/month.

3.4 Cost Estimation

To complete the project in 3 months, we needed 4 people. Let's say the labor rate is ₹ 5,000 per person per month.

Total Efforts = Efforts * (Number of people) * Months
= 1*4*3
= 12 person-months

Cost to be paid to 4 people for 3 months = 12*5,000
= ₹ 60,000
Therefore, Project Costing = ₹ 60,000

3.5 Risk Management

Risk is the possibility of suffering loss, and total risk exposure to a specific project will account for both the probability and the size of the potential loss. Identifying and aggregating risks is the only predictive method for capturing the probability that a software development project will experience unplanned or inadmissible events. These include terminations, discontinuities, schedule delays, cost underestimation, and overrun of project resources.

Impact Values -
1) Catastrophic
2) Critical
3) Marginal
4) Negligible

RISKS	CATEGORY	PROBABILITY	IMPACT
WEBSITE CRASH	TECHNICAL RISK	10%	1
LESS FRIENDLY USER INTERFACE	PROJECT RISK	20%	3
REQUIREMENTS CHANGES	PROJECT RISK	30%	3
LACK OF SKILL	PROJECT RISK	20%	2

RMMM Plan (Risk Mitigation, Monitoring, Management)

- The goal of the risk mitigation, monitoring, and management plan is to identify as many potential risks as possible. The project will then be analyzed to determine any project-specific risks.

- When all risks have been identified, they will then be evaluated to determine their probability of occurrence. Plans will then be made to avoid each risk, to track each risk to determine if it is more or less likely to occur, and to plan for those risks should they occur.

- It is the organization's responsibility to perform risk mitigation, monitoring, and management in order to produce a quality product. The quicker the risks can be identified and avoided, the smaller the chances of having to face that particular risk's consequence. The better the product, the smoother the development process.

- Risk Analysis and Management are a series of steps that help a software team to understand and manage uncertainty.

- Many problems can plague a software project. A risk is a potential problem- it might happen or it might not. But regardless of the outcome, it's a really good idea to identify.

- It assesses its probability of occurrence, estimates its impact, and establishes a contingency plan should the problem actually occur.

- Software is a difficult undertaking. A lot of things can go wrong, and frankly many often do.

Risk Management

- It's for this reason that being prepared understanding the risks and taking proactive measures to avoid or manage them – is a key element of good software project management. Recognizing what can go wrong is the first step called "Risk Identification".

- Next, each risk is analyzed to determine the likelihood that it will occur and the damage that it will do if it does occur. Once this information is established, risks are ranked, by probability and impact.

- Finally, a plan is developed to manage those risks with high probability and high impact. The work product is the “Risk Mitigation, Monitoring and Management (RMMM) Plan” or a set of risk information sheets is produced.

RISK 1: WEBSITE CRASH

- MITIGATION

The cost associated with website crashes resulting in a loss of data is crucial. A website crash itself is not crucial, but rather the loss of data. A loss of data will result in not being able to deliver the product to the customer. This will result in not receiving a letter of acceptance from the customer. Without the letter of acceptance, the group will receive a failing grade for the course. As a result, the organization is taking steps to make multiple backup copies of the software in development and all documentation associated with it, in multiple locations.

- MANAGEMENT

The lack of a stable environment is extremely hazardous to a software development team. In the event that the website environment is found unstable, the development team should cease work on that system until the environment is made stable again or should move to a system that is stable and continue working there.

RISK 2: LESS FRIENDLY INTERFACE

- MITIGATION

The website in its very initial stages can be a little bit less friendly to the user. This can decrease the interest of the user and belief to use the website. As we all know the major requirement of the website is making it as simple and easy to handle as possible. If this issue is avoided it might lead the user to non-acceptance of the app and thus it would not be a happy gesture for the organization.

- MONITORING

Like all the issues the team is also involving major time and work in making the interface easy at their best.

- MANAGEMENT

To make a good interface the team is examining and using the interface from a user's point of view. And they will resolve the features as much as possible. But still, if the user gets some small issues with the interface the team will resolve and satisfy the consumer.

RISK 3: REQUIREMENT CHANGES

- MITIGATION

In order to prevent this from happening, meetings (formal and informal) will be held with the customer on a routine business. This ensures that the product we are producing, and the requirements of the customer are equivalent.

- MONITORING

The meetings with the customer should ensure that the customer and our organization understand each other and the requirements for the product.

- MANAGEMENT

Should the development team come to the realization that their idea of the product requirements differs from those of the customer, the customer should be immediately notified and whatever steps necessary to rectify this problem should be taken. Preferably a meeting should be held between the development team and the customer to discuss at length this issue.

RISK 4: LACK OF SKILL

- MITIGATION

Select the most talented and experienced member to join the Project Team to avoid this scenario.

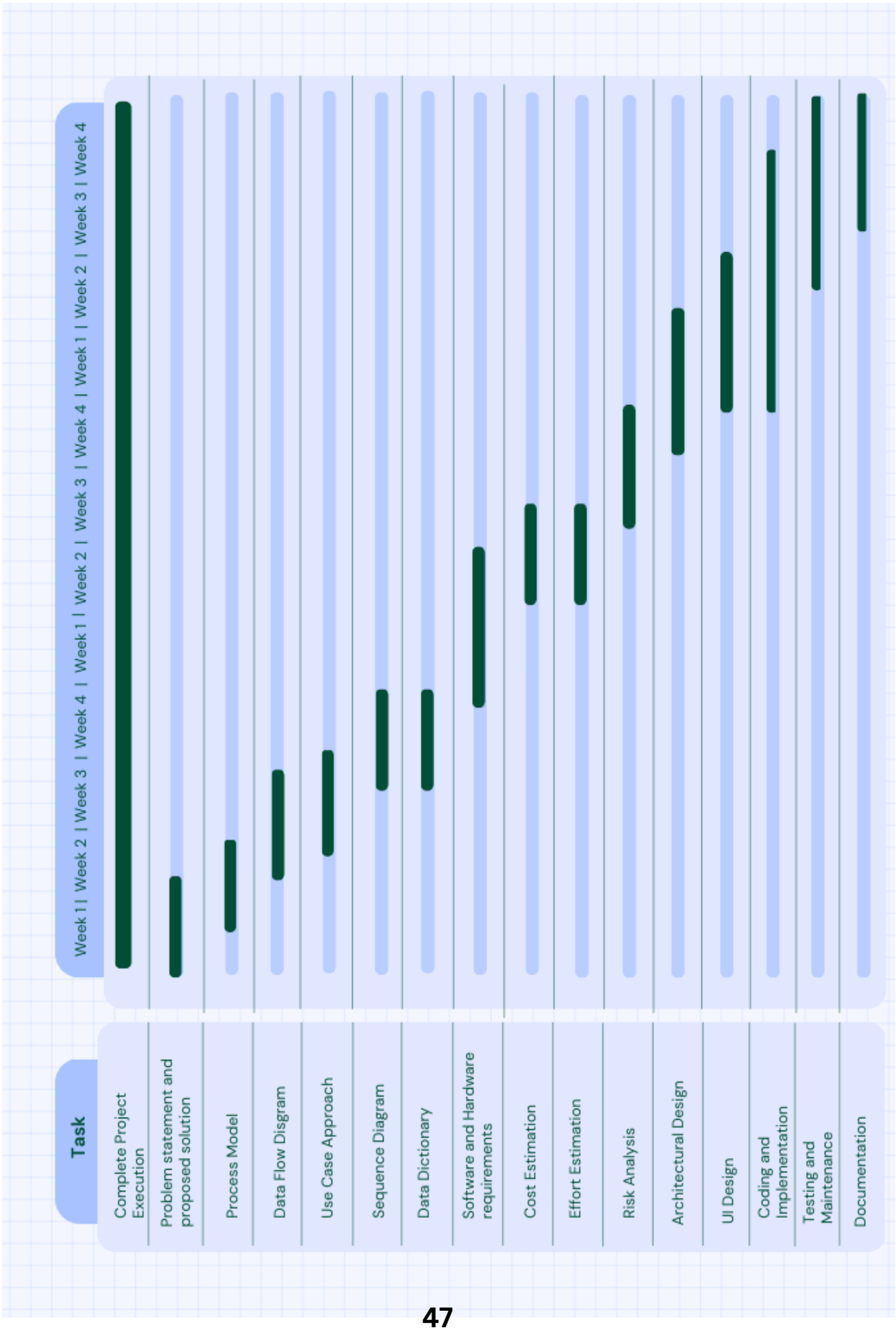
- MONITORING

Talk to the client about the project extension and apologize to him.

- MANAGEMENT

External resources would help and will try to include dynamic members to be part of the project.

3.6 TimeLine Chart

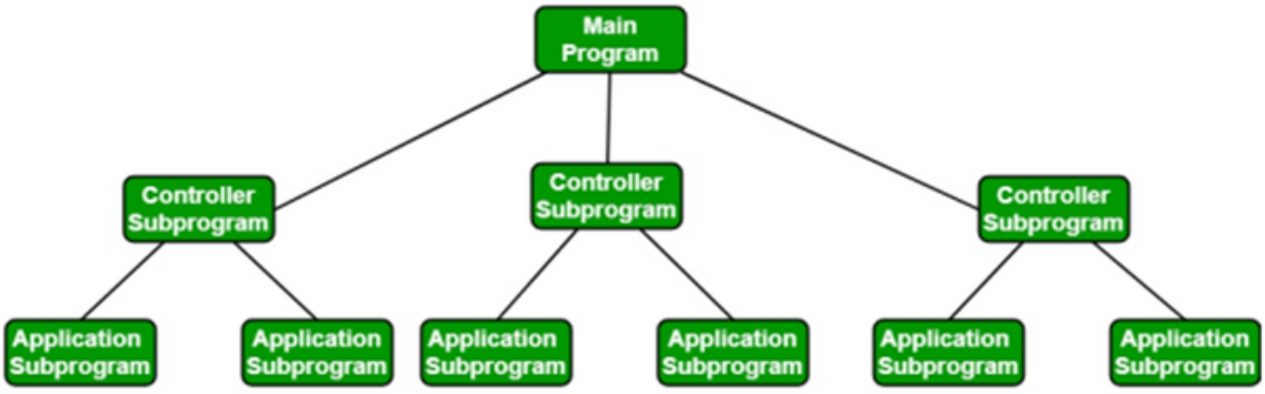


Design Engineering

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.

The software design process can be divided into the following three levels of phases of design :

1. Interface Design
2. Architectural Design
3. Detailed Design



DESIGN ENGINEERING

4.1 Architectural Design

Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored.
The architectural design adds important details ignored during the interface design. The design of the internals of the major components is ignored until the last phase of the design.

4.2 Pseudocode Explanation for Code

- **Module for Generation of Bars**

Function generatebars(num)

```
For int i from 0 to n
    Select a random value v from 9 to 180
    Create a div bar
    Add bar div to CSS class ‘bar’
    Set div height as v/2
    Create a bar_label with value v
    Add label to CSS class “bar_id”
```

```
If(num>80)
   Don’t display bar_label
End if
   If(num<=40)
      If num<=10
         Set bar_label size as XXXL
      End if
      If num<=20
         Set bar_label size as XXL
      End if
      If num >20 && num<=30
         Set bar_label size as XL
      End if
      If num>30
         Set bar_label size as L
      End if
   End if
   Add bar_label to the bar
   Add bar to the container
End for
```

Function generate()

```
Var num = take number of bars input from user
If(num>400)
    Throw an Alert upper bound is 400 bars.
    n.value = 400;
    generate();
end if
else
    generatebars(num)
```

- **Module for setting speed**

Function Delayset()

```
Delay = 5000
Var s = get element by id range
Var v = s.value;
Delay = Delay/v;
```

- **Module for Bubble Sort**

```
function BUBBLESORT(ARRAY)
  loop IDX from 0 to size of ARRAY – 1
    loop IDX2 from 0 to size of ARRAY – 2 – IDX
      if ARRAY[IDX2] > ARRAY[IDX2 + 1] then
        # swap elements if they are out of order
        TEMP = ARRAY[IDX2]
        ARRAY[IDX2] = ARRAY[IDX2 + 1]
        ARRAY[IDX2 + 1] = TEMP
      end if
    end loop
  end loop
end function
```

- **Module for Selection Sort**

```
function SELECTIONSORT(ARRAY)
   loop IDX from 0 to size of ARRAY – 2
  MINIDX = 0
     loop IDX2 from IDX to size of ARRAY – 1
       if ARRAY[IDX2] < ARRAY[MINIDX] then
         MINIDX = IDX
     TEMP = ARRAY[MINIDX]
     ARRAY[MINIDX] = ARRAY[IDX]
     ARRAY[IDX] = TEMP
```

- **Module for Insertion Sort**

```
function INSERTIONSORT(A)
    i ← 1
    while i < length(A)
        j ← I
        while j > 0 and A[j-1] > A[j]
            swap A[j] and A[j-1]
            j ← j - 1
        end while
        i ← i + 1
    end while
```

- **Module for Quick Sort**

```
function QUICKSORT(A, START, END)
    if START >= END then
        return
    PIVOTINDEX = PARTITION(A, START, END)
    QUICKSORT(A, START, PIVOTINDEX – 1)
    QUICKSORT(A, PIVOTINDEX + 1, END)
end function
```

```
function PARTITION(A, START, END)
    PIVOTVALUE = A[END]
    PIVOTINDEX = START
    loop INDEX from START to END
        if ARRAY[INDEX] <= PIVOTVALUE
            TEMP = A[INDEX]
            A[INDEX] = A[PIVOTINDEX]
            A[PIVOTINDEX] = TEMP
            PIVOTINDEX = PIVOTINDEX + 1
        end if
    end loop
    return PIVOTINDEX – 1
```

- **Module for Randomized Quick Sort**

```
function RANDOMQUICKSORT(A, S, E)
     if S >= E then
         return
    PIVOTINDEX = RANDOMPARTITION(A, S, E)
    RANDOMQUICKSORT(A, S, PIVOTINDEX – 1)
    RANDOMQUICKSORT(A, PIVOTINDEX + 1, E)
end function
```

```
function RANDOMPARTITION(A, START, END)
    PIVOTVALUE = Random value in [Start,End]
    PIVOTINDEX = START
    loop INDEX from START to END
        if ARRAY[INDEX] <= PIVOTVALUE
            TEMP = A[INDEX]
            A[INDEX] = A[PIVOTINDEX]
            A[PIVOTINDEX] = TEMP
            PIVOTINDEX = PIVOTINDEX + 1
        end if
    end loop
    return PIVOTINDEX – 1
```

- **Module for Merge Sort**

```
function MERGESORT(A, S, E)
    if E - S + 1 == 1 then    return
    if E - S + 1 == 2 then
        if A[S] > A[E] then
            TEMP = A[S]
            A[S] = A[E]
            A[E] = TEMP
        return
    HALF = int((S + E) / 2)
    MERGESORT(A, S, HALF)
    MERGESORT(A, HALF + 1, E)
    MERGE(A, S, HALF, E)
```

```
function MERGE(A, START, HALF, END)
    TEMPARRAY = new array[END – START + 1]
    INDEX1 = START
    INDEX2 = HALF + 1
    NEWINDEX = 0
    while INDEX1 <= HALF and INDEX2 <= END
        if A[INDEX1] < A[INDEX2] then
            TEMPARRAY[NEWINDEX] = A[INDEX1]
            INDEX1 = INDEX1 + 1
        else
            TEMPARRAY[NEWINDEX] = A[INDEX2]
            INDEX2 = INDEX2 + 1
        NEWINDEX = NEWINDEX + 1
    while INDEX1 <= HALF
        TEMPARRAY[NEWINDEX] = A[INDEX1]
        INDEX1 = INDEX1 + 1
        NEWINDEX = NEWINDEX + 1
    loop while INDEX2 <= END
        TEMPARRAY[NEWINDEX] = A[INDEX2]
        INDEX2 = INDEX2 + 1
        NEWINDEX = NEWINDEX + 1
    end loop
    loop INDEX from 0 to size of TEMPARRAY – 1
        A[START + INDEX] = TEMPARRAY[INDEX]
    end loop
end function
```

- **Module for Heap Sort**

Heapify(A as array, n as int, i as int)

```
    max = i
    leftchild = 2i + 1
    rightchild = 2i + 2

    if (leftchild <= n) and (A[i] < A[leftchild])
        max = leftchild
    else
        max = i
    if (rightchild <= n) and (A[max]  > A[rightchild])
        max = rightchild
    if (max != i)
        swap(A[i], A[max])
        Heapify(A, n, max)
```

Heapsort(A as array)

```
{
   n = length(A)
   for i = n/2 downto 1
     Heapify(A, n ,i)
   for i = n downto 2
     exchange A[1] with A[i]
     A.heapsize = A.heapsize - 1
     Heapify(A, i, 0)
```

- **Module for Counting Sort**

countingSort(array, size)

```
max <- find largest element in array
initialize count array with all zeros
for j <- 0 to size
    find the total count of each unique element and
    store the count at jth index in count array
for i <- 1 to max
    find the cumulative sum and store it in count array
itself
for j <- size down to 1
    restore the elements to array
    decrease count of each element restored by 1
```

- **Module for Gnome Sort**

gnomeSort(a[]):

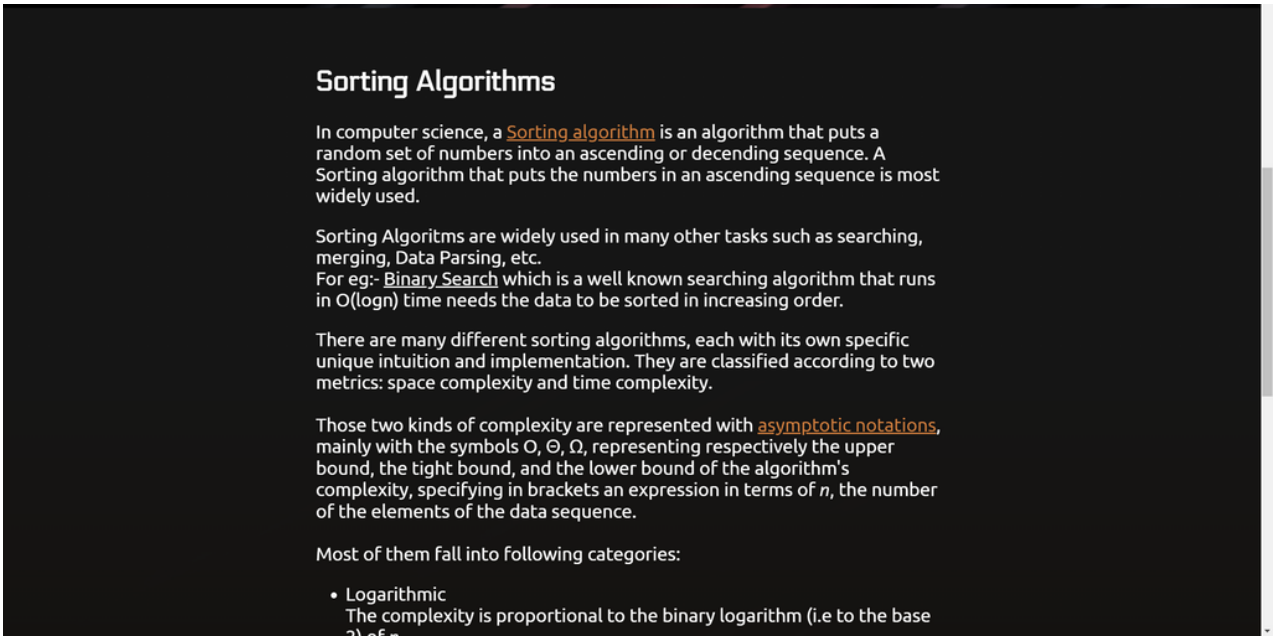
```
pos := 0
while pos < length(a):
  if (pos == 0 or a[pos] >= a[pos-1]):
     pos := pos + 1
  else:
     swap a[pos] and a[pos-1]
     pos := pos - 1
```

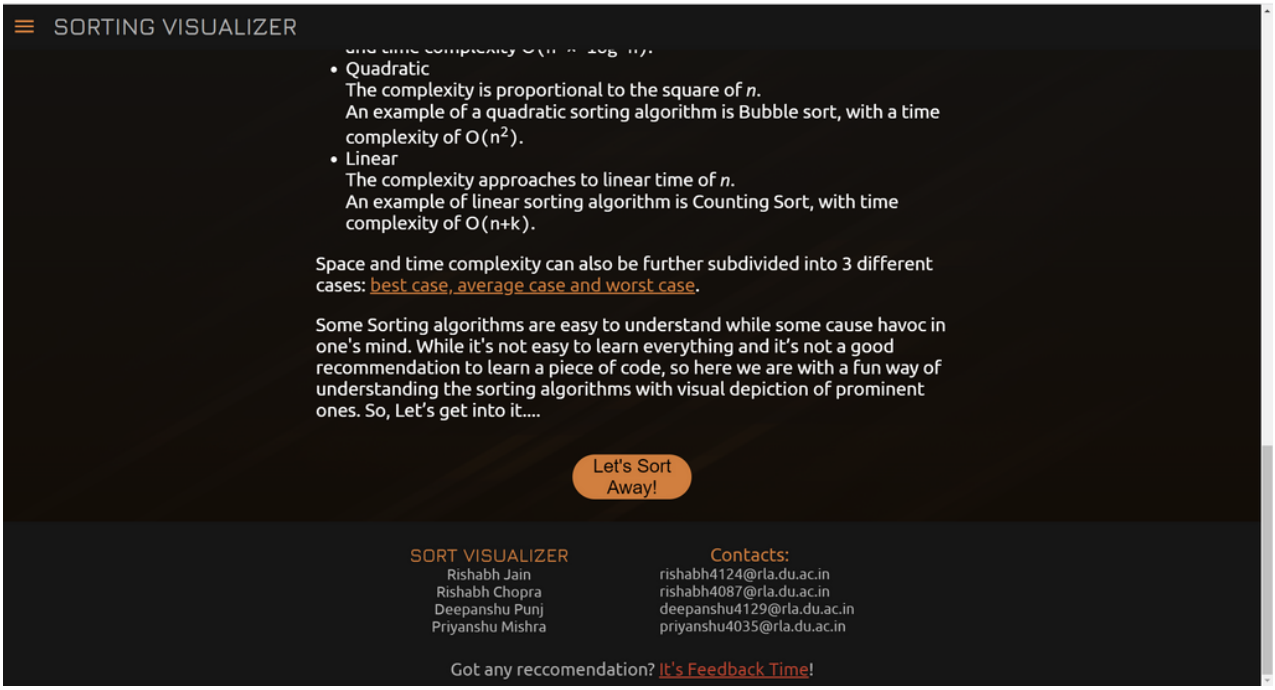
- **Module for Stooge Sort**

```
StoogeSort(A, i, j)
    if A[i]>A[j]
        then swap A[i] and A[j]
    if i+1>=j
        then return
    k=[(j-i+1)/3]
    Stoogesort(A, i, j-k)
    Stoogesort(A, i-k, j)
    Stoogesort(A, i, j-k)
```

4.3 UI Design

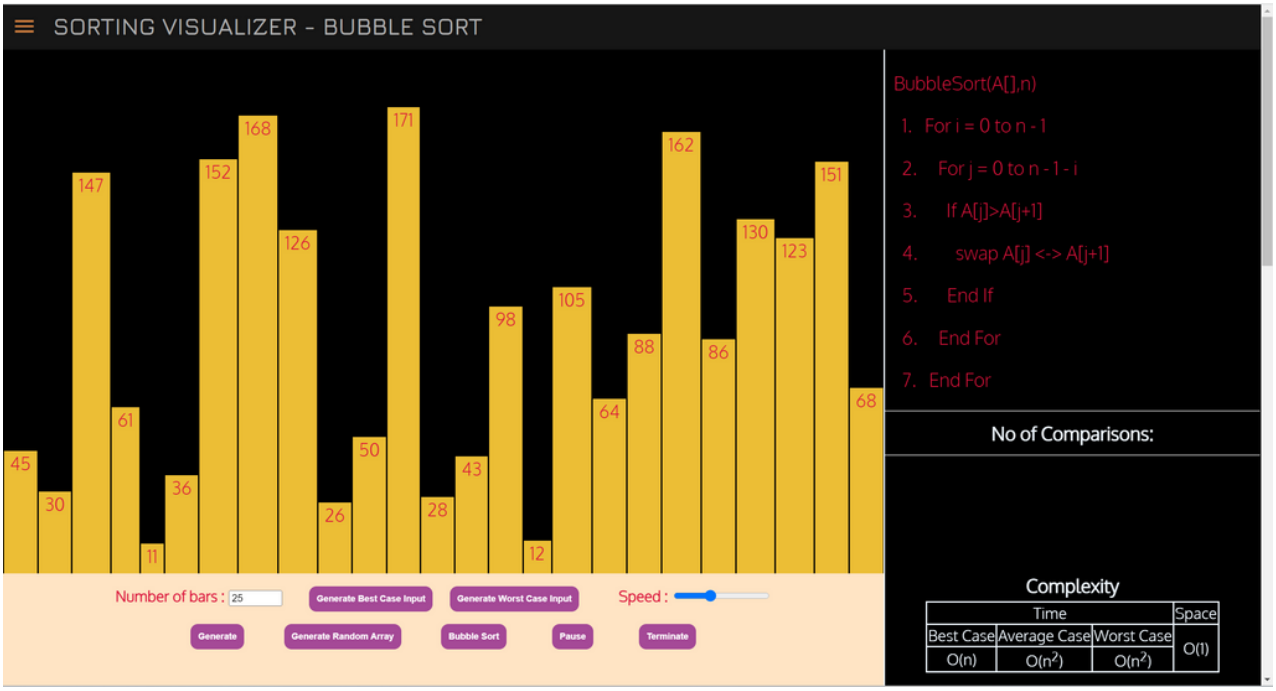
4.3.1 Start Screen



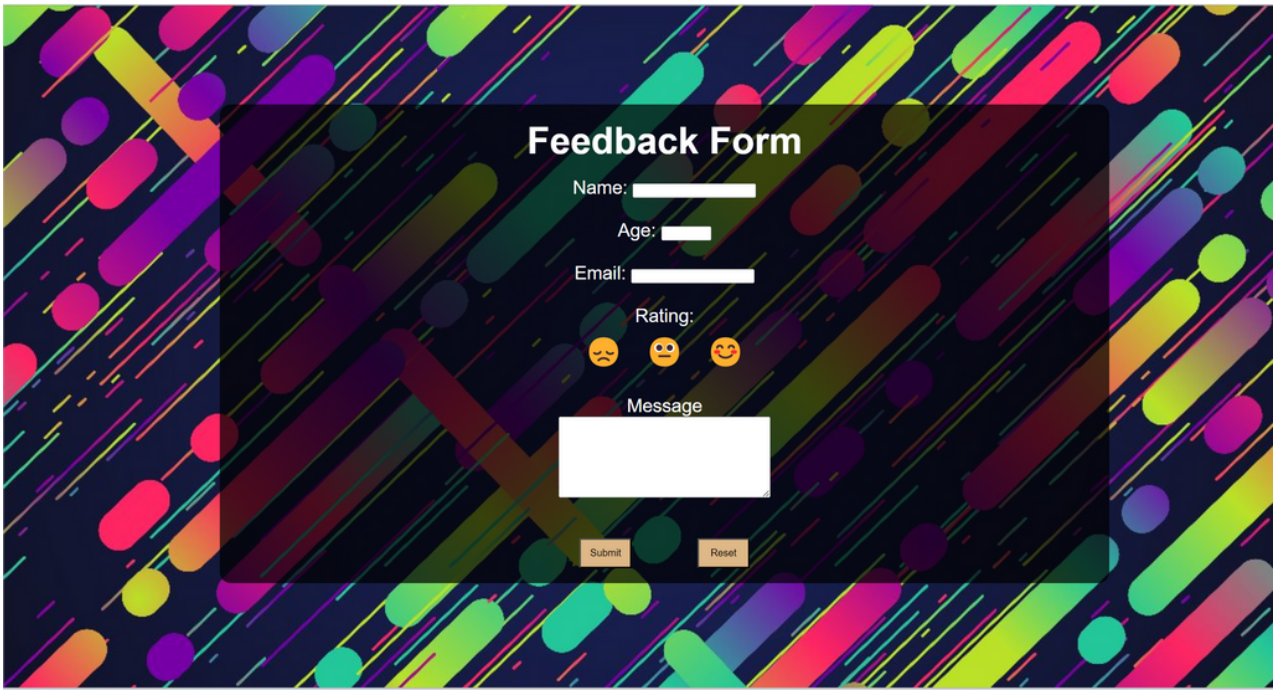


4.3.2 Selecting Sorting Algo

4.3.3Sorting Algo UI



4.3.4FeedBack Form



Implementation

5.1.1 index.html

```
<!DOCTYPE html>
<html

 <head>
  <title>iSort - The Sort Visualizer </title>

      <meta name="viewport" content="width=device-width, initial-scale=1.0">
                  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <meta name="language" content="English">

  <link href="css/SideNavIcon.css" rel="stylesheet">
  <link href="css/font.css" rel="stylesheet">
      <link rel= "stylesheet" type= "text/css" href= "css/style1.css">
      <link rel= "stylesheet" type= "text/css" href= "css/style2.css">
      <link rel= "stylesheet" type= "text/css" href= "css/style.css">
  <link rel="icon" href="favicon.ico" type="image/x-icon">
 </head>
```

```
<body>
                                                    <script
src="https://unpkg.com/aos@2.3.1/dist/aos.js">
</script>
                        <script        type="text/javascript"
src="javascript/index.js"></script>
   <div class="sidenav" id="sidenav">
          <a  href="index.html"  class="sidenav-title  no-
remove"><i    class="material-icons    icon">home</i>
Home</a>
    <div id="sep" class="no-remove"></div>
              <div   class="sidenav-title   no-remove"><i
class="material-icons icon">sort</i> Sorts</div>
        <a  href="sorts/bubbleSort.html"  class="sidenav-
element no-remove">Bubble Sort</a>
       <a href="sorts/selectionSort.html" class="sidenav-
element no-remove">Selection Sort</a>
       <a href="sorts/insertionSort.html" class="sidenav-
element no-remove">Insertion Sort</a>
          <a  href="sorts/quickSort.html"  class="sidenav-
element no-remove">Quick Sort</a>
              <a   href="sorts/randomizedQuickSort.html"
class="sidenav-element          no-remove">Randomized
Quick</a>
```

```
<a href="sorts/mergeSort.html" class="sidenav-element no-remove">Merge Sort</a>
 <a href="sorts/heapSort.html" class="sidenav-element no-remove">Heap Sort</a>
  <a href="sorts/countingSort.html" class="sidenav-element no-remove">Counting Sort</a>
   <a href="sorts/gnomeSort.html" class="sidenav-element no-remove">Gnome Sort</a>
   <a href="sorts/stoogeSort.html" class="sidenav-element no-remove">Stooge Sort</a>
 </div>

 <div class="topnav">
  <button class="sidenav-btn topnav-element ripple open" id="sidenav-menu"><i class="material-icons icon">menu</i></button>
   <a href="/" class="topnav-element">SORTING VISUALIZER</a>
 </div>

<div id="header"><h1 class="header__title"><center><center>iSORT </center> The Witty Visualizer</center></h1></div>
```

```
<div id="cover">
    <div id="description-box">
        <h2 style="font-family: 'Electrolize', Courier, monospace;">
        Sorting Algorithms
      </h2>
      <div class="description-content">
        <p>
                    In computer science, a <a href="https://brilliant.org/wiki/sorting-algorithms/">Sorting algorithm</a> is an algorithm
            that puts a random set of numbers into an ascending or decending sequence.
        A Sorting algorithm that puts the numbers in an ascending sequence is most widely used.
      </p>
        <p>
            Sorting Algoritms are widely used in many other tasks such as searching, merging, Data Parsing, etc. <br>
         For eg:- <u>Binary Search</u> which is a well known searching algorithm that runs in O(logn) time needs the data to be sorted in
        increasing order.
        </p>
```

There are many different sorting algorithms, each with its own specific unique intuition and implementation.
They are classified according to two metrics: space complexity and time complexity.

Those two kinds of complexity are represented with asymptotic notations,
mainly with the symbols O, Θ, Ω, representing
respectively the upper bound, the tight bound, and the lower bound of the algorithm's complexity, specifying in brackets an expression in terms of
<code><var>n</var></code>, the number of the elements of the data sequence.

Most of them fall into following categories:

Logarithmic
The complexity is proportional to the binary logarithm (i.e to the base 2) of <code><var>n</var></code>.

An example of a logarithmic sorting algorithm is Quick sort, with space and time complexity O<code>(n × log n)</code>.

Quadratic
The complexity is proportional to the square of <code><var>n</var></code>.
An example of a quadratic sorting algorithm is Bubble sort, with a time complexity of O<code>(n²)</code>.

Linear
The complexity approaches to linear time of <code><var>n</var></code>.
An example of linear sorting algorithm is Counting Sort, with time complexity of O<code>(n+k)</code>.

<p>
Space and time complexity can also be further subdivided into 3 different cases: <a href="https://www.geeksforgeeks.org/worst-average-and-best-case-analysis-of-algorithms/">

```
best case, average case and worst case</a>.
 </p>

 <p>
 Some Sorting algorithms are easy to understand while some cause havoc in one's mind. While it's not easy to learn everything and it’s not a good recommendation
 to learn a piece of code, so here we are with a fun way of understanding the sorting algorithms with visual depiction of prominent ones. So, Let’s get into it....
 </p>
<button id="sort-btn" class="ripple-sort-btn open">Let's Sort Away!</button>
      </div>
    </div>
  </div>
<footer class="footer">

    <div class="footer-container">

            <div class="footer-content" style="margin-left:0;">
<span class="footer-title" style="font-family: 'Jura', Courier, monospace;">SORT VISUALIZER</span>
<br>
```

```
<center>
        Rishabh Jain<br>
          Rishabh Chopra<br>
        Deepanshu Punj<br>
             Priyanshu Mishra
      </center>
       </div>

       <div class="footer-content">

              <center><span  class="footer-title">Contacts:
</span></center>
        rishabh4124@rla.du.ac.in<br>
             rishabh4087@rla.du.ac.in<br>
       deepanshu4129@rla.du.ac.in<br>
             priyanshu4035@rla.du.ac.in
       </div>
      </div>
      <div class="credits">
                              Got  any  reccomendation?  <a
href="Feedback/feedbackmain.html">It's      Feedback
Time</a>!
      </div>
    </footer>
  </body>
</html>
```

5.1.2 feedbackform.html

```
<!DOCTYPE html>
<html>
<head>
 <title>
 Feedback Form
 </title>
 <link rel="stylesheet" href="feedbackcss.css">
 <script>
  function validateForm() {
        var name = document.forms["feedbackForm"]
["name"].value;
         var age = document.forms["feedbackForm"]
["name"].value;
        var email = document.forms["feedbackForm"]
["email"].value;
       var message = document.forms["feedbackForm"]
["message"].value;
       var regEmail = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*
(\.\w{2,3})+$/g;  //Javascript reGex for Email
Validation.
      var regName = /\d+$/g;                  // Javascript
reGex for Name validation
    if (name == "" || regName.test(name)) {
      alert("Name must be filled out");
      return false;
    }
```

```
if (age == "") {
        alert("Name must be filled out correctly");
        return false;
      }

      if (email == "" || !regEmail.test(email)) {
        alert("Email must be filled out correctly");
        return false;
      }

      if (message == "") {
        alert("Message must be filled out");
        return false;
      }

      return true;
    }
  </script>
</head>
<body      background="bg.jpg"      style="background-
repeat:no-repeat; background-size:100% 100vh;">
  <center>
    <div id="div1" style="background-color: rgba(0, 0, 0,
0.7)">
```

```
<h1 font-size : 70px style="margin:20px;">Feedback Form</h1>
                        <form name="feedbackform" action="mailto:deepanshu4129@rla.du.ac.in; priyanshu4035@rla.du.ac.in; rishabh4124@rla.du.ac.in; rishabh4087@rla.du.ac.in" method="post"
                        onsubmit="return validateForm()" enctype="text/plain">

    <label for="name">Name:</label>
        <input type="text" id="name" name="name" required><br><br>

    <label for"age">Age:</label>
        <input type="number" min="15" max="100" name="Age" required><br><br>

    <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br><br>
    <label for="" text-align:left>Rating:</label>
    <div id="response">
        <input type="radio" id="sad" name="response" value="sad" class="radio" />
                        <label class="emoji emoji1" for="sad">&#128542</label>
```

```
<input type="radio" name="response" id="average" value="average" class="radio" />
            <label class="emoji emoji2" for="average">&#128528</label>

  <input type="radio" name="response" id="happy" value="happy" class="radio" />
     <label class="emoji emoji3" for="happy">&#128522</label>
 </div>
<br>
                                    <label id="label1" for="message">Message</label> <br>
       <textarea id="message" name="message" rows="7" cols="40"></textarea><br><br>

                  <input type="submit" value="Submit" class="button">
            
                      <input type="reset" value="Reset" class="button">
     </form>
   </div>
  </center>
</body>
</html>
```

5.1.3 bubbleSort.js

```
const container = document.querySelector(".vrsplit1");

let isPlaying = false;
const pausePlayBtn = document.getElementById("pauseButton");
let terminate = false;
var count = 0;
const compare = document.getElementById("comp");

function generatebars(num) {
  container.innerHTML=""
  for (let i = 0; i < num; i += 1) {
      const value = Math.floor(Math.random() * (180-9)+9) + 1;
    const bar = document.createElement("div");
    bar.classList.add("bar");
    bar.style.height = `${value/2}%`;
    const barLabel = document.createElement("label");
    barLabel.classList.add("bar__id");
    barLabel.innerHTML = value;
    if (num>80) {
      barLabel.style.display='none';
    }
```

```
if (num<=40) {
      if (num<=10) {
        barLabel.style.fontSize = 'xxx-large';
      }
      else if (num<=20) {
        barLabel.style.fontSize = 'xx-large';
      }
      if (num>20 && num<=30) {
        barLabel.style.fontSize = 'x-large';
      }
      else if (num<=40) {
        barLabel.style.fontSize = 'large';
      }
    }
    bar.appendChild(barLabel);
    container.appendChild(bar);
  }
}
generatebars(25);
function generate() {
  var n = document.getElementById("nele");
  var numele = parseInt(n.value);
  if (numele>400) {
        window.alert("Upper bound is 400 bars. Kindly
choose a value in that range!");
```

```
n.value=400;
 generate();
 }
 else {
 generatebars(numele);
 }
}
function generate2() {
  const value = Math.floor(Math.random() * 80) + 1;
  generatebars(value);
}

function generatebarsWorst(num) {
  container.innerHTML=""
    let values = Array.from({length: num}, (_, i) =>
Math.max(num - i + 9, 10));
  for (let i = 0; i < num; i += 1) {
    const value = values[i];
    const bar = document.createElement("div");

    bar.classList.add("bar");
    bar.style.height = `${value/2}%`;
const barLabel = document.createElement("label");
    barLabel.classList.add("bar__id");
    barLabel.innerHTML = value;
```

```
if (num>80) {
      barLabel.style.display='none';
    }

    if (num<=40) {
      if (num<=10) {
        barLabel.style.fontSize = 'xxx-large';
      }
      else if (num<=20) {
        barLabel.style.fontSize = 'xx-large';
      }
      if (num>20 && num<=30) {
        barLabel.style.fontSize = 'x-large';
      }
      else if (num<=40) {
        barLabel.style.fontSize = 'large';
      }
    }
    bar.appendChild(barLabel);
    container.appendChild(bar);
  }
}
function generatebarsBest(num) {
  container.innerHTML=""
```

```
let values = Array.from({length: num}, (_, i) => i + 1 +
9);
 for (let i = 0; i < num; i += 1) {
 const value = values[i];
const bar = document.createElement("div");

    bar.classList.add("bar");
    bar.style.height = `${value/2}%`;

    const barLabel = document.createElement("label");
    barLabel.classList.add("bar__id");
    barLabel.innerHTML = value;

    if (num>80) {
      barLabel.style.display='none';
    }
    if (num<=40) {
      if (num<=10) {
        barLabel.style.fontSize = 'xxx-large';
      }
else if (num<=20) {
        barLabel.style.fontSize = 'xx-large';
      }
      if (num>20 && num<=30) {
        barLabel.style.fontSize = 'x-large';
      }
```

```
else if (num<=40) {
        barLabel.style.fontSize = 'large';
      }
    }
    bar.appendChild(barLabel);
    container.appendChild(bar);
  }
}
function generatebest() {
  const value = Math.floor(Math.random() * 80) + 1;
  generatebarsBest(value);
}
function generateworst() {
  const value = Math.floor(Math.random() * 80) + 1;
  generatebarsWorst(value);
}
function disable() {
  document.getElementById("Button1").disabled = true;
document.getElementById("Button1").style.backgroun
dColor = "#d8b6ff";
  document.getElementById("Button2").disabled = true;
document.getElementById("Button2").style.backgroun
dColor = "#d8b6ff";
  document.getElementById("Button3").disabled = true;
document.getElementById("Button3").style.backgroun
dColor = "#d8b6ff";
```

```
document.getElementById("Button4").disabled = true;
 document.getElementById("Button4").style.backgroun
dColor = "#d8b6ff";

 document.getElementById("Button5").disabled = true;
 document.getElementById("Button5").style.backgroun
dColor = "#d8b6ff";
}
var delay = 5000;

async function BubbleSort() {
 count = 0;
 let bars = document.querySelectorAll(".bar");

 var l0 = document.getElementById("line0");
 l0.style.backgroundColor = "lightgreen";
 for (let i = 0; i < bars.length; i++) {
   var l1 = document.getElementById("line1");
   l1.style.backgroundColor = "cyan";
await new Promise((resolve) => setTimeout(() => {
resolve(); }, delay));
   l1.style.backgroundColor = null;

   var c2 = 1;
```

```
for (let j = 0; j < bars.length - i - 1; j++) {
      var l2 = document.getElementById("line2");
      l2.style.backgroundColor = "cyan";
      while (isPlaying) {
        if (terminate) {
          l2.style.backgroundColor = null;
          l0.style.backgroundColor = null;
           document.getElementById("Button1").disabled =
false;
document.getElementById("Button1").style.backgroun
dColor = "#a54997";
           document.getElementById("Button2").disabled =
false;
 document.getElementById("Button2").style.backgroun
dColor = "#a54997";
           document.getElementById("Button3").disabled =
false;
 document.getElementById("Button3").style.backgroun
dColor = "#a54997";
           document.getElementById("Button4").disabled =
false;
 document.getElementById("Button4").style.backgroun
dColor = "#a54997";
           document.getElementById("Button5").disabled =
false;
```

```
document.getElementById("Button5").style.backgroun
dColor = "#a54997";
 isPlaying = false;
 pausePlayBtn.textContent = 'Pause';
compare.textContent=' ' + " No of Comparisons: ";
        for (let k=0;k<bars.length;k++) {
              bars[k].style.backgroundColor="rgb(236, 190,
53)";
        }
        terminate = !terminate;
        return;
       }
       await new Promise((resolve) => setTimeout(() => {
resolve(); }, 1000));
     }
     if (terminate) {
       l2.style.backgroundColor = null;
       l0.style.backgroundColor = null;
         document.getElementById("Button1").disabled =
false;
document.getElementById("Button1").style.backgroun
dColor = "#a54997";
         document.getElementById("Button2").disabled =
false;
document.getElementById("Button2").style.backgroun
dColor = "#a54997";
```

```
document.getElementById("Button3").disabled = false;
document.getElementById("Button3").style.backgroun
dColor = "#a54997";
        document.getElementById("Button4").disabled =
false;
 document.getElementById("Button4").style.backgroun
dColor = "#a54997";

        document.getElementById("Button5").disabled =
false;
document.getElementById("Button5").style.backgroun
dColor = "#a54997";

      for (let k=0;k<bars.length;k++) {
            bars[k].style.backgroundColor="rgb(236, 190,
53)";
      }
   compare.textContent=' ' + " No of Comparisons: ";
      terminate = !terminate;
      return;
    }
                                    var value1 =
parseInt(bars[j].childNodes[0].innerHTML);
                  var value2 = parseInt(bars[j +
1].childNodes[0].innerHTML);
```

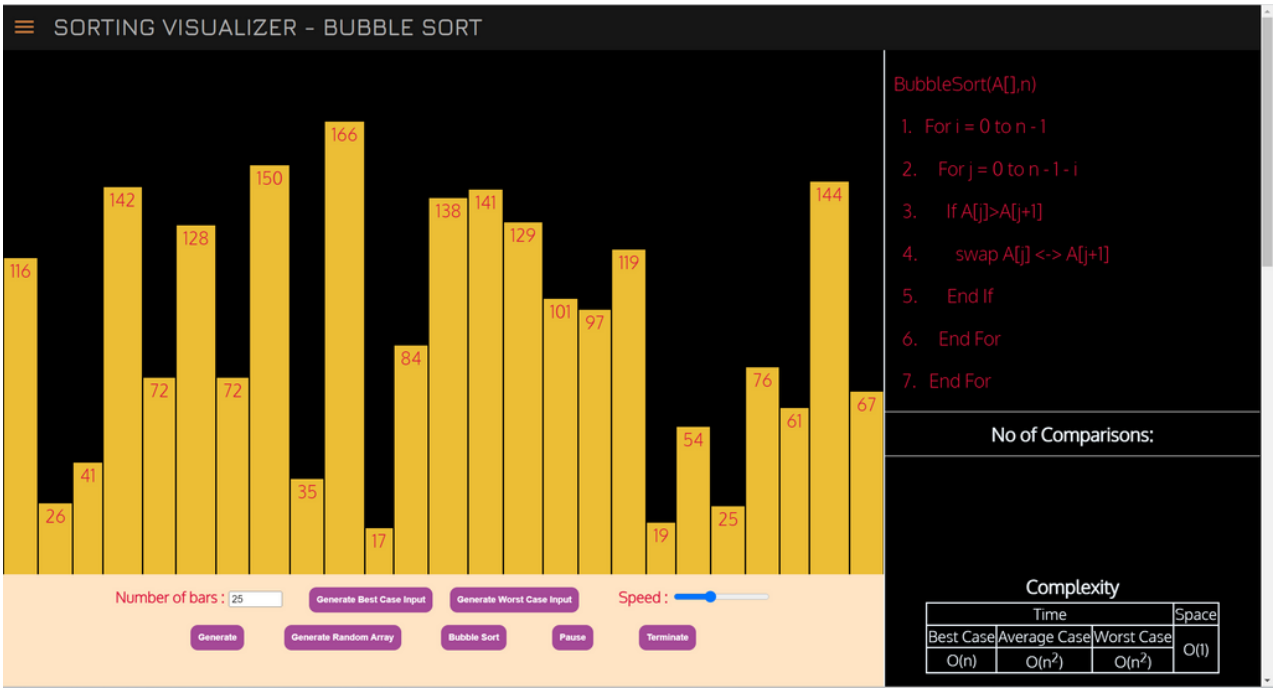
```
var l3 = document.getElementById("line3");
    l3.style.backgroundColor = "cyan";
    if (value1 > value2) {
      await new Promise((resolve) => setTimeout(() => {
resolve(); }, delay));
      var l4 = document.getElementById("line4");
      l4.style.backgroundColor = "cyan";
      bars[j].style.backgroundColor = "red";
    await new Promise((resolve) => setTimeout(() => {
resolve(); }, delay));
      bars[j + 1].style.backgroundColor = "red";
      var temp1 = bars[j].style.height;
      var temp2 = bars[j].childNodes[0].innerText;

      await new Promise((resolve) => setTimeout(() => {
resolve(); }, delay));

      bars[j].style.height = bars[j + 1].style.height;
            bars[j].childNodes[0].innerText = bars[j +
1].childNodes[0].innerText;
      bars[j + 1].style.height = temp1;
      bars[j + 1].childNodes[0].innerText = temp2;

      l4.style.backgroundColor = null;
    }
    c2 = c2+1;
```

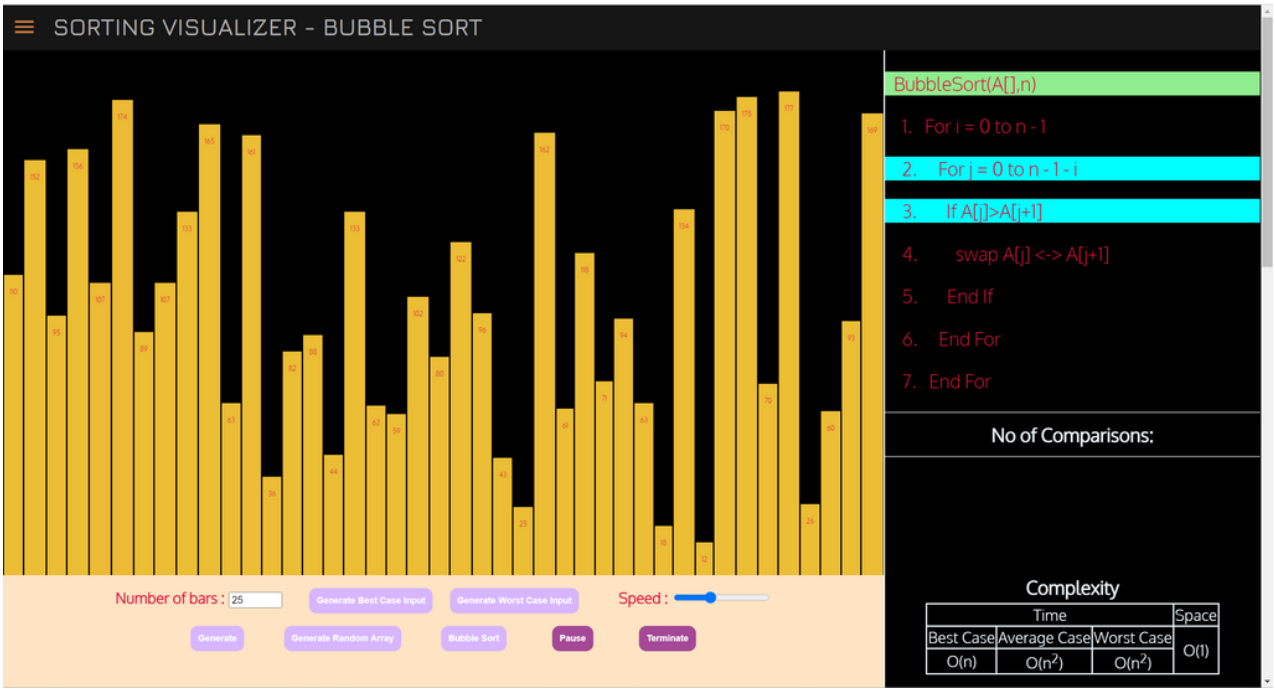
```
var l5 = document.getElementById("line5");
     l5.style.backgroundColor = "cyan";
     l3.style.backgroundColor = null;
      await new Promise((resolve) => setTimeout(() => {
resolve(); }, delay));
     l5.style.backgroundColor = null;
     bars[j].style.backgroundColor = "rgb(236, 190, 53)";
      bars[j + 1].style.backgroundColor = "rgb(236, 190,
53)";
      await new Promise((resolve) => setTimeout(() => {
resolve(); }, delay));
     l2.style.backgroundColor = null;
   }
   count = count + c2;
   compare.textContent=' ' + " No of Comparisons:
"+count;
   var l6 = document.getElementById("line6");
   l6.style.backgroundColor = "cyan";
      bars[bars.length - i - 1].style.backgroundColor =
"rgb(49, 226, 13)";
    await new Promise((resolve) => setTimeout(() => {
resolve(); }, delay));
   l6.style.backgroundColor = null;
  }
  var l7 = document.getElementById("line7");
  l7.style.backgroundColor = "cyan";
```

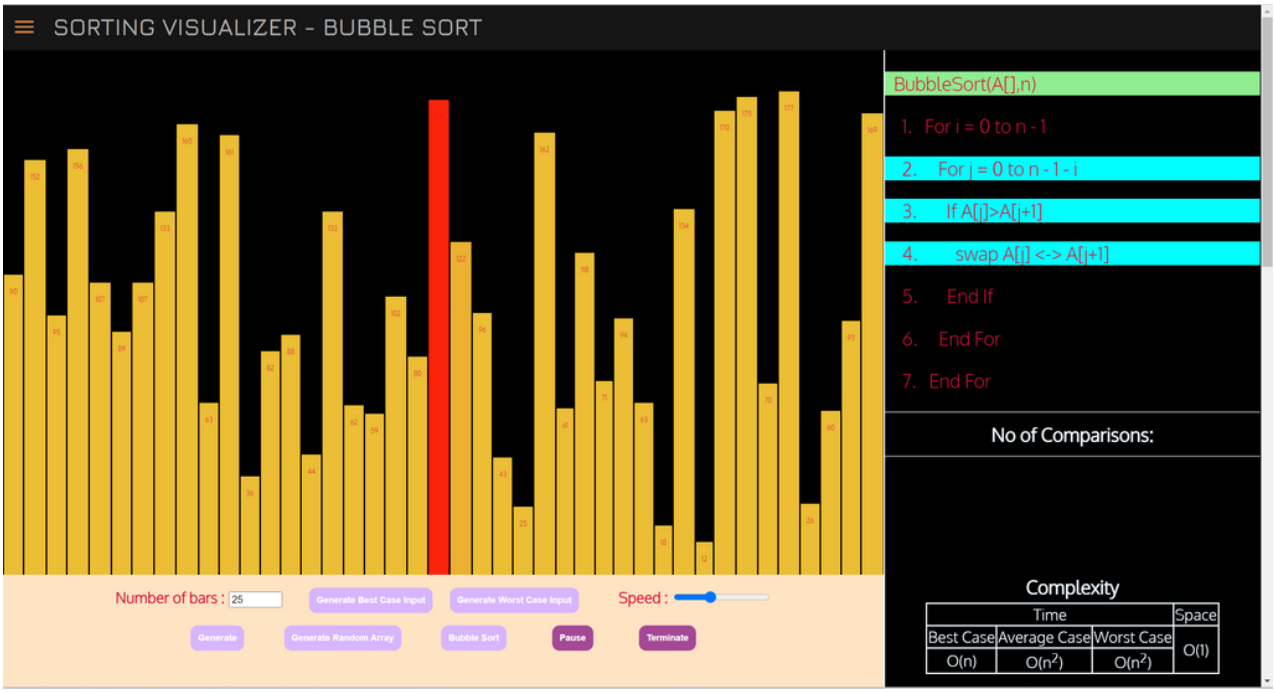
```
for (let i = 0; i < bars.length; i++) {
    bars[i].style.backgroundColor = "rgb(49, 226, 13)";
  }
   await new Promise((resolve) => setTimeout(() => {
resolve(); }, delay));
  l7.style.backgroundColor = null;
        document.getElementById("Button1").disabled    =
false;
document.getElementById("Button1").style.backgroun
dColor = "#a54997";
        document.getElementById("Button2").disabled    =
false;
document.getElementById("Button2").style.backgroun
dColor = "#a54997";
        document.getElementById("Button3").disabled    =
false;
document.getElementById("Button3").style.backgroun
dColor = "#a54997";
        document.getElementById("Button4").disabled    =
false;
document.getElementById("Button4").style.backgroun
dColor = "#a54997";
        document.getElementById("Button5").disabled    =
false;
document.getElementById("Button5").style.backgroun
dColor = "#a54997";
```

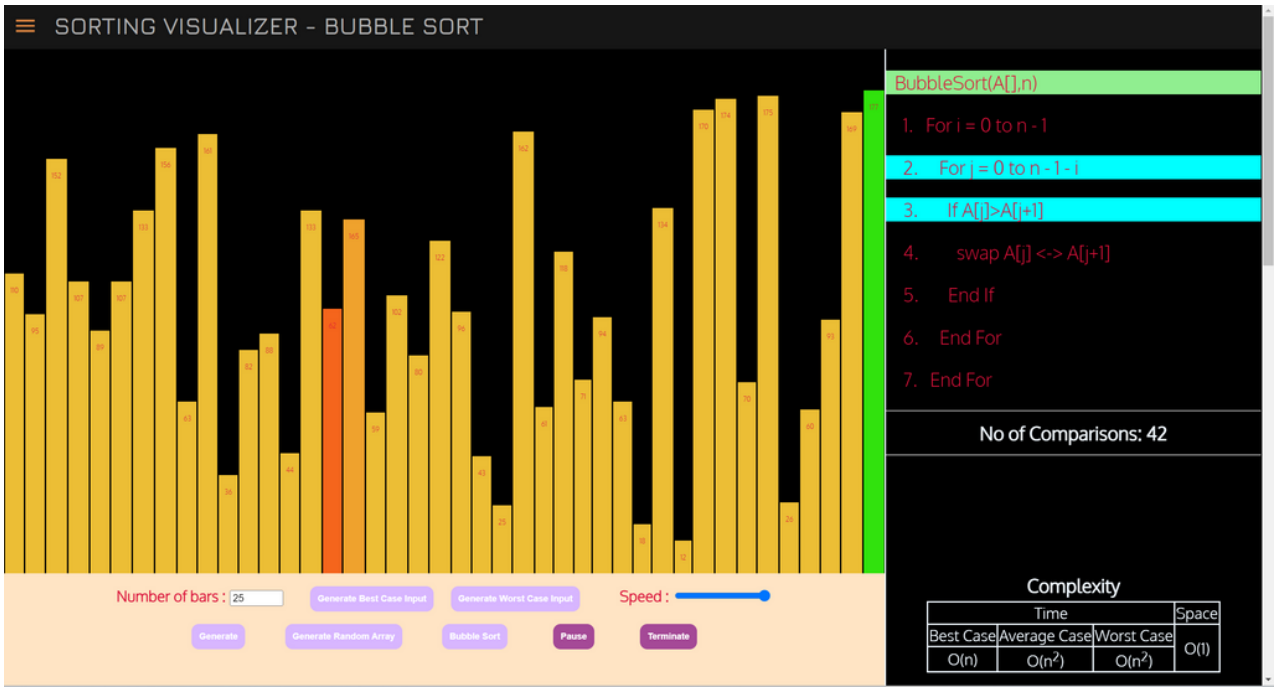
```
compare.textContent=' ' + " No of Comparisons: "+count;

   await new Promise((resolve) => setTimeout(() => { resolve(); }, delay));
  l0.style.backgroundColor = null;
}

function delaySet() {
  delay = 5000;
  var s = document.getElementById("speeder");
  var d = parseInt(s.value);
  delay=delay/d;
}

pausePlayBtn.addEventListener('click', () => {
  if (isPlaying) {
   isPlaying = false;
   pausePlayBtn.textContent = 'Pause';
  } else {
   isPlaying = true;
   pausePlayBtn.textContent = 'Resume';
  }
 });
```

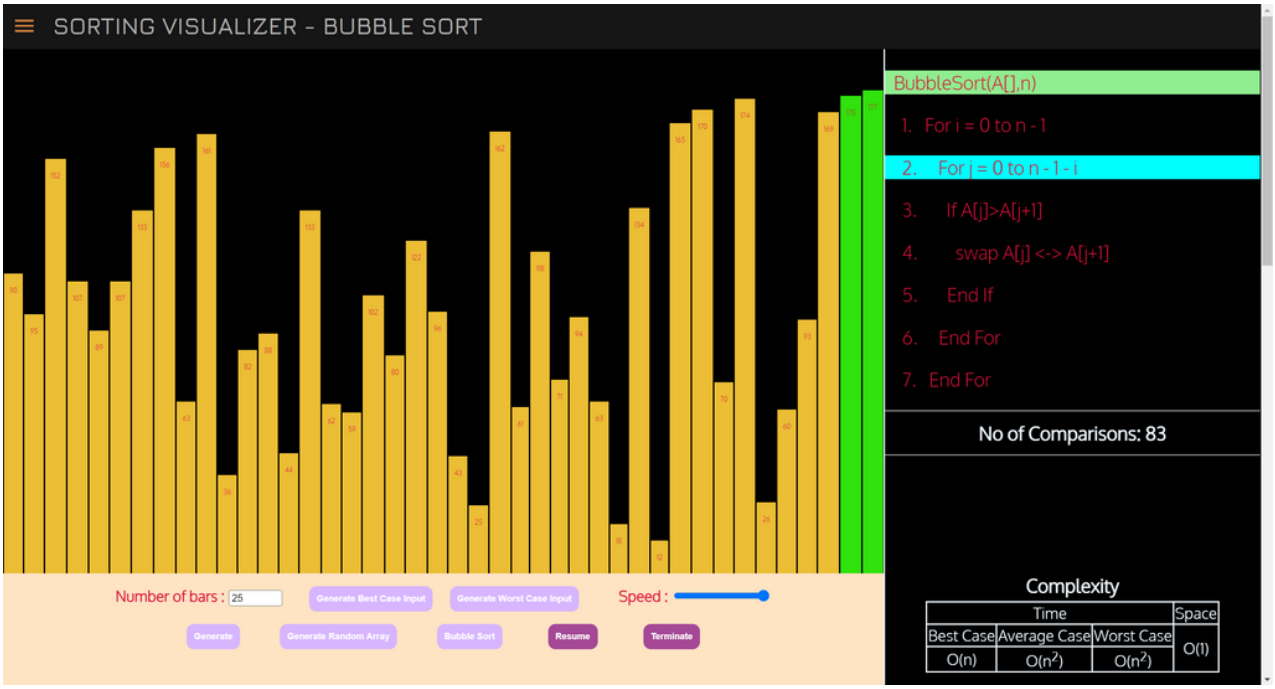
5.2 Screenshots of working project

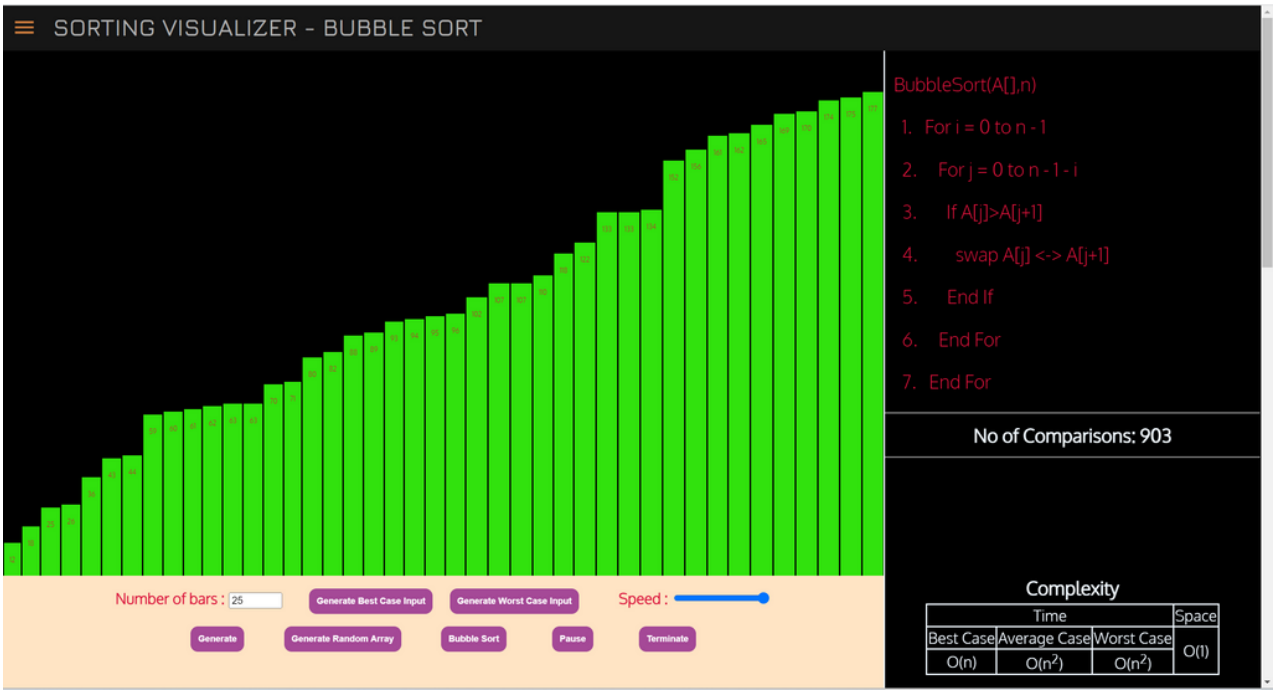





SORTING VISUALIZER - BUBBLE SORT
BubbleSort(A[],n)
1. For i = 0 to n - 1
2. For j = 0 to n - 1 - i
3. If A[j]>A[j+1]
4. swap A[j] <-> A[j+1]
5. End If
6. End For
7. End For
No of Comparisons:
Number of bars : 25
Generate Best Case Input
Generate Worst Case Input
Speed :
Generate
Generate Random Array
Bubble Sort
Pause
Terminate
Complexity
Time
Space
Best Case
Average Case
Worst Case
O(n)
O(n²)
O(n²)
O(1)


SORTING VISUALIZER - BUBBLE SORT
BubbleSort(A[],n)
1. For i = 0 to n - 1
2. For j = 0 to n - 1 - i
3. If A[j]>A[j+1]
4. swap A[j] <-> A[j+1]
5. End If
6. End For
7. End For
No of Comparisons: 42
Number of bars : 25
Generate Best Case Input
Generate Worst Case Input
Speed :
Generate
Generate Random Array
Bubble Sort
Pause
Terminate
Complexity
Time
Space
Best Case
Average Case
Worst Case
O(n)
O(n²)
O(n²)
O(1)


SORTING VISUALIZER - BUBBLE SORT
BubbleSort(A[],n)
1. For i = 0 to n - 1
2. For j = 0 to n - 1 - i
3. If A[j]>A[j+1]
4. swap A[j] <-> A[j+1]
5. End If
6. End For
7. End For
No of Comparisons: 83
Number of bars : 25
Generate Best Case Input
Generate Worst Case Input
Speed :
Generate
Generate Random Array
Bubble Sort
Resume
Terminate
Complexity
Time
Space
Best Case
Average Case
Worst Case
O(n)
O(n²)
O(n²)
O(1)


SORTING VISUALIZER - BUBBLE SORT
BubbleSort(A[],n)
1. For i = 0 to n - 1
2. For j = 0 to n - 1 - i
3. If A[j]>A[j+1]
4. swap A[j] <-> A[j+1]
5. End If
6. End For
7. End For
No of Comparisons: 903
Number of bars : 25
Generate Best Case Input
Generate Worst Case Input
Speed :
Generate
Generate Random Array
Bubble Sort
Pause
Terminate
Complexity
Time
Space
Best Case
Average Case
Worst Case
O(n)
O(n²)
O(n²)
O(1)

Software Testing

"Testing is the process of executing the program with the intent of finding errors."

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation. Once source code has been generated, software must be tested to uncover as many tests as possible before delivery to the customer.

6.1 Software Testing Fundamentals

A strategy for software testing provides a road map that describes the steps to be conducted as part of testing, when these steps are planned and then undertaken, and how much effort, time and resources will be required. Therefore, any testing strategy must incorporate test planning, test case design, test execution, and resultant data collection and evaluation. At the same time, it must be rigid enough to encourage reasonable planning and management tracking as the project progresses. It must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

The terms verification & validation are often confused and used interchangeably but have different meanings. Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the condition imposed at the start of that phase. Hence verification activities are applied to early phases in SDLC such as requirements, design, planning etc. We check and review the documents generated after completion of every phase in order to ensure that what comes out of that phase is what we expected to get. Whereas validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies the specific requirements. Therefore, validation requires actual execution of the program and is also known as computer-based testing. We experience failures and identify the causes of the failure. Hence testing includes both validation and verification. The terms alpha and beta testing are used when the software is developed as a product for anonymous customers. Hence formal acceptance testing is not possible in such cases. However, some potential customers are identified to get their views about the product.

The alpha tests are conducted at the developer's site by a customer. These tests are conducted in a controlled environment. The beta tests are conducted by the customers/end users at their sites. Unlike alpha testing, developers are not present here. Beta testing is conducted in a real environment that cannot be controlled by the developer. Customers are expected to report failures, if any, to the company. After receiving such failure reports, developers modify the code and fix the bug and compare the product after the final release.

6.2 Test Case Design

Test No.	Description	Test Data	Excepted Result	Actual Result
1.	Selection of Sorting algorithm by user .	Data required by the user.	Interface of the selected sort should open.	Pass
2.	Starting the sort by selecting whether best or worst case or random generated input.	Data required to be entered by the user.	Should start with required speed on selected input.	Pass
3.	Generating the array of sorted bars with no. comparison taken by the algorithm.	No Data required to be entered by the user.	Applying the respective sorting function and displaying the correct number of comparisons.	Pass

6.3 Black Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

Advantages:

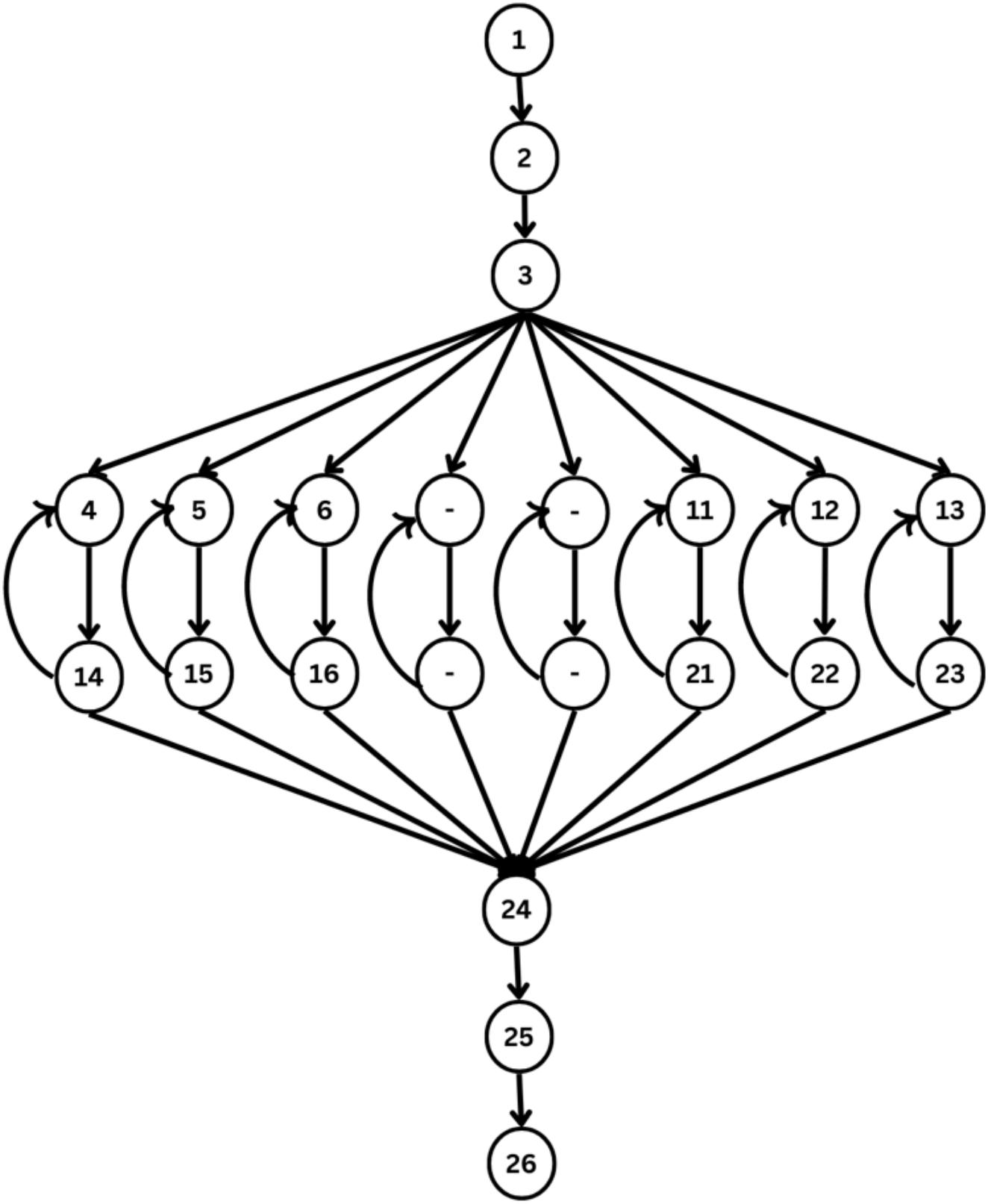
- Well suited and efficient for large code segments.
- Code access is not required.
- Clearly separates the user's perspective from the developer's perspective through visibly defined roles.
- Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.

6.4 White Box Testing

White-box testing is the detailed investigation of the internal logic and structure of the code. White-box testing is also called glass testing or open-box testing. In order to perform white-box testing on an application, a tester needs to know the internal workings of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

Advantages:

- As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.
- It helps in optimizing the code.
- Extra lines of code can be removed which can bring in hidden defects.
- Due to the tester's knowledge of the code, the maximum coverage is attained during test scenario writing.



Control Flow Graph

6.5 Cyclomatic Complexity

Cyclomatic Complexity = E – N + 2
where E = Number of edges in the program flow graph and,
N = Number of nodes in the program flow graph.
Here, E = 44, N = 26
Cyclomatic Complexity = 44 – 26 + 2 = 20
Since the cyclomatic complexity is 20, therefore the number of independent paths will be 20.

MAINTENANCE

The term System Maintenance actually refers to the changes that need to be made to a system after it has been implemented. Changes may be required to fix bugs or to make enhancements to the system in order to accommodate new user requirements or adapt to new hardware or software. Users may notice an error in certain types of files that are not compatible with the software. Maintenance would be required to fix this bug. Whenever a user notices an error in the software or an enhancement is required, he passes on the finding and requirements to the software maintenance department. The analyst in the department identifies the impacted items (problem areas), corrects them, tests the package, and requests the user to check if the request has been serviced properly.

Software maintenance can be defined as the process of changing a system after it has been delivered and is in use. It is basically the process of changing a system to maintain its ability to survive. Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

7.1 Need For Maintenance

Software Maintenance must be performed in order to:

- Correct faults.
- Improve the design.
- Implement enhancements.
- Interface with other systems.
- Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
- Migrate legacy software.
- Retire software.

7.2 Categories of Software Maintenance

Maintenance can be divided into the following categories:

- **Corrective maintenance**: Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.
- **Adaptive maintenance**: This includes modifications and updations when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.

- **Perfective maintenance**: A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer's demands.
- **Preventive maintenance**: This type of maintenance includes modifications and updates to prevent future problems with the software. It aims to attend to problems, which are not significant at this moment but may cause serious issues in the future.

Future Work

- **Adding more Sorting Techniques**
- **Testing the visualizer with real-world data**
- **Implementing more advanced visualization techniques**
- **Optimization of the code**
- **Implementing other algorithms to upgrade the sorting visualizer to the algorithm visualizer**
- **Adding sounds to sorting visualizers can make the process more engaging, help users stay focused, and provide auditory feedback to complement the visual feedback.**

BIBLIOGRAPHY

- **Software Engineering: A Practitioner's Approach. 8th edition. McGraw-Hill**
- **Aggarwal, K. K., & Singh, Y. (2007). Software Engineering**
- **https://www.google.com**
- **https://devdocs.io/**
- **http://www.devdoc.net/web/developer.mozilla.org/en-US/docs/en/HTML.html**
- **https://stackoverflow.com/**
- **https://www.geeksforgeeks.org/**
- **https://www.w3schools.com/**