2009 International Conference on Advances in Recent Technologies in Communication and Computing

Visualizing Sequence of Algorithms for Searching and Sorting

Bremananth R
Dept. of Computer Applications,
Sri Ramakrishna Engineering College,
Coimbatore, Tamil Nadu, India-641022
e-mail: bremresearch@gmail.com.

Radhika V, Thenmozhi S
Dept. of Computer Applications,
Sri Ramakrishna Engineering College,
Coimbatore, Tamil Nadu , India
e-mail:vj.radhika@gmail.com,tthenmozhis@gmail.com.

***Abstract*—**In this paper sequence of execution of algorithms are explained visually in an interactive manner. It helps to realize the fundamental concept 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 of each algorithm 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, Searching, 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 of data, Understand hidden steps of algorithms, Memory usages and Time management strategy. The first well-known visualization presented by Baecker, it was in videotape format. It shows the animation of nine different sorting algorithms. This videotape allows students to watch the behavior of the algorithm rather than try to imagine its actions from a verbal explanation or from several static images [1]. *Brown Algorithm Simulator and Animator (BALSA)* is a major interactive algorithm animation system developed at Brown University [2]. In this system Students were able to control the animation by starting, stopping and replaying the animations. A later version *BALSA-II* added color and some rudimentary sounds. Brown University created another algorithm animation system TANGO. It does not erase and redraw each image as the previous animation systems did. It is able to produce smoother more cartoon-like animations. A later version of this system is XTANGO [3]. *A New Interactive Modeler for Animations in Lectures (ANIMAL)* is a newer visualization system incorporating lessons learned from pedagogical research Developed at the University of Siegen in Germany [4]. In this paper, some of searching and sorting algorithms are explained visually. Interaction with this tool can be achieved through the exploration of existing default visualizations, through the direct manipulation of graphics objects. This tool is designed for students, instructors and software developers. This will be very interactive which means the user verifies the algorithms by different set of data. We explain the following algorithms in this paper. Searching Algorithms: Sequential Search and Binary Search Sorting Algorithms: Selection Sort and Bubble Sort.

Section 3 describes visualization of searching algorithms Sorting algorithms have been discussed in section 4. Section 5 deals result and analysis of both kinds of algorithms. Finally, concluding remarks and future enhancements are described in section 6.

II. VISUALIZING SEARCHING TECHNIQUES

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) which we want and also select the algorithm based on the selected concept. The inputs for the selected algorithm are obtained from the user. Visualization process starts by clicking the start button. The Buttons Pause and Resume are used to suspend and resume the process of visualization. In both Searching and sorting algorithms, an appropriate message is displayed for each process. In Searching algorithms, when process starts, the component (labels) that contains the element to be searched moves through the list based on the selected algorithm until a match is found. In sorting algorithms, when process starts, the positions of the components (labels) that contain the elements to be sorted are interchanged. This process continues until all the elements are sorted. It contains the following algorithms.

III.SEARCHING ALGORITHM

In searching algorithms the user has to give the total numbers of inputs, the set of data and the element to be searched. Then select the particular algorithm from the list and then the visualization of the selected algorithm is shown with the given inputs.

A. Sequential Search

A Sequential search, also known as linear search that is suitable for searching a minimum number of inputs. It checks every element of a list one at a time in a sequential order until a match is found.

Algorithm: Sequential_Search(List,Target,N)
List represents the elements to be searched, *Target* is the value being searched for, *N* is the number of elements in the list, *pos* is the value of the position from where repaint starts.
Step1: call ct.d() //function that starts the timer
Step2: For i=1 to N do
Step3: if (Target=List[i]) then
Step4: Stop the program
Step5: End if
Step6: pos= i//stores the position

978-0-7695-3845-7/09 $26.00 © 2009 IEEE
DOI 10.1109/ARTCom.2009.20

Step7: call repaint();// It moves searching element through out the list until a match is found
Step8: End for
Step9: call ct.d1() //function that stops the timer

After choosing the algorithm from the given list, Timer is started using the function *ct.d()*. The element to be searched is called the *Target*. For elements 1 to *n* the *Target* is compared with the elements in the *List* starting from the first element. In each step the index value of the element in the *List* with which the *Target* is compared is stored in variable *pos*. Then *repaint()* function is called to move the element to the next position of the *List*. When the *Target* matches with the element of the *List* the program stops and timer is stopped by calling the function *ct.d1()*.

B. Binary Search

In binary search, we first compare the target with the element in the middle position of the array. If there's a match, we can return immediately. If the target is less than the middle element, then the target must lie in the lower half of the array; if the target is greater than the middle element then target must lie in the upper half of the array. So we repeat the procedure on the lower (or upper) half of the array.

Algorithm: Binary_Search (List, Target, N)
List represents the elements to be searched, *Target* is the value being searched for, *N* is the number of elements in the list, *pos* is the value of the position from where repaint starts *thread_var* is variable to control the movement of the labels to be exchanged
Step1: call ct.d()//function that starts the timer
Step2: start=1
Step3: end=N
Step4: while start<=end do
Step5: middle= (start+end)/2
Step6: if (List[middle] < Target) then
Step7: start = middle + 1 ; pos=start
Step8: While (thread_var<3) do
Step9: call repaint( )// It moves searching element through specified list until a match is found
Step10: End While
Step11: End if
Step12: if (List[middle] > Target) then
Step13: end = middle - 1; pos =end;
Step14: call repaint( ) // It moves searching element through specified list until a match is found
Step15: End if
Step16: if (List[middle] =Target) then
Step17: Stop the program
Step18: End if
Step19: End while
Step20: call ct.d1()//function that stops the timer

The Binary search option is chosen from the list. The program starts and the timer is also started using the function *ct.d()*. The value 1 is stored in variable *start* and *N* in variable *end*. It finds the *middle* value using the formula *middle = (start + end)/2*. It compares the *middle* element with the *Target*. When the *Target* is greater than the *middle* element, it starts searching the upper half by assigning start=*middle+1* and storing the start value in variable *pos* and *repaint()* function is called to move the *Target* element to the first position of the upper half. It applies the above steps for the upper half until the *Targe*t is found.

It stops when *Target* is found. When the *Target* is less than the *middle* element ,the program starts searching in the lower half by assigning *end=middle-1* and *end* value is stored in variable *pos* and *repaint()* function is called to move the *Target* element to the lower half. It applies the above steps for the lower half until the *Target* is found. When *Target* is equal to the *middle* element the *middle* value is stored in variable *pos* and *repaint()* function is called to place the *Target* in middle. Then the program stops and the timer is stopped using the function *ct.d1()*. It searches only in a sorted list. When the list is unsorted, it sorts the list first and then starts searching.

IV.SORTING ALGORITHM

In sorting algorithms user has to give the total numbers of inputs and the set of data. Then select the required algorithm from the list and the visualization of the selected algorithm is shown with the given inputs.

A. Selection sort

The algorithm first finds the minimum value in the list and swaps it with the value in the first position. This process is repeated for the remaining elements of the list for consecutive positions .

Algorithm: Selection_Sort (List, N)
List represents the elements to be sorted, *N* is the number of elements in the list, *pos* is the position of the first element to be exchanged, *pos1* is the position of the second element to be exchanged, *t* is temporary variable, *min* is a variable to store the minimum value of the list, *thread_var* is variable to control the movement of the labels to be exchanged.
Step1: call ct.d() //Starts the timer
Step2: For i=0 to N-1 do
Step3: min =i
Step4: For j=i+1 to N do
Step5: if (List[j] <List [min]) then
Step6: min=j
Step7: End if
Step8: End for
Step9: pos=i ; pos1=min
Step10: if (min! =i)
Step11: while (thread_var<3)
Step12: Call repaint () //It Exchanges the elements
Step13: End While
Step14: End If
Step15: t =List [i]; List [i] = List [min]
Step16: List [min] =t; thread_var =0
Step17: End For
Step18: call ct.d1()//function that stops the timer

The algorithm starts by selecting the selection sort option from the list and the timer is started using the function ct.d().From the first element to the last element, find the smallest element and its position is assigned to the variable min. Variable pos is assigned the value i that represent the

position in the array starting from 1 and variable pos1 is assigned the value min. Then repaint() function is called to swap the values in the positions given in variable pos and pos1.The process continues until the list is sorted and the timer is stopped using the function ct.d1().

B.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.
Algorithm: Bubble_Sort (List, N)
List is the elements to be put in order, N is the number of elements in the list, *pos* is the position of the first element to be exchanged, *pos1* is the position of the second element to be exchanged, *t* is temporary variable *thread_var* is Variable to control the movement of the labels to be exchanged.
Step1: call ct.d() //function to start the timer
Step2: For i=N-1 to 0 Step -1 do
Step3: For j=0 to i do
Step4: if (List[j]>List [j+1]) then
Step5: pos=j; pos1=j+1
Step6: while (thread_var<3)
Step7: Call repaint () //It Exchanges the elements
Step8: End While
Step9: t=List[j]; List[j] =List [j+1]
Step11: List [j+1] =t; thread_var=0
Step12: End if
Step13: End for j
Step14: End for i
Step15: call ct.d1()//function that stops the timer

Choose the algorithm from the given list. The program starts and the timer is started using the function ct.d().The bubble sort algorithm makes N-1 number of passes through the list of elements where N is the number of input values. On each pass it compares adjacent element value. If they are out of order their position are stored in variables pos and pos1 and repaint() function is called to swap the elements in the two positions. The program stops after the *N-1* number of passes and the *List* is sorted. The timer is stopped using the function *ct.d1()*.

V.RESULT ANALYSIS

A.Analysis of Searching Algorithms

Table I depicts the comparison of searching algorithms with average values. In this, number of inputs are 8. Its average runtime is given in seconds. From analyzing Fig.1, we came to know that binary search would be time consuming than sequential search. This is due to sorting the data before searching the elements. But basically binary search will be less time consuming one while entering a sorted list as input to this algorithm. Sequential search does not need a sorted list.

TABLE I COMPARISON OF SEARCHING ALGORITHMS-WITH AVERAGE VALUES

No. of Inputs	Sequential Searching in Seconds	Binary Searching in Seconds
0	0	0
1	1.3	1.3
2	1.9	3.1
3	2.6	7.8
4	3.2	23.2
5	4.3	27
6	6.4	32.3
7	8	51.2
8	9.6	60.8

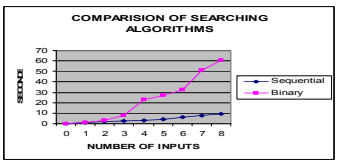

Figure.1 Comparison of searching algorithms with average values

B. Analysis of Sorting Algorithms

Table II illustrates the comparison of the sorting algorithms with average values. These are analyzed with 8 input values. Its average runtime is given in seconds. From analyzing Fig.2, we came to know that Selection sort is less time consuming when compared to Bubble sort. Among the sorting algorithms Bubble sort will be the most time consuming algorithm because of the number of passes it makes.

TABLE II COMPARISON OF SORTING ALGORITHMS-WITH AVERAGE VALUES

No. of Inputs	Selection Sorting in Seconds	Bubble Sorting in Seconds
0	0	0
1	0	0
2	2.8	2.8
3	5.8	6.7
4	7.6	13.4
5	9.6	25
6	16.3	35.5
7	16.8	40.3
8	17.3	55.7

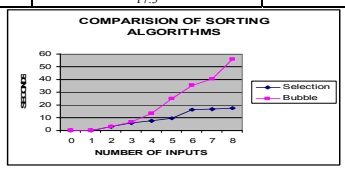

Figure.2 Comparison of sorting algorithms with average values

VI.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 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, we can compare their order of complexity and space complexity. The system can include more algorithms for searching and sorting. Visualization can also be done for other kinds of algorithms. Voice can be included to the system, to give more interaction for the users.

VII. REFERENCES

[1] Baecker, R. *Sorting out Sorting*, Narrated colors videotape, 30 minutes, presented at ACM SIGGRAPH, 1981.
[2] Marc. H.Brown and J. Hershberger, *Color and sound in algorithm animation,* IEEE Computer, 25(12) 1992,pp.:52-63.
[3] J.T.Stasko, *TANGO, A framework and system for algorithm Animation computer*, 23(9),1990,pp:27-39.
[4] G.Rossling, M.Schuler, and B.Freisleben, *The ANIMAL algorithm Animation Tool*, Proceedings of the ItICSE 2000 conference, pp 37- 40.
[5] Jeffrey J.McConnell, *Analysis of Algorithms*, Narosa Publications pvt.ltd, 2001.
[6] Ellis Horowitz, Sartajsahni and Sanguthevar Rajasekaran *Fundamentals of Computer Algorithms*, Galgotia Publications, 2007.