CALIFORNIA STATE UNIVERSITY, SAN BERNARDINO

Department of Computer Science
And Engineering

CSCI 455

# {AlgorithmA}; 2010


# Software Requirement Specification (SRS) First Iteration

**CS455, Inc.**

CEO: Dr. Concepcion
Project Manager: Patrick O'Connor
Assistant Managers: Danny Vargas and Abdelrahman Kamel
Documentation Team: Erick Behr, Tyler Cannon,
Charles Korma, Kathleen Daugherty

# Table of Contents

# 1. Introduction

The CEO of CS455 Inc. has requested improvements and repairs to {AlgorithmA}; that will be prepared during iteration one. The following document is a list of the proposed solutions for the CEO's expectations. Please review this document and verify the proposed solutions are correct interpretations of the requirements specified. Upon the approval from the CEO of this document, development will commence immediately.

## 1.1 Purpose

The main goal of {AlgorithmA}; 2010 is to provide an interface to the academic community, in particular new students interested in the field of Computer Science and Engineering, to explore and learn about various algorithms. End users will have the ability to view a step-by-step animation for each of the algorithms that illustrate how they operate.

Several generations of {AlgorithmA}; have introduced mathematical concepts for end-users with mathematical knowledge and/or interest in order to provide a basic understanding of them. {AlgorithmA}; 2010 will continue to build upon these introductions, providing graphics where appropriate and refining the already existing explanations to better suit the end user, {AlgorithmA}; 2010 will address the following requests:

- Animation improvement
- Application loading and execution response
- Fix known faults
- Generate technical documentation for server installation
- Generate system architecture documentation
- Create an application user's manual
- We will re-engineer {AlgorithmA}; 2009.
- We will recall {AlgorithmA}; 2009 from open source.

## 1.2 Scope

{AlgorithmA}; 2010 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}; 2010 will continue on forward with the progress of {AlgorithmA}; 2009 by identifying and fixing all program faults. {AlgorithmA}; 2010 will also reengineer the java animation with JavaScript code. JavaScript is easier for the software engineer to use and reduces the system requirements needed to run the applications.

The following algorithms will be re-implemented:

Data Structures
- Deque
- Priority Queue
- Linked List
- Queue
- Stack

Search
- Binary Search Tree
- Sequential
- Depth First Search
- Breadth First Search

Sort
- Bubble
- Insertion
- Merge Sort
- Quick Sort

# 1. Definitions and Acronyms

**{AlgorithmA};**

PattE's sister project, which stands for "Algorithm Animation."

**Animation**

A visual display that depicts selected algorithms being studied. The display is based on the Cartesian x-y coordinate system.

**Computer Science**

Study of information and computation.

**CS**

Acronym for Computer Science.

**Deployment Diagram**

Deployment diagrams serve to model the hardware used in system implementations and the associations between those components.

**Fault**

An abnormal condition or defect at the component, equipment, or subsystem level, which may lead to failure. It is informally linked with "bug."

**Graphical User Interface**

A GUI is a method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text.

**HTML**

Acronym for Hypertext Markup Language, the authoring language used to create documents on the World Wide Web.

**HTTP**

Acronym for Hypertext Transfer Protocol. It is the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web Servers and browsers should take in response to various commands.

**IDE**

Acronym for Integrated Development Environment. IDEs assist computer programmers in developing software.

**Interface**

The communication boundary between two entities such as software and its users.

**Iteration**

The repetition of a process.

**Java**

A high-level programming language developed by Sun Microsystems.

**Model-View-Controller**

A software architecture that separates an application's data model, user interface, and control logic into three distinct components so that modification to one component can be made with minimal impact to the others.

**Mozilla Firefox**

A free, cross-platform, graphical web browser that complies with many of today's standards on the World Wide Web. The most important standards for {AlgorithmA}; 2010 will be the W3C web standards.

**Open Source Software**

Software whose source code is published and made available to the public, enabling anyone to copy, modify and redistribute the source code without paying royalties or fees. Open source code evolves through community cooperation.

**PHP**

Self-referential acronym for PHP: Hypertext Preprocessor, an open source, server-side, HTML embedded scripting language used to create dynamic Web pages. In an HTML document, PHP script (similar syntax to that of Perl or C) is enclosed within special PHP tags.

**Software Requirements Specification**

An SRS is used to describe all the tasks that go into the instigation, scoping, and definition of a new or altered computer system.

**SRS**

An acronym for Software Requirements Specification.

**W3C**

An acronym for the World Wide Web Consortium.

**World Wide Web Consortium**

An international organization that works to define standards for the World Wide Web.

## 1.3 References

Wikpedia, http://en.wikipedia.org/

Bruegge, Bernd, and Allen H. Dutoit. Object-Oriented Software Engineering. 3rd ed. New Jersey: Pearson Prentice Hall, 2010.

W3schools, http://www.w3schools.com/jsref/default.asp

"{AlgorithmA}; 2008 SRS Prototype 2." CS455 Inc. Management Team. February 4, 2008.

Fowler, Martin. UML Distilled Third Edition. A Brief Guide to the Standard Object Modeling Language. Boston: Pearson Education, Inc. 2009.

IEEE SRS Std. 830-1998.

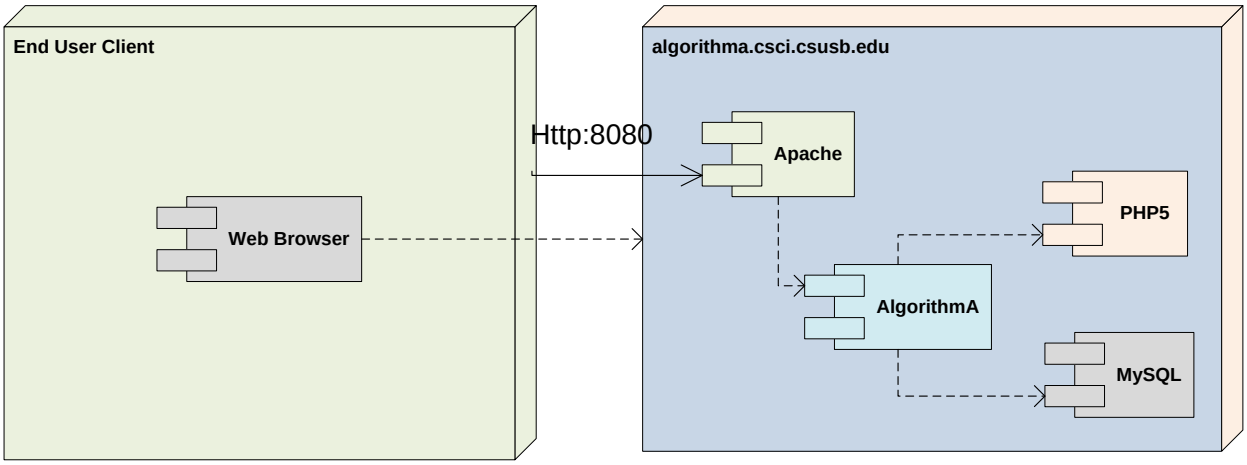
http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel4%2F5841%2F15571%2F00720574.pdf&authDecision=-203

# 2. OVERALL DESCRIPTION

## 2.1 Product Description

### 2.1.1 System Interfaces

The deployment diagram pictured below interprets the system interactions for iteration #1 of {AlgorithmA}; 2010. The entire application is divided into a "front-end" and "back-end".

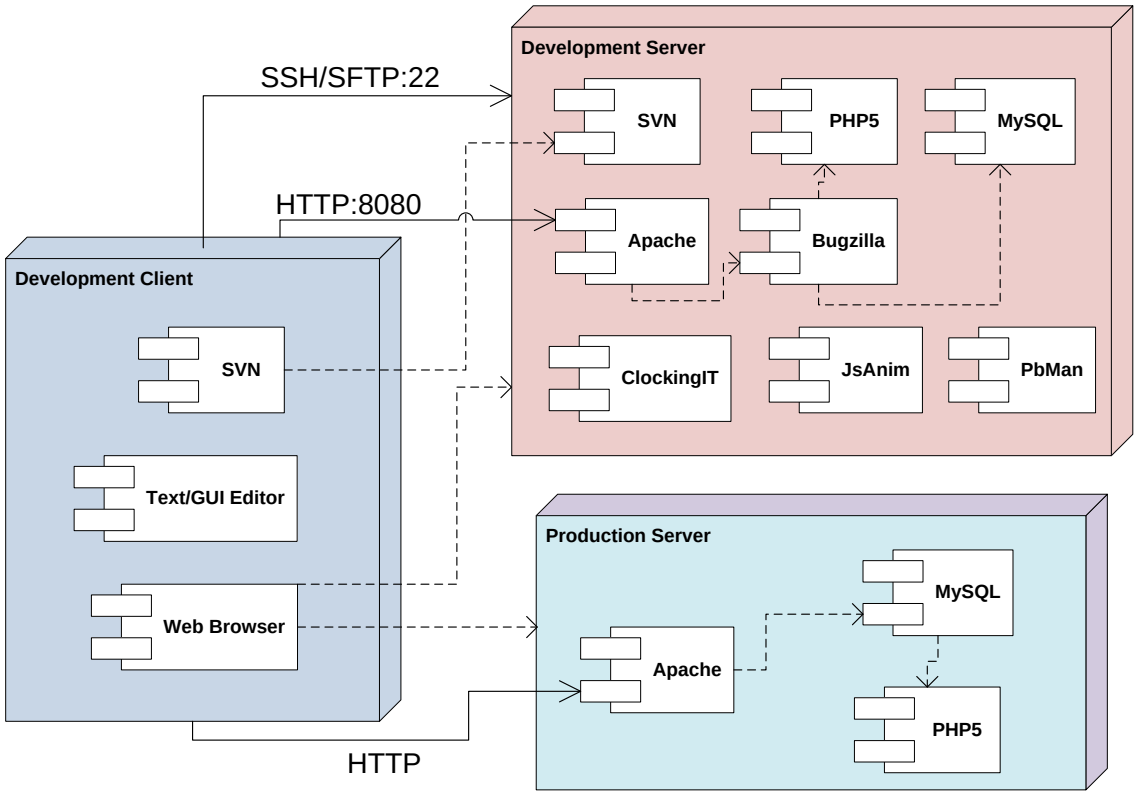



**Deployment Server Diagram**

The initial installation of the system development infrastructure will consist on the installation of two separate application servers running Centos. The first server, a development server, will house all development applications including Bugzilla, SubVersion, Apache HTTP server, PHP, and MySQL. The second server, production server, will be used as the final repository of the final product, which can be used for end users to connect to {AlgorithmA}; 2010.

When the iteration reaches completion, the Subversion repository for iteration #1 will be exported to a production server. The project will be given a web address that users can access through HTTP. Browsers will continue to be supported. The production server will now only contain the latest milestone release. Milestones will be considered a completed iteration. We expect to give our final presentation from this server.

### 2.1.1.2 Development Server




**Deployment Servers Diagram**

Pictured above is the deployment diagram planned by the server team. The purpose of a development deployment is to outline what services will be required in each Autonomous System.

## 2.1.2 User Interfaces

{AlgorithmA}; 2010 consists of four basic user interfaces. The first interface will consist of the website itself and will serve to provide the navigation between all of the other interfaces in {AlgorithmA}; 2010. The second interface will be an animation interface, which will provide a graphical animation of an algorithm. The third interface will be a walk‐through interface, which will consist of two panes, one for the animation of an algorithm, and the other for a walk‐through of code that corresponds to the animation. The fourth interface will provide specific context-sensitive help for hovering and operating each animation. In addition, a structured non-technical file will be provided within {AlgorithmA}; 2010 explaining mathematical concepts for each algorithm.

#### 2.1.2.1 Website Interface

The primary interface will consist of the website itself and will serve to provide the navigation between all of the other interfaces in {AlgorithmA}; 2010.
In iteration 1, the website interface will change, reflected in the following ways:

- A new {{AlgorithmA}; 2010 logo.
- A new menu system that is easy to access and not cluttered with unfinished details
- A new color scheme to fit the overall design of both the logo and characters (if any) therein.
- Compact design to maintain a visual preference for those with screen resolutions 1024x768 and up.
- Simple overall design so as to not overwhelm the user with information and action.

#### 2.1.2.2 Animation Interface

This first iteration will redo the entire animation interface. The overall goal is to keep the system simple and concise. There only need to be the bare minimum of interactive interfaces for any user to have for the majority of the animations we create. Our primary focus is to maintain an educational overview of how a specific algorithm functions by making the process as simple as possible.

#### 2.1.2.3 Walk-Through Interface

We shall also be including a walk-through interface that was founded back in 2008 {AlgorithmA}; . We will carry this interface through to our reconstruction in JavaScript but by making the code found within to be easier for those without a programming background can observe and understand easily.

#### 2.1.2.4 General Information Interface

Along with the animation in this iteration, we will include an interface which will provide general information about the algorithm presented. This interface will provide a brief description of the algorithm, diagrams, and history of the algorithm.

### 2.1.3 Software Interfaces

{AlgorithmA}; 2010 requires a web browser to be viewed. Any web browser may be used that follows the W3C web standards.

### 2.1.4 Communication Interfaces

{{AlgorithmA}; 2010 will be implemented using JavaScript. All client-side communication with the application shall use HTTP from the client's internet browser. Server-side communication will be controlled by the web-server.

### 2.1.5 Memory Constraints

{AlgorithmA}; 2010 will require 128 MB of RAM to be viewed.

### 2.1.6 Operations

{AlgorithmA}; 2010 will be maintained during the winter quarter (January to March of 2010) and operated 24 hours a day, 7 days a week throughout the year. The maintenance will be conducted by CS455, Inc., and the hosting will be provided by the CSE Department of California State University, San Bernardino.

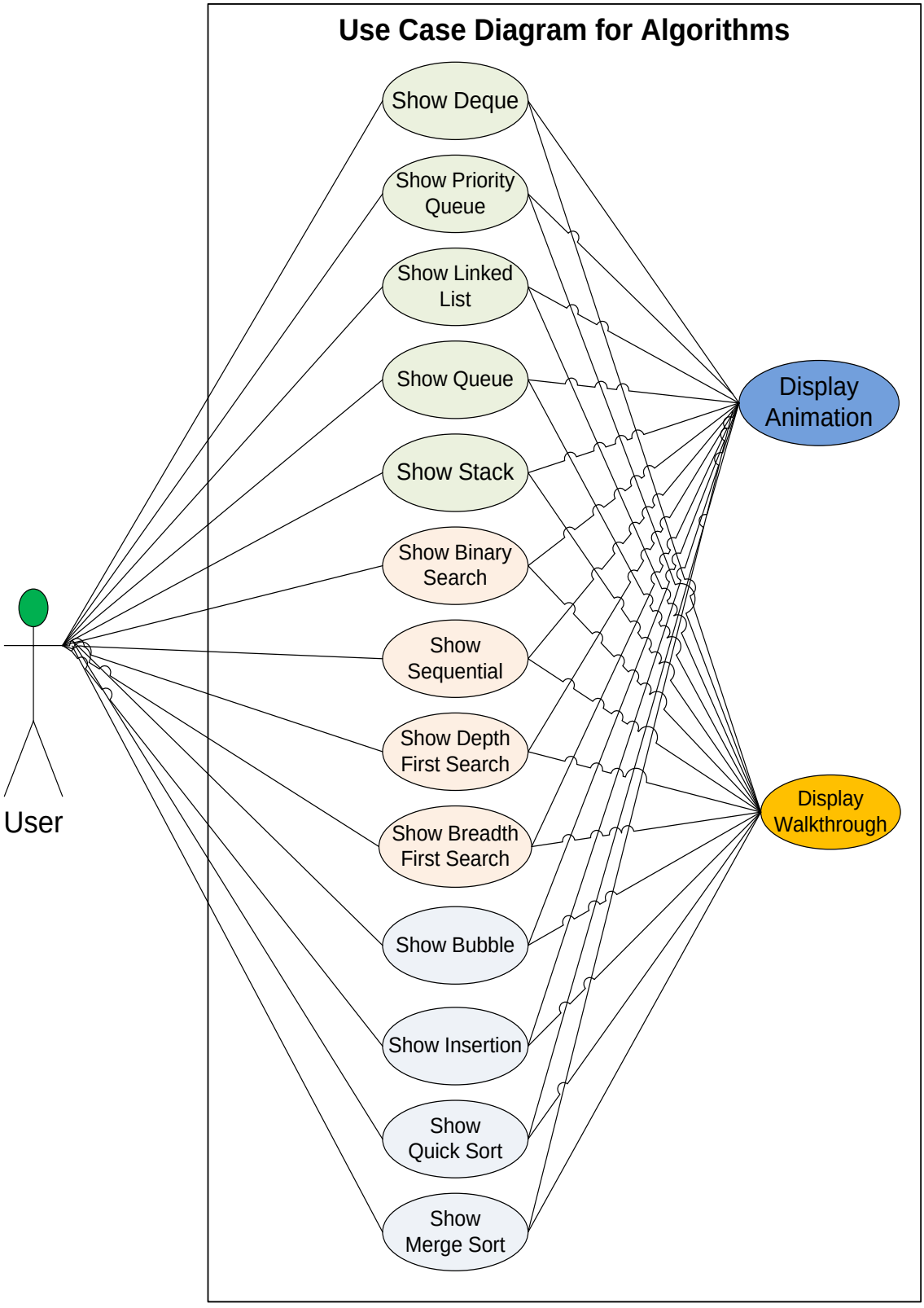
### 2.1.7 Site Adaptation Requirements

The on-campus workstations meet the current requirements of the JavaScript. Home workstations will need to ensure that the requirements in section 2.1.5 are met.

## 2.2 Product Functions

The use case diagram presented below illustrates which functions an end-user may choose to perform in {AlgorithmA}; 2010 as far as the scope of sorting algorithms is concerned. Walkthroughs and animations will be ported into iteration 1.
Small enhancements will be made to the currently existing animation system to further ensure that the user understands the meaning behind the animation. Such enhancements include:

1. Show the pseudo-code by default
2. Change the colors of the text to make them easier to read


Use Case Diagram for Algorithms
Show Deque
Show Priority Queue
Show Linked List
Show Queue
Show Stack
Show Binary Search
Show Sequential
Show Depth First Search
Show Breadth First Search
Show Bubble
Show Insertion
Show Quick Sort
Show Merge Sort
Display Animation
Display Walkthrough
User

## 2.3 User characteristics

The target users of {AlgorithmA}; 2010 are students that attend Computer Science 201 and 202 courses (Introduction to C++ and Intermediate C++ respectively). These users are expected to have no prior knowledge in programming languages. {AlgorithmA}; 2010 is meant to visually display algorithms and common data structures in order to better facilitate a means of learning for these students.

## 2.4 Constraints

{AlgorithmA}; 2010 and algorithms module shall be functioning and deployed for presentation for the client by Finals week of the winter quarter, 2010. All program's faults must identified and fixed prior to the final demonstration, a presentation of its progress will be delivered during an analysis that is to occur 7 weeks into the project.

## 2.5 Assumptions and Dependencies

{AlgorithmA}; 2010 assumes that all previous modules and documentation are available and working at an acceptable level. It assumes that the MVC architecture implemented by the previous CSCI 455 Inc., is functional and ready for inclusion of code necessary to facilitate {AlgorithmA}; 2010's core functionality.

# 3. SPECIFIC REQUIREMENTS

## 3.1 External Interfaces Requirements

The logo designed for {AlgorithmA}; 2010:

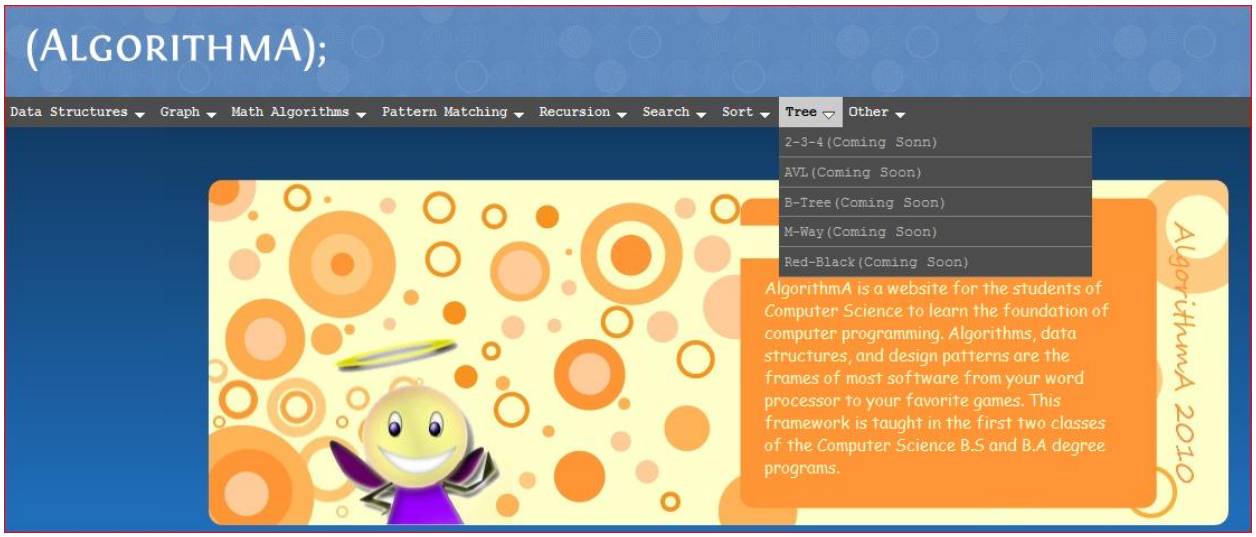

**{AlgorithmA}; 2010 Logo**

### 3.1.1 User Interfaces

When the user first visits the {AlgorithmA}; website, there will be a navigation bar on top, header, footer, and the main frame at the center of page. The navigation bar will have options of the different algorithms associated with computer science. Then each of those topics will feature a drop menu showing the algorithms associated with that topic.

#### 3.1.1.1 General Interface

When the end-user first visits the {AlgorithmA}; 2010 website. In addition, this page will contain the logo and a synopsis of {AlgorithmA}; 2010 capabilities and functions. Below is a concept of the introduction page of {AlgorithmA}; 2010.




**{AlgorithmA}; 2010 Main Webpage**

### 3.1.2 Hardware Interfaces

a. Server side
The web application will be hosted on one of the departments' Linux servers and connecting to one of the school Internet gateways. The web server is listening on port 8080.

b. Client side
The system is a web based application; clients are requiring using a high speed Internet connection and using a up-to-date web browser such as Microsoft Internet Explorer and Mozilla Firefox.

### 3.1.3 Software Interfaces

#### 3.1.3.1 JavaScript

JavaScript will be implemented throughout the website in order to display the correct feature the user requested. Once the user makes a request to see a specific walkthrough or animation, JavaScript sends a request to provide that particular feature.

#### 3.1.3.2 Communication interfaces

{AlgorithmA}; 2010 is designed to be viewed on any internet browser, provided that:

1. JavaScript is enabled
2. Images are enabled

Performance may vary slightly between browsers. However, the functionality of the site should not be impaired.

## 3.2 Functional Requirements

The functions specified in this section directly correspond to work that will be conducted on the {{AlgorithmA}; 2010 project. Given the large scope of the project, it may not be possible to complete all specified components in the time frame that is given. If this is the case, the {{AlgorithmA}; 2010 team will perform whatever measures are necessary to complete the specified components and render them in a functional state. This means constructing modules that follow the appropriate architecture while avoiding technical, artistic, and time consuming elements that are implied given the nature of the project. More on this is specified in the SPMP and SQAP documents.

### 3.2.1 Data Structures – Deque

#### 3.2.1.1 Overview

Deque is a data structure much like a queue or stack but can be accessed inserted and removed from any position in the structure. 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.

#### 3.2.1.2 Layout

Already classified under Data Structures, the user will be presented with a brief introduction of the function and the basic already "useful" deque 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 walkthrough that is highlighted as actions are performed.

#### 3.2.1.3 Functionality

Deque should only contain the bare minimum. This includes:

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

#### 3.2.1.4 Data Structures - Priority Queue

#### 3.2.1.5 Overview

The user will be presented with a tree and two branches. Also the array on numbers to be sorted will be presented at the bottom of screen. To the left of the animation work area will be a pseudo-code walkthrough that is highlighted as actions are performed.

#### 3.2.1.6 Layout

In order to run Priority Queue, the user will input a number. The number is then inserted into a random tree that has already been generated. It starts at the bottom as a leaf and if it is larger than its parent, it replaces the parent and becomes the new parent. The old parent takes the place that the newly inserted leaf would have occupied. The while loop is triggered for this swap and then it checks the next parent to see if it is larger than the current node. If so, it moves up again. This continues until the leaf that was pushed is placed in the right position in relation to the other numbers that are already in the tree.

#### 3.2.1.7 Functionality

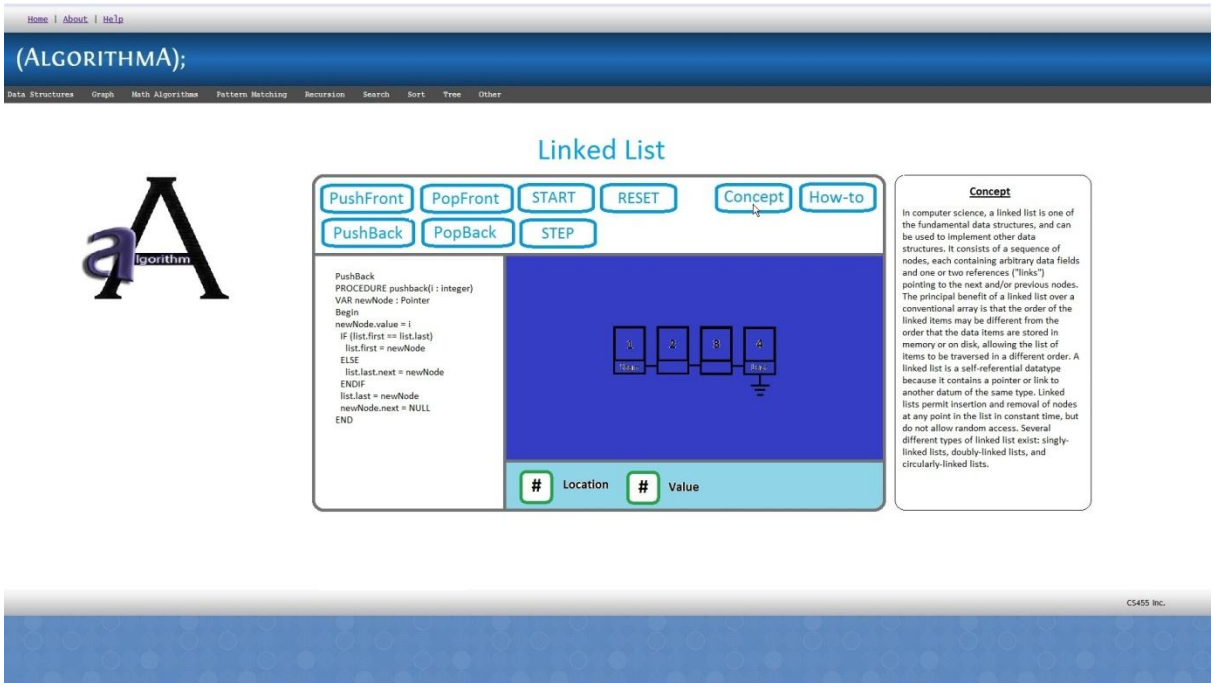
The queue should only contain the bare minimum. This includes:

- Push Back – Push a random element to the structure to the back of the queue
- Pop Front – Remove an element from the front of the queue
- Front – Report the front element
- Back – Report the back element
- Reset – Reset to the initial layout when first loaded.

### 3.2.2 Data Structures – Linked List

#### 3.2.2.1 Overview

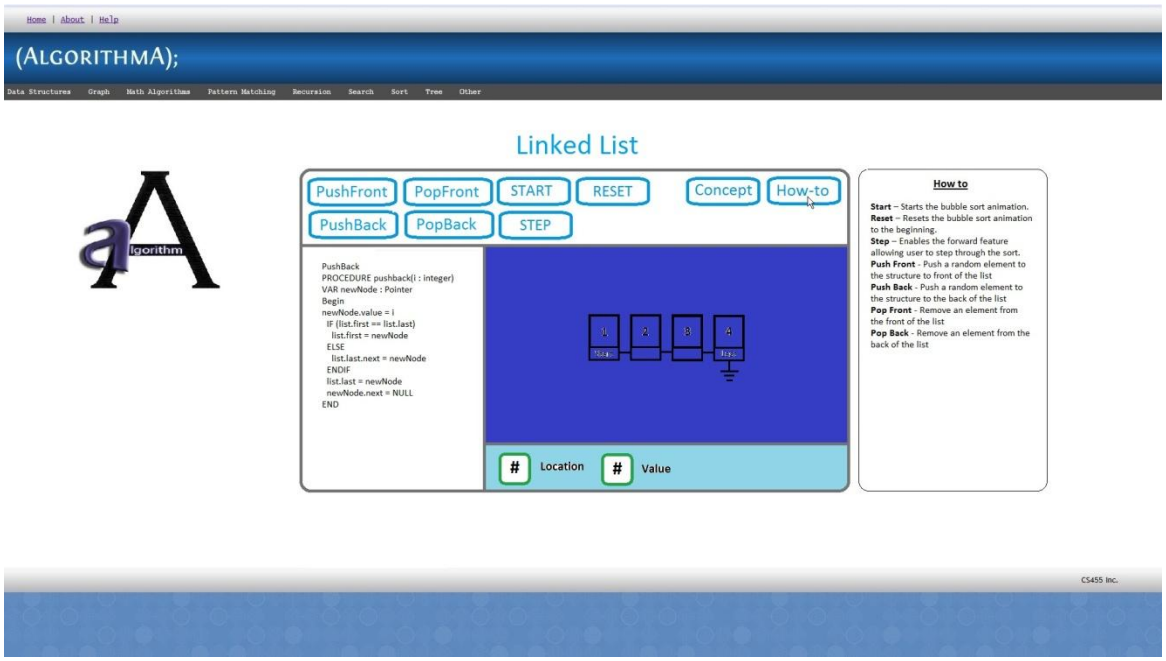
A linked list is a data structure that can add and remove elements from the front and the back of the list. Traversing can only be done using an iterator and removal of any element is relatively easy. In order to show the bare minimum of its workings, the walk-through process must step a user through the process of adding, removing, and possibly iteration through the structure itself.




**Linked List with Concept help**

### 3.2.2.2 Layout

Already classified under Data Structures, the user will be presented with a brief introduction of the function and the basic already "useful" linked list 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 walkthrough that is highlighted as actions are performed.




**Linked List with Function help**

#### 3.2.2.3 Functionality

The linked list should only contain the bare minimum. This includes:

- Push Front – Push a random element to the structure to the front of the list
- Push Back – Push a random element to the structure to the back of the list
- Pop Front – Remove an element from the front of the list
- Pop Back – Remove an element from the back of the list
- Remove Random – Remove a random element from the list
- Reset – Reset to the initial layout when first loaded.

### 3.2.3 Data Structures – Queue

#### 3.2.3.1 Overview

A queue is a data structure that can only add from the back and remove from the back. 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.

#### 3.2.3.2 Layout

Already classified under Data Structures, the user will be presented with a brief introduction of the function and the basic already "useful" queue 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 walkthrough that is highlighted as actions are performed.

#### 3.2.3.3 Functionality

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

- Push Back – Push a random element to the structure to the back of the queue
- Pop Front – Remove an element from the front of the queue
- Front – Report the front element
- Back – Report the back element
- Reset – Reset to the initial layout when first loaded.

### 3.2.4 Data Structures – Stack

#### 3.2.4.1 Overview

Heap is a data structure that can only add and remove elements from the top (first in first out). 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.

#### 3.2.4.2 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 walkthrough that is highlighted as actions are performed.

#### 3.2.4.3 Functionality

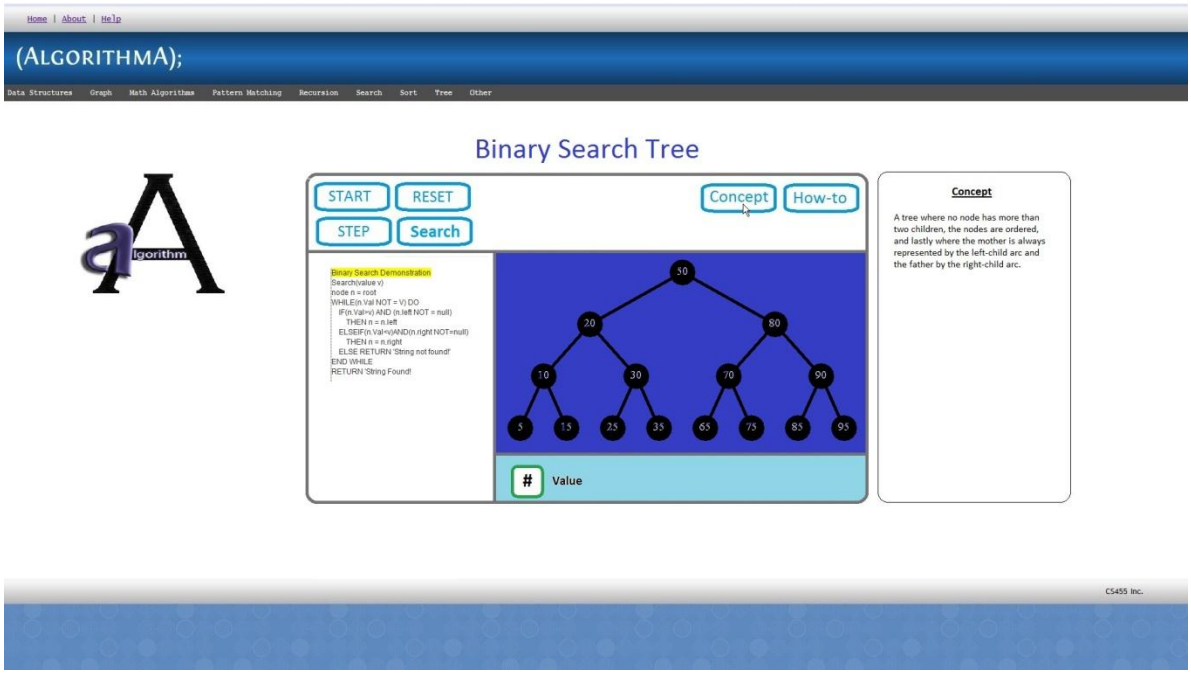
Stack should only contain the bare minimum. This includes:

- Push Back – Push a random element to the structure to the top of the stack.
- Pop Back – Remove the top element of the stack.
- Back – Reports to the user the last element of the stack.
- Reset – Reset to the initial layout when first loaded.

### 3.2.5 Search – Binary Search Tree

#### 3.2.5.1 Overview

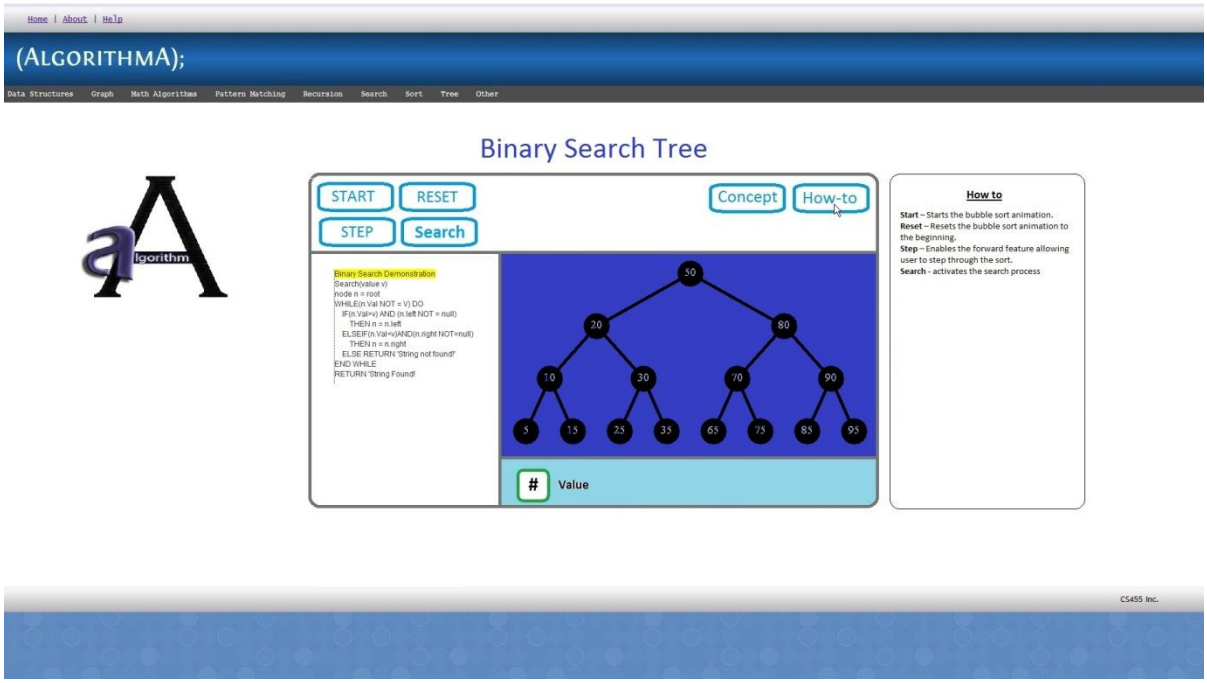
Much like a heap, this is a tree search algorithm to search for sequential nodes using a generic "if greater than, lower than" equality. In order to show the bare minimum of its workings, the walk-through process must step a user through the process of finding any element in a given tree.




**Binary Search Tree with Concept help**

#### 3.2.5.2 Layout

Reclassify this under Search. The user will be presented with a brief introduction of the function and the basic already "useful" tree to perform a search on (must be a tree with at least 4 levels to it). To the left of the animation work area will be a pseudo-code walkthrough that is highlighted as actions are performed.




**Binary Search Tree with Function help**

### 3.2.5.3 Functionality

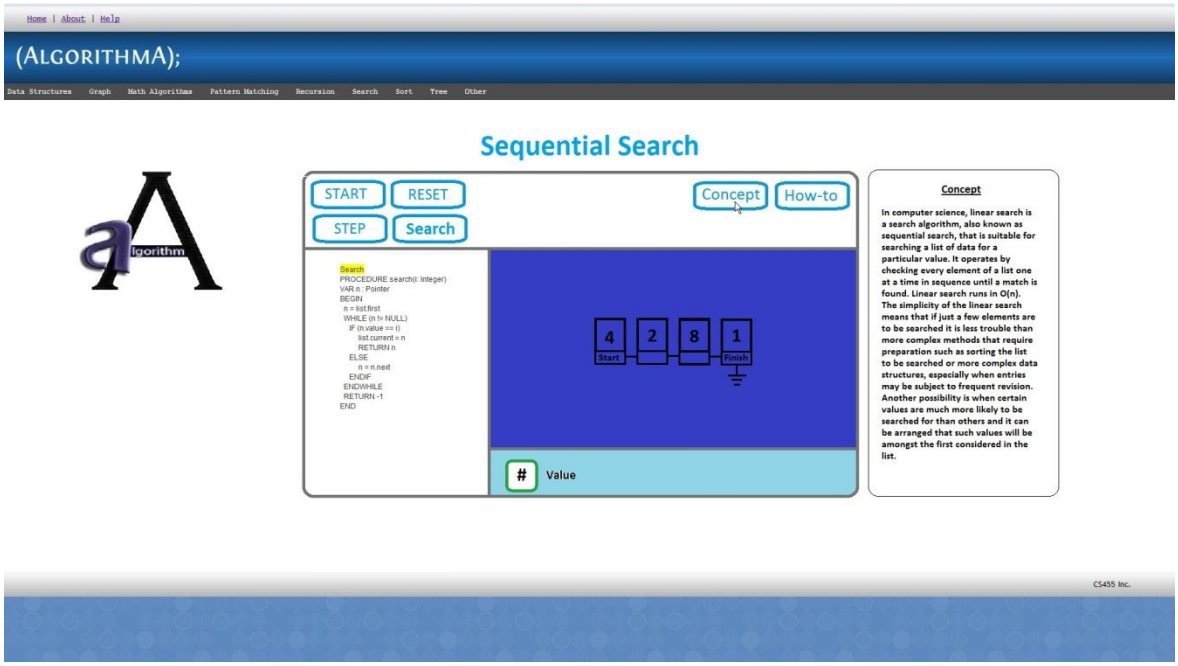
BST should only contain the bare minimum. This includes:

- Search – Search for a random element found in the tree.
  - This includes 1-2 instances of a failed search
- Step Through (Check box) – Will perform the next search with a prompt or button to allow the user to move on to the next step manually.
- Pause/Play – A single button that will stop and start an already started animation.
- Reset – Reset to the initial layout when first loaded.

## 3.2.6 Search – Sequential Search

### 3.2.6.1 Overview

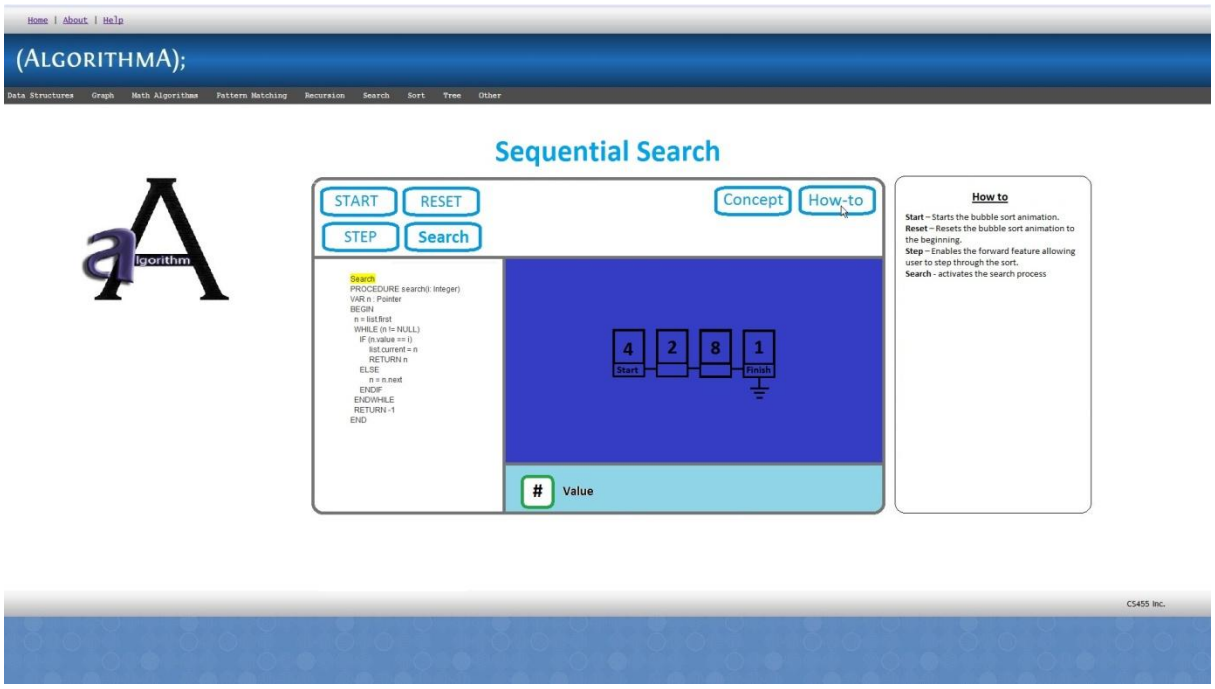
Sequential search is the most basic search where the algorithm searches sequentially though any data structure. In order to show the bare minimum of its workings, the walk-through process must step a user through the process of finding any element in a given data structure.




**Sequential Search with Concept help**

### 3.2.6.2 Layout

Already classified under Search, the user will be presented with a brief introduction of the function and the basic already "useful" list to perform a search on (must be a list with at least 4 elements in it). To the left of the animation work area will be a pseudo-code walkthrough that is highlighted as actions are performed.




**Sequential Search with Function help**

#### 3.2.6.3 Functionality

Sequential Search should only contain the bare minimum. This includes:

- Search – Search for a random element found in the tree.
- This includes 1-2 instances of a failed search
- Step Through (Check box) – Will perform the next search with a prompt or button to allow the user to move on to the next step manually.
- Pause/Play – A single button that will stop and start an already started animation.
- Reset – Reset to the initial layout when first loaded.

### 3.2.7 Search – Depth First Search

#### 3.2.7.1 Overview

Breadth First Search is a graph search algorithm 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. This is typically implemented with a queue.

#### 3.2.7.2 Layout

When breadth first search is selected, the basic layout of the window authoring system will be presented. The pseudo code for representing the use of a queue will be displayed, and will be able to see how it works through an animation.

#### 3.2.7.3 Functionality

- The breadth first search will have the following functionalities:
- Start – Start the animation of what the pseudo code does at run-time.
- Break – Will stop the animation
- Step – Allow the user stop at one line to better understand what the code is doing.

### 3.2.8 Search – Breadth First Search

#### 3.2.8.1 Overview

Depth first search is (1) any search algorithm that considers outgoing edges of a vertex before any neighbors of the vertex that is, outgoing edges of the vertex's predecessor in the search. Extremes are searched first. This is easily implemented with recursion. (2) An algorithm that marks all vertices in a directed graph in the order they are discovered and finished, partitioning the graph into a forest.

#### 3.2.8.2 Layout

The applet has two different layouts. One layout for a walkthrough of the algorithm and another layout for an animation displaying a depth first search.

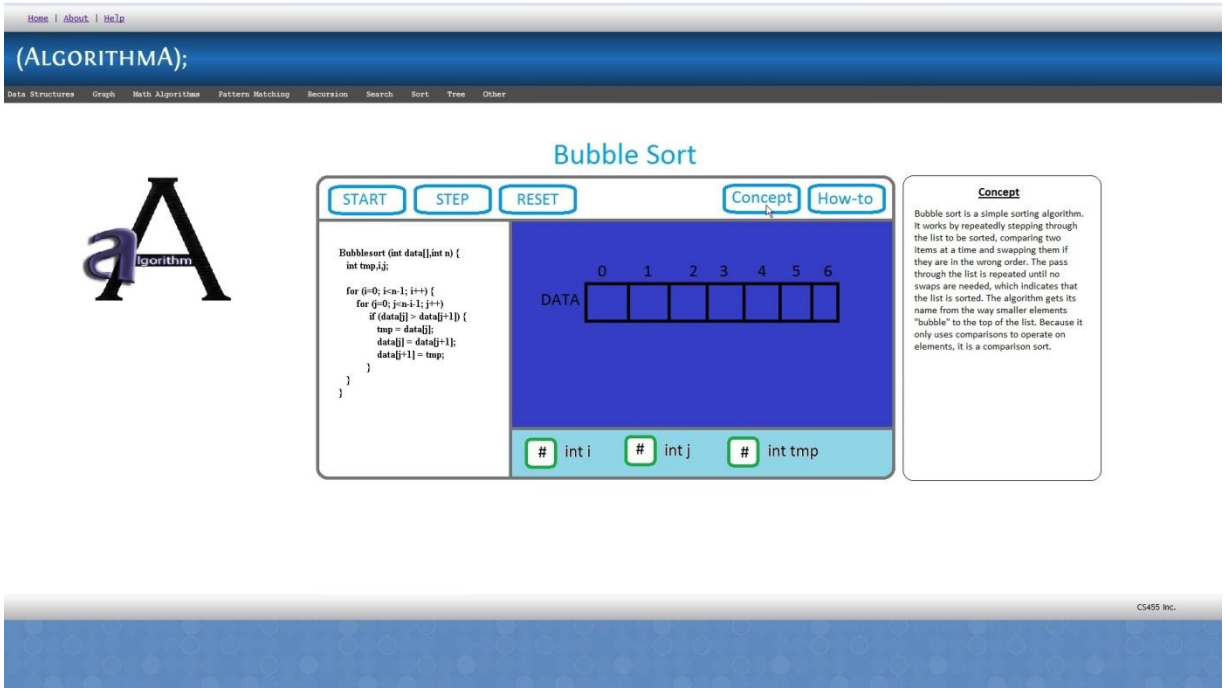
### 3.2.8.3 Functionality

When a user wants to progress through the algorithm they will press the "Forward" button inside the applet.

## 3.2.9 Sort – Bubble

### 3.2.9.1 Overview

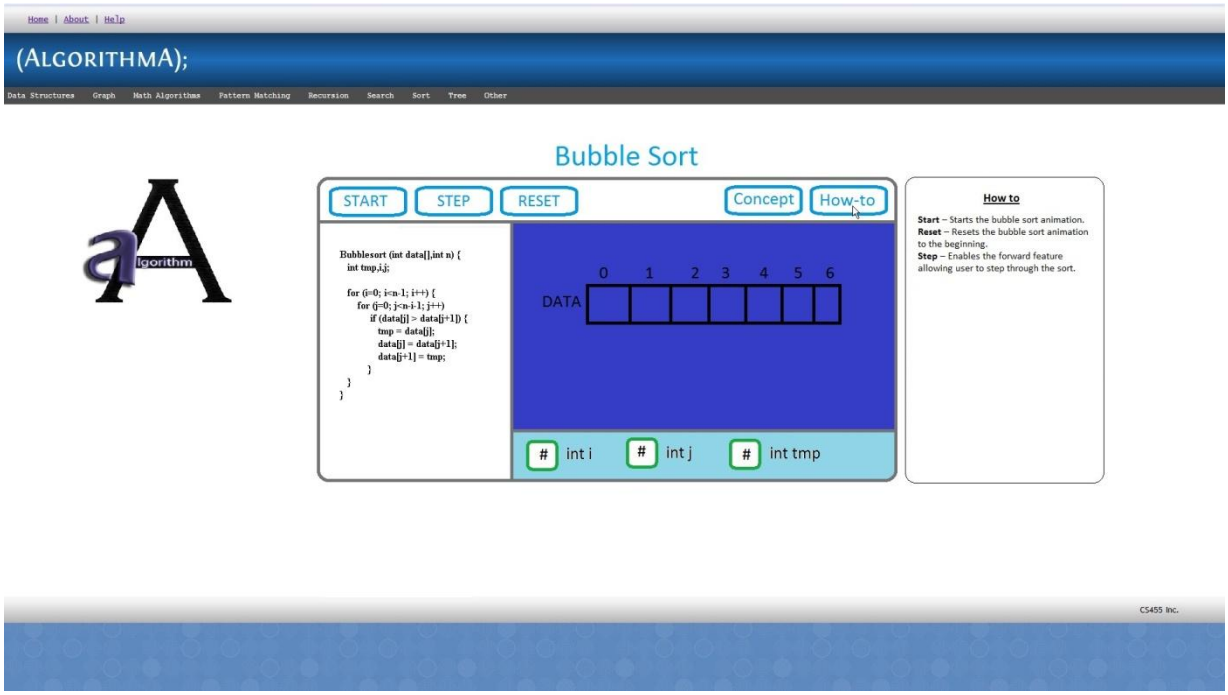
Bubble Sort is the most basic sorting algorithm in which elements are moved one by one depending on a general "less than, greater than" equality. In order to show the bare minimum of its workings, the walk-through process must step a user through the process of finding any element in a given data structure.




**Bubble Sort Animation with concept help**

### 3.2.9.2 Layout

Already classified under Sort, the user will be presented with a brief introduction of the function and the basic already "useful" list to perform a sort on (must be a list with at least 10 elements in it for a minimum number of 100 steps). To the left of the animation work area will be a pseudo-code walkthrough that is highlighted as actions are performed.




**Bubble Sort Animation with Function help**

### 3.2.7.2 Functionality

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

- Sort – Sorts the given data structure while showing it steps with a low delay.
- Step Through (Check box) – Will perform the next sort or step while sorting with a prompt or button to allow the user to move on to the next step manually.
- Pause/Play – A single button that will stop and start an already started animation.
- Reset – Reset to the initial layout when first loaded.

## 3.2.8 Sort – Insertion

### 3.2.8.1 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.

#### 3.2.8.2 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.

#### 3.2.8.3 Functionality

- Start – Starts the Insertion sort animation.
- Reset – Resets the Insertion sort animation to the beginning.
- Pause – Pauses the animation at a desired point in the sort.
- Step – Enables the forward feature allowing user to step through the sort.
- Forward – Allows the user to step through the sort one step at a time.
- Show Code – Allows user to see the Insertion sort source code.
- Show Animation – Allows user to see Insertion sort animation.
- Speed – Allows user to speed up or slow down the animation.

### 3.2.9 Sort – Merge

#### 3.2.8.1 Overview

Merge is a comparison-based sorting algorithm in which the divide and conquer methodology is employed. In order to show the bare minimum of its workings, the walk-through process must step a user through the process of finding any element in a given data structure.

#### 3.2.8.2 Layout

Already classified under Sort, the user will be presented with a brief introduction of the function and the basic already "useful" list to perform a sort on (must be a list with at least 10 elements in it for a minimum number of 10 steps). To the left of the animation work area will be a pseudo-code walkthrough that is highlighted as actions are performed.

#### 3.2.8.3 Functionality

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

- Sort – Sorts the given data structure while showing it steps with a low delay.
- Step Through (Check box) – Will perform the next sort or step while sorting with a prompt or button to allow the user to move on to the next step manually.
- Pause/Play – A single button that will stop and start an already started animation.
- Reset – Reset to the initial layout when first loaded.

### 3.2.9 Sort – Quick

#### 3.2.9.1 Overview

Quick sort is a sorting algorithm that makes use of partitioning and comparisons to get most of the work done. In order to show the bare minimum of its workings, the walk-through process must step a user through the process of finding any element in a given data structure.

#### 3.2.9.2 Layout

Already classified under Sort, the user will be presented with a brief introduction of the function and the basic already "useful" list to perform a sort on (must be a list with at least 10 elements in it for a minimum number of 10 or 100 steps). To the left of the animation work area will be a pseudo-code walkthrough that is highlighted as actions are performed.

#### 3.2.9.3 Functionality

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

- Sort – Sorts the given data structure while showing it steps with a low delay.
- Step Through (Check box) – Will perform the next sort or step while sorting with a prompt or button to allow the user to move on to the next step manually.
- Pause/Play – A single button that will stop and start an already started animation.
- Reset – Reset to the initial layout when first loaded.

### 3.3.1 Walkthroughs

The walkthrough interface allows the user the interactively step through an algorithm animation.

## 3.4 Performance Requirements

Unlike previous versions of AlgorithmA, performance is the key to our version. The move from Java to JavaScript is based almost entirely on the fact that Java is not meant to be used in such an environment that we are using it for. The requirements therefore reflect the need for a much smoother and readily available interface. Animations must be smooth; data retrieval must be fast, and navigation. We can offer no guarantee of the expected time delivery of the content to the requester due to varying rates at which data may be sent via the Internet.

## 3.5 Design Constraints

Since we are starting from scratch, having to port legacy code like in previous AlgorithmA projects will not be performed. We are therefore free to design and implement to at our leisure. But we must ultimately conform to an easy to document and understand standard. We must also constrain ourselves to the Coding Standards document.

## 3.6 Software System Attributes

The legacy for the future generations of AlgorithmA, otherwise known as {AlgorithmA}; 2010, will be an implemented architecture that will bring simplicity to maintaining and extending the current code base. Besides coding, a much larger emphasis will be placed on documenting the iteration, by providing a knowledgebase of information. Such documentation include: detailed descriptions of code in the actual source code, and a comprehensive log of all activity via the wiki dedicated to {AlgorithmA}; 2010. [http://wiki.algodev.ias.csusb.edu:8080/wiki/Main_Page]

## 3.7 Other Requirements

{AlgorithmA}; 2010 must be supportable: It has to be maintained well enough to be smoothly taken over by the next CS455, Inc. {AlgorithmA}; 2010 must be easy to use for instructors and students interested in learning about sorting and mathematical algorithms. {AlgorithmA}; 2010 must be reliable.
Moreover, {AlgorithmA}; 2010 must continue to run on the computer science school web site well after development for future project groups to access. It should be available 24 hours a day, 7 days a week, and 365 days a year. The only time when the site will become unavailable is during short maintenance periods.

## 3.8 Documentation

Three levels of documentation will be provided with the {AlgorithmA}; 2010. A hyper context-sensitive help pop up screen will be implemented throughout each algorithm. A detailed installation technical guide will be developed as supporting documentation for the next generations of {AlgorithmA}; . And finally, a User's Manual will be created based on the implementation of the new application. The User's manual will cover explanation behind the theory of each algorithm as well as how to execute and manipulate each control presented to the user when executing each module. The final documentation will be available in a PDF format.

Proofreading: Patrick O'Connor
Danny Vargas