Hajee Mohammad Danesh Science and Technology University, Dinajpur-5200

Project Name: Visual Representation of Data Structures

Course Title: Application Development Sessional

Course Code: CSE 252

Department: Computer Science & Engineering

Level: 2, Semester: II

By

Md. Mostafijur Rahman

Student ID: 1902073

TABLE OF CONTENTS

ABSTRACT

A data structure is an arrangement of data in a computer's memory. An example of several common data structures are arrays, linked lists, stacks, queues, binary trees, graphs etc. Algorithms are used to manipulate the data contained in these data structures such as searching and sorting. Algorithms and data structures are important areas of computer science. In spite of the fact, the most common algorithms and data structures have already been implemented in many programming languages and they are also available in many libraries, every programmer and student of computer science should know their principles. Algorithm visualizations can be an effective way to explain these principles. These Visualizations of the execution of the data structures which can promote better understanding of the data structures and its underlying algorithms. The aim of this project is to visualize some data structures which gives us the outside view of data structures. In this project, the data structures are visualized by using the Java language where Object Oriented concepts are considered to perform all algorithms and for Graphical User Interface Java Swing is used.

Keywords: algorithm, data structure, visualization.

1. INTRODUCTION

1.1 Introduction

The algorithm in general is a sequence of steps, which leads to the desired solution of a problem. In computer science, algorithms are widely used, because every computer requires an exact sequence of instructions to solve a problem. Algorithms in computer science must also be deterministic, finite and effective. Tightly coupled with algorithms is also the area of data structures. Data structure is generally “any data representation and its associated operations”.

There are a big number of algorithms and data structures in computer science, some of them are trivial and easy to understand, but there are also algorithms which require a deeper analysis to understand them. One of the methods for explaining more difficult algorithms can be a pseudocode, since it can be written very quickly and it can also describe an algorithm clearly. However, the explanation of algorithms using a pseudocode fails when the algorithm itself consists of more non-trivial functions. We can get rid of this problem in this situation by using visualization. The visualization helps students to understand how algorithms and data structures work, but many computer science educators don’t use it, because of time and effectiveness reasons. Therefore when we want to implement a visualization platform, it should be effective and easy-to-use.

2. LITERATURE REVIEW

2.1 Literature Review

Data structures are a conceptually demanding topic which confronts many Computer Science students early in their course. The topic has a strong conceptual basis and often proves difficult for many to grasp. A number of previous studies have examined that the use of interaction and visualization within the systems can motivate a student to engage in the learning process. This literature review investigates the effectiveness of these systems that were and are being used today for teaching and learning of data structures to novice Computer Science students. It also explores the different techniques that are used to develop the intelligent tutoring systems and concludes on presenting which techniques are most effective.

Based on analysis of existing solutions of each data structure, we decided to start developing our own algorithm visualization system named "Visual Representation of Data Structures".

In this platform, we effectively use the algorithms and visualize these algorithms using java swing. Anyone can use this to understand the algorithms perfectly and can be able to input any data which they want.

3. METHODOLOGY

3.1 Introduction

A Data Structure is a particular way of storing and organizing data in a computer so that it can be stored, retrieved, or updated efficiently. Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by an address- a bit string that can be itself stored in memory and manipulated by the program.

Visualization of Data Structure is more important to computer science to understand the algorithms better. We've developed an interactive GUI project with some animations for a variety of data structures and algorithms. We use the Java language and the Swing classes and components are also used in our project to represent the graphical things easily.

3.2 Technique

To develop a system too is and technologies play the most important role. The tools denote the components that are used to create the techniques environment for the programmers. Technology is a collection of skills, methods and processes used in the production of goods or services or in the accomplishment of the objectives, such as scientific investigation. Technology is the knowledge of techniques, processes etc. or it is embedded in machines, computers, devices and factories, which can be operated by individuals without detailed knowledge of the workings of such things .Technology describes the way to code a particular project.

The tools and technologies that we used for the implementation for our project are:

>Language: Java

>For GUI: Java Toolkit- Java Swing

>Integrated Development Environment: NetBeans

>Operating System: Windows 10

> RAM: 8GB

3.3 Adding & Searching Representation

Adding is a process where elements are inserted one by one in an array list. After adding a specific number of elements or a data set we can apply the searching data structure to find a desired element. There are different types of searching algorithms but here we have used Linear Search to perform the searching method.

Adding Operation:

- Get the value from the JText field where we insert an element.
- Insert the element in a Jlabel area sequentially.

Searching Operation:

Searching is the process of fetching a specific element in a collection of elements. The collection can be an array or a linked list. If we find the element in the list, the process is considered successful, and it returns the location of that element and prints the element of that location. To perform the searching we have used the linear search algorithm.

Linear search, often known as sequential search, is the most basic search technique. In this type of search, we go through the entire list and try to fetch a match for a single element. If we find a match, then the address of the matching target element is returned. On the other hand, if the element is not found, then it returns a NULL value.

Following is a step-by-step approach employed to perform Linear Search Algorithm:

- First, read the search element (Target element) in the array.
- In the second step compare the search element with the first element in the array.
- If both are matched, display "Target element is found" and terminate the Linear Search function.
- If both are not matched, compare the search element with the next element in the array.
- In this step, repeat steps 3 and 4 until the search (Target) element is compared with the last element of the array.
- If the last element in the list does not match, the Linear Search Function will be terminated, and the message "Element is not found" will be displayed.

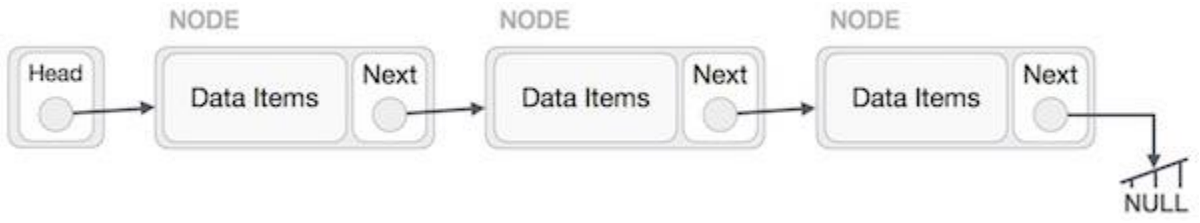
3.4 Linked List Representation

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- Link − each link of a linked list can store a data called an element.
- Next − each link of a linked list contains a link to the next link called Next.
- Linked List − A Linked List contains the connection link to the first link called First.

Linked List Representation:

Linked list can be visualized as a chain of nodes, where every node points to the next node.



As per the above illustration, following are the important points to be considered.

- Linked List contains a link element called first.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Linked List Operations in our system:

Following are the basic operations supported by our system.

- Insertion − Adds an element at the beginning of the list.

  − Adds an element at the last of the list

  − Adds an element by using a specific location.

- Deletion − Deletes or Removes an element at the beginning of the list.

  − Remove an element at the last of the list.

  − Removes an element from the desired location.

3.5 Stack Representation

A Stack is a Linear Data Structure that follows a particular order in which the operations are performed. It is just like a pile of books that are kept on top of each other. We can do the following with such a pile of books:-Put a new book on top of all the books. Remove the top book from the pile of books. If you want the book at the top of the piled-up books, such arrangement is called ***Last In First Out — the last item is the first item to go out.***

Stack Operations in our system:

There are mainly two operations that can be performed on a stack:

- Push: Putting/Placing an item on top of the stack.
- Pop: Removing an item from the top of the stack.

There are few other operations that are used for stack implementation:

- Is Empty: Check if the stack is empty.
- Is Full: Check if the stack is full (overflow).
- Peek: Get the value of the top element of the stack without removing it.
- Delete: Delete the top element.

Now's Let's Have a Look at the Working of the Stack Data Structure:

Following are the steps used for stack implementation:

1. A TOP pointer is used which points to the top element in the stack.
2. Initially, the TOP pointer is set to -1 so that we can check if the stack is empty or not by comparing TOP = -1.
3. While pushing an element on the stack, first increment the value of the TOP pointer and then place the new element in the position pointed by the TOP pointer.

4. But remember before pushing the element, we must check if the stack is already full or not.
5. On popping an element, we return the element pointed to by TOP and reduce its value.
6. Note that before popping, we must check if the stack is already empty or not.

3.6 Queue Representation

A Queue is a Linear Data Structure that follows a particular order in which the operations are performed. The order is First in First out (FIFO). An example of a queue is the ticket counter where a person who came first receives the ticket first and the last standing person receives the ticket last.

We can perform the following operations on a queue:-

- Adding an item to the end/rear of the queue is called Enqueue.
- Remove an item from the front of the queue that is called Dequeue.

Queue Operations in our system:

There are mainly two operations that can be performed on a queue:

1. *Enqueue:* Adding an item to the end or rear of the queue.
2. *Dequeue:* Removing an item from the front of the queue.

There are few other operations that are used for queue implementation:

1. IsEmpty: Check if the queue is empty.
2. IsFull: Check if the queue is full.
3. Peek: Get the value of the front or rear element of the queue without removing it.

Let’s See the Working of the Queue:

For Queue implementation, two-pointers FRONT and REAR are used.

1. FRONT: this pointer points to the first element of the queue.
2. REAR: this pointer points to the last element of the queue.

Note that initially, FRONT and REAR are set to -1.

Enqueue Operation:

- Check if the queue is full
- For the first element, set the value of FRONT to 0
- Increase the REAR index by 1
- Add the new element at the position pointed by REAR

Dequeue Operation:

- check if the queue is empty
- return the value pointed by FRONT
- increase the FRONT index by 1
- When there are no elements in the queue, reset the values of FRONT and REAR to -1. (i.e after removing the last element in the queue).

3.7 Sorting Representation

3.7.1 Bubble Sort

Bubble Sort is one of the simplest sorting algorithms which works by swapping the adjacent elements repeatedly if they are in the wrong order.

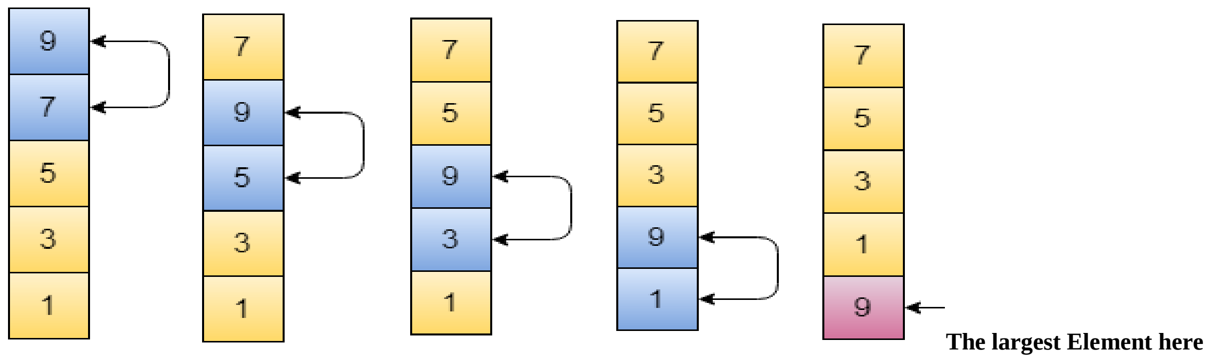
Basically, if there are n elements in the array then there would be n-1 passes required to sort them. For sorting the elements in the ascending order, in the 1st pass, the largest element settles down at its correct position. In the 2nd pass, the second largest element settles down at its correct position and so on.

Bubble Sort Operations in our system:

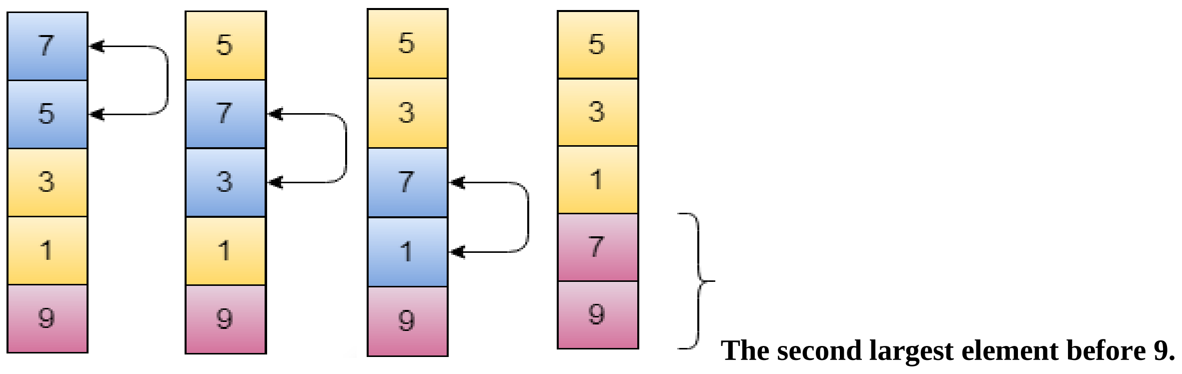
1. Initially we input 10 elements in the array which are randomly generated by java swing components.
2. In the Bubble sort algorithm two elements will be compared. In the array list we denoted the compared these two elements into two different colors- pink and cyan color.
3. In every pass the largest element will be inserted and settled down in the right side of the array.

The all passes are given below for the array 9, 7,5,3,1.

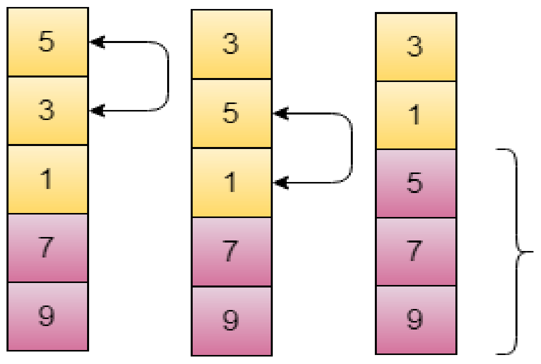
Pass 1:

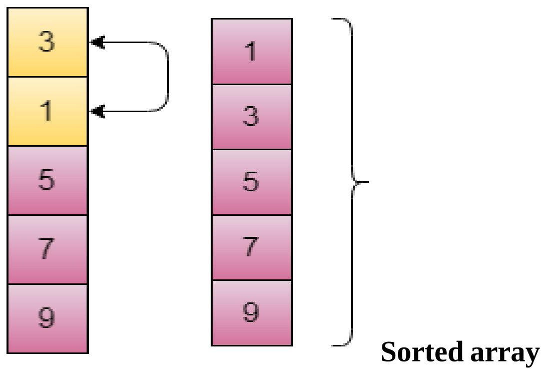


Pass 2:



Pass 3: **Pass 4:**





3.7.2 Radix Sort

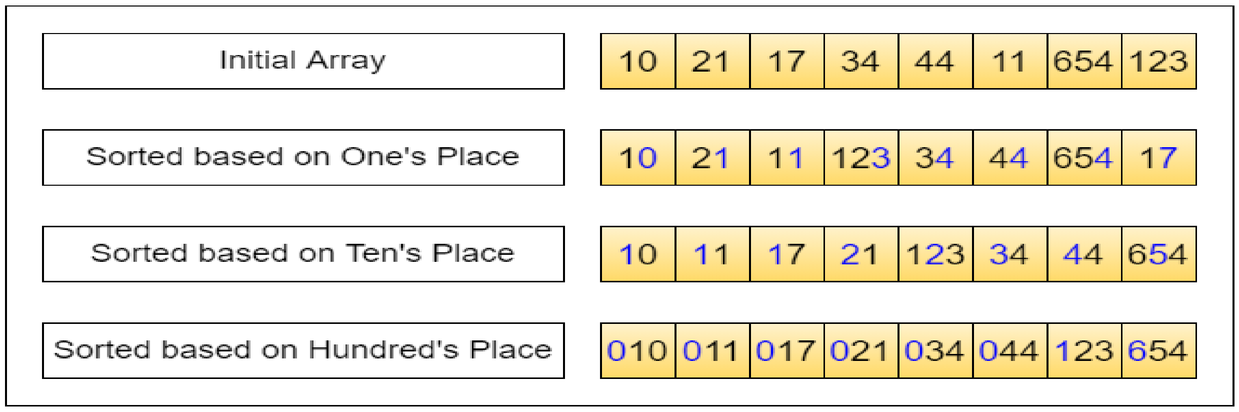
Radix sort is a small method that many people intuitively use when alphabetizing a large list of names. Specifically, the list of names is first sorted according to the first letter of each name, that is, the names are arranged in 26 classes.

Intuitively, one might want to sort numbers on their most significant digit. However, Radix sort works counter-intuitively by sorting on the least significant digits first. On the first pass, all the numbers are sorted on the least significant digit and combined in an array. Then on the second pass, the entire numbers are sorted again on the second least significant digits and combined in an array and so on.

Steps in our system for the completion of Radix sort:

1. In our system we get an array with 20 elements with 3 digits maximum which are randomly generated by java components.
2. To sort the all places elements we take 10 columns to insert these elements.
3. To sort one’s place elements the step button has to be frequently clicked by the user.
4. These process will be running until the all places are not sorted
5. Finally we get the sorted array which is randomly generated.

Radix Sort basic design:



Initial Array	10	21	17	34	44	11	654	123
Sorted based on One's Place	10	21	11	123	34	44	654	17
Sorted based on Ten's Place	10	11	17	21	123	34	44	654
Sorted based on Hundred's Place	010	011	017	021	034	044	123	654

3.7.3 Insertion Sort

Insertion sort algorithm picks elements one by one and places it to the right position where it belongs in the sorted list of elements. In the following C program we have implemented the same logic.

Before going through the program, let's see the steps of insertion sort with the help of an example.

Steps in our system for the completion of Insertion sort:

1. Initially we input 10 elements in the array which are randomly generated by java swing components.
2. Insertion sort involves going through a pile, taking one item, comparing it to the first, swapping places if one item is larger than another and continuing this process until the minimum item is in the correct location.
3. In the Insertion sort algorithm two elements will be compared. In the array list we denoted the compared these two elements into two different colors- pink and cyan color.

Input elements: 89, 17, 8, 12, 0

Step 1: **89,** 17, 8, 12, 0 (the bold elements are sorted list and non-bold unsorted list)

Step 2: **17, 89,** 8, 12, 0 (each element will be removed from unsorted list and placed at the right position in the sorted list)

Step 3: **8, 17, 89,** 12, 0

Step 4: **8, 12, 17, 89,** 0

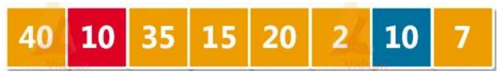
Step 5: **0, 8, 12, 17, 89 (the sorted element).**

3.7.4 Selection Sort

Selection sort is another sorting technique in which we find the minimum element in every iteration and place it in the array beginning from the first index. Thus, a selection sort also gets divided into a sorted and unsorted subarray.

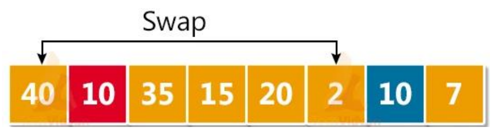
Steps in our systems for the completion of Selection sort:

1. Initially we input 10 elements in the array which are randomly generated by java swing components.
2. In every iteration, the selection sort algorithm selects the smallest element from the whole array and swaps it with the leftmost element of the unsorted sub-array.
3. In the array list we denoted the sorted element with pink color which will be compared with the minimum element of the unsorted parte that is denoted by cyan color.
4. The number of steps will be less than one from the total elements in the array list.

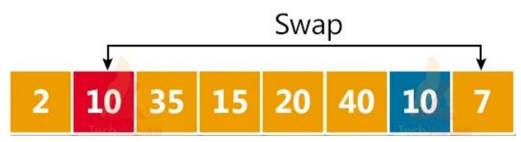


The all passes are given below for the array list

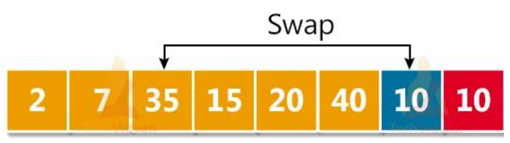
Pass 1:



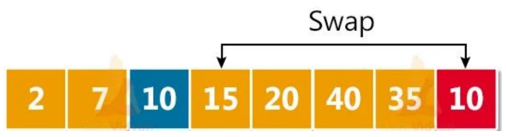
Pass 2:



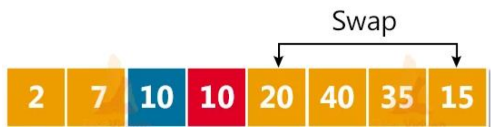
Pass 3:



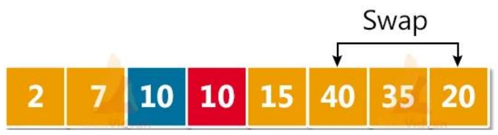
Pass 4:



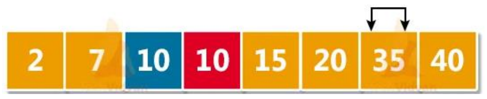
Pass 5:



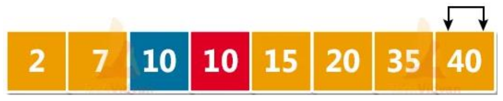
Pass 6:



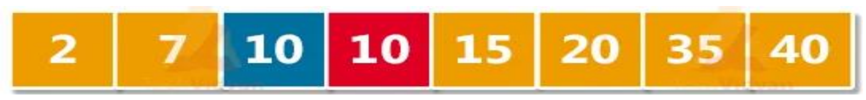
Pass 7:



Pass 8:

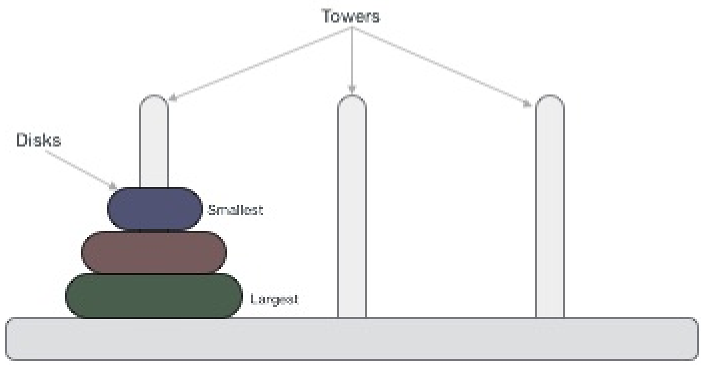


Thus, the final array after selection sort is:



3.8 Tower of Hanoi Representation

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one disks is as depicted −



These disks are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

Rules of TOH:

The mission is to move all the disks to some other tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk. B

Proposed system design for Tower of Hanoi:

To design the implementation of Tower of Hanoi, first we need to learn how to solve this problem with a smaller amount of disks, say → 1 or 2. We mark three towers with name, source, destination and aux (only to help moving the disks). If we have only one disk, then it can easily be moved from source to destination peg.

If we have 2 disks −

- First, we move the smaller (top) disk to aux peg.
- Then, we move the larger (bottom) disk to the destination peg.
- And finally, we move the smaller disk from aux to destination peg.

So now, we are in a position to design an algorithm for Tower of Hanoi with more than two disks. We divide the stack of disks in two parts. The largest disk (nth disk) is in one part and all other (n-1) disks are in the second part.

Our ultimate aim is to move disk n from source to destination and then put all other (n1) disks onto it. We can imagine applying the same in a recursive way for all given sets of disks.

The steps to follow are −

Step 1 − Move n-1 disks from source to destination.

Step 2 − Move nth disk from source to destination.

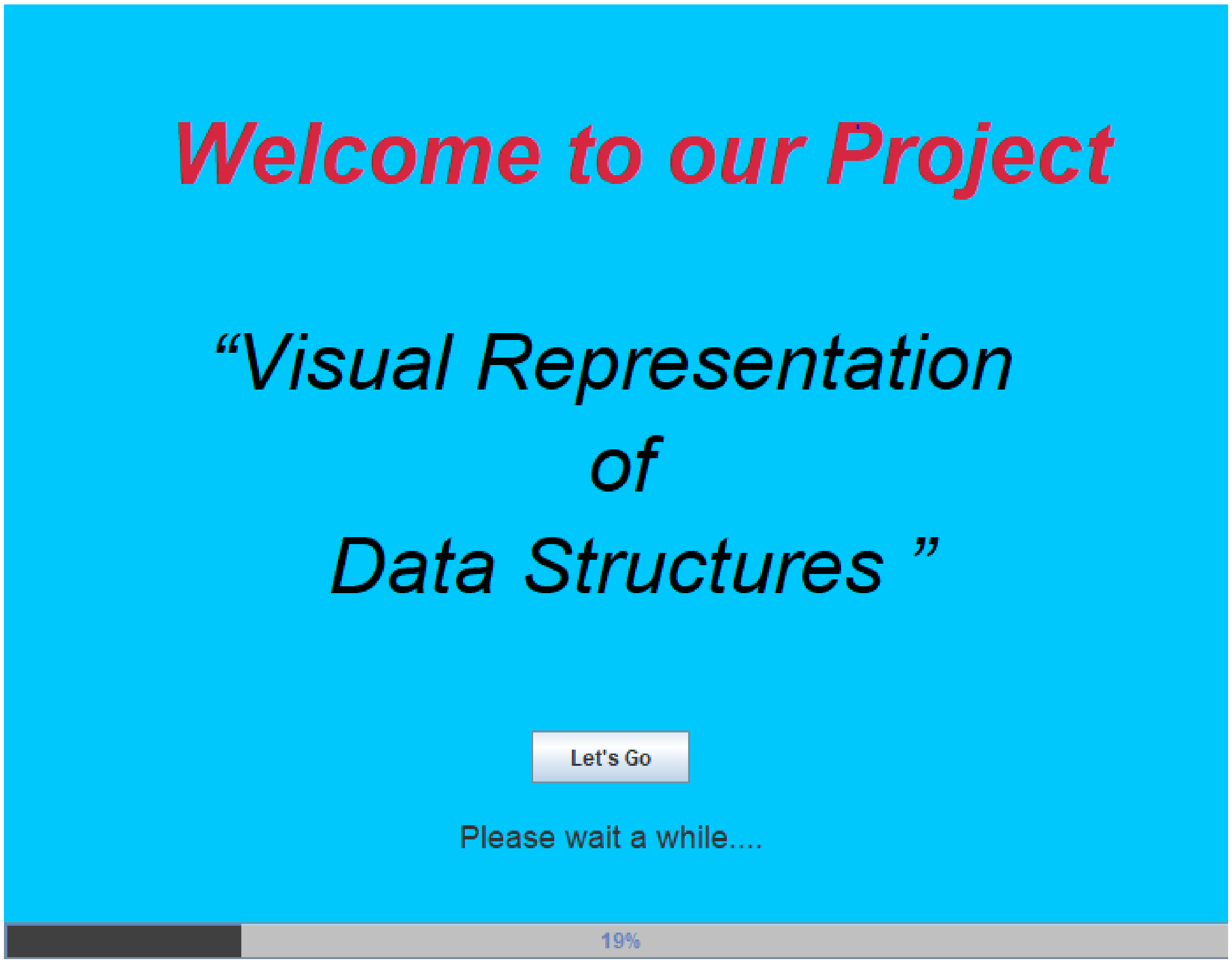
Step 3 − Move n-1 disks from aux to destination.

4. RESULT AND DISCUSSION

4.1 Introduction

After implementation, we get our system ready with some output. We apply all our algorithms perfectly and after running this system in our platform we get the User Interface to understand which task will be worked for each algorithm. It will also be very easy to understand to the general people what happens in these algorithms.

4.2 Project Outputs


Welcome to our Project

“Visual Representation of Data Structures ”

Figure 5.1: Welcome Page.

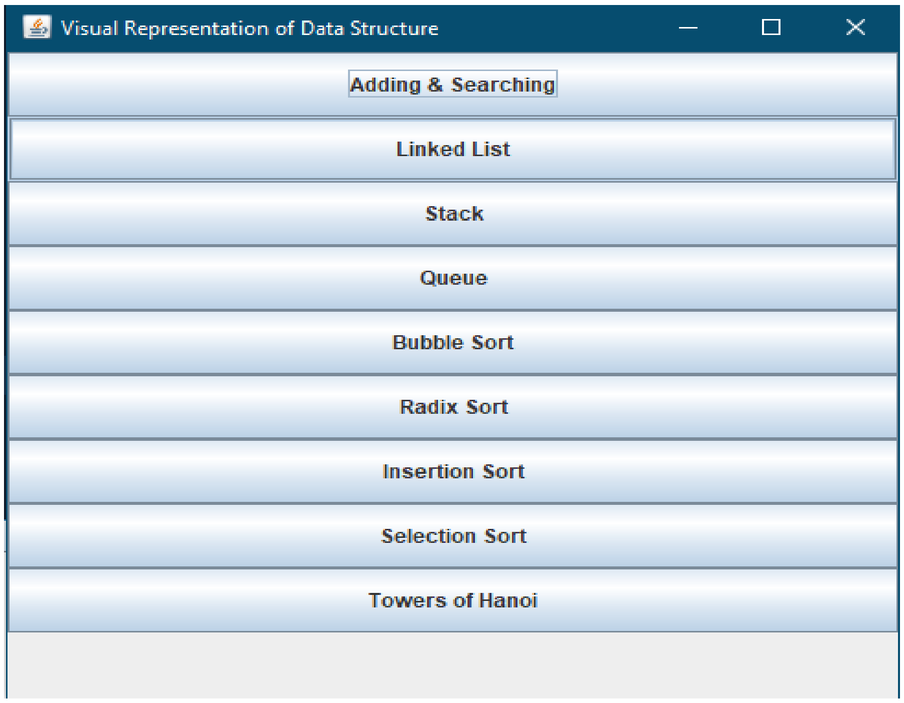
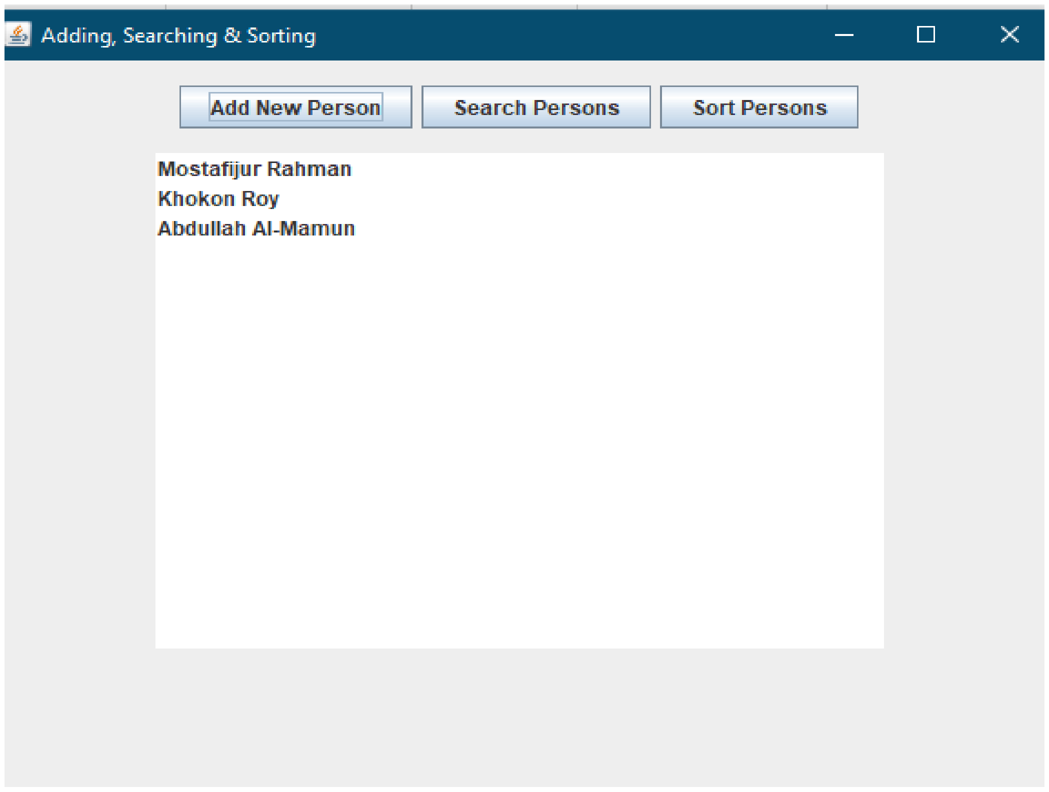


Figure 5.1: Home Page



Mostafijur Rahman
Khokon Roy
Abdullah Al-Mamun

Figure 5.2: Adding, Searching & Sorting Page.

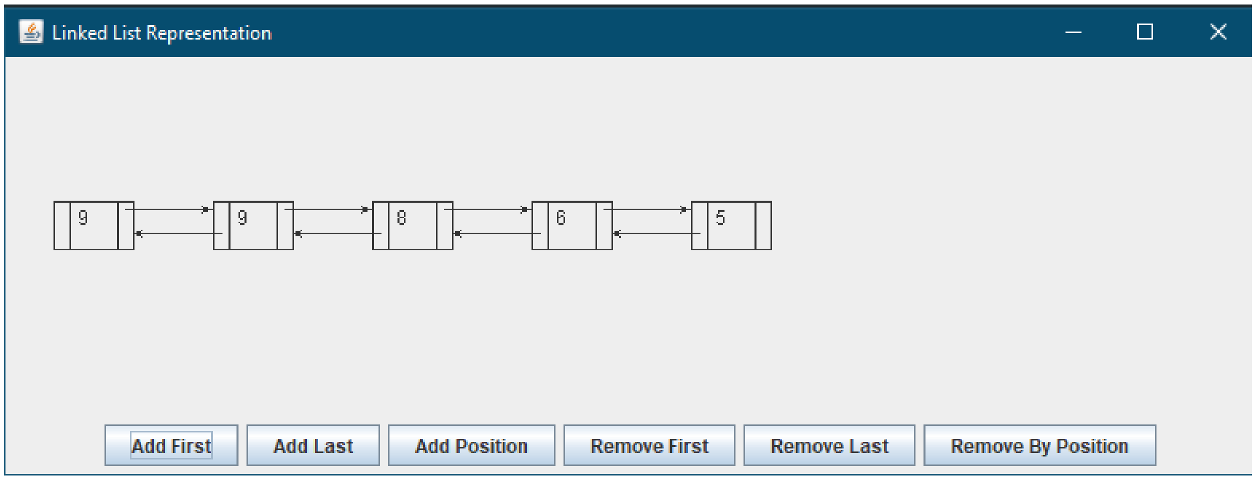


Figure 5.3: Linked List Page.

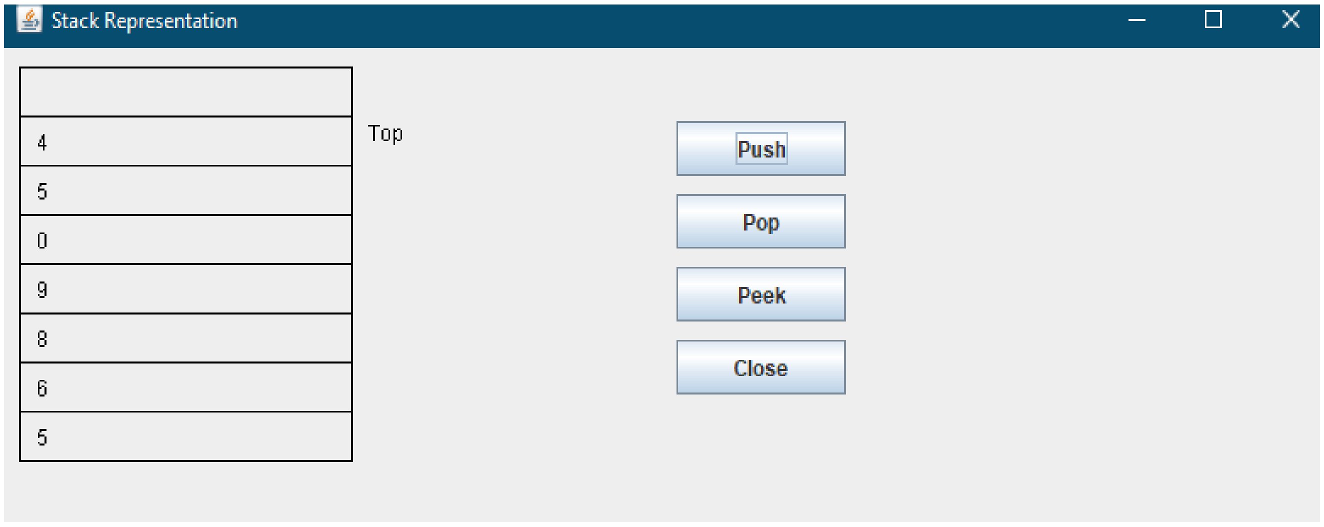


Figure 5.4: Stack Representation Page

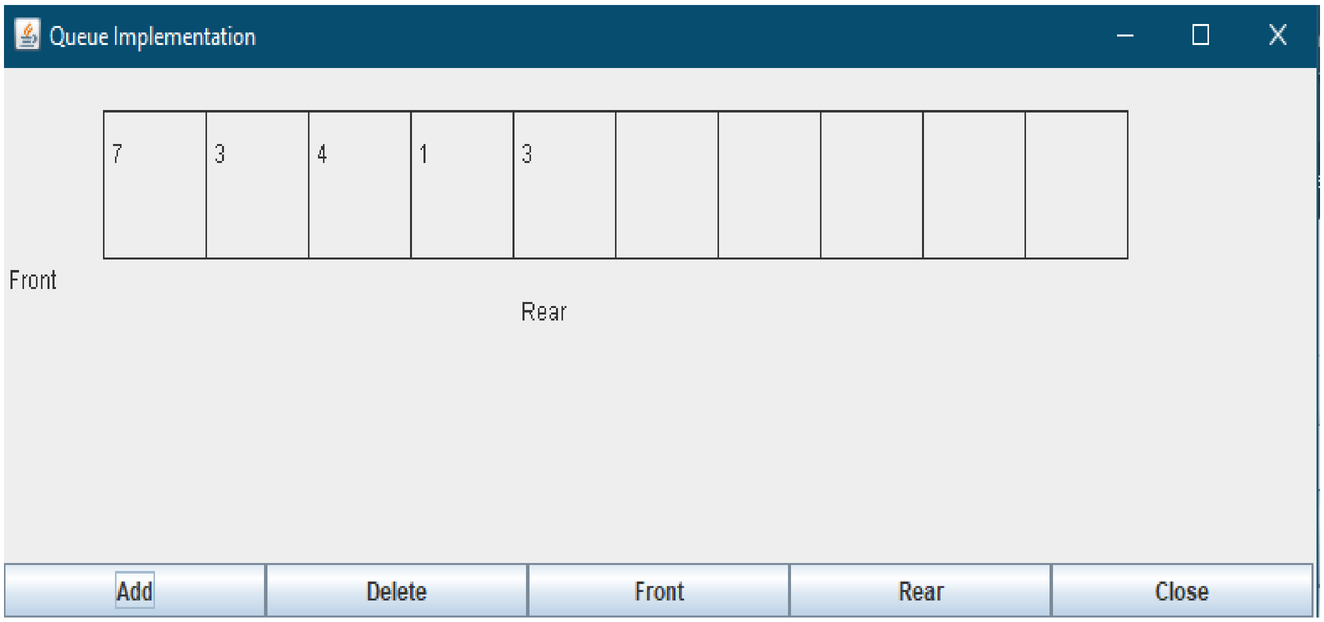


Figure 5.5: Queue Representation Page

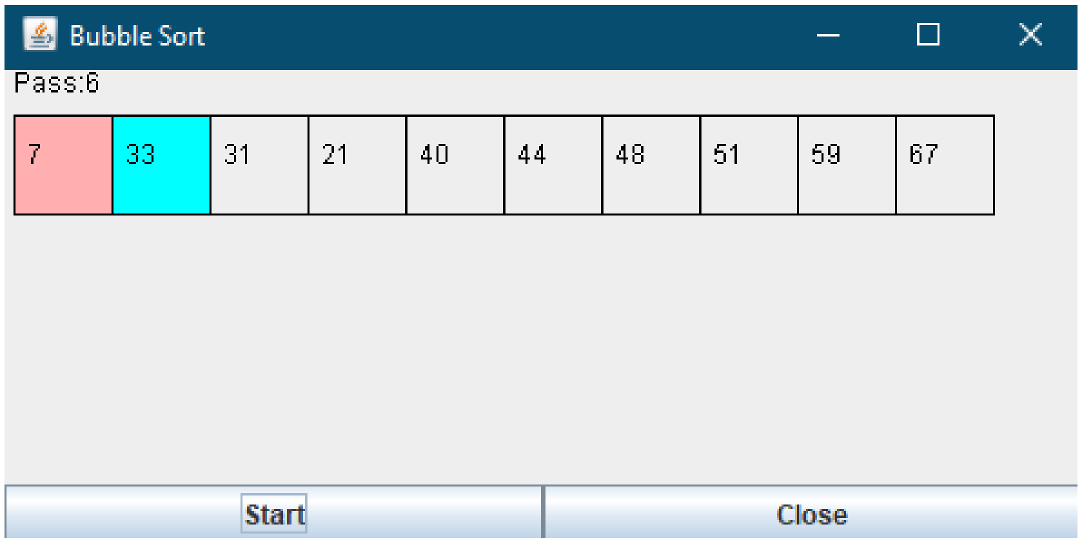


Figure 5.6: Bubble Sort Representation Page

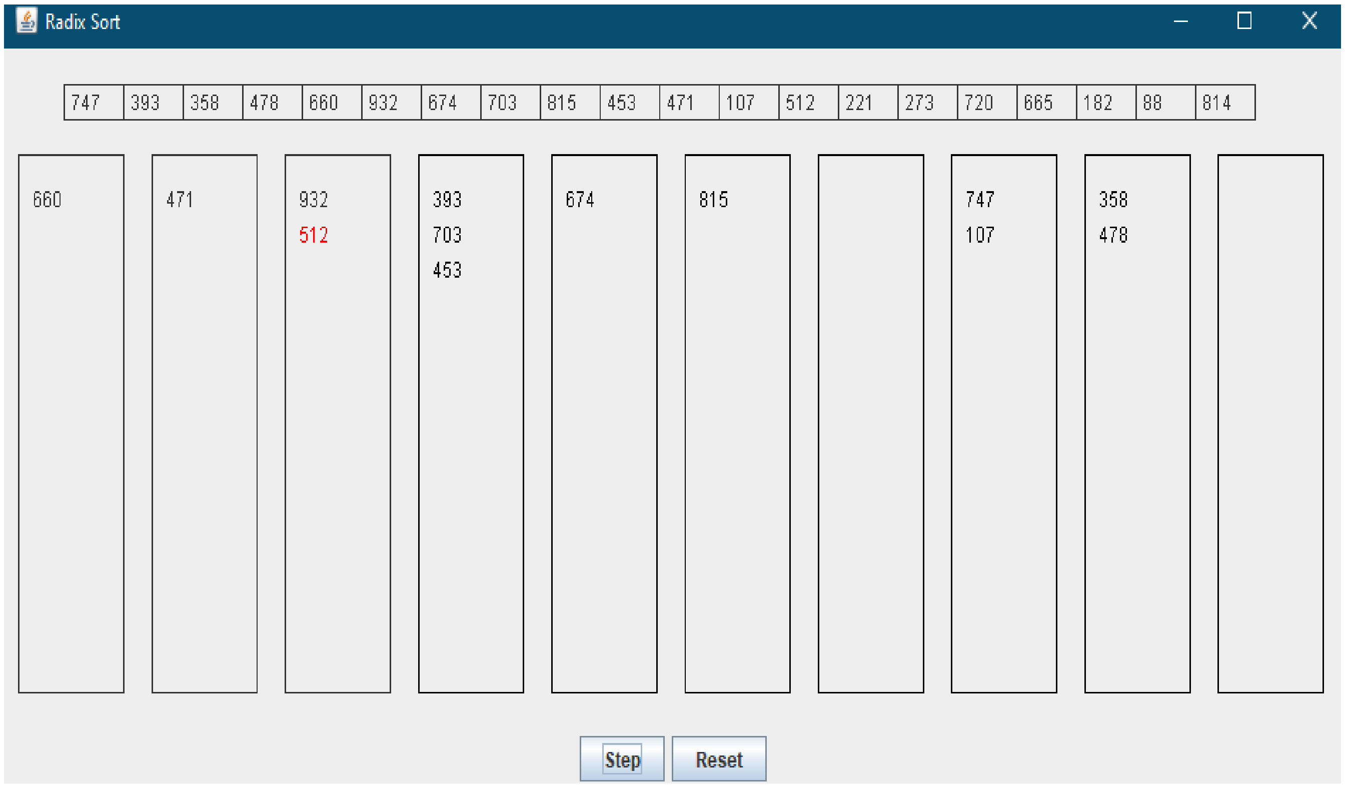


Figure 5.6: Radix Sort Representation Page

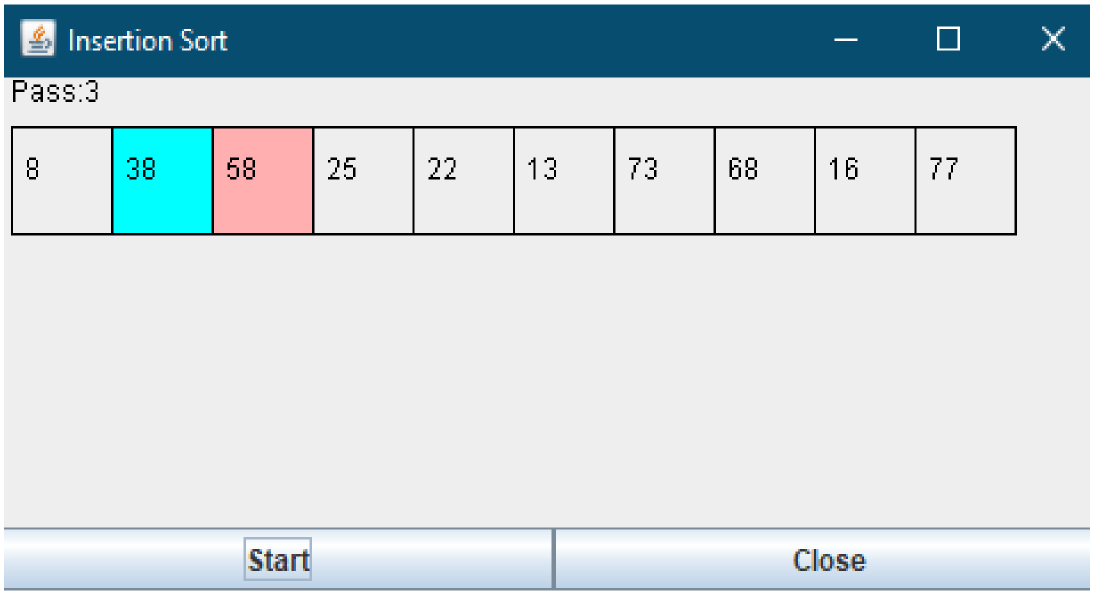


Figure 5.6: Insertion Sort Representation Page

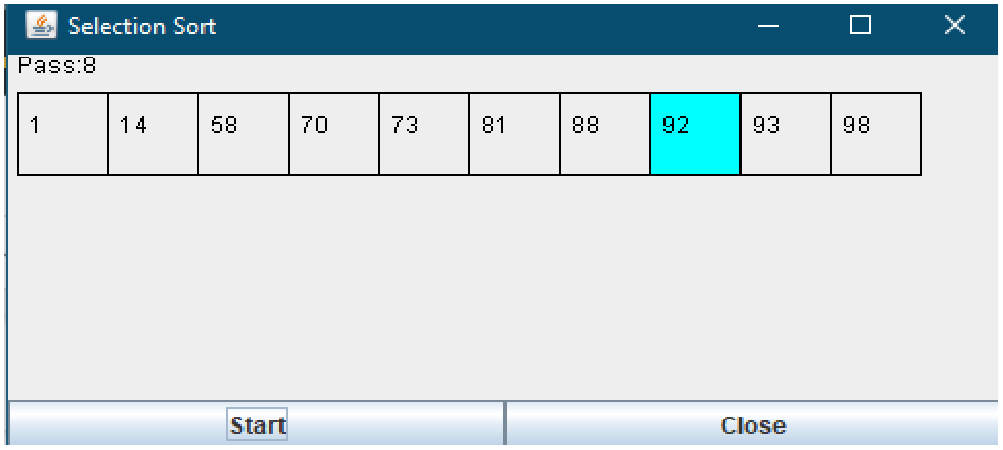


Figure 5.6: Selection Sort Representation Page

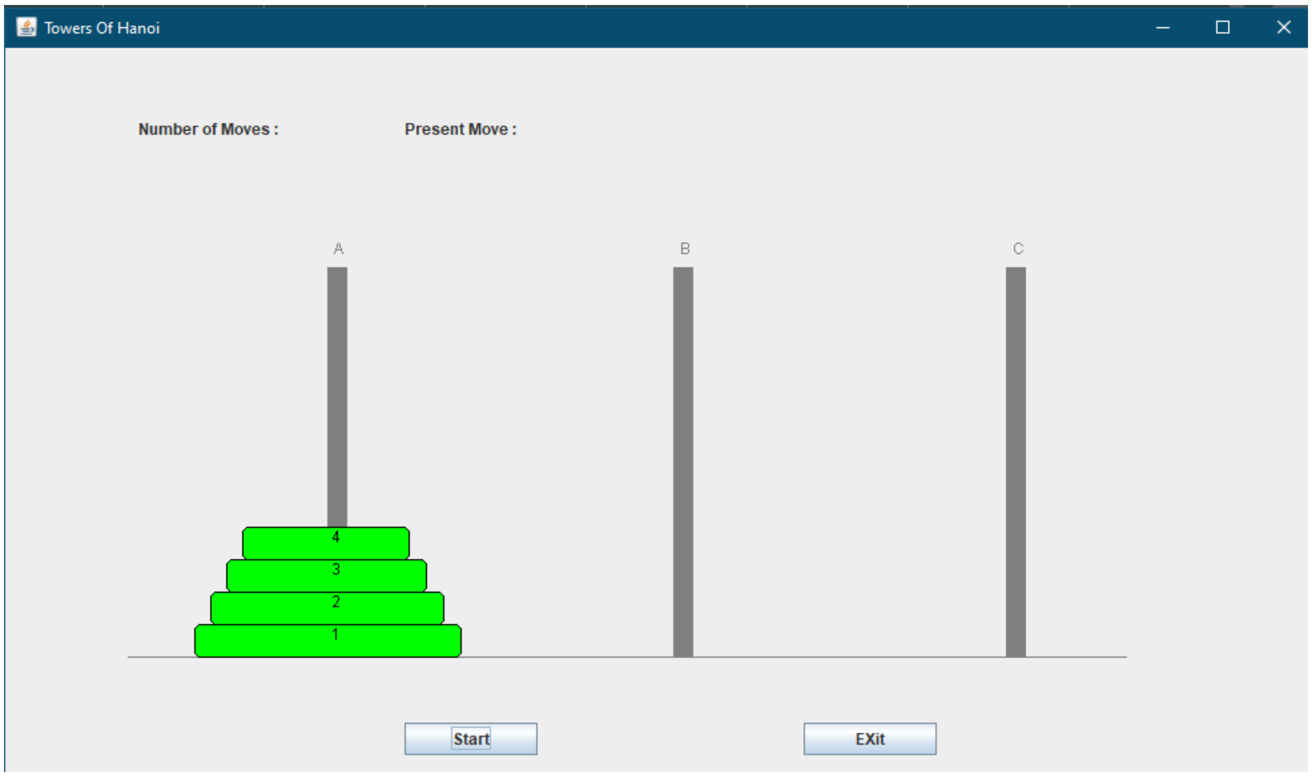


Figure 5.6: Tower of Hanoi Representation Page

5. CONCLUSION

In conclusion, data structures are a great tool to computer science and the professionals who utilize them. Data structures have their advantages and disadvantages like everything in our lives. Only advanced users can make changes to data structures, and any problem involving data structure will need a professional to rectify. In this system every computer science student will be able to see the implementation by seeing these graphically. Here some data structures are taken to show the use of these graphical presentations. In this project report we discuss the mechanisms and how we can design these in our project and summarize these algorithms and how they work and show the example of these in graphically. The significance of the project is that any one will be able to understand the basic concept of these algorithms which are used in our project. Because each step is visualized graphically in every section in our project.