CALIFORNIA STATE UNIVERSITY, SAN BERNARDINO

School of Computer Science
And Engineering

CSE 455

{AlgorithmA }; 2011

Software Requirement Specification
User Manual

CEO: Dr. Concepcion

Project Manager: Anthony Thomas

Assistant Manager: Erwin Soekianto

Team Leads: Matthew Compean,Patrick Ridge, Esmirna Nolasco, and Idalia A. Flamenco

Table Of Contents

1 Introduction

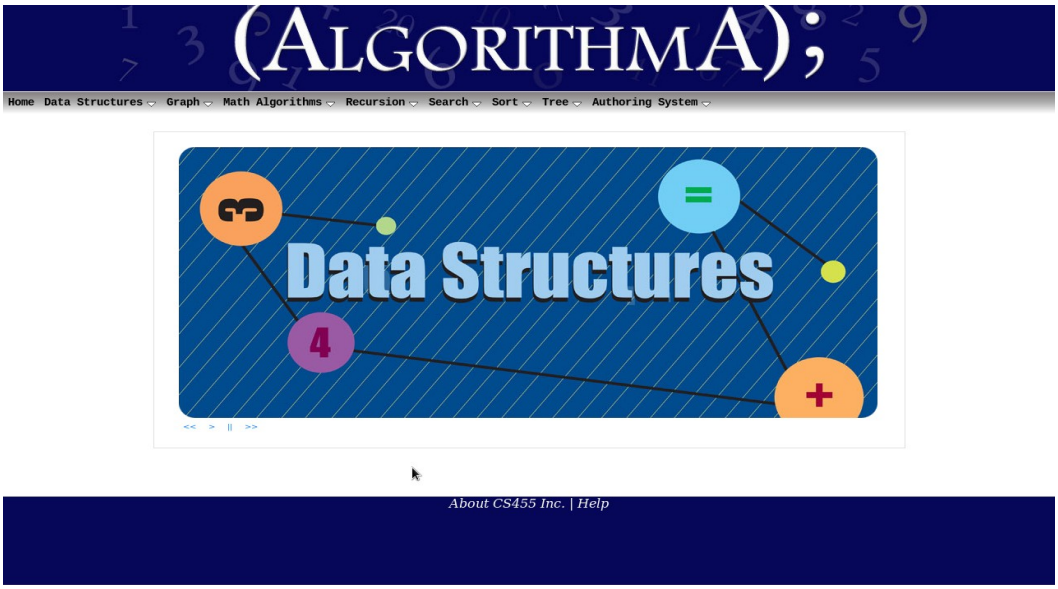
1.1 Who is CSE 455, Inc?

CSE 455, Inc is a fictitious software development company focused to develop educational software for the academic community in Computer Science. This software company is managed by a CEO, a Program Manager, two Assistant Managers, four Team Leads and a group of competent programmers highly qualified in several high-level computer languages.

1.2 What is AlgorithmA 2011?

{AlgorithmA}; 2011 is an end user application designed to provide a depth insight of Computer Science and Engineering areas to students and faculty to explore various mathematical algorithms. The main goal of {AlgorithmA}; is to provide step-by-step visualization of the various types of algorithms already implemented in the preceding {AlgorithmA}; versions.

{AlgorithmA}; 2011 will continue on forward with the progress of {AlgorithmA}; 2010 by continuing the re-implementation of algorithms from previous versions of {AlgorithmA}; that were not re-implemented by the CS455 Inc. 2010. Furthermore, {AlgorithmA}; 2011 will contain an authoring system that will allow the user to write algorithm's and see the result of their pseudo code animated in real time using the animation system implemented by {AlgorithmA}; 2010.



2 Algorithms

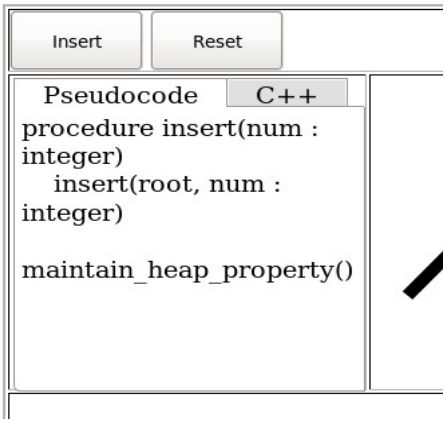
2.1 Pseudo Code and C++ Converter

2.1.1 Overview

This feature allow users to view, enter or edit either Pseudo-code or C++ Code, to see the animations of the code they loaded. System would accept the C++ code or pseudo-code and convert to pseudo-code or c++ code, and compile the code to display the animations. This feature is useful because our target users, CSE 201 and CSE 202 students, are familiar with C++, so it would help them to understand better with the language they are familiar with.

2.1.2 Layout

Here's the layout of Pseudo-code and C++ Converter user interface below



2.1.3 Functionality

- Pseudo Code Tab – To view the pseudo-code
- C++ Tab – to view C++ code
- Load Code – To load the code for animations

3 Data Structures

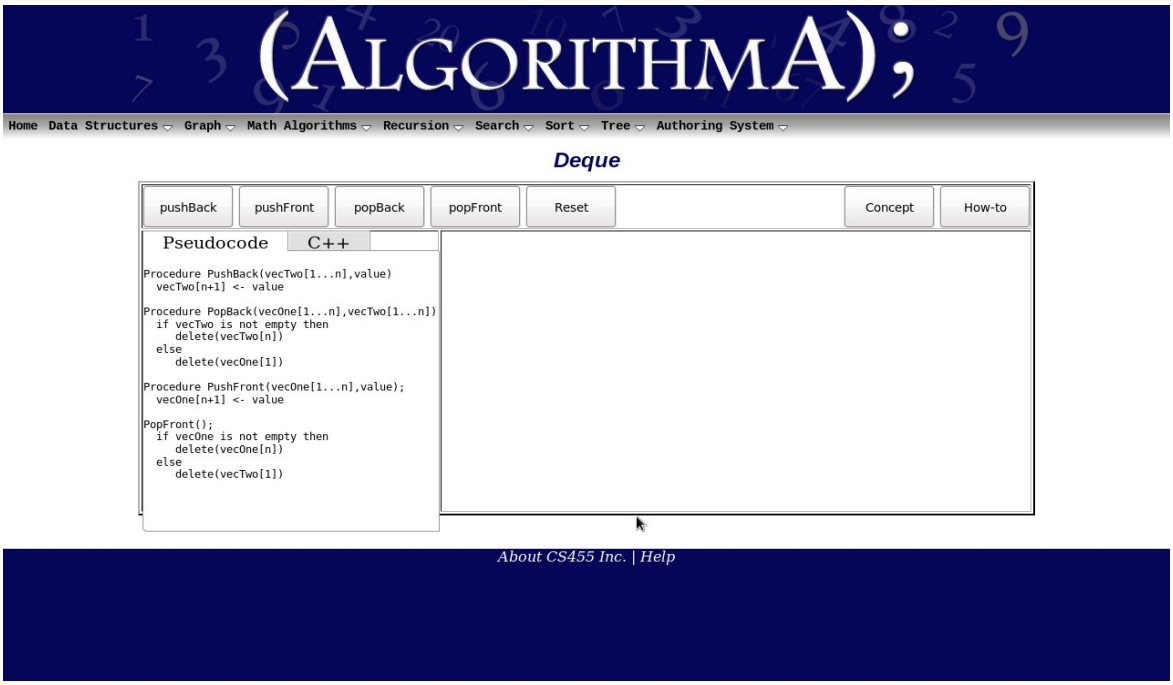
3.1 Deque

3.1.1 Overview

Deque (double-ended queue) is an abstract data structure that implements a queue for which elements can only be added to or removed from the front (head) or back (tail). In a doubly-linked list implementation, the time complexity of all Deque operations is O(1). Additionally, the time complexity of insertion or deletion in the middle, given an iterator, is O(1); however, the time complexity of random access by index is O(n).

3.1.2 Layout

The user will be presented with buttons detailing the concept and the how-to of the function and the data structure algorithm to be used in creating a list. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



3.1.3 Functionality

o Push Front – Push a random element to the structure to the front of the list

o Push Back – Push a random element to the structure to the back of the list

o Pop Front – Remove an element from the front of the list

o Pop Back – Remove an element from the back of the list

o Reset – Reset to the initial layout when first loaded.

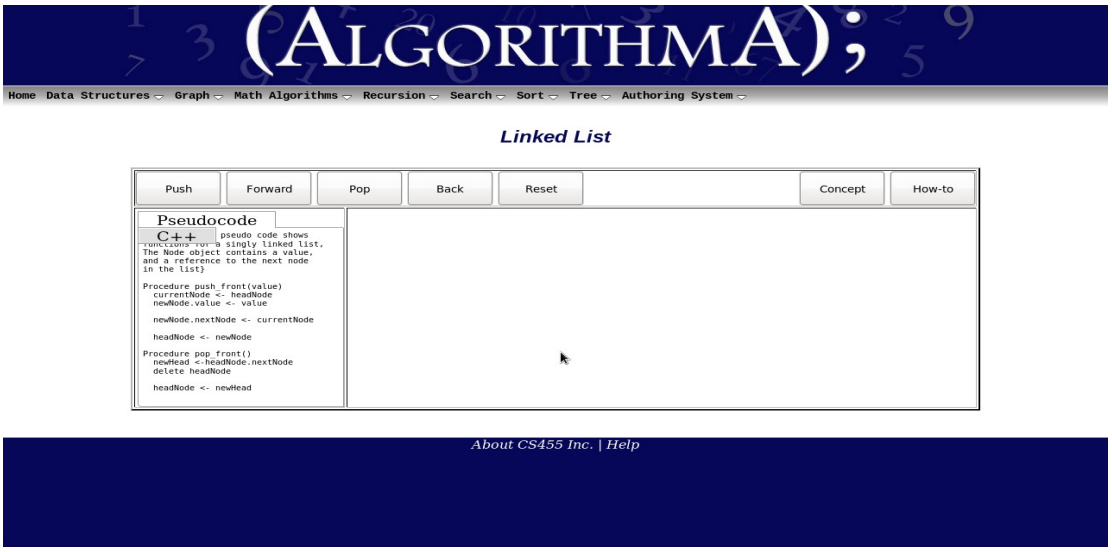
3.2 Linked List

3.2.1 Overview

Linked List is a data structure that consists of a sequence of data records such that in each record there is a field that contains a reference to the next record in the sequence. Linked Lists are among the simplest and most common data structures, and are used to implement many important abstract data structures such as stacks, queues, hash tables, symbolic expressions, skip lists, and many more. Each record of a linked list is often called an element or node. The field of each node that contains the address of the next node is usually called the next link or next pointer. The remaining fields are known as the data, information, value, or payload fields. The head of a list is its first node, and the tail is the list minus that node. In LISP and some derived languages, the tail may be called the CDR of the list, while the payload of the head node may be called the CAR.

3.2.2 Layout

The user will be presented with buttons detailing the concept and the how-to of the function and the data structure algorithm to be used in creating a list. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



3.2.3 Functionality

o Push – Push an element onto the structure of the list

o Forward – Steps through the pseudo code one line at a time in a forward direction

o Pop – Remove an element from the list

o Back – Steps through the pseudo code one line at a time in a backward direction

o Reset – Reset to the initial layout when first loaded.

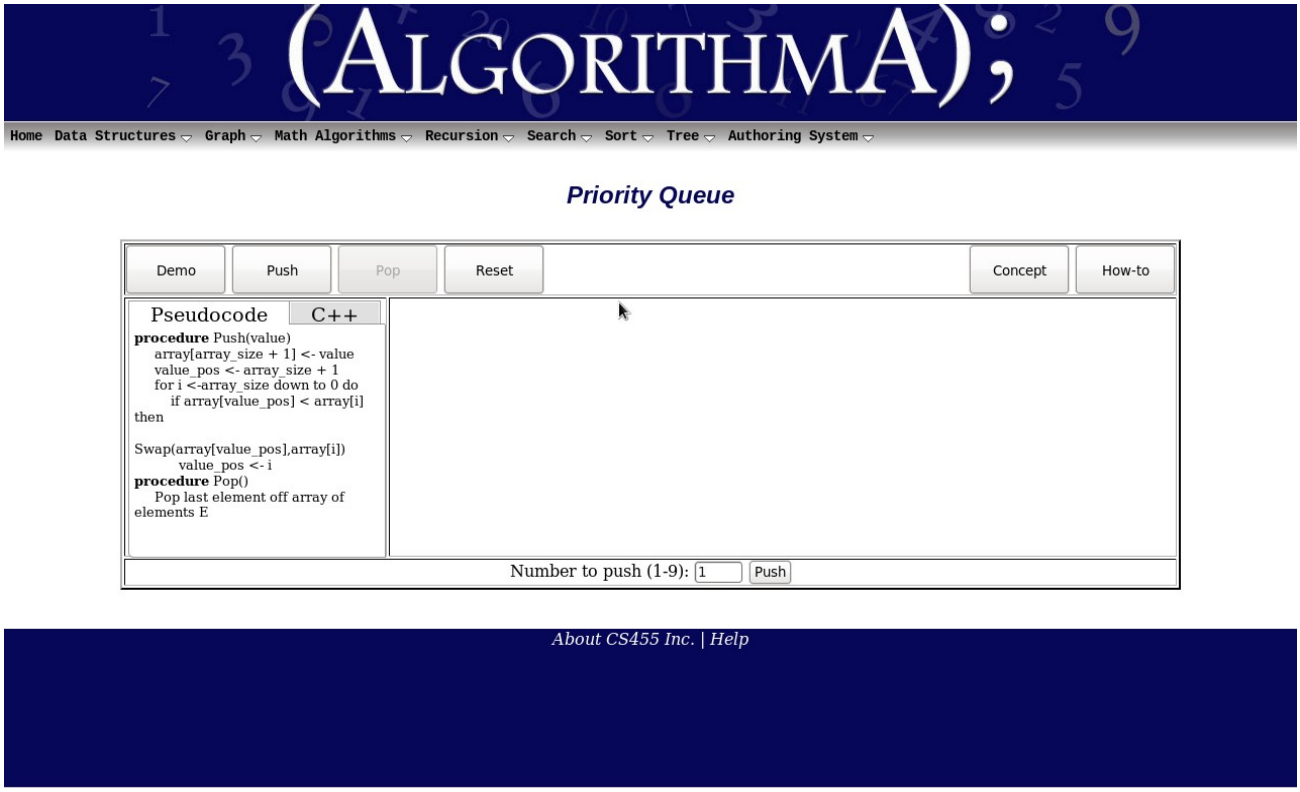
3.3 Priority Queue

3.3.1 Overview

Priority queue is an abstract data type which efficiently finds the item with the highest priority across a series of operations. One can imagine a priority queue as a modified queue, but when one would get the next element off the queue, the highest-priority one is retrieved first. Stacks and queues may be modeled as particular kinds of priority queues. In a stack, the priority of each inserted element is monotonically increasing; thus, the last element inserted is always the first retrieved. In a queue, the priority of each inserted element is monotonically decreasing; hence, the first element inserted is always the first retrieved.

3.3.2 Layout

The user will be presented with buttons detailing the concept and the how-to of the function and the data structure algorithm to be used in creating a list. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



3.3.3 Functionality

o Demo – Starts the priority queue animation

o Push – Push an element onto the structure of the list

o Sort – Activates the sorting feature

o Prioritize – Organizes the list in ascending order

o Pop – Remove an element from the list

o Reset – Reset to the initial layout when first loaded.

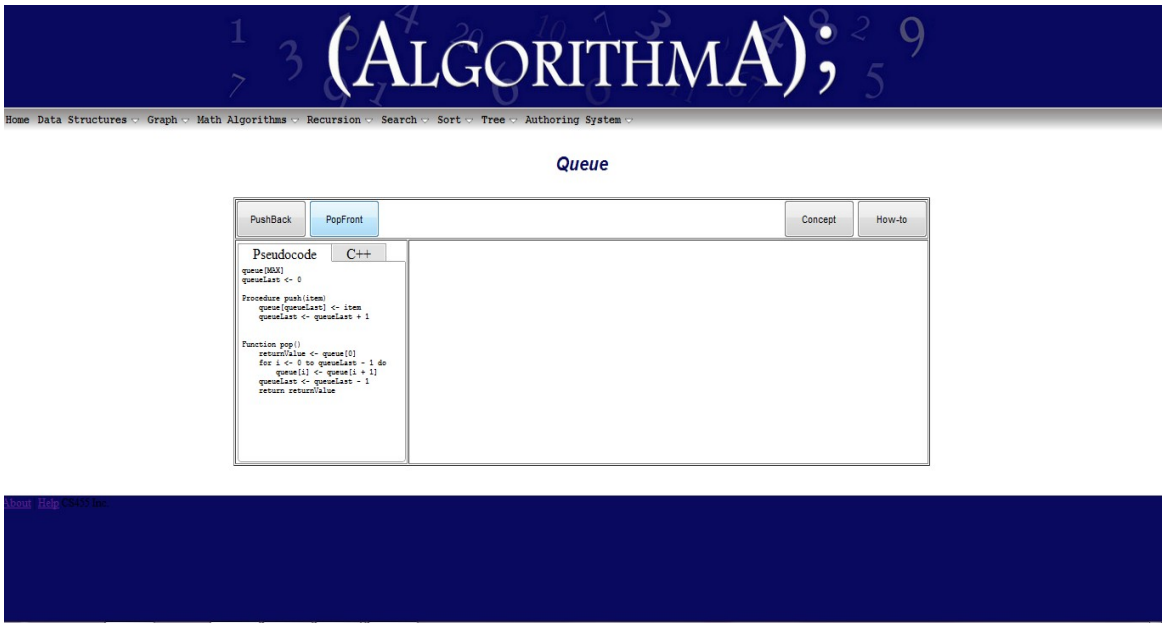
3.4 Queue

3.4.1 Overview

A queue is a particular kind of collection in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position and removal of entities from the front terminal position. This makes the queue a First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that whenever an element is added, all elements that were added before have to be removed before the new element can be invoked. A queue is an example of a linear data structure.

3.4.2 Layout

The user will be presented with buttons detailing the concept and the how-to of the function and the data structure algorithm to be used in creating a list. To the left of the animation work area will be a pseudo-code walk through that is highlighted as instructions are being executed.



3.4.3 Functionality

o Push Back – Push a random element to the structure to the back of the list

o Pop Front – Remove an element from the front of the list

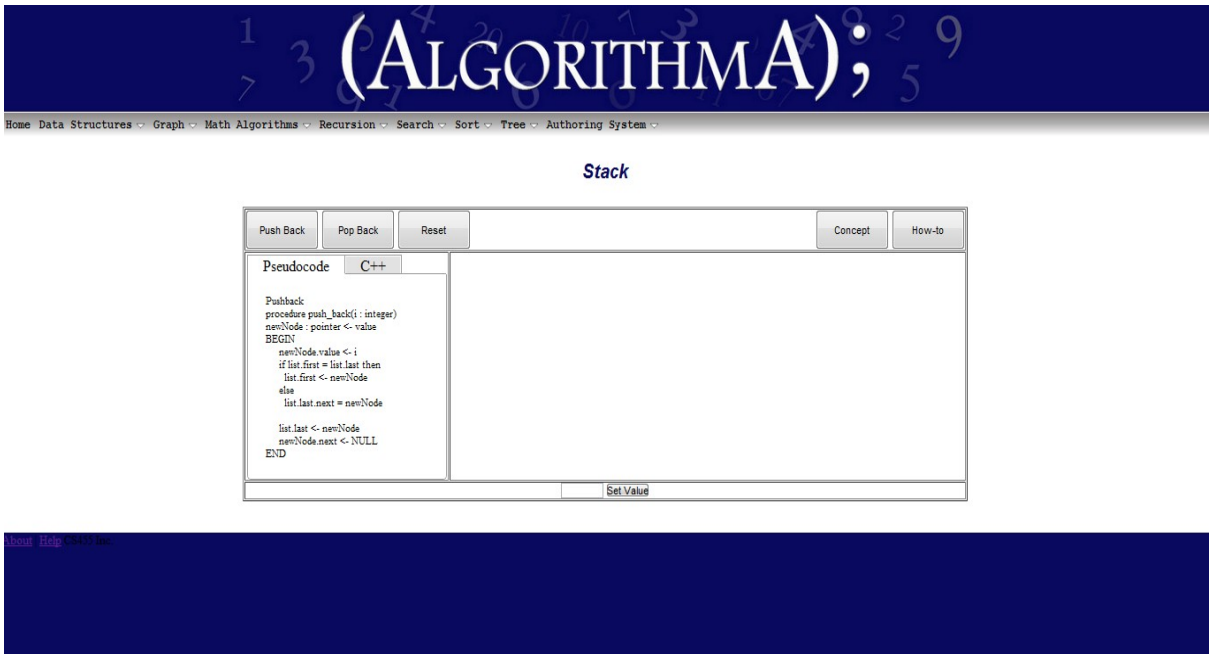
3.5 Stack

3.5.1 Overview

Stack is a last in, first out (LIFO) abstract data type and data structure. A stack can have any abstract data type as an element, but is characterized by only two fundamental operations: push and pop. The push operation adds to the top of the list, hiding any items already on the stack, or initializing the stack if it is empty. The pop operation removes an item from the top of the list, and returns this value to the caller. A pop either reveals previously concealed items, or results in an empty list. A stack is a restricted data structure, because only a small number of operations are performed on it. The nature of the pop and push operations also mean that stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition: therefore, the lower elements are typically those that have been in the list the longest.

3.5.2 Layout

The user will be presented with buttons detailing the concept and the how-to of the function and the data structure algorithm to be used in creating a list. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



3.5.3 Functionality

o Push Back – Push an element onto the structure to the back of the list

o Pop Back – Remove an element from the back of the list

o Reset – Reset to the initial layout when first loaded.

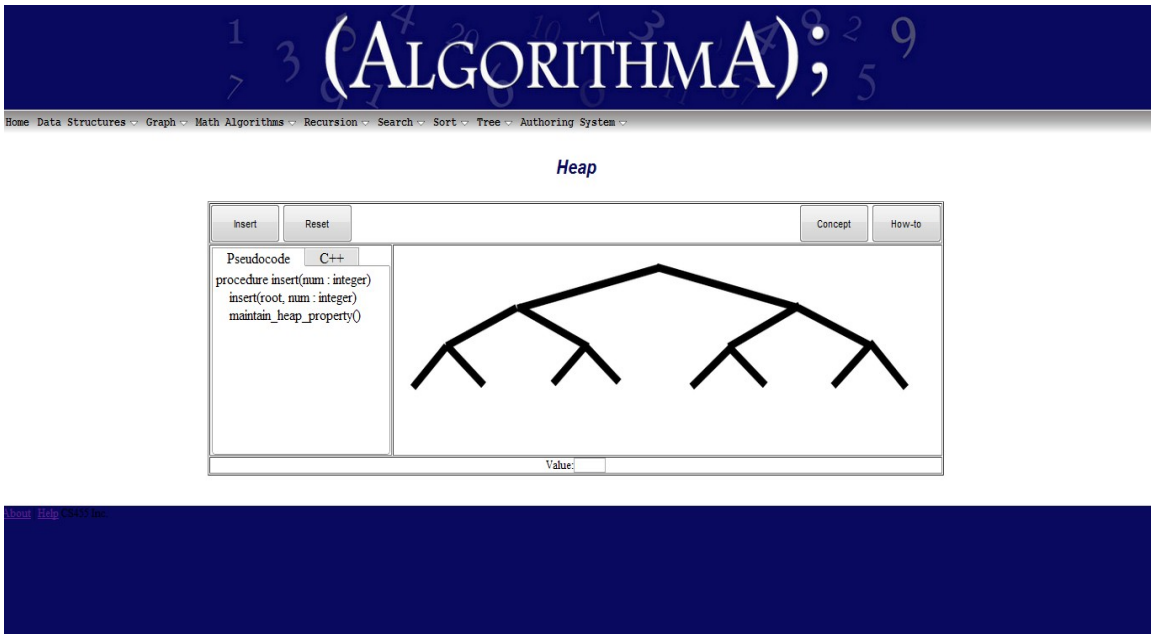
3.6 Heap

3.6.1 Overview

Heap is a data structure that is a specialized tree that works similarly to a priority queue. It follows a strict structure called the heap property where the largest element in the tree is always the root. In order to show the bare minimum of its workings, the walk-through process must step a user through the process of adding, removing, and possibly iteration through the structure itself.

3.6.2 Layout

Already classified under Data Structures, the user will be presented with a brief introduction of the function and the basic already "useful" heap to add and remove from (must have at least 3 elements already present). To the left of the animation work area will be a pseudo-code walk-through that is highlighted as actions are performed.



3.6.3 Functionality

Heap should only contain the bare minimum. This includes:

- Add – Add a random element to the structure to its proper position.

- Remove – Remove a random element from the heap.
- Reset – Reset to the initial layout when first loaded.

4 Graphs

4.1 Critical Path

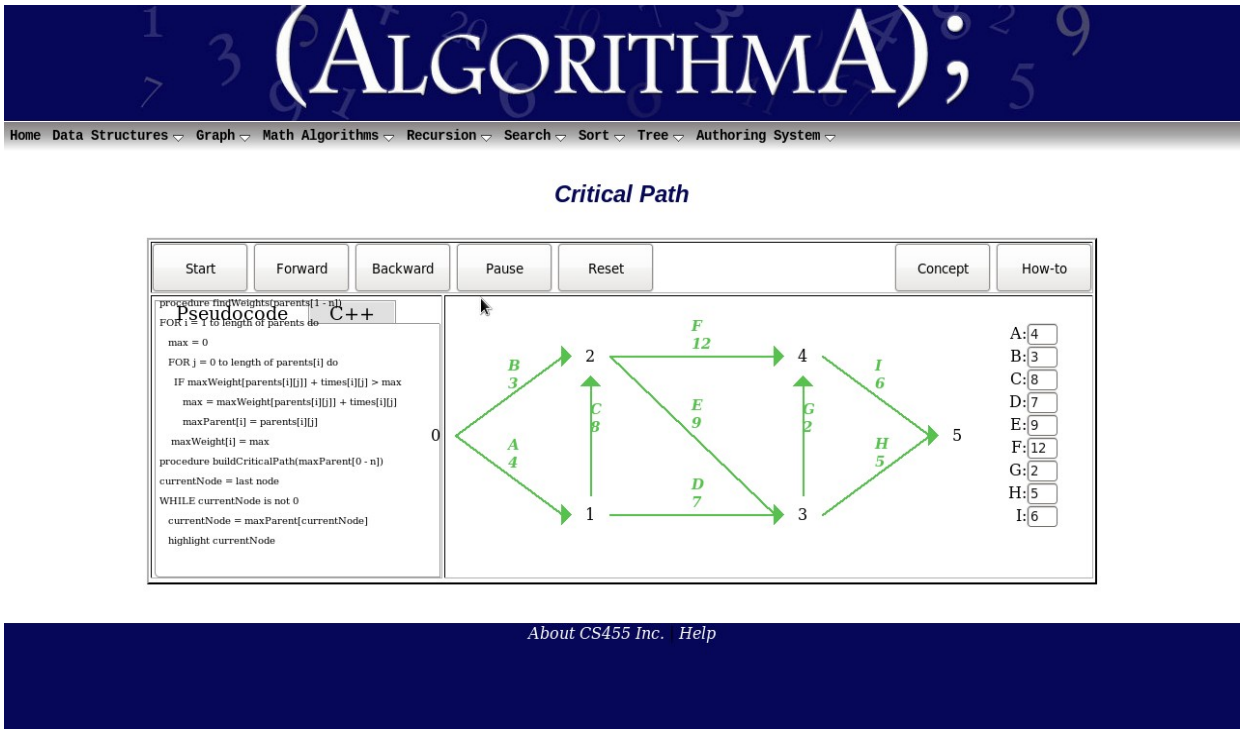
4.1.1 Overview

The critical path method (CPM) or critical path analysis is a mathematically based algorithm for scheduling a set of project activities. It is an important tool for effective project management. The essential technique for using critical path is to construct a model of the project that includes the following:

1. A list of all activities required to complete the project (typically categorized within a work breakdown structure),
2. The time (duration) that each activity will take to completion, and
3. The dependencies between the activities

4.1.2 Layout

The user will be presented with a brief introduction of the function and the basic already "useful" graph to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



4.1.3 Functionality

o Start – Starts the bubble sort animation.

o Forward – Allows the user to step through the sort one step at a time.

o Backward – Steps through the pseudo code one line at a time in a backward direction

o Pause – Interactively allows the user to stop and continue the animation

o Reset – Resets the bubble sort animation to the beginning.

4.2 Dijkstra's Shortest Path

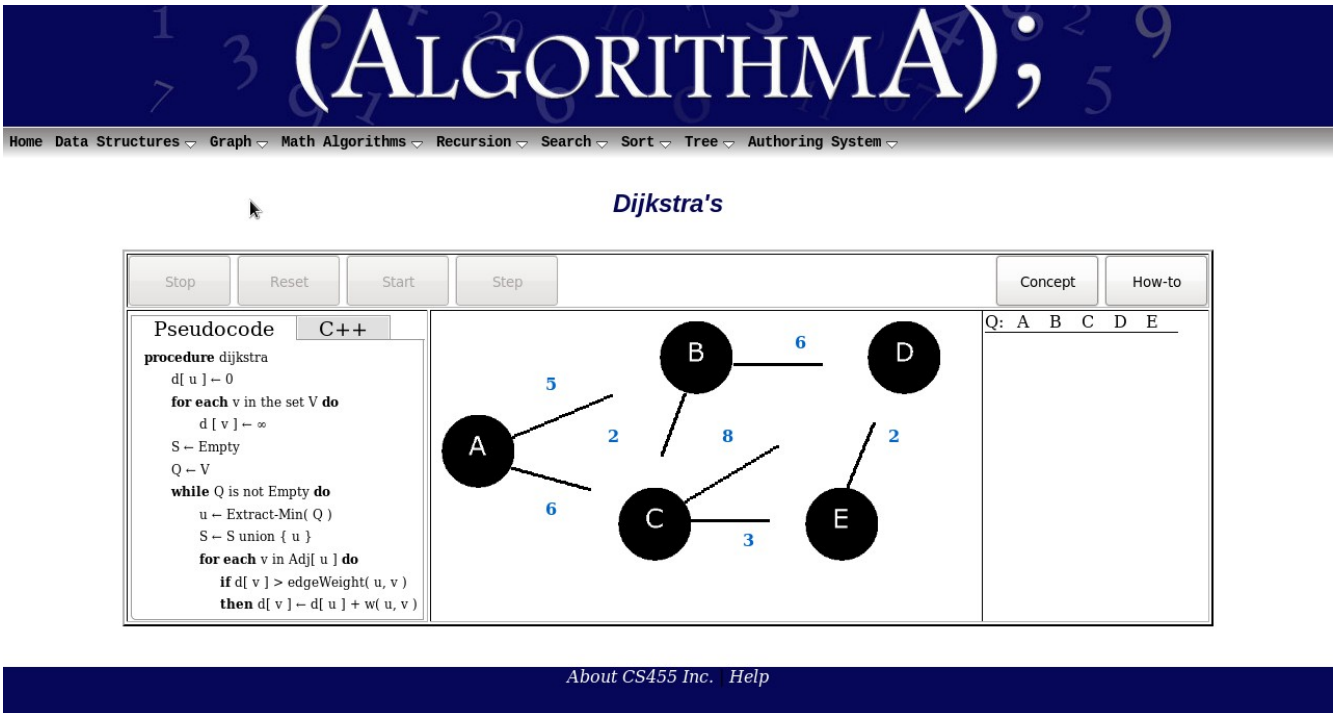
4.2.1 Overview

Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing.

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the verticals of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First).

4.2.2 Layout

The user will be presented with a brief introduction of the function and the basic already “useful” graph to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



4.2.3 Functionality

o Pause – Allows the user to stop the animation and restart it when pressed again.

o Reset – Reset to the initial layout when first loaded.

o Start – Allows the user to run the animation.

o Step – Allows the user to step through the animation one step at a time.

4.3 Kruskal's Minimum Tree

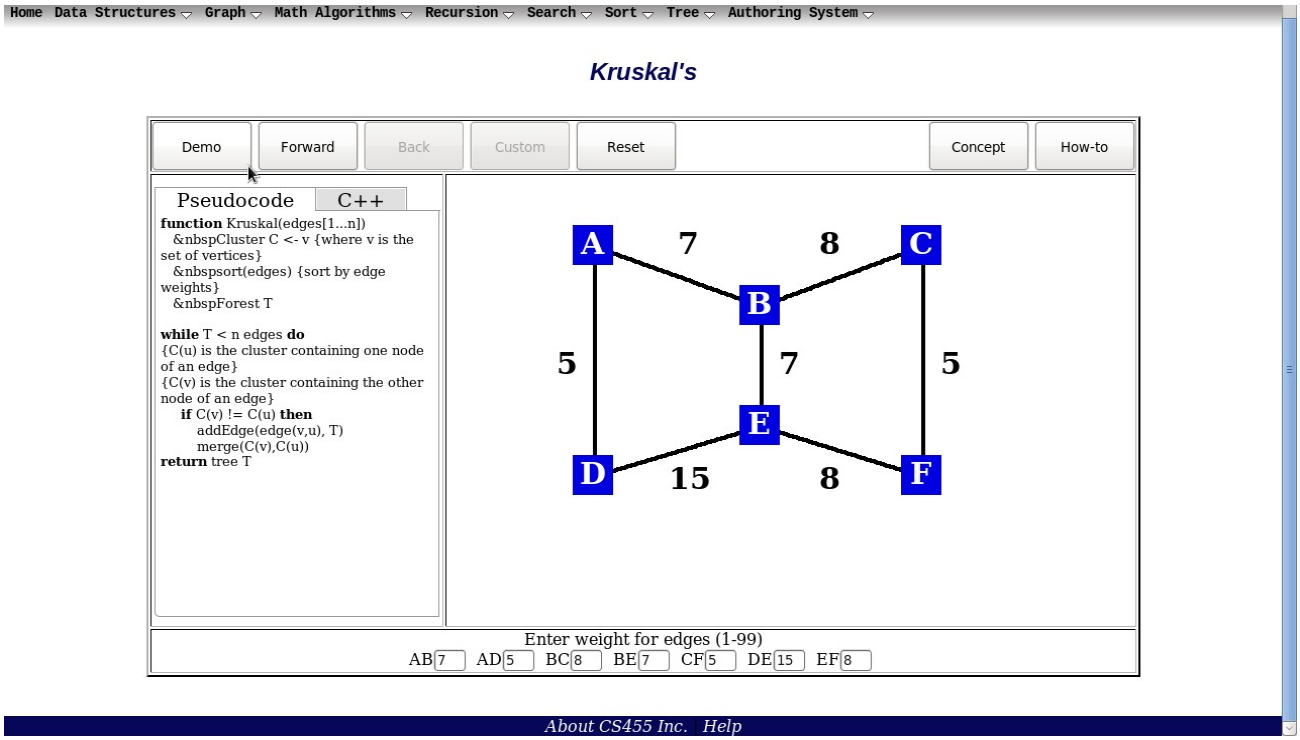
4.3.1 Overview

Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each

connected component). Kruskal's algorithm is an example of a greedy algorithm.

4.3.2 Layout

The user will be presented with a brief introduction of the function and the basic already “useful” graph to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



4.3.3 Functionality

o Demo – Allows user to run a demo of the algorithm.

o Forward – Will allow user to step through the animation one step at a time.

o Back – Will allow user to step backward through animation one step at a time.

o Custom – Allows the user to create an animation of the algorithm from scratch.

o Reset – Reset to the initial layout when first loaded.

4.4 Prim's Minimum Tree

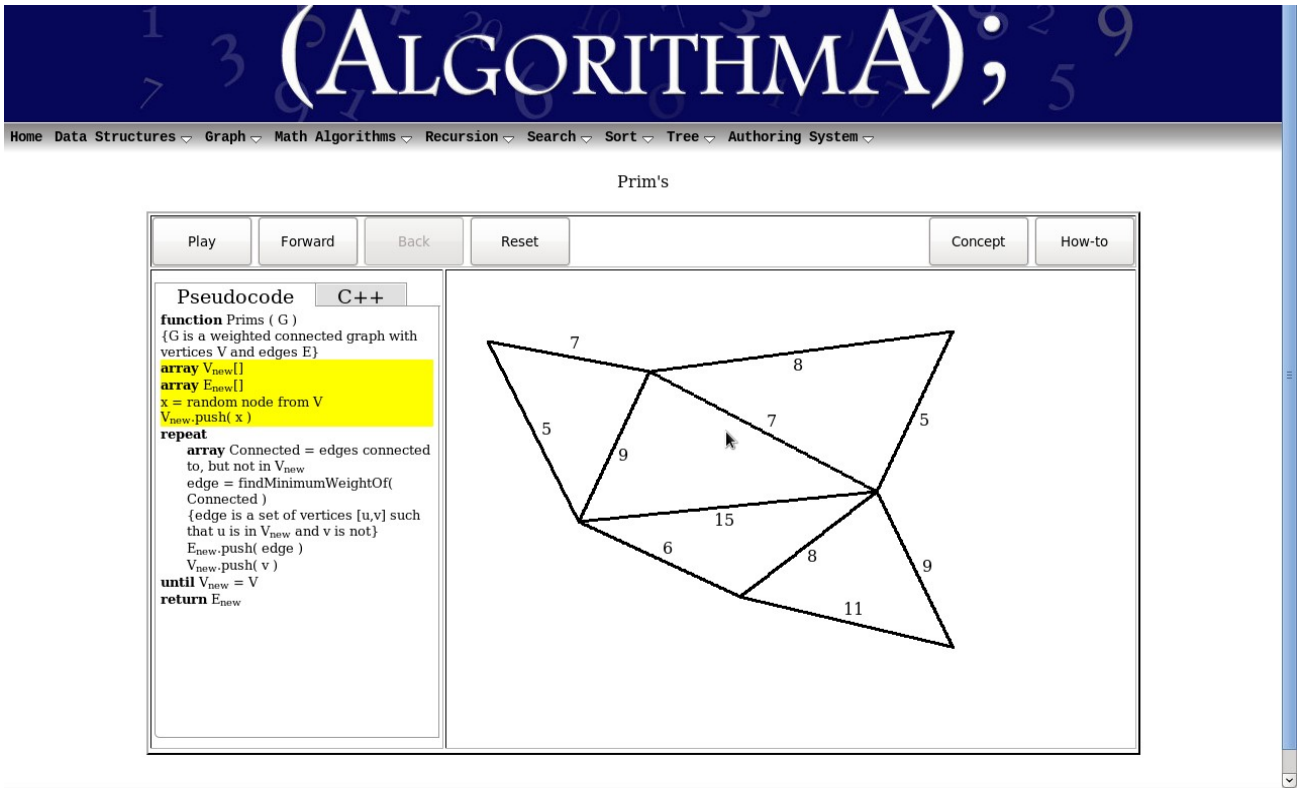
4.4.1 Overview

Prim's algorithm closely resembles Dijkstra's algorithm because they both rely on a similar approach of finding the "next closest" vertex. Prim's algorithm slowly grows a minimum spanning tree, starting from a single vertex and adding in new edges that link the partial tree to a new vertex outside of the tree. Using the cut property, we know that if we have a partial tree, then to add a new vertex to the tree, all we actually need to do is find the vertex that is closest to the tree (the elements in the tree and the elements outside the tree are the two sets in the cut property, so we just choose the smallest edge bridging the two).

A significant difference between Prim's algorithm and Dijkstra's algorithm is that in Dijkstra's algorithm, the distance being checked is to a source node whereas in Prim's algorithm, the distance is only to the minimum spanning tree. The running time, too, is similar to Dijkstra's algorithm. When a heap is used to pick the next vertex to consider, the running time is O(|E| * log|V|), whereas when an unsorted list of verticals is searched in linear time, the running time is O(|V|^2 + |E|). V stands for number of verticals and E is the number of edges.

4.4.2 Layout

The user will be presented with a brief introduction of the function and the basic already "useful" graph to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



4.4.3 Functionality

o Pause – Interactively allows the user to stop and continue the animation

o Forward – Will allow user to step through the animation one step at a time.

o Back – Will allow user to step backward through animation one step at a time.

o Reset – Reset to the initial layout when first loaded.

5 Math Algorithms

5.1 Gaussian Elimination

5.1.1 Overview

Gaussian elimination is an algorithm for solving systems of linear equations, finding the rank of a

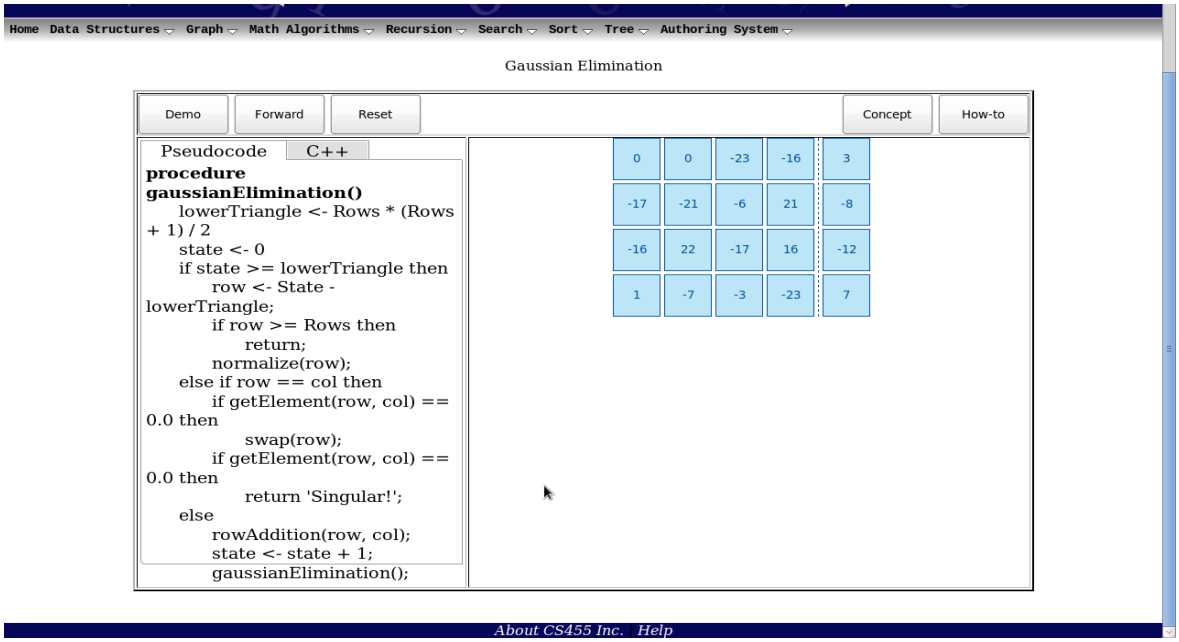
matrix, and calculating the inverse of an invertible square matrix. The method is named after Carl Fried rich Gauss but it was not invented by him.

Elementary row operations are used to reduce a matrix to row echelon form. Gauss–Jordan elimination, an extension of this algorithm, reduces the matrix further to reduced row echelon form.

Gaussian elimination alone is sufficient for many applications, and requires fewer calculations than the-Jordan version.

5.1.2 Layout

Here's the layout of Gaussian Elimination user interface below



5.1.3 Functionality

- Demo – Start the algorithm and let it run until finished
- Forward – Step through the algorithm line by line
- Back – Step back to the previous node.

- Reset – Reset to the initial layout when first loaded

6 Recursion

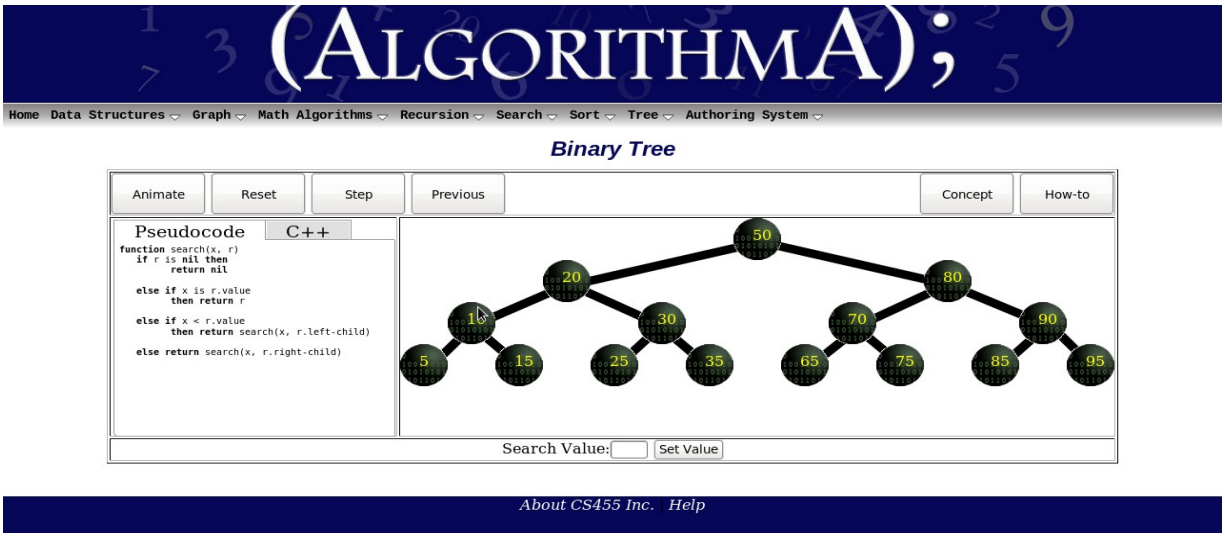
6.1 Binary Search Tree

6.1.1 Overview

The user will be presented with a binary tree and two branches. To the left of the animation work area will be a pseudo-code walk-through that is highlighted as actions are performed. This pseudo code will contain the recursive algorithm for a binary search.

6.1.2 Layout

In order to run the recursive Binary Search, the user will input a number to be found and click the start button. The pseudo code will highlight as the algorithm is stepped through line by line with the animation view on the right changing as appropriate.



6.1.3 Functionality

The search should only contain the bare minimum. This includes:

- Animate- Starts animation
- Reset- Restore to the main

- Step- Goes step-by-step
- Previous- Previous step

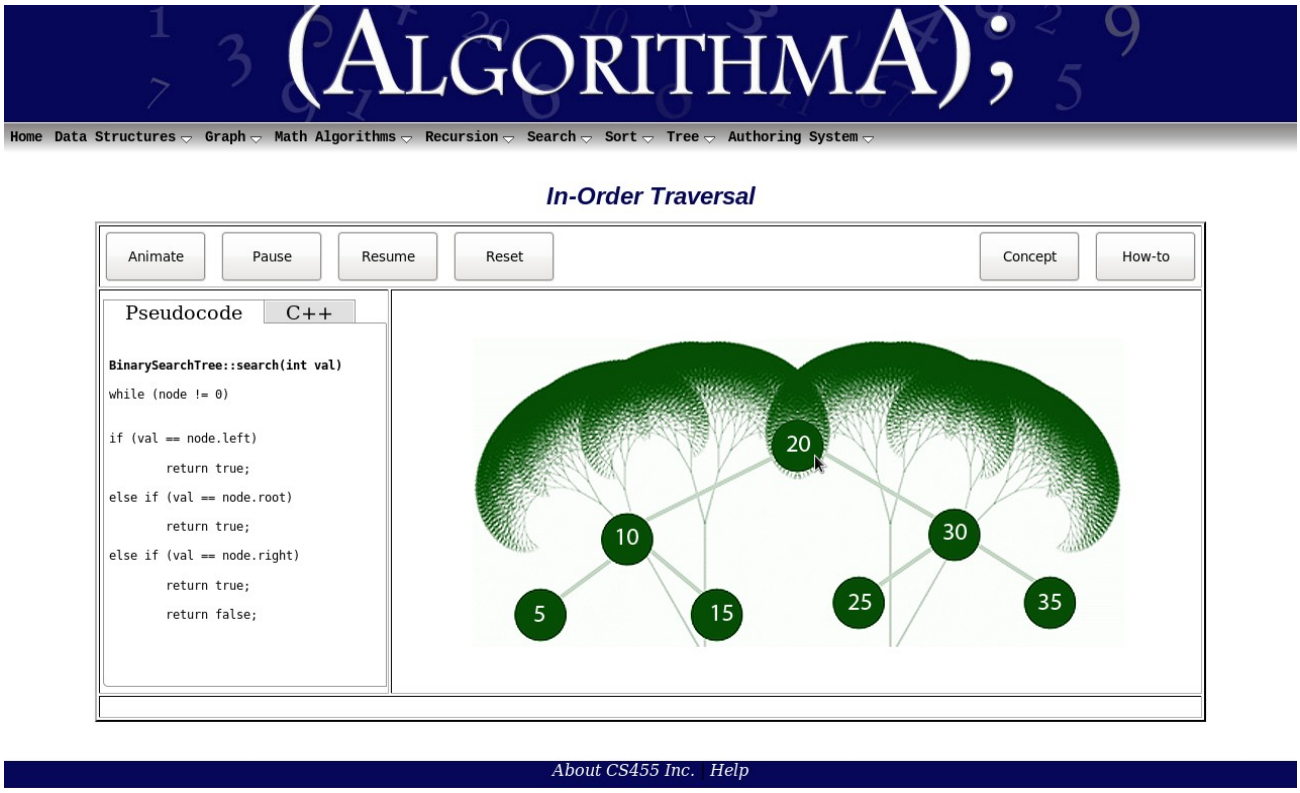
6.2 In-order

6.2.1 Overview

In order traversal is one of the three methods of visiting every node in a tree exactly once. This is done by the LNR method, visiting the left child node first, then the root node (parent node), then the right child node. This algorithm is best done recursively and is an important aspect of mastering the use of binary trees.

6.2.2 Layout

The user will be presented with a basic ready-to-use binary tree. To the left of the animation work area will be a pseudo-code walk-through that is highlighted as actions are performed.



6.2.3 Functionality

The in-order traversal should contain at the bare minimum

- Start – Start the algorithm and let it run until finished
- Step – Step through the algorithm line by line
- Reset– Reset to the initial layout when first loaded

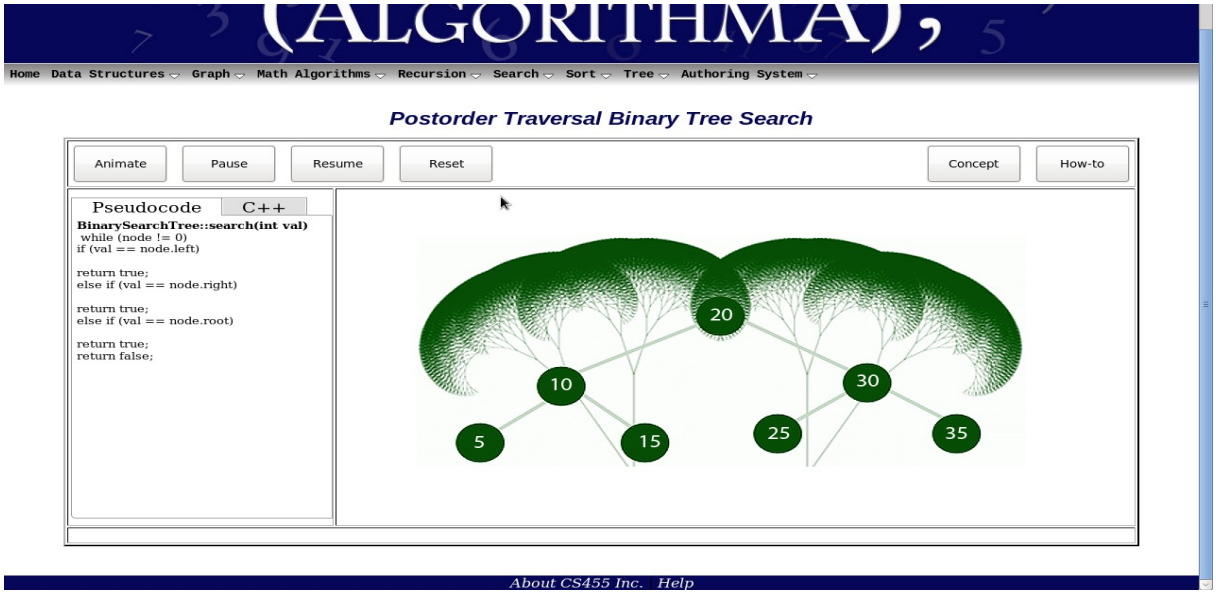
6.3 Post-order

6.3.1 Overview

Post-order traversal is one of the three methods of visiting every node in a tree exactly once. This is done by the LRN method, visiting the left child first, then the right child, then the root node (parent node). This algorithm is best done recursively and is an important aspect of mastering the use of binary trees.

6.3.2 Layout

The user will be presented with a basic ready-to-use binary tree. To the left of the animation work area will be a pseudo-code walk-through that is highlighted as actions are performed.



6.3.3 Functionality

The traversal should only contain the bare minimum. This includes:

- Start – Start the algorithm and let it run until finished
- Step – Step through the algorithm line by line
- Reset– Reset to the initial layout when first loaded

6.4 Pre-order

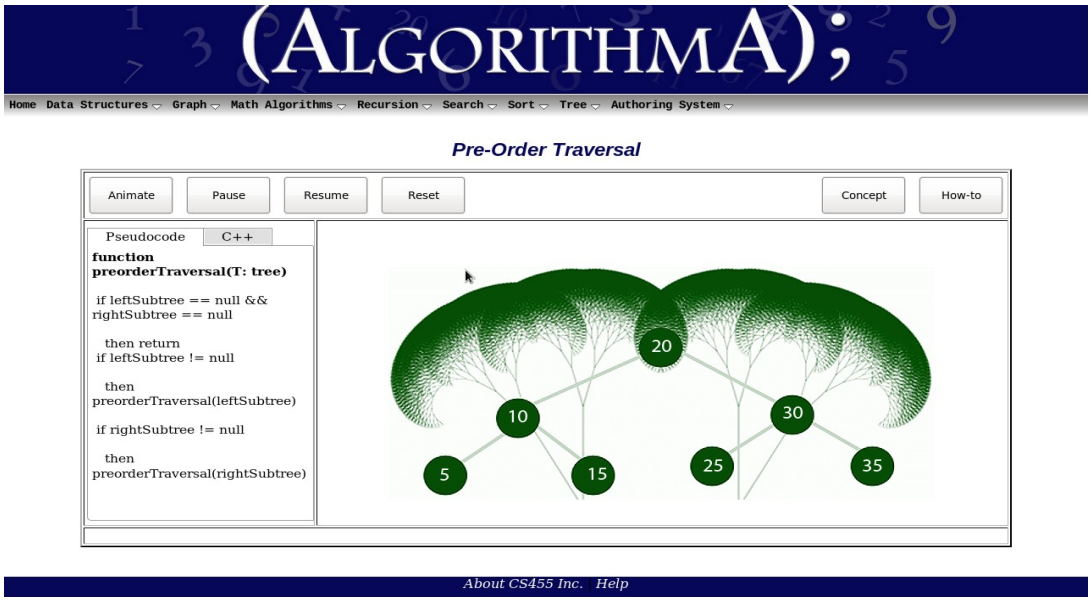
6.4.1 Overview

Pre-order traversal is one of the three methods of visiting every node in a tree exactly once. This is done by the NLR method, visiting the root node (parent node) first, then the left child node, then the right child node. This algorithm is best done recursively and is an important aspect of

mastering the use of binary trees.

6.4.2 Layout

The user will be presented with a basic ready-to-use binary tree. To the left of the animation work area will be a pseudo-code walk-through that is highlighted as actions are performed.



6.4.3 Functionality

The traversal should only contain the bare minimum. This includes:

- Start – Start the algorithm and let it run until finished
- Step – Step through the algorithm line by line
- Reset– Reset to the initial layout when first loaded

7 Search algorithms

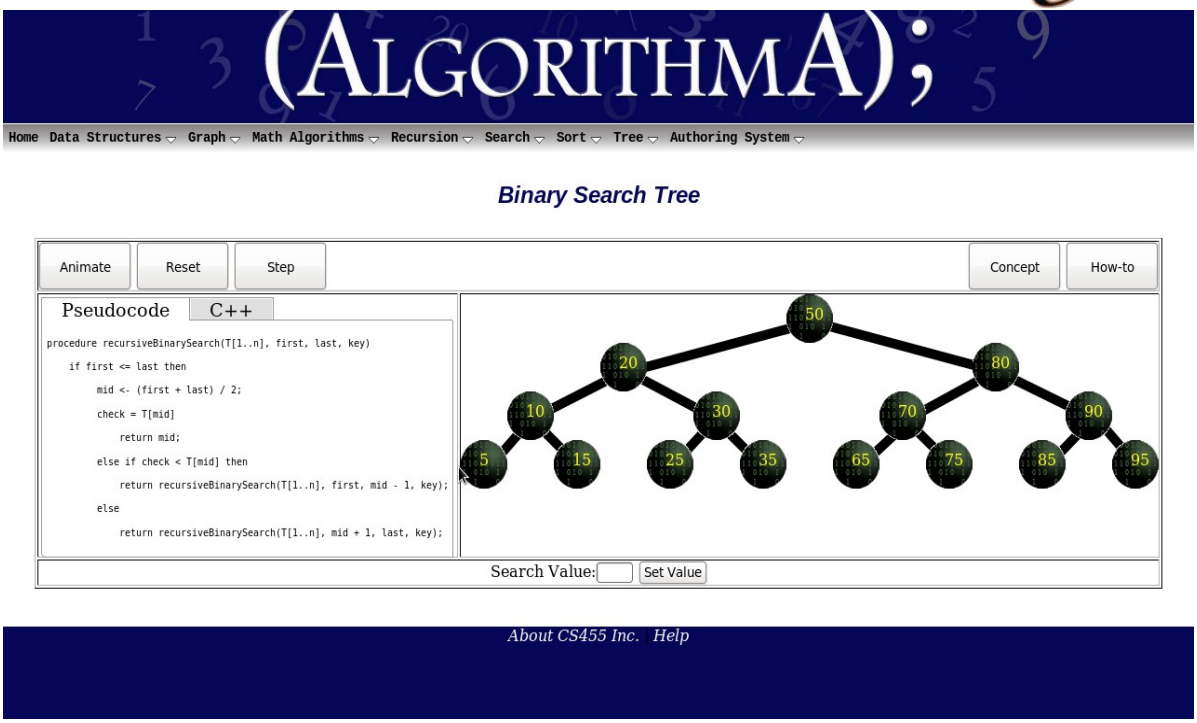
7.1 Binary Search Tree

7.1.1 Overview

A binary tree is a tree data structure in which each node has at most two children. Typically the first node is known as the parent and the child nodes are called left and right. Binary trees are commonly used to implement binary search trees and binary heaps. Every left node has a value less than or equal to its parent node and every right node has a value greater than or equal to its parent. A new node is always added as a leaf, following the specified rule above.

7.1.2 Layout

The user will be presented with buttons detailing the concept and the how-to of the function and a basic tree to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



7.1.3 Functionality

o Search Value/Set Value – User will enter a value to be set. This value will then be searched in the tree.

o Animate – Will perform the full search after a search value has been entered.

o Reset – Reset to the initial layout when first loaded.

o Step – Will allow user to run through animation one step at a time.

o Previous – Will allow user to back up through the animation one step at a time.

7.2 Breadth First Search

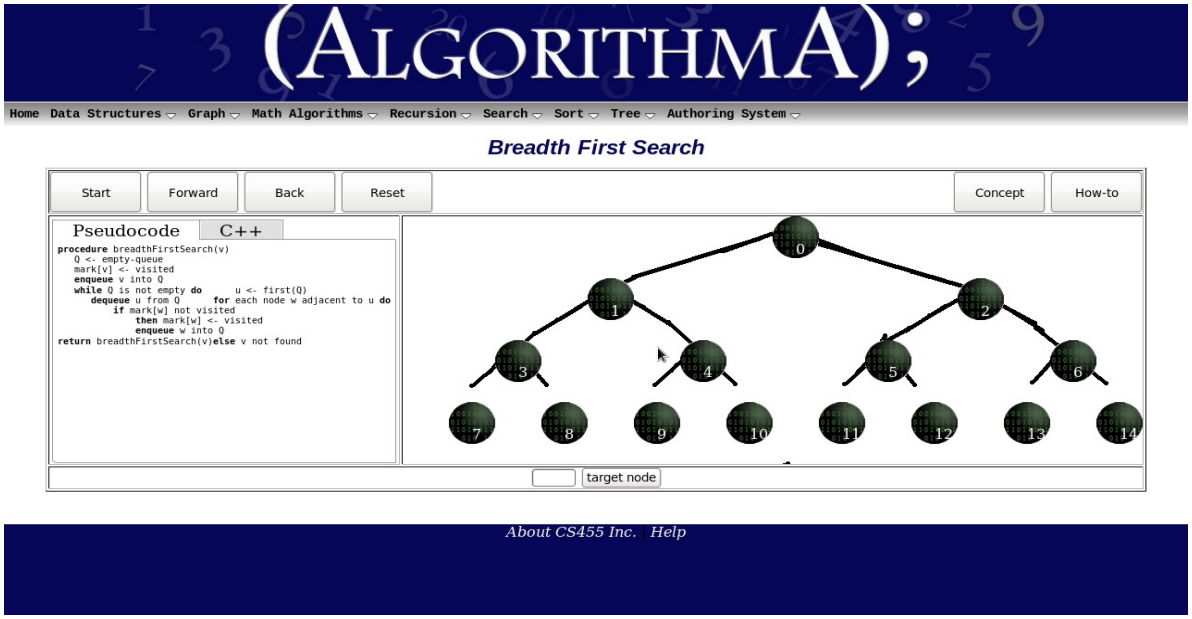
7.2.1 Overview

Breadth-First Search (BFS) is a search that begins at the root node and explores all the

neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal. BFS is an uninformed search method that aims to expand and examine all nodes of a graph or combination of sequences by systematically searching through every solution. In other words, it exhaustively searches the entire graph or sequence without considering the goal until it finds it. It does not use a heuristic algorithm.

From the standpoint of the algorithm, all child nodes obtained by expanding a node are added to a FIFO queue. In typical implementations, nodes that have not yet been examined for their neighbors are placed in some container (such as a queue or linked list) called "open" and then once examined are placed in the container "closed".

7.2.2 Layout



The user will be presented with buttons detailing the concept and the how-to of the function and a basic tree to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.

7.2.3 Functionality

o Target Node – User will enter a value to be searched for.

o Start – Will perform the full search after a search value has been entered.

o Forward – Will allow user to step through animation.

o Back – Will allow user to step through animation backward one step at a time.

o Reset – Reset to the initial layout when first loaded.

7.3 Depth First Search

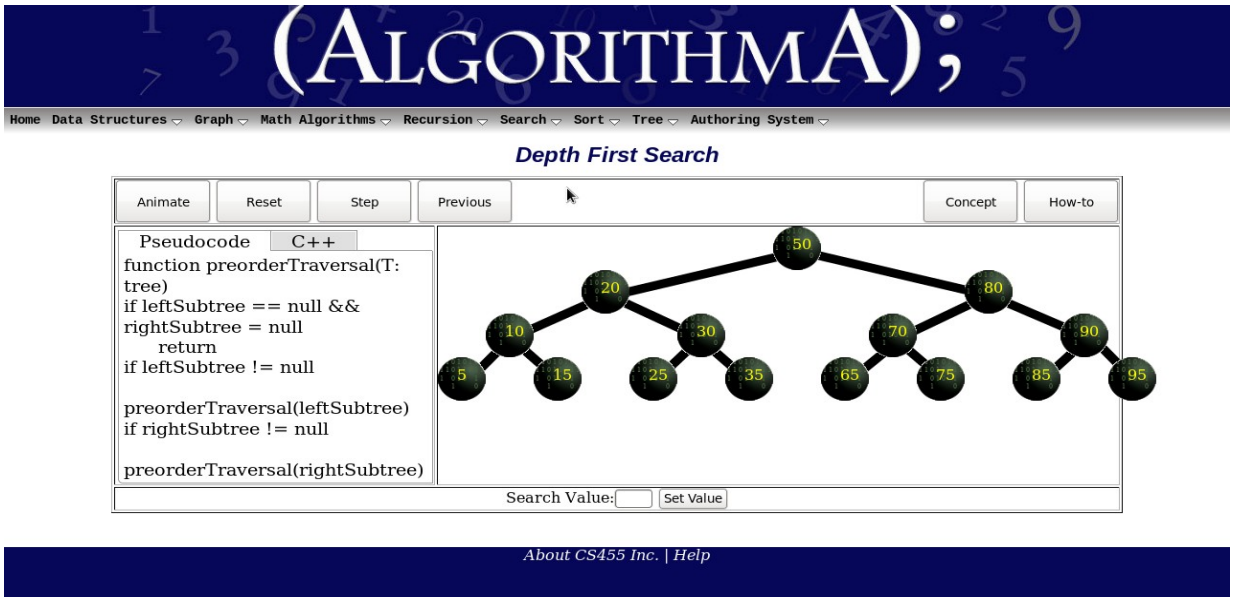
7.3.1 Overview

Depth-First Search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking. DFS is an uninformed search that

progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it hasn't finished exploring. In a non-recursive implementation, all freshly expanded nodes are added to a stack for exploration.

7.3.2 Layout

The user will be presented with buttons detailing the concept and the how-to of the function and a basic tree to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



7.3.3 Functionality

o Search value – User will enter a value to be searched for.

o Animate – Will perform the full search after a search value has been entered.

o Step – Will allow user to pause through animation.

o Previous – Will allow user to step through animation backward one step at a time.

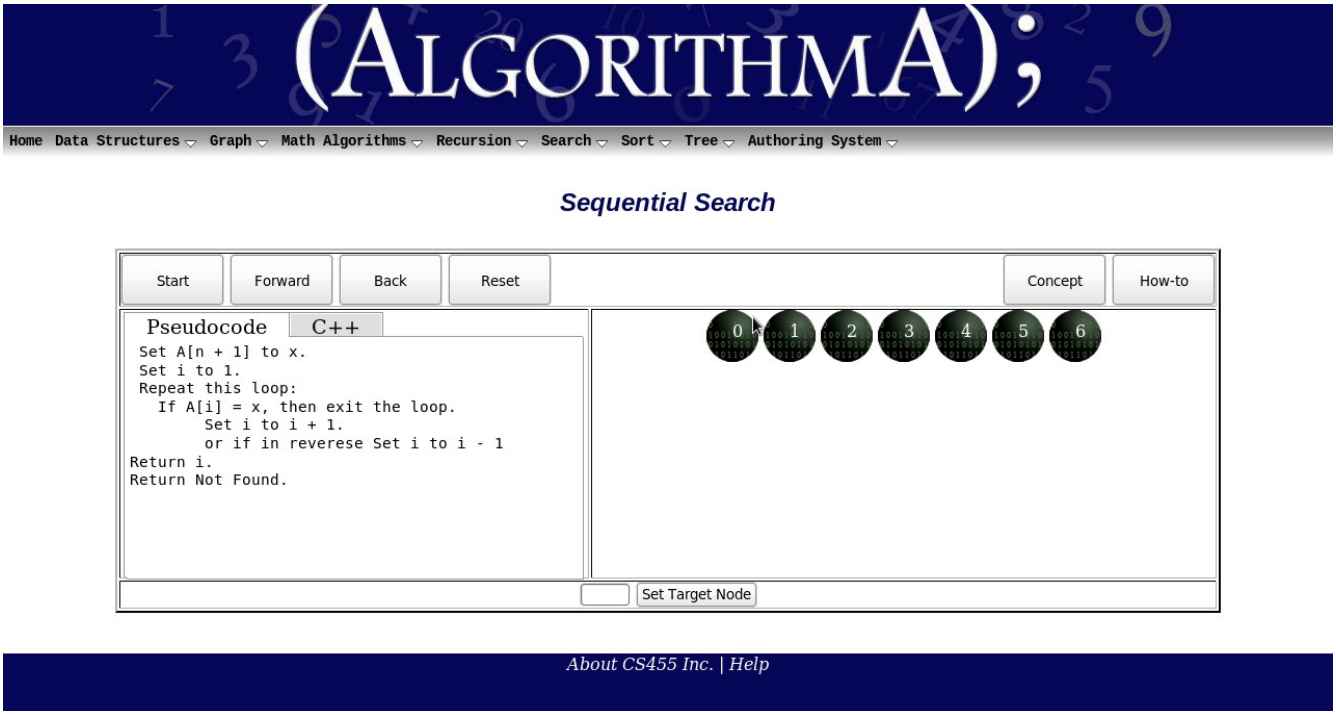
o Reset – Reset to the initial layout when first loaded.

7.4 Sequential Search

7.4.1 Overview

Sequential search, also called linear search, is a method for finding a particular value in a list that consists in checking every one of its elements, one at a time and in sequence, until the desired one is found. Sequential search is the simplest search algorithm; it is a special case of brute-force search. Its worst case cost is proportional to the number of elements in the list; and so is its expected cost, if all list elements are equally likely to be searched for. Therefore, if the list has more than a few elements, other methods (such as binary search or hashing) may be much more efficient.

7.4.2 Layout



The user will be presented with buttons detailing the concept and the how-to of the function and the basic list to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.

7.4.3 Functionality

o Set Target Node – Will allow user to enter and set a value to be searched for.

o Start – Will perform the search either with or without a target value set.

o Forward – Will allow user to step through the animation one step at a time.

o Back – Will allow user to step backward through animation one step at a time.

o Reset – Reset to the initial layout when first loaded.

8 Sorting Algorithms

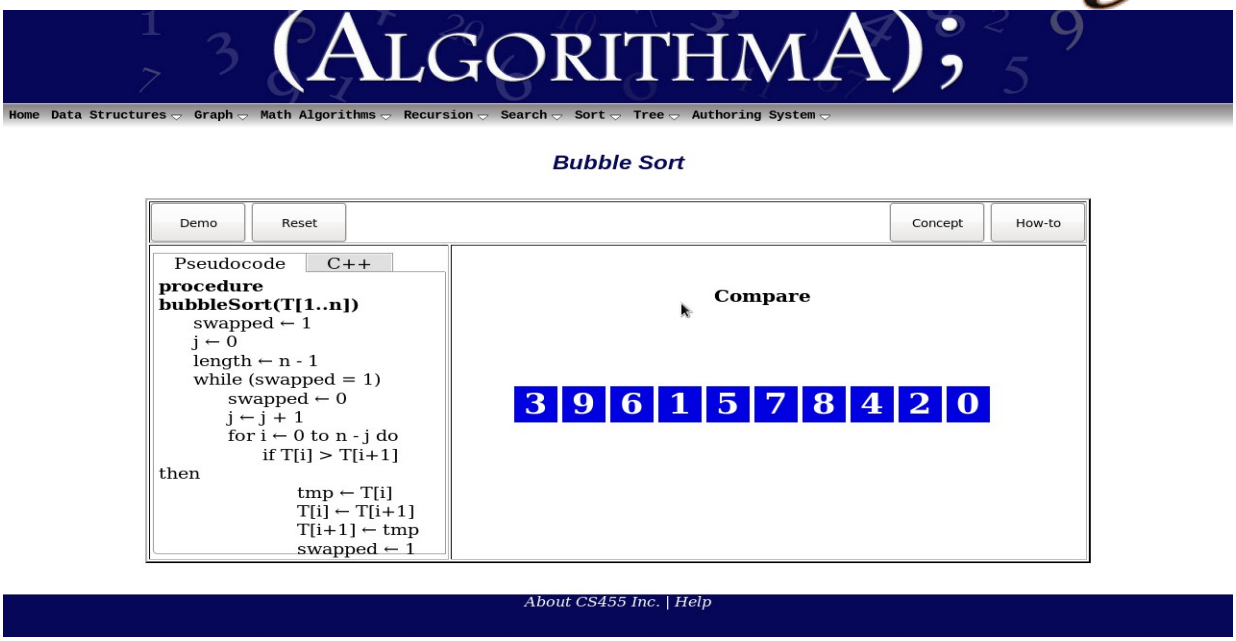
8.1 Bubble Sort

8.1.1 Overview

The bubble sort is the oldest and simplest sort in use. Unfortunately, it's also the slowest. The bubble sort works by comparing each item in the list with the item next to it, and swapping them if required. The algorithm repeats this process until it makes a pass all the way through the list without swapping any items (in other words, all items are in the correct order). This causes larger values to "bubble" to the end of the list while smaller values "sink" towards the beginning of the list. It is generally considered to be the most inefficient sorting algorithm in common usage. Under best-case conditions (the list is already sorted), the bubble sort can approach a constant O (n) level of complexity. General-case is an abysmal O (n2).

8.1.2 Layout

Bubble sort will be classified under Sort. When Bubble sort is selected from sort the basic layout will be displayed. The animation will be displayed by default with an option to select viewable source code. Start, Reset, Pause, Forward, Step check box, Animation check box and Speed setting bar will be displayed.



8.1.3 Functionality

o Start – Starts the bubble sort animation.

o Forward – Allows the user to step through the sort one-step at a time.

o Back – Steps through the pseudo code one line at a time in a backward direction

o Reset – Resets the bubble sort animation to the beginning.

8.2 Insertion Sort

8.2.1 Overview

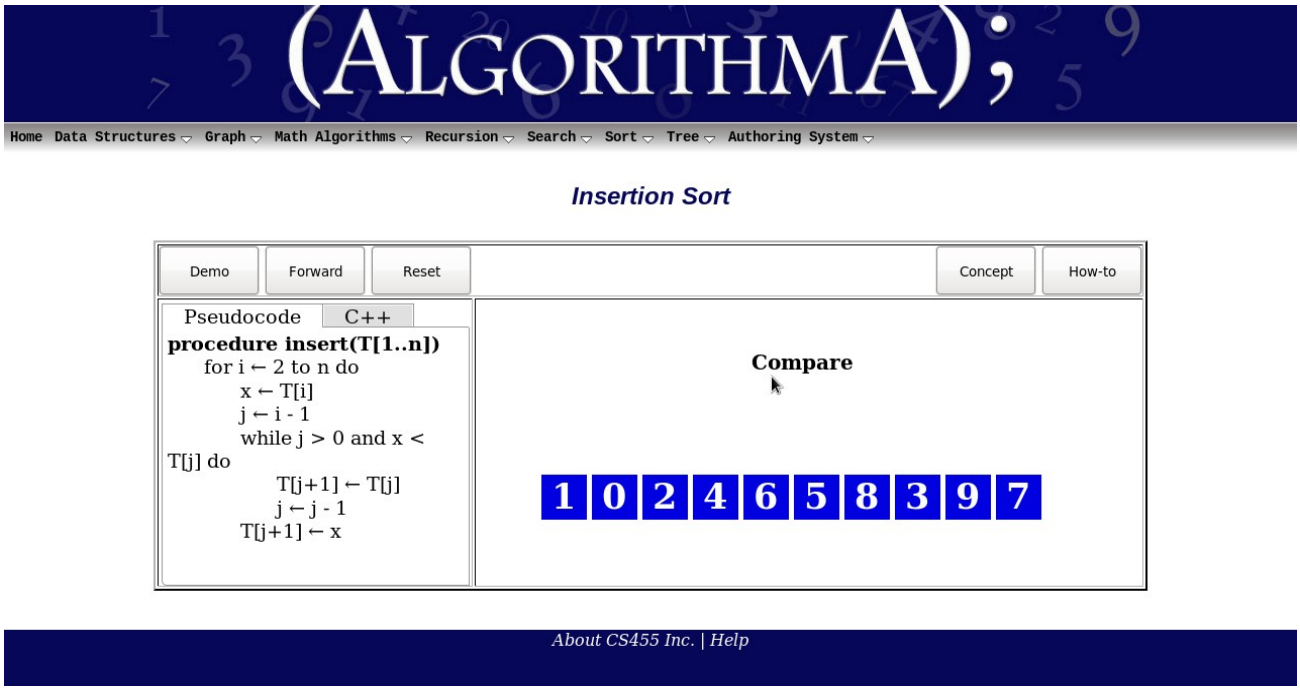
The insertion sort works just like its name suggests - it inserts each item into its proper place in the final list. The insertion sort works by taking the values one by one and inserting each one into a new list that it constructs, constantly maintaining the condition that the elements of the new list are in the desired order with respect to one another. Clearly, this condition will not be maintained if each element is added to the new list at the beginning, instead, the insertion sort adds each element at a carefully selected position within the new list, placing the new element

after each previously placed element that precedes it according to the given precedence rule, but before every such element that it precedes. The simplest implementation of this requires two list structures - the source list and the list into which sorted items are inserted. To save memory, most implementations use an in-place sort that works by moving the current item past the already sorted items and repeatedly swapping it with the preceding item until it is in place.

Like the bubble sort, the insertion sort has a complexity of O (n 2). Although it has the same complexity, the insertion sort is a little over twice as efficient as the bubble sort.

8.2.2 Layout

Insertion sort will be classified under Sort. When Insertion sort is selected from sort the basic layout will be displayed. The animation will be displayed by default with an option to select viewable source code. Start, Reset, Pause, Forward, Step check box, Animation check box and Speed setting bar will be displayed



8.2.3 Functionality

o Demo – Starts the Insertion sort animation.

o Forward – Only activated during walk through mode. The next line of code will not be executed until this button has been pressed.

o Back – Steps through the pseudo code one line at a time in a backward direction

o Reset – Cancels the animation or walk through and reinitialized data seen on screen.

8.3 Selection Sort

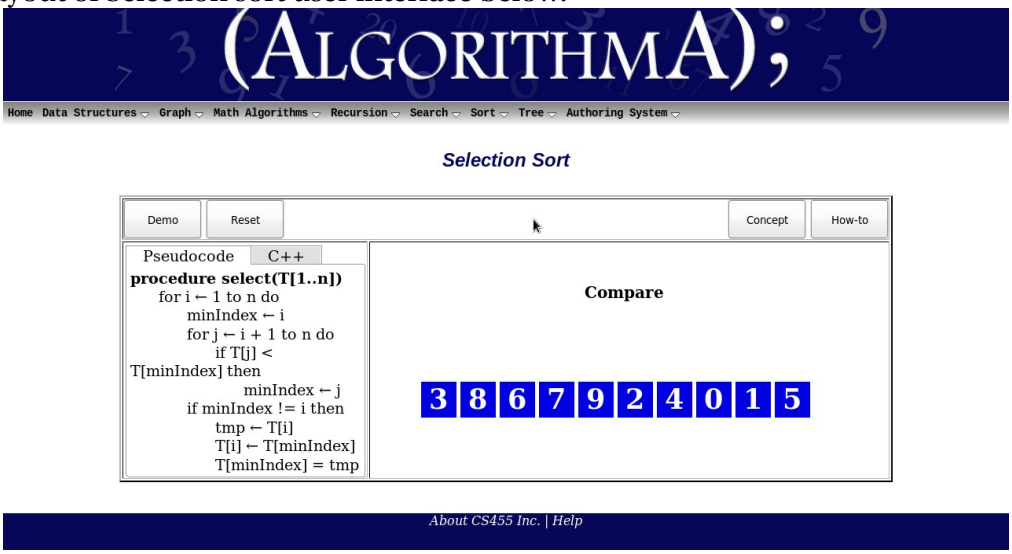
8.3.1 Overview

Selection sort is a sorting algorithm. With a O(n^2) complexity, selection sort is unfit to handle large lists, but is notable for its programming simplicity. It almost always outperforms bubble sort and gnome
sort, and performs worse than insertion sort. Selection sort finds the minimum value in the list and then swaps it with the value in the first position. Continue to repeat those two steps for the

remainder
of the list while starting at the second position and advancing each time.

8.3.2 Layout

Here's the layout of Selection sort user interface below.



8.3.3 Functionality

- Demo – Start the algorithm and let it run until finished
- Forward – Step through the algorithm line by line
- Back – Step back to the previous node.
- Reset – Reset to the initial layout when first loaded

8.4 Merge Sort

8.4.1 Overview

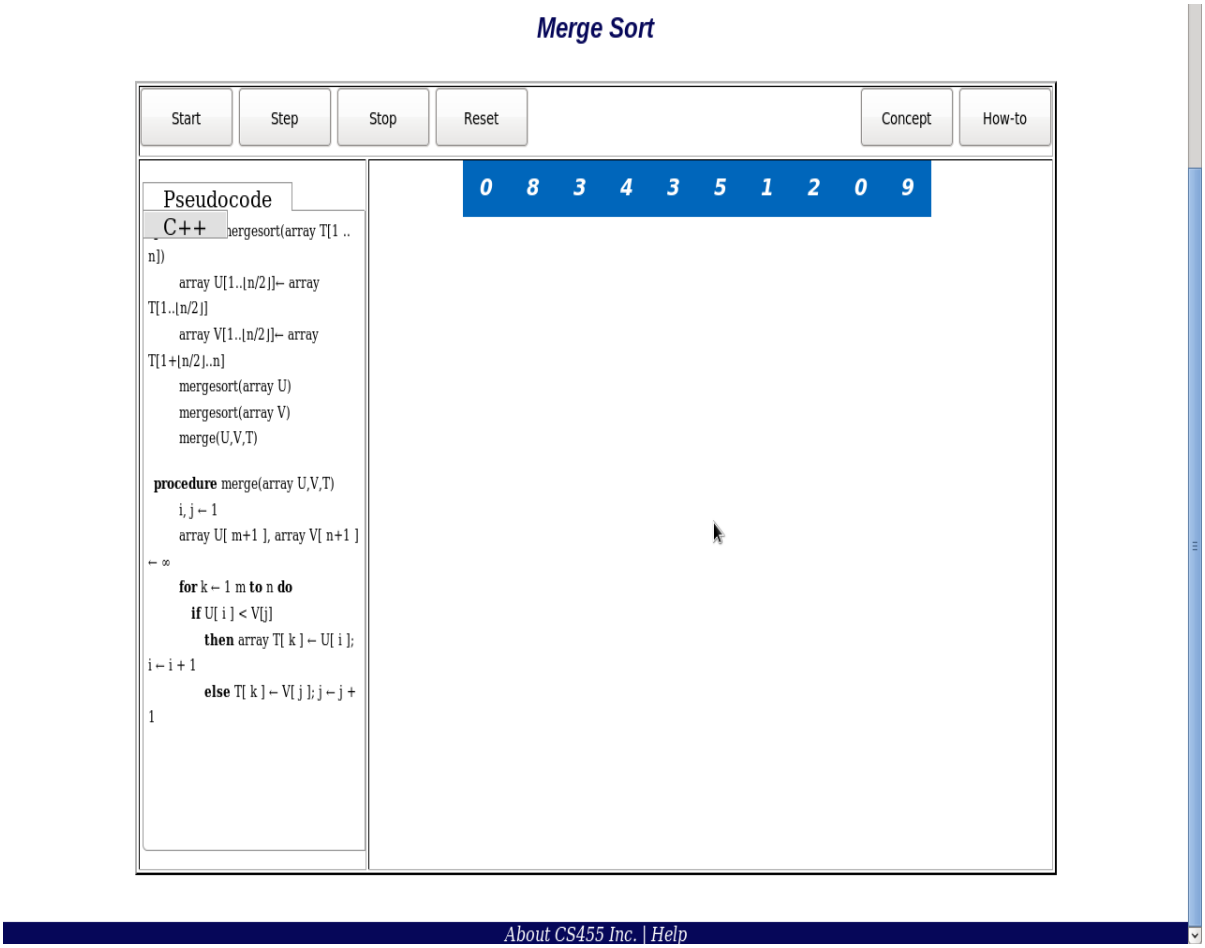
The Merge Sort algorithm is a sorting algorithm that is similar to Quick Sort. Merge Sort has a complexity of O(n log(n)) and is therefore an optimal sort. The Merge Sort is based on a “divide and conquer” strategy. First, the sequence of data to be sorted is divided into two approximately equal halves. Each half is recursively sorted and then the two halves are merged into a sorted

sequence.

In AlgorithmA 2010, the animation and walk through views are merged into one. The animation view is the default. To initiate the animation, the user must click on the “Start” button. Once initiated, the animation will proceed until the sort is complete. To view the walk through, the user must press the “Step” button and then the “Forward” button to proceed through the walk through.

8.4.2 Layout

The Merge Sort algorithm is already classified under Sort. When selected, the user will be given a short explanation of how the algorithm works.



8.4.3 Functionality

o Start – Starts either the animation or walk through depending on whether the Step button is checked or not. See the Step button functionality for more detail.

o Stop – Halts the execution of the animation if the current state of the animation is in execution mode.

o Step – When this is checked, the view is that of the walk through. When not checked, the view is that of the animation.

o Reset – Stops the animation or walk through and initializes the data again.

8.5 Quick Sort

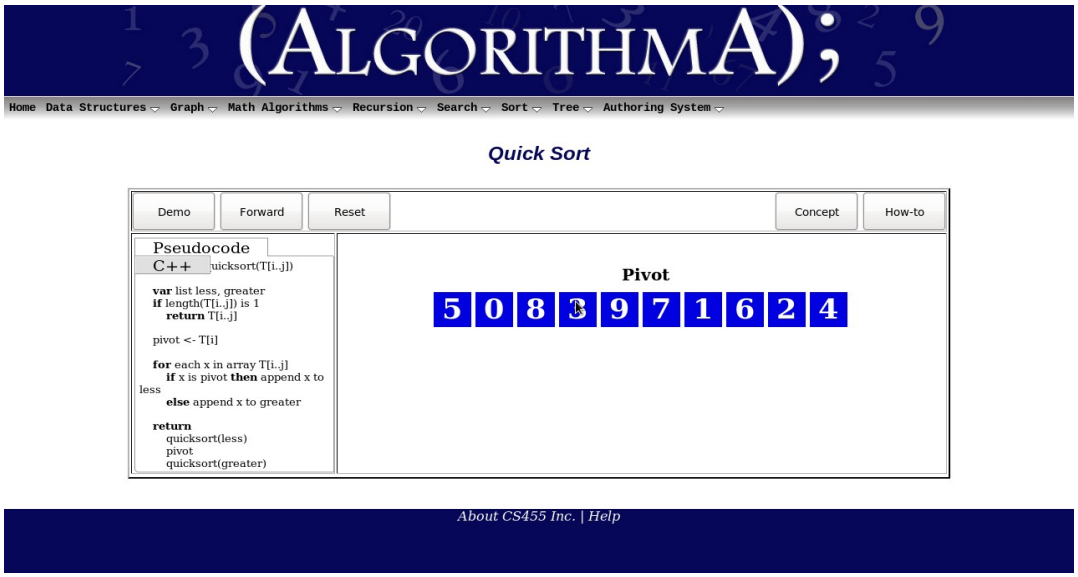
8.5.1 Overview

The Quick Sort algorithm is a sorting algorithm whose complexity can range from O(nlogn) to O(n2). It first chooses a pivot element, and creates three lists. The first list is all elements that are less than the pivot value. The second list contains all elements that are equal to that value. The third list contains all elements that are greater than that value. The algorithm is recursively called onto the first and last list, and continues until there is only one element in each list. Doing so will give the desired result.

For AlgorithmA 2010, the animation and walk through components are merged into one. By default, the animation is set to automatic. Upon pressing the start button, the sorting algorithm will continue to sort without any interaction. If the user wants to see a walk through, the “step” button can be pressed, which will stop the next frame of animation until the user desires.

8.5.2 Layout

The Quick Sort algorithm is already classified under Sort. When selected, the user will be given a quick explanation of the algorithm. The explanation is insufficient from previous iterations and will be improved to provide more detail and a thorough explanation of how the algorithm works.



8.5.3 Functionality

- Demo – Starts the Quick Sort animation
- Forward – Only activated during walk through mode. The next line of code will not be executed until this button has been pressed.
- Back – Steps through the pseudo code one line at a time in a backward direction
- Reset – Cancels the animation or walk through and reinitialized data seen on screen.

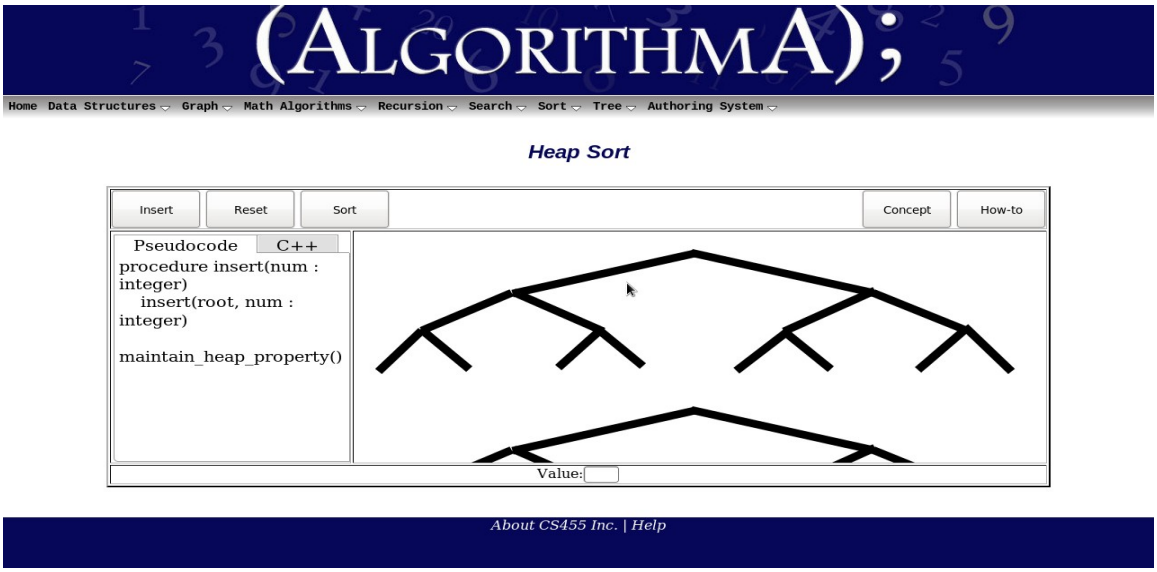
8.6 Heap Sort

8.6.1 Overview

Heap sort is a comparison-based sorting algorithm, and is part of the selection sort family. Although somewhat slower in practice on most machines than a well implemented quick sort, it has the advantage of a more favorable worst-case O(n log n) run time.

8.6.2 Layout

Here's the layout of Heap sort user interface below



8.6.3 Functionality

- Demo – Start the algorithm and let it run until finished
- Forward – Step through the algorithm line by line
- Back – Step back to the previous node.
- Reset – Reset to the initial layout when first loaded

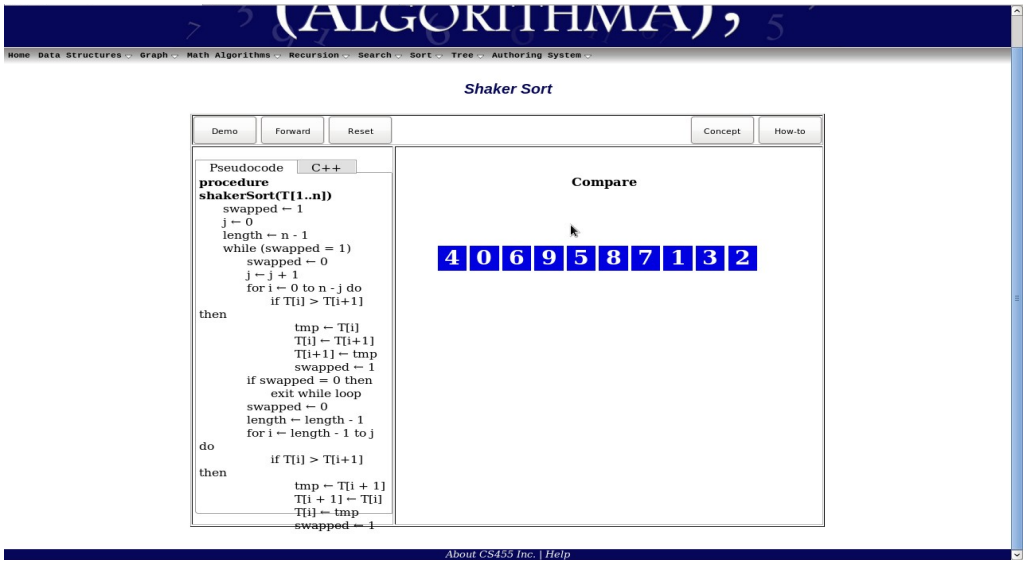
8.7 Shaker Sort

8.7.1 Overview

Shaker sort is much like the bubble sort except that it sorts in both directions on each pass through an array. Shaker sort is marginally more difficult to implement than the bubble sort, however it solves the problem of 'turtles,' the event that a low number appears at the top of the array and thus takes longer to move to the bottom of the array, in bubble sort. Under best-case conditions (the list is already sorted), the shaker sort can approach a constant O (n) level of complexity. General-case is an abysmal O.

8.7.2 Layout

Here's the layout of Shaker sort user interface above.



8.7.3 Functionality

- Demo – Start the algorithm and let it run until finished
- Forward – Step through the algorithm line by line
- Back – Step back to the previous node.
- Reset – Reset to the initial layout when first loaded

9 Trees

9.1 2-3-4 Tree

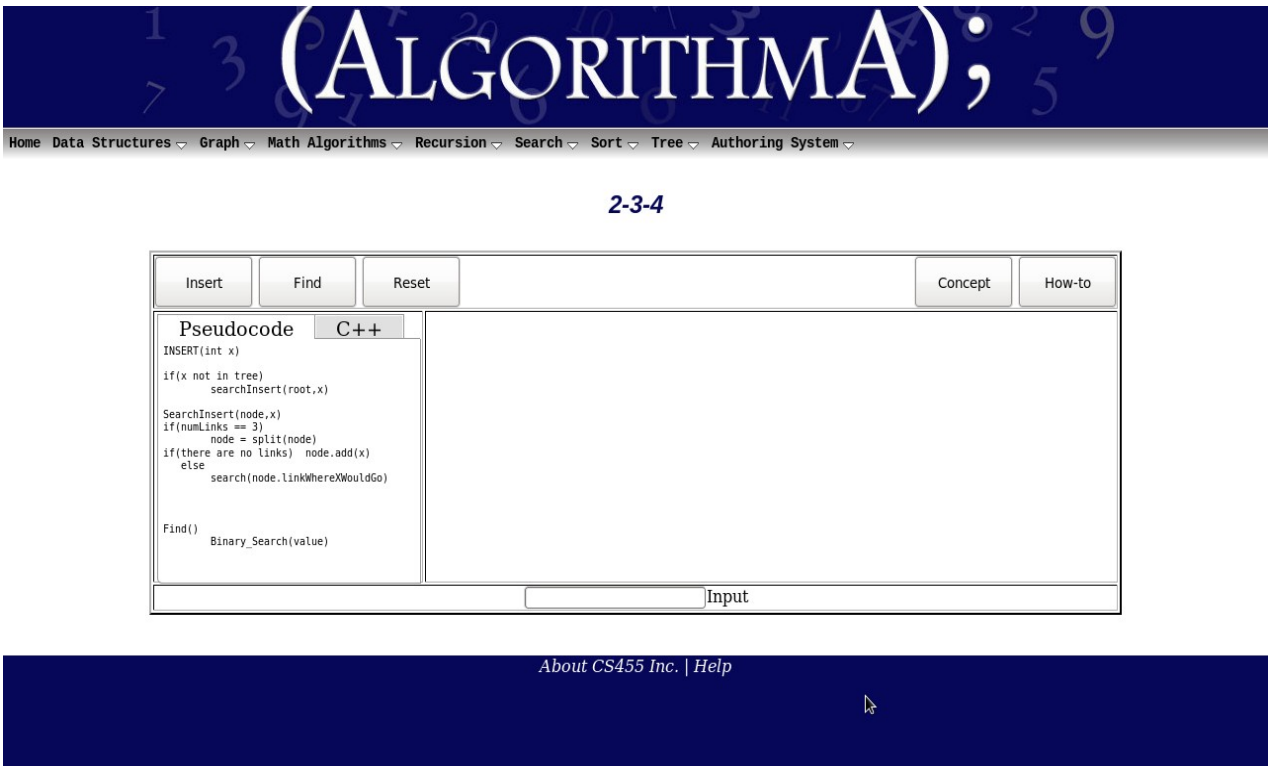
9.1.1 Overview

A 2-3-4 tree (also called a 2-4 tree) is a self-balancing data structure that is commonly used to implement dictionaries. 2-3-4 trees are B-trees of order 4, and can search, insert and delete in

O(log *n*) time. One property of a 2-3-4 tree is that all external nodes are at the same depth. 2-3-4 trees can be difficult to implement in most programming languages because of the large number of special cases involved in operations on the tree.

9.1.2 Layout

The user will be presented with a brief introduction of the function and the basic already "useful" tree to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



9.1.3 Functionality

- Input – Allows entering a whole numeric value.
- Insert – Will insert nodes with an input numeric value.
- Find – After a value in input, find will attempt to locate it.

- Reset – Reset to the initial layout when first loaded.

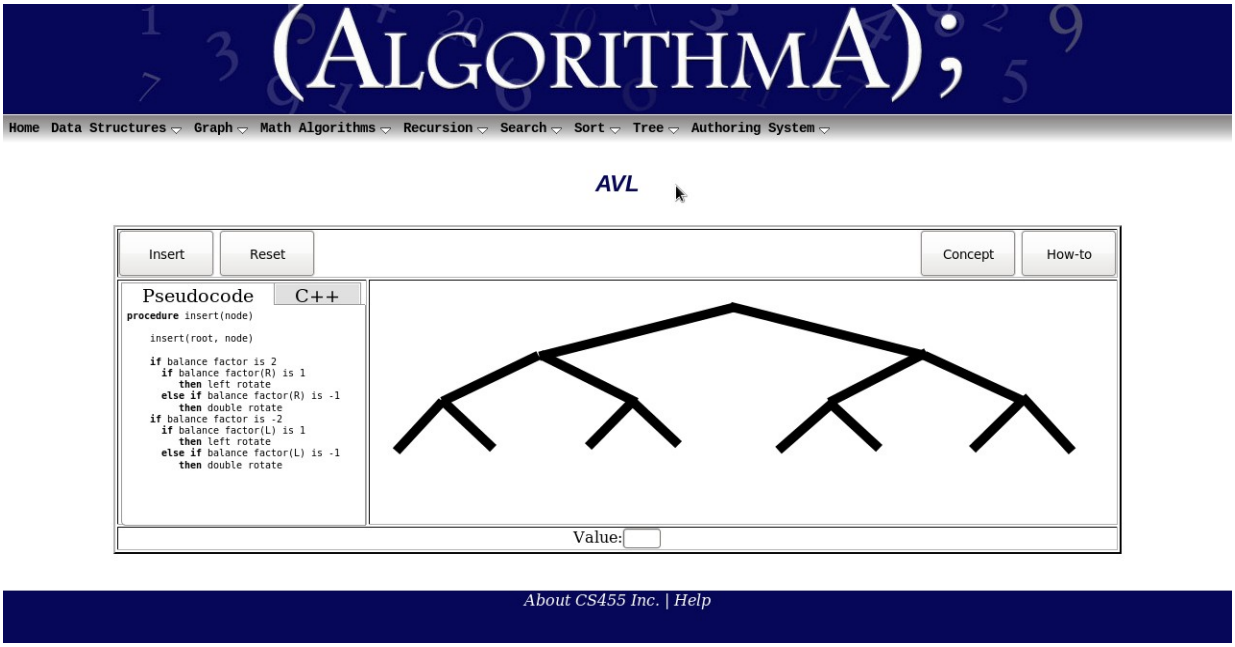
9.2 AVL Tree

9.2.1 Overview

An AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two child sub-trees of any node differ by at most one; therefore, it is also said to be height-balanced. Look up, insertion, and deletion all take O(log *n*) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations. The balance factor of a node is the height of its right sub tree minus the height of its left sub tree, and a node with balance factor 1, 0, or −1 is considered balanced. A node with any other balance factor is considered unbalanced and requires re-balancing the tree. The balance factor is either stored directly at each node or computed from the heights of the sub trees.

9.2.2 Layout

The user will be presented with a brief introduction of the function and the basic already “useful” tree to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



9.2.3 Functionality

o Input – Allows entering a whole numeric value.

o Insert – Will insert nodes with an input numeric value.

o Reset – Reset to the initial layout when first loaded.

9.3 Red-Black Tree

9.3.1 Overview

A red-black tree is a type of self-balancing binary search tree, typically used to implement associative arrays. It is complex, but has good worst-case running time for its operations and is efficient in practice: it can search, insert, and delete in O(log *n*) time, where *n* is total number of elements in the tree. Red-black trees, like all binary search trees, allow efficient in-order traversal of their elements. The search-time results from the traversal from root to leaf, and therefore a balanced tree, having the least possible tree height, results in O(log *n*) search time. A red-black tree is a binary search tree where each node has a *color* attribute, the value of which is either *red* or *black*. In addition to the ordinary requirements imposed on binary search trees, the following

additional requirements apply to red-black trees:

1. A node is either red or black.

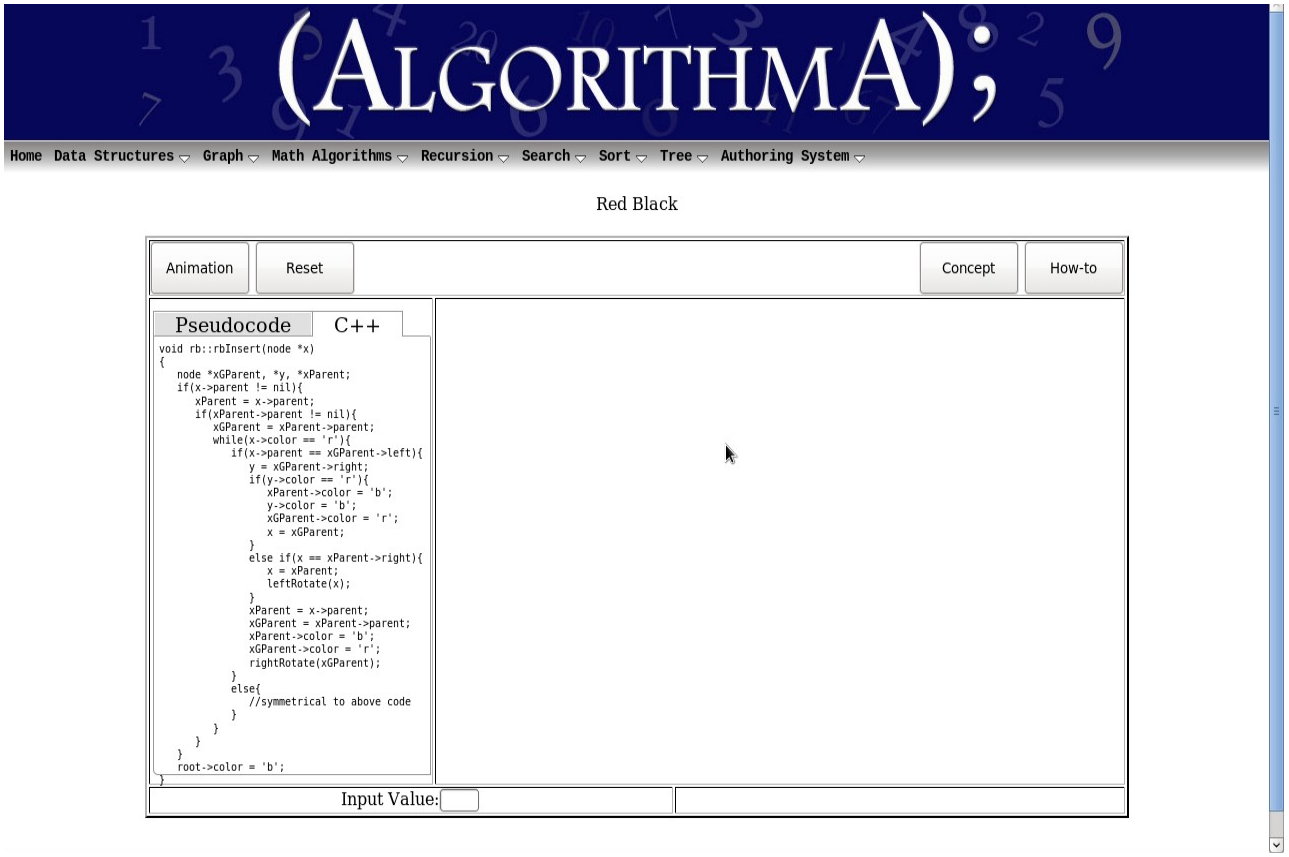
2. The root is black. All leaves are black.

3. Both children of every red node are black.

4. Every simple path from a given node to any of its descendant leaves contains the same number of black nodes.

9.3.2 Layout

The user will be presented with a brief introduction of the function and the basic already "useful" tree to perform a search on. To the left of the animation work area will be a pseudo-code walk through that is highlighted as actions are performed.



9.3.3 Functionality

o Animation – Runs the animation.

o Reset – Reset to the initial layout when first loaded.

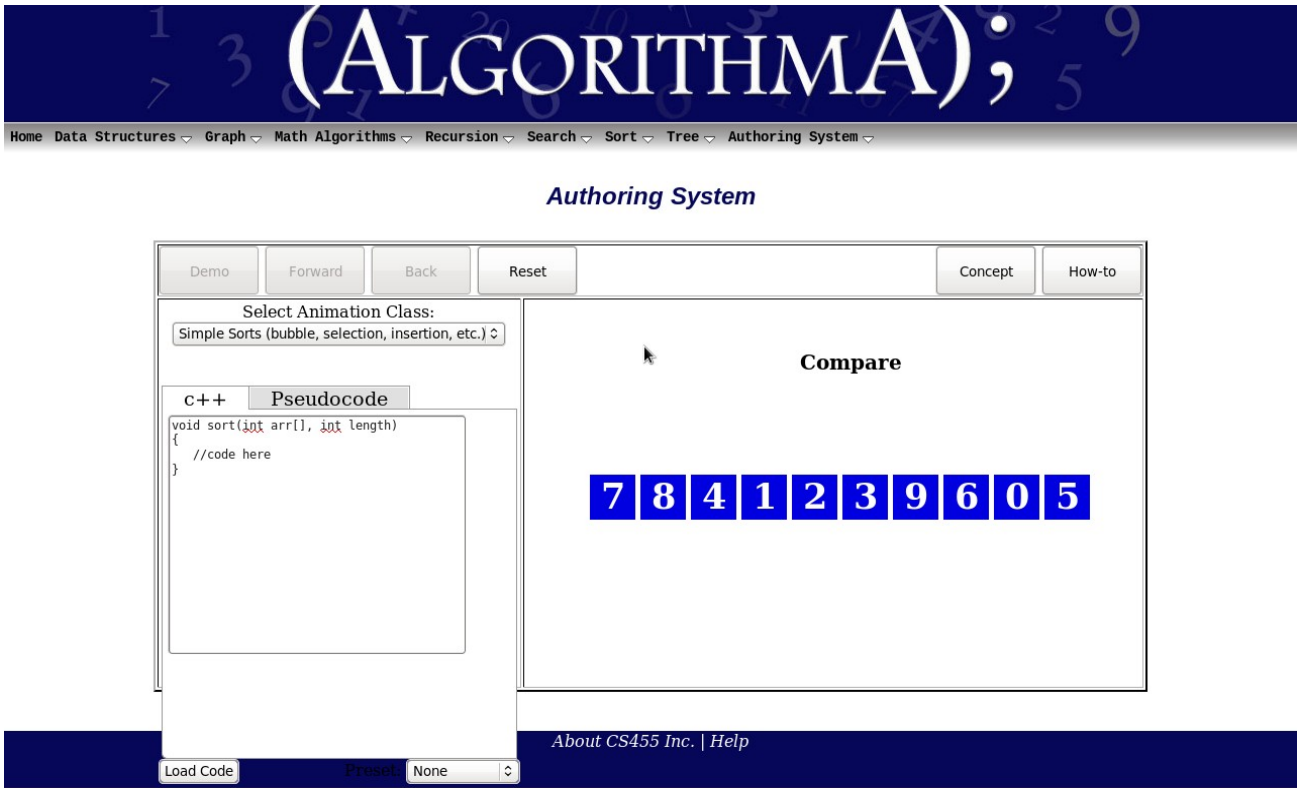
10 Authoring System

10.1 Overview

Authoring system is the innovation of {AlgorithmA} 2011, it would allow user to enter or edit the pseudo-code or c++ code, and see the changes in animations accordingly. However there would be some limitation on entering and editing the code to prevent the errors such as infinite loop, syntax error, etc. But this feature would allow user to understand the algorithms better. This authoring system in this iteration would cover array intensive sort (Merge Sort, Quick Sort, etc),basic sort algorithms(Insertion sort, Bubble sort, Selection sort, etc) and basic programmings concept(for loop, while loop, etc)

10.1.1 Layout

Here's the layout of Authoring system user interface below



10.1.2 Functionality

- Pseudo Code Tab – To enter, edit or view the pseudo-code
- C++ Tab – to enter, edit or view C++ code
- Load Code – To load the code for animations
- Select Animations Class – To select type of animations
- Preset code – To load preset code to be displayed
- Demo – Start the algorithm and let it run until finished

- Forward – Step through the algorithm line by line
- Back – Step back to the previous node.
- Reset – Reset to the initial layout when first loaded