Visualization Visualizer

Abstract — Sequences of execution of algorithms in an interactive manner using multimedia tools are employed in this paper. It helps to realize the concept of fundamentals of algorithms such as searching and sorting method in a simple manner. Visualization gains more attention than theoretical study and it is an easy way of learning process. We propose methods for finding runtime sequence of each algorithm in an interactive way and aims to overcome the drawbacks of the existing character systems. System illustrates each and every step clearly using text and animation. Comparisons of its time complexity have been carried out and results show that our approach provides better perceptive of algorithms

Keywords — Algorithms, Sorting, Visualization.

I. INTRODUCTION

VISUALIZATION of algorithms’ sequence is an important process to learn various hidden steps, which are involved dynamically. The advantages of visualizing algorithms are: Easy to learn with different set data. Understand hidden steps of algorithms, Memory usages and Time management strategy. Algorithms and data structures are two essential courses for any computer science curriculum. Both teachers and students face constant challenges to teach and learn the concepts of algorithms and data structures. Students struggle to comprehend theoretical models and core concepts of these courses. Becker et al. [1] states that it is often difficult to let students understand a working knowledge of the creation and operation of data structures by using traditional communication and delivery methods.

We explain the following algorithms in this paper.

Sorting Algorithms: 1) Selection Sort
2) Bubble Sort
3) Insertion Sort

II. VISUALIZING SEARCHING TECHNICS

Interaction with our system can be achieved through the exploration of existing default visualizations, through the direct manipulation of graphical objects. This will provide the way by selection of concepts (Searching or Sorting)

III. SORTING ALGORITHM

In sorting algorithms the user has to give how many number of inputs and the set of data. Then select the particular algorithm from the list and then the visualization of the selected algorithm is shown with the given inputs.

A. Selection sort

The algorithm works as follows:
1. Find the minimum value in the list
2. Swap it with the value in the first position
3. Repeat the steps above for remaining of the list (to the consecutive positions)

Algorithm Selection Sort (List, N)
List -the elements to be put in order
N -the number of elements in the list
pos -the position of the first element to be exchanged
pos1 -the position of the second element to be exchanged
t -temporary variable
min-variable to store the Minimum value
thread_var–variable to control the movement of the label to be exchanged

Step1: calls ct.d() //Starts the timer
Step2: For i=0 to N-1 do
Step3: min =i
Step 4: For j=i+1 to N do
Step 5: if (List[j] < List[min]) then
Step 6: min=j;
Step 7: End if
Step 8: End for
Step 9: pos=i
Step 10: pos1=min
Step 11: if (min!=i)
Step 12: while (thread_var<3)
Step 13: Calls repaint () //It exchanges the elements
Step 14: End While
Step 15: End If
Step 16: t=List[i]
Step 17: List[i]=List[min]
Step 18: List[min]=t
Step 19: thread_var=0
Step 20: End For
Step 21: calls ct.d1() //function that stops the timer

A. Bubble sort

The bubble sort algorithm makes number of passes through the list of elements. On each pass it compares adjacent element values. If they are out of order, they are swapped. We start each of the passes at the beginning of the list. On first pass, once the algorithm reaches the largest element, it will be swapped with all of the remaining elements, moving it to the end of the list. The second pass will move the second largest element down the list until it is in the second to last location. The process continues with each additional pass moving one more of the larger values down in the list. If on any pass there are no swaps, all of the elements are now in order and the algorithm can stop.

Algorithm Bubble Sort (List, N)
List - the elements to be put in order
N - the number of elements in the list
pos1 - the position of the first element to be exchanged
pos - the position of the second element to be exchanged
t - temporary variable
thread_var - Variable to control the movement of the labels to be exchanged

Step 1: calls ct.d1() //function to start the timer
Step 2: For i=N-1 to 0 Step -1 do
Step 3: For j=0 to i do
Step 4: if (List[j]>List[j+1] then
Step 5: pos=j
Step 6: pos1=j+1
Step 7: while (thread_var<3)
Step 8: Calls repaint () //It exchanges the elements
Step 9: End While
Step 10: t=List[j]
Step 11: List[j]=List[j+1]
Step 12: List[j+1]=t
Step 13: thread_var=0
Step 14: End if
Step 15: End for j
Step 16: End for i
Step 17: calls ct.d1() //function that stops the timer

B. Insertion sort

The Insertion sort is a sorting algorithm, which sorts the array by shifting the elements one at a time. It iterates the input elements by growing the sorted array at each iteration. It compares the current element with the largest value in the sorted array. If the current element is greater, then it leaves the element in its place and moves on to the next element else it finds its correct position in the sorted array and moves it to that position. This is done by shifting all the elements, which are larger than the current element, in the sorted array to one position ahead.

Algorithm for Insertion Sort

Step 1 – If it is the first element, it is already sorted. return 1;
Step 2 – Pick next element
Step 3 – Compare with all elements in the sorted sub-list
Step 4 – Shift all the elements in the sorted sub-list that is greater than the value to be sorted
Step 5 – Insert the value
Step 6 – Repeat until list is sorted

IV. RESULTS AND DISCUSSIONS

In the establishing requirements activity, we conducted a detailed survey on the existing literature and we have identified a set of pedagogy, usability, and accessibility goals and their relevant features that are listed in Table I, II and III. The incorporation of these requirement goals and features in an AV tool can promote students’ engagement and learning of the concepts of algorithms and data structures.

TABLE I. FEATURES INFLUENCING PEDAGOGICAL GOALS

<table>
<thead>
<tr>
<th>Goals</th>
<th>Features</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase Understanding</td>
<td>Narrative content and textual explanations Integrating explanations with the step of the visualization Feedback on student action Integrating context sensitive help and help files Structural view of algorithms Data input facilities</td>
<td>[5]</td>
</tr>
<tr>
<td>Promote Active Engagement</td>
<td>Ask students to construct or to customize an AV Allow users to provide input to the algorithm Manipulate and direct aspects of visualization Rewind capability Speed variation Restart/pause algorithm Step forward Complete algorithm Structural view of algorithms</td>
<td>[1]</td>
</tr>
</tbody>
</table>
It’s important that the system be memorable so that the learners don’t have to re-learn it when they come back. This also reduces stress and allows learners to jump right to the study of algorithms.

**TABLE II. FEATURES INFLUENCING USABILITY GOALS**

<table>
<thead>
<tr>
<th>Goals</th>
<th>Features</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learnability</td>
<td>Familiarity, e.g., familiar controls Consistency across top- and sub-elements Generalizability to similar systems Predictability of system operation Simplicity of interface</td>
<td>[5]</td>
</tr>
<tr>
<td>Memorability</td>
<td>Common interface (template) that supports multitude of animations Discoverable/intuitive controls and behaviors Minimize cognitive load</td>
<td>[6]</td>
</tr>
<tr>
<td>Easy-to-use</td>
<td>Common interface (template) that supports multitude of animations Discoverable/familiar controls Clarity of model and concepts</td>
<td>[7]</td>
</tr>
<tr>
<td>Robustness</td>
<td>Integration with database for course management reasons Hypertext explanations of the visual display Integration of context sensitive help and help files Integration of predefined defaults Implementation of forward and backward error recovery</td>
<td>[5]</td>
</tr>
<tr>
<td>Effectiveness</td>
<td>Visibility on system status Consistency and standard User control and freedom Error prevention Recognition rather than recall Flexibility and efficiency of use Help users recognize, diagnose, and recover from errors</td>
<td>[1]</td>
</tr>
</tbody>
</table>

V. CONCLUSION AND FUTURE ENHANCEMENT

This system is implemented for visualizing some of the searching and sorting algorithms. This is a helpful tool for all kinds of learners/scholars to easily understand the implicit sequences of algorithm. Here the users are allowed to select the options, either searching or sorting.

Then they are allowed to give input and they can select the algorithms from the list and the algorithm is explained visually. In future to enhance and continue this project, the system may include more algorithms for searching and sorting. Visualization can also be done for other kinds of algorithms. Voice can further be included to the system, to give more interaction for the end users.

**REFERENCES**


