Teaching Interactive Computer Science

Group 7

Alexander Kjellén
Björn Delin
Erik Skogby
Jan-Erik Bredahl
Joakim Israelsson

Requirements Document

Preface

The expected readership of this document is the project team members.

Version history:

Removing the compiling function in the system

The first version of the system was supposed to have its own compiler so other people could add new data structures and functions to the system. Later we decided that it wasn't realistic and decided not to have it in the system.

Limit and redefine the data structures

The second version of the system had the data structures: Array, Linked list, binary tree, Heap, Skip list and Double linked list. After realizing how much work that was needed for this we decided to skip the data structures: Heap, Skip list and Double linked list. Therefore we redefined the data structures Array, Linked list and Binary tree to Vector, Singly linked list and Sorted binary tree. We found it better to focus on those and make them look good.

Introduction

Who are the users and what problem does the system solve for them?

For beginners in computer science and programming, it can be hard to understand how various data structures and algorithms (D&A) work. These people are not used to code and have problems understanding D&A through code alone. Today, algorithms are visualized on white boards, black boards, overhead projectors or using simple post-it notes. All of these methods can become very messy when demonstrating more complex algorithms. To help people grasp these things, we shall develop a visualization tool to help demonstrate D&A visually. This way, people who are not used to code does not have to struggle with pseudo code, but can focus on the concept of a particular data structure or algorithm. When learning a new data structure or algorithm, a clear visualization will explain things much more clearly than for example a messy white board.

The main users are students studying computer science and are beginning to learn programming, and others who want a deeper understanding of how some algorithms and data structures work. Students will also benefit when teachers use our software during classes since it will make learning easier. At the very least, they must have a beginning knowledge in data structures, algorithms and programming, such as a few weeks into an introductory programming course.

The main uses of the system

What this program does is visually demonstrating algorithms, when using a particular data structure. The user will be able to load a data structure and the accompanying algorithms and functions from a menu. The data structure will then be visually represented on the screen, in its current state. The user can then run the functions belonging to the data structure. The visualization will then be animated to display how the function works, how objects interact with each other and how objects are created and destroyed. The objects will move around, numbers will be updated and pointer arrows will change target.

Timothy is a computer science student who doesn't like homework. Tomorrow he and a few other students have to discuss some algorithms with their teacher. To do this, they had to read about 120 pages about these algorithms, something which Timothy of course hasn't done. He is now staring at the book, bored out of his mind. Thinking there has to be an easy shortcut, he starts browsing the internet for some simpler info than that in his book. Although he finds some Wikipedia articles, they are far too long for Timothy, who's tired after watching TV all day. Then after awhile, he stumbles upon the TICS website. According to the website, TICS has animated visualizations of all the algorithms he need to know about. Timothy's delighted when he sees the screenshots and immediately downloads the program. Spurred by the lazy man's energy, he searches through the visualizations available, knowing that one of them shows the algorithms he needs. One after one, he finds the visualizations, loads them and clicks play to watch them animate. Grabbing a bag of potato chips, he leans back and watches the animation. Using his cordless mouse on his thigh, he clicks on "restart" to see the animation again several times.

After awhile he feels confident that he can make the teacher believe that Timothy knows what he's talking about. Timothy goes to sleep, smugly aware that he has fooled the system.

It is possible for computer science teachers to use this program as a presentation tool while teaching students how data structures and algorithms work. Here is another example:

Walter is a computer science teacher that wants to explain to the class how a binary tree works. He brings his computer to the class but goes over the structure on the whiteboard first to explain how it works, and what it can be used for. Now he connects his computer to the projector, starts the program and loads the Binary Tree. The structure gets drawn up so that all students can see how it looks like in a graphical way. He starts the algorithm "insert" and in the animation window objects begin to move around, displaying how pointers get redirected and values changed. When inserting another value into the tree he chooses to step through the animation and for each step explain what is happening. Then Walter goes over a few other functions such as find and delete in the same way.

The context/environment in which the system is to be used

The user requirements

This software is targeted at an academic environment to teach students.
The user should know how to start a java application, know the basics in Windows XP and should be able to understand English. Also, to get any use of this program, the user should have some knowledge about A&D.

The software requirements for the system

- 32-bit operating system “Windows XP”
- SUNs Java 1.5 runtime environment

To be able to download this program from the internet you will need an internet connection and a web browser.

The hardware requirements for the system

The minimal and also recommended hardware requirement for the system is:

- A Windows based PC
- 800 MHz single or dual core processor
- 512 MB of ram
- A graphics card and screen that support the resolution 1024x768
- 50 MB free disc space.

Even though Windows XP and java only needs 300MHz processor, 128 MB of ram and 2 GB disc space we cannot guarantee that our program will run smoothly on such system.

The scope of the system

This tool is not intended to replace teachers or literature, but to complement them. Therefore we will not include extensive documentation on the theory of these data structures. We will however have some documentation on how each function works.

Topic	***In***	***Out***
Show objects and how they relate to each other visually.	X	
Animate this visualization when algorithms are executed on it.	X	
Ability to control the animation speed, pausing it, reversing it, etc.	X	
Show the algorithm code.		X
Adding own data structures and algorithms.		X
Manipulate visualized data structures through supplied functions.	X	
Manipulate visualized data structures directly in the animation window.		X
Extract code documentation from data structure source files.		X
3D graphics.		X
Animated Java2D graphics.	X	

The main factors that need to be taken in to account when designing and building the system

- The visualization has to work in a pedagogic and clear way so that it does not confuse people more than it helps them.
- The program should be able to run on a win XP computer with Java 1.5 installed.
- An error in an animation might cause students to learn the structure incorrectly.
- The program must be able to handle incorrect or otherwise inappropriate input from the user.
- The visualization must scale well so that we can get meaningful views of both smaller and larger structures.
- To make this program meaningful, the right algorithms and data structures must be chosen.

Technologies & Risks

The technology we will use is Java version 1.5. The newest version is 1.6, but it is not supported on all platforms. We believe that Java 1.5 is a stable technology that is good enough for our purposes. We see no serious risks with it. For the custom graphics part, we will use Java2D. We are fairly confident that Java2D is up to the task, performance wise. Still, there is a slight chance that Java2D might not be able to handle the graphic complexity we'll throw at it. In this case, we will simply reduce the graphical complexity of our software.

Another risk is that we have a pretty vague idea on how complex our project really is. This gives us a fairly high risk of underestimating the complexity. Our strategy here is to have a priority on each feature, and if the project gets too complex, start dumping features until we can handle it.

There is also a pedagogical risk here. Our visualizations need to help more than they confuse. If they don't, then there is no reason to do this software. Our estimate is that there is a low to medium chance of this happening. Since this is simply not an option, we will have to test the system on people unfamiliar with the subject during the development cycle. We might also contact some HCI (Human-Computer Interaction) people to help us on this one.

For the project to be useful, we need to have a certain amount of data structures and algorithms to show. One risk is that we might not have enough to show that people will find useful. In this case, we need to increase the complexity of the software, which might conflict with the risk above, which the complexity can get out of hand. Our strategy here is to minimize this risk by making it as easy as possible for us to add new content. This will be a high priority when designing the system.

Glossary

- **A&D** – Algorithms and data structures
- **Algorithms** – A definite list of well defined instructions for completing a task. In computer science there are lots of known algorithms that that is proved to meet the goal in an efficient way.
- **Animation** – A rapid display of sequence of images in order to create an illusion of movement. In this program the animation will consist of objects moving from one place to another.
- **Computing Science -** The study of the theoretical foundations of information and computation and their implementation and application in computer systems.
- **Data structures** – A way of storing data in a computer so that it can be used efficiently.
- **Function –** A portion of code within a larger program which performs a specific task, in our case it runs a specific algorithm.
- **Graphical User Interface -** a type of user interface which allows people to interact with a computer and computer-controlled devices
- **GUI -** Graphical User Interface
- **Index –** An integer which identifies an element or data structure which enables fast lookup.
- **Parameter** – A variable which takes on the meaning of a corresponding argument passed in a call to a function.
- **Singly linked list –** A data structure consisting of a sequence of nodes each containing arbitrary data fields and one reference pointing to the next node.
- **Sorted binary tree –** A data structure consisting of nodes where every node has at most two children. The “left” pointer points to a node with a lower value than itself and the “right” pointer points at a node with at least as high value.
- **TICS** – Teaching interactive computer science, the name of our system.
- **Vector** – A one dimensional data structure consisting of a group of elements that are accessed by indexing.
- **Visualization** – A technique for creating animations to communicate a message. In this case it means creating an animation that will describe an algorithm or a data structure.

User requirements definition

Functional requirements

Explanation of functional requirements headers
Input: *The information that the functional requirement needs to work as specified. How wrong or insufficient input is handled is specified in the Error Handling section.*
Output: *The result of the functional requirement assuming successful execution.*
Dependencies: *Which, if any, functional requirements that must be implemented for the current one to work.*
Definition: *Explanation of what the functional requirement shall do if the input is correct.*
Error handling: *How possible errors such as invalid input shall be handled.*
Testing: *How we will verify that the functional requirement is implemented correctly. Since the requirements are relatively high-level, we will not include all the unit-tests required for different implementations.*

General functional requirements

Create a data structure
Input: All necessary parameters for a certain data structure.
Output: The data structure is created and displayed.
Dependencies: At least one of the data structures below must be implemented.
Definition: A data structure is created according to the parameters and displayed on the screen. Any data structure already created shall be removed.
Error handling: If the parameters are incorrect in some way, the user shall be notified of this and the data structure will not be created. Any data structure already created shall not be removed.
Testing: All available data structure creations will be tested using both erroneous and correct input parameters. Removal of previously created data structures shall also be tested.

Run a function
Input: All necessary parameters for the function.
Output: A paused animation.
Dependencies: A data structure must have been created.
Definition: The function is executed on the current data structure. An animation that visualizes the flow of the execution of this function is created and set into a paused state.
Error handling: If the parameters are incorrect in some way, the user shall be notified of this and the function shall not be executed, nor the animation created.
Testing: All available data structure functions will be tested using both erroneous and correct input parameters.

Data structures

These requirements regard which data structures should be implemented for visualization in the program. The definitions are about what shall be visualized, not necessarily how it should be implemented.

Array

Input: None.
Output: None.
Dependencies: None.
Definition: This is the basic data structure where the elements are stored in a contiguous block of memory that can be randomly accessed. The array will have a minimum size of 0 and a maximum size of 32 elements. The elements stored in the array shall be integers.
Error handling: Error handling is done by the functions operating on the data structure, not the data structure itself.
Testing: When all the functions operating on this data structure pass their test, then the data structure is also correct.

Singly linked list

Input: None.
Output: None.
Dependencies: None.
Definition: The data structure will have a minimum size of 0 and a maximum size of 32 elements. The elements stored in the singly linked list shall be integers.
Error handling: Error handling is done by the functions operating on the data structure, not the data structure itself.
Testing: When all the functions operating on this data structure pass their test, then the data structure is also correct.

Sorted binary tree

Input: None.
Output: None.
Dependencies: None.
Definition: The data structure will have a minimum size of 0 and a maximum size of 31 elements. The elements stored in the sorted binary tree shall be integers.
Error handling: Error handling is done by the functions operating on the data structure, not the data structure itself.
Testing: When all the functions operating on this data structure pass their test, then the data structure is also correct.

Available functions

Some general error handling for all functions is that all inputs will be checked for validity. Any invalid input will cancel the function and the user will be notified about the invalid input.

Add first

Input: The function shall take a integer with a value from 0 to 10000
Output: None
Dependencies: Array and Singly linked list must have been implemented.
Implemented on: The function shall work on Array and Singly linked list.
Definition: An animation will be created showing how the supplied value is inserted first into the structure. When the animation has ended the data structure shall be updated to include the new element.

Error handling: If the parameters are incorrect in some way, the user shall be notified of this and the function shall not be executed. If the data structure is at the maximum size, the user shall be notified and the function will not execute.
Testing: Will be tested by adding an element to the data structure that empty, not empty and when it is at its maximum size. Will also test by adding positive and negative integers as well as characters.

Insert

Input: An integer between 0 and 10000 inclusive, and an index where the integer will be inserted, where the index ranges from 0 to N-1 where N is the structure size and,.
Output: None.
Dependencies: Array and Singly linked list must have been implemented.
Definition: An animation will be created showing how the supplied value is inserted at the supplied index in the data structure. When the animation has ended, the data structure shall be updated to include the new element.
Error handling: If the parameters are incorrect in some way, the user shall be notified of this and the function shall not be executed.
Testing: Will be tested by adding an element to the data structure that empty, not empty and when it is at its maximum size. Will also test adding positive, negative integers and also adding characters. Will also test invalid indexes.

Search

Input: An integer from 0 to 10000.
Output: Boolean value
Dependencies: Array, Singly linked list and Sorted binary tree must have been implemented.
Implemented on: The function shall work with Array, Singly linked list and Sorted binary tree.
Definition: Will create an animation showing how the data structure is searched for the supplied element. If the value is found, TRUE is returned, otherwise FALSE.
Error handling: If the parameters are incorrect in some way, the user shall be notified of this and the function shall not be executed.
Testing: Will be tested by searching for characters and positive and negative integers.

Set random values

Input: An integer from 0 to the structure's maximum size.
Output: None
Dependencies: Array, Singly linked list and Sorted binary tree must have been implemented.
Implemented on: The function shall work with Array, Singly linked list and Sorted binary tree.
Definition: Will update the data structure. The supplied input will decide the size of the data structure. All elements in the updated data structure will have random values.
Error handling: If the parameters are incorrect in some way, the user shall be notified of this and the function shall not be executed.
Testing: Will be tested by setting the size to a character or positive and negative integer.

Remove

Input: An integer from 0 to 10000.
Output: Boolean value
Dependencies: Array, Singly linked list and Sorted binary tree must have been implemented.
Implemented on: The function shall work with Array, Singly linked list and Sorted binary tree.

Definition: Will create an animation of how the data structure is searched for the supplied element. If the element is found, the animation will also show how the element is removed from the data structure. If the element was removed, True is returned, otherwise False. If the element was found, then the data structure shall be updated to exclude the removed element when the animation has ended.
Error handling: If the parameters are incorrect in some way, the user shall be notified of this and the function shall not be executed.
Testing: Will be tested by trying to remove an element with value of a character, positive integer and a negative integer.

Remove by index

Input: An integer from 0 to 31.
Output: Boolean value.
Dependencies: Array must have been implemented.
Implemented on: The function shall work with Array.
Definition: Will create an animation of how the data structure is searched for the supplied element. If the element is found, the animation will also show how the element is removed from the data structure. If the element was removed, True is returned, otherwise False. If the element was found, then the data structure shall be updated to exclude the removed element when the animation has ended.
Error handling: If the parameters are incorrect in some way, the user shall be notified of this and the function shall not be executed.
Testing: Will be tested by trying to remove an element at a index that is a character, positive integer and a negative integer.

Search

Input: An integer from 0 to 10000.
Output: Boolean value.
Dependencies: Array, Singly linked list and Sorted binary tree must have been implemented.
Implemented on: The function shall work with Array, Singly linked list and Sorted binary tree.
Definition: Will create an animation of the data structure is searched for the first occurrence of the supplied element. If the element was found, True is returned, otherwise False.
Error handling: If the parameters are incorrect in some way, the user shall be notified of this and the function shall not be executed.
Testing: Will be tested by trying to search for an element with a value of a character, positive integer and a negative integer.

Insertion sort

Input: None.
Output: None.
Dependencies: Array and Singly linked list must have been implemented.
Implemented on: The function shall work with Array and Singly linked list.
Definition: Will create an animation of how the data structure is sorted by using the insertion sort algorithm. When the animation has ended, the data structure shall have been updated to represent the sorted data structure.
Error handling: None.

Testing: Several test cases will be created to test when the data structure is empty, only has one element, is full, is already sorted, is sorted in reverse order, when all elements have the same value and when the elements have random values.

Mergesort

Input: None.
Output: None.
Dependencies: Array must have been implemented.
Implemented on: The function shall work with Array.
Definition: Will create an animation of how the data structure is sorted by using the mergesort algorithm. When the animation has ended, the data structure shall have been updated to represent the sorted data structure.
Error handling: None.
Testing: Several test cases will be created to test when the data structure is empty, only has one element, is full, is already sorted, is sorted in reverse order, when all elements have the same value and when the elements have random values.

Quicksort

Input: None.
Output: None
Dependencies: Array must have been implemented.
Implemented on: The function shall work with Array.
Definition: Will create an animation of how the data structure is sorted by using the quicksort algorithm. When the animation has ended, the data structure shall have been updated to represent the sorted data structure.
Error handling: None.
Testing: Several test cases will be created to test when the data structure is empty, only has one element, is full, is already sorted, is sorted in reverse order, when all elements have the same value and when the elements have random values.

Animation control

Play

Input: A paused animation.
Output: The animation starts running.
Dependencies: The user has loaded a data structure and ran a function.
Implemented on: All functions except for “Set Random Values”.
Definition: Plays all the animation steps of the last run function from the current state to finish.
Error handling: None.
Testing: Will be tested by pressing play when an animation is paused, has been stopped and when the animation has not begun running.

Pause

Input: No input
Output: The user sees an animation
Dependencies: The user has loaded a data structure and ran a function. The animation control must also be in play state.

Implemented on: All functions except for “Set Random Values”.
Definition: Pause the animation of the playing function
Error handling: None.
Testing: Will be tested by pressing pause when an animation is paused, is running and has been stopped.

Step Forward

Input: No input
Output: The user sees an animation
Dependencies: The user has loaded a data structure and ran a function. The animation control must also be in paused state.
Implemented on: All functions except for “Set Random Values”.
Definition: Steps one step forward in a paused animation, animating the step.
Error handling: None.
Testing: Will be tested by pressing step forward when an animation is paused, is running and has been stopped.

Step Backwards

Input: No input
Output: The user sees an animation
Dependencies: The user has loaded a data structure and ran a function. The animation control must also be in paused state.
Implemented on: All functions except for “Set Random Values”.
Definition: Steps one step backward in a paused animation.
Error handling: None.
Testing: Will be tested by pressing step backwards when an animation is paused, is running and has been stopped.

Stop

Input: No input
Output: The user sees an animation
Dependencies: The user has loaded a data structure and ran a function. The animation control must also be in paused state.
Implemented on: All functions except for “Set Random Values”.
Definition: Goes to the first step of the animation and pauses it.
Error handling: None.
Testing: Will be tested by pressing stop when an animation is paused, is running and has been stopped.

Non-functional requirements

- **System –** The computer must be a PC with at least a processor with 800 MHz and 512 MB ram. The program will be used on a 32-bit PC with the operating system “Windows XP”. The user must have SUN Java 1.5 installed on his computer. Everything you do in the program shall have a response-time less than 1 second.

- **Scalability –** Our program will work in the resolution 1024x768 pixels even though it is scalable we do not guarantee everything will fit in other resolutions.
- **Documentation –** All functions and data structures must be well documented so that users can find out what they are looking at and what happens.
- **Space and delivery –** Since the program will be distributed through the internet, the total size of the program shall not be larger than 20 Megabyte.
- **Standards –** Data structures shall be visualized according to the standards used in books. For example, tree nodes shall be drawn with the father centered over the children nodes. The same thing for algorithms such as quicksort.

Use cases

Load a data structure

Primary actor: Student
Level: Summary
Stakeholder and interests:

1. Student – To get a better understanding of the algorithm without having to read a lot in his algorithm book
2. Teacher – To as fast as possible get the student to understand an algorithm so that he can continue with other parts of his course.

Minimal guarantees: The student is shown a list of available data structures.
Success guarantees: The student loads a data structure.
Precondition: The computer is turned on; the program is properly installed and started.
Main Success scenario:

- The student chooses load from a list of options.
- Here he chooses the data structure “Linked list”.
- He specifies the number of random posts that the list will be filled with.

The use case ends.
Extensions:

2. a) The data structure “Linked List” is not found.
   He cancels the loading procedure or loads another data structure.
3. a) The amount entered is not a valid number.
   He won’t be able to load the structure.

Use a function with parameters

Primary actor: Student
Level: Summary
Stakeholder and interests:

1. Student – To get a better understanding of the algorithm without having to read a lot in his algorithm book
2. Teacher – To as fast as possible get the student to understand an algorithm so that he can continue with other parts of his course.

Minimal guarantees: The data structure does not change if he cancels the function.
Success guarantees: The student is shown an animation of the function with his chosen parameters.
Precondition: He has loaded a data structure.
Main Success scenario:

1. The student selects the function named Insert.
2. He specifies the input parameters.
3. The student gets shown an animation of the function with his chosen parameters.

The use case ends.
Extensions:

1. a) The function Insert does not exist.
   He closes the program or runs another function.
2. a) He inputs invalid arguments
   The program does not run unless correct input arguments are entered.
   b) The input is valid but not intended.
   The program runs as usual

Use a function without parameters

Primary actor: Student
Level: Summary
Stakeholder and interests:

1. Student – To get a better understanding of the algorithm without having to read a lot in his algorithm book
2. Teacher – To as fast as possible get the student to understand an algorithm so that he can continue with other parts of his course.

Minimal guarantees: The data structure does not change if he cancels the function.
Success guarantees: The student is shown an animation of the function.
Precondition: He has loaded a data structure.
Main Success scenario:

1. The student selects the function named quicksort.
2. The student gets shown an animation of the function with his chosen parameters.

The use case ends.
Extensions:

1. a) The function quicksort does not exist.
   He closes the program or runs another function.

Animation control

Primary actor: Student
Level: Summary
Stakeholder and interests:

1. Student – To get a better understanding of the algorithm without having to read a lot in his algorithm book
2. Teacher – To as fast as possible get the student to understand an algorithm so that he can continue with other parts of his course.

Minimal guarantees: The student sees an animation of the desired algorithm.
Success guarantees: The student understands the algorithm that well that he can explain it to others.
Precondition: The student has loaded a data structure, filled it with some values and ran a function.
Main Success scenario:

1. The student presses play.
2. He watches an animation of how the list is being sorted by the algorithm.
3. He chooses to pause the animation.
4. He steps forward and watches as the animation makes one step at a time in the algorithm.
5. He steps backward a couple of step and watches as the animation goes a couple of steps backward in the algorithm.
6. He uses stop and the animation steps directly to the first step of the animation.

The use case ends.
Extensions:

4. a) The animation is at the end.
   He runs the animation from the beginning or steps backwards.
5. a) The animation is at the beginning.
   He presses play or step to go forward again.
6. a) The animation is already at the beginning.
   Nothing happens.

System architecture

TICS' system architecture will be divided into two parts, the Graphical User Interface (GUI) and the data structure module. The data structure module will encapsulate all the data structures. This module handles the entire GUI not handled by the animation window or the animation control. This includes menus, options, etcetera.

Animation Window

This is the part of the GUI that draws the animation. It will have access the information from an animation structure and visualize it.

Animation Control

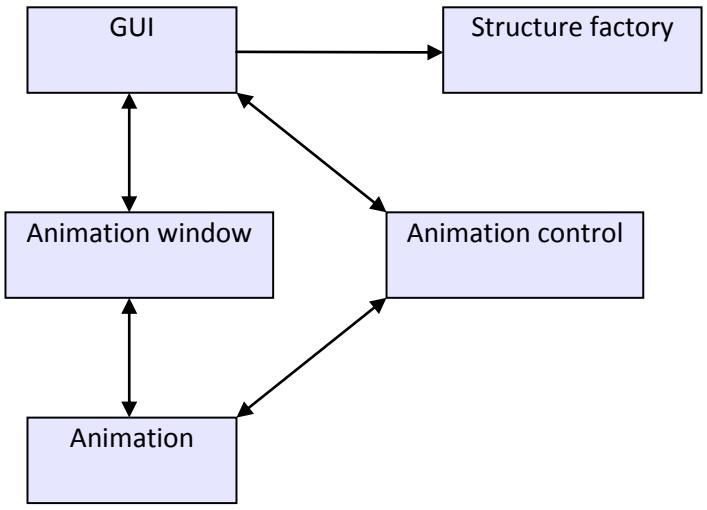
This is the part of the GUI that lets the user control the current animation, such as start, pause, etc.

Animation Structure

This is the data structure that contains an animation. It should keep track of the whether the animation is playing, paused, etc.

Data Structures and Functions

This is all the data structures and algorithms that are visualized in this program. This module will keep track of the available data structures and the functions that they provide.



System requirements specification

Create a data structure

Input: All the parameters needed to create a data structure, including which data structure should be created. This input is provided by the user through the GUI subsystem.
Output: A data structure which is sent to the GUI.
Action: Parse all the parameters and create the appropriate data structure.
Pre-condition: No animation may be running.
Post-condition: A new data structure is created and visualized in the animation window. Any previously created data structure and/or animation data will be removed.

Run a function

Input: All the parameters needed for this function, including which data structure it is operating on and which function to run. This input comes from the user through the GUI system.
Output: An animation structure that is sent to the animation window where it will be rendered.
Action: The function is executed and an animation structure visualizing the function process is created.
Pre-condition: A data structure must be loaded.
Post-condition: The data structure is updated by the function and an animation structure has been created.

Animation control

Input: An animation control is activated by the user.
Output: The state of the current animation in the animation window is changed.
Action: The animation state is changed depending on which animation control is activated.
Pre-condition: A data structure is loaded and an animation is shown in the animation window.
Post-condition: The animation state has changed.

System evolution

The use of algorithms and data structures will probably not decrease. On the contrary, the amount and use of A&D in this computer centralized society will increase as time goes by. Therefore the need of knowing how A&D works will increase in the future.

Algorithms and data structures evolve as well as hardware. This means that in the future, new A&D will be invented. However; the most basic A&D has not been changed for a while and will most probably not improve in the near future, making this system usable for a long time ahead.

Hardware improvements do not have much effect on our system since it is not performance dependent. It is designed to work on an 800MHz laptop and will not benefit on being run on faster hardware.

Appendices

Algorithms

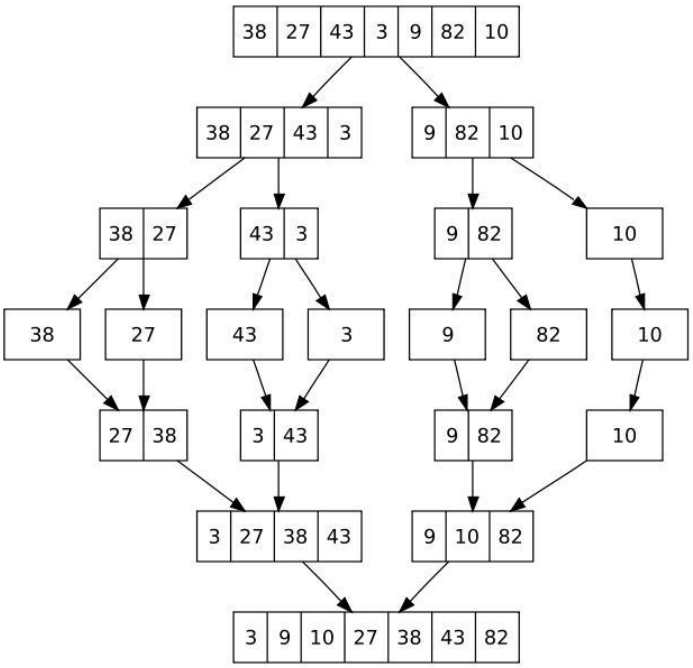
Quicksort

Quicksort is one of the most effective algorithms today. In worst case scenario, it will sort data in $O(n^2)$ but usually it will sort in O(n log(n)). Quicksort works by choosing a pivot element and moves all the elements that are smaller than the pivot to one side of it and the rest to the other side. After this is done, it will choose a new pivot element for each side of the previous pivot element moving the elements like before. This will be done recursive until it is sorted.

The pivot element can be chosen in different ways, one way to choose it is just by take a random element, but a more common way is to take the median element of the first, the middle and the last element in the array.

Mergesort

Mergesort is a sorting algorithm that sorts data in O(n log(n)). An example of how it works can be seen in the picture below taken from Wikipedia.



Source: http://en.wikipedia.org/wiki/Merge_sort

Insertion sort

Insertion sort is a very simple algorithm which takes each element in the list and inserts it in the right place in a new list (or the same list). This algorithm is not very effective on large data, but when it is small lists or the data is already partly sorted, then this algorithm is quite effective and easy to implement.

Table of contents

Index