IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 5, Ver. V (Sep. - Oct. 2016), PP 75-85
www.iosrjournals.org

Fundamental Research on Sorting Algorithms for Numeric Arrays in Animated Examples

Lupu Costică [1]
[1] *"Vasile Alecsandri" University of Bacău, 157 Mărăşeşti Street, Bacău, 600115, România, Pre- and In-Service Teacher Training Department*

Abstract: *Data structures and manipulation algorithms are found in every branch of computer science. Although implementations are virtually everywhere, few students fully understand their execution on the proposed data structures.*
The importance of understanding these algorithms is not related only to their use, but also to the parallel development of other, more complex algorithms, depending on customer requirements.
When working with a database, which relies on a lot of information that should be sorted in one way or another, the most accurate algorithm should be chosen to improve application performance.
From our point of view, the biggest problem related to understanding these algorithms is the lack of abstract, formal and conceptual thinking. To overcome this obstacle, there is proposed the method of visual learning, which has several strengths such as: - better and faster understanding of the processes and their functionalities; - formation and development of practical skills; - better anticipation of solving problems; - transmission of large amounts of factual information; - facilitation of the formation of mental representations; - development of brain power to store and recall images.
To this effect, there was created an application that helps students better understand the most important sorting algorithms used in programming on a daily basis.
Keywords: *animating arrays, application development, animating the algorithms, fundamental research, JavaScript, sort algorithm, system of animation.*

I. Introduction

Technological advances in computer science have led to the development of computing power and memory complexity, in all spheres of human activity. The computer has become an indispensable tool for the present generation. Using the Internet and accessing information resources, broadening the range of applications contribute to the computer imposing itself as the most important tool in many fields. Consequently, the demand to create programs in all fields of human activity has also increased.

The programming activity has two main components: a technology component and a scientific component. The scientific part includes the data structures and the algorithms required for the operations involved in using these data structures and the independence of these structures from programming languages. The information processed by a program is stored using data structures that group data in an effective form.

Programs process the information organized into calculation structures. How we represent data structures determines the clarity, concision, running speed and storage capacity of the program. Before implementing a program in a specific language it is necessary to specify an algorithm for solving by: problem analysis, understanding the problem, elaborating a solving plan, performing the calculations, finding the solution. There is, however, no algorithm that can solve every problem, but there are a number of techniques to develop algorithms, called programming techniques.

II. Theoretical Aspects

An algorithm is composed of a finite set of steps, each requiring one or several operations. To be implementable on computer, these operations must first be defined, namely stating clearly what needs to be executed. Secondly, the operations must be effective, which means that - in principle at least - a person equipped with pencil and paper should be able to perform any step in a finite time interval. We consider that an algorithm should finish after a finite number of operations in a reasonable amount of time.The program is the expression of an algorithm in a programming language. Before learning general concepts, one should have already gained some practical experience in the field. Assuming we already know the programs in a high level language, sometimes there still occur difficulties in formulating the solution to a problem or in selecting the best algorithms. The study of algorithms includes the following steps: 1. Elaboration of the algorithm; 2. Presentation of the algorithm (in a program, the presentation must be clear and concise); 3. Validation of the algorithm (the algorithm will be validated to ensure that it is correct, independent of the language in which it will subsequently be written); 4. Analysis of algorithms (refers to the computation time and the memory

required); 5. Testing the programs (debugging and profiling). As stated by E. W. Dijkstra, however, by troubleshooting we can highlight the presence of errors but not also their absence. Running is the correct execution of the program on various dates by determining the computation time and the memory required.

We should ideally find, for a given problem, several algorithms, and then choose the optimal one. The efficiency of an algorithm can be expressed by: - practical analysis of how the algorithm behaves in different cases; - theoretical analysis of the algorithm, by determining the resources (time, memory etc.). The size of a case x, marked by $|x|$, is the number of bits needed to represent this case. The advantage is that the theoretical analysis does not depend on the computer, or the programming language or the skill of the programmer. Theoretical analysis allows us to study the effectiveness of the algorithm for cases of all sizes.

We can also analyse an algorithm by a hybrid method that theoretically determines the efficiency of the algorithm, and the numerical values of the parameters are then determined practically, concretely. The theoretical efficiency of an algorithm is given by the principle of invariance that, assuming we have two implementations that require the times $t_1(n)$, respectively $t_2(n)$ to solve a problem of size n, then there is always a positive constant c, so that $t_1(n) \leq c \cdot t_2(n)$ for any n that is large enough. The efficiency is expressed within the limits of a multiplicative constant. We say that an algorithm requires time in the order of t, for a given function t, if there is a positive constant c and an implementation of the algorithm capable of solving each case of the problem within a period not exceeding $c \cdot t(n)$.If an algorithm requires a value of the order of n, the value is said to be linear. Analogously, an algorithm is quadratic, cubic, polynomial, or exponential if it requires values of the order n^2, n^3, n^k, respectively c^n, where $k \in N^*$, and $c \in R_+^*$.

III. Sorting Algorithms

III.1. The bubble sort algorithm

The bubble sort algorithm implies passing through a series of elements and comparing each element with its predecessor or successor, depending on the direction. During the first passing through, the maximum value element will have the final position for head-to-tail stepping through or, respectively, the minimum value on the first position for head-to-tail stepping through. On the second passing through, the item coming next in value after the maximum element, respectively the minimum element, will have the penultimate / second position. Thus, the series is passed through several times until no further swapping of the elements can be applied. In this case, the elements in our list are sorted.

The bubble sort algorithm will sort the elements of the sequence s [1], s [1], s [3], ..., s [n] so that after sorting they will occur in the order $s[1] \leq s[2] \leq s[3] \leq \ldots \leq s[n]$. The algorithm consists in passing through the list of elements several times until it becomes ordered. Each step consists of comparing two consecutive elements s[i], s[i+1]. If s[i] > s [i+1], then these elements are swapped. With each comparison of the elements, the positions rise by one unit so that the item with the highest position reaches the limit of passing through the list. With each passing through, this limit will decrease by one unit. Finally, when the limit will be equal to 1, our algorithm will be next to completion and the list will be ordered.JavaScript example: for(i = n; i > 0, i--){ for (p = 0; p < i; p++){ if(s[p] > s[p+1]){ temp = s[p+1]; s[p+1] = s[p]; s[p] = temp;}} }. In the example above, there was also used a temp variable that temporarily holds the value of the successor element, to change the values of the elements if necessary.

III.2. The insertion sort algorithm

The basic idea of sorting by insertion is the insertion of a specific element in the already sorted list of its predecessors. For example, given an unsorted list $s[1], s[2], s[3], \ldots, s[n]$, ..., s [n] that we want to order in ascending order. The first element is maintained as such, being considered an already ordered substring, then move on to the next item to be compared with each element of the substring, and if there is found an item higher than it and its predecessor is smaller than the selected item, then the selected item is inserted between these two 2 elements. This procedure will be repeated until the position of the selected item will be the final position of our list.

JavaScript example: for(i = 2; i <= n; i++){ p=i; while(s[p] < s[p-1] && p>1){ temp = s[p]; s[p] = v[p-1]; s[p-1] = temp; p--;}}.The algorithm requires a time in n^2. However, for the best case, when the initial list is ordered in ascending order, time is in Q(n), and $Q(n^2)$ when the list is ordered in descending order.

III.3. The selection sort algorithm

Sorting by selecting the minimum (or maximum) is the algorithm that selects an item and places it on its final position directly in the list to be sorted. For example, we have an unsorted list s[1], s[2], s[3], …, s[n] and we want to sort it in ascending order. Starting from the first element, the algorithm searches for the minimum value in the list. Once we have found it, there will be swapped the element of the position from which we started s[1] and the minimum element s[k]. Then, it will continue from the second element, until reaching the last element of the array.

JavaScript example:
for (i=0; i < n-1; i++){pos_min = i; for (p=i-1; p < n; p++){ if(s[p] < s[pos_min]){ pos_min = p; }} if(pos_min != i){ temp = s[i]; s[i] = v[pos_min]; s[pos_min] = temp; } } The complexity order of this algorithm is $Q(n^2)$. For this algorithm we can not say that we have a favourable or unfavourable case since the number of performed steps is $n(n-1)/2$ regardless of the positioning of the elements.

III.4. The merge sort algorithm

The merge sort is a sorting algorithm invented by John von Neumann in 1945. The basis of this algorithm is the principle of "Divide et impera". The algorithm execution divides the list in two substrings and sorting them. This is done recursively, so that the substrings eventually contain no more than two items.

This algorithm performs the following steps: 1. If the list length is 0 or 1, then it is already sorted; 2. Divide the unsorted lists into two approximately equal sublists; 3. Sort each recursive sublist by reapplying the merge sort algorithm; 4. Interclass the two lists to achieve the initial list, which is already sorted.
Implementation in JavaScript:
function merge(left, right){ var result = [], il = 0, ir = 0; while(il < left.length && ir < right.length){ if (left[il] < right[ir];){result.push(left[il++]);}else {result.push(right[ir++]);}} result = result. Conca t(left.slice(il)). concat(right.slice(ir)); return result; } function mergeSort(items){ if (items.length < 2) { return items; } var middle = Math.floor(items.length / 2), left = items.slice(0, middle), right = items. Slice (middle); return merge(mergeSort(left), mergeSort(right));}The merge sort algorithm perfectly illustrates the principle of "Divide et Impera". Its complexity order is Q(n^{logn}), therefore the given algorithm is faster than the ones described above.

III.5. The quick sort algorithm

The algorithm is based on the principle of "Divide et impera". Unlike the merge sort, the non-recursive part of the algorithm is focused on building them, but not their combination.
While running the program, the algorithm performs the following steps: 1. Choose any element from the list as pivot; 2. Reorder the list so that all the elements smaller than the pivot may be on the left side and the higher elements on the right side. 3. Sort recursively with the same algorithm the sublist on the right side of the pivot and the other one on the left. An array of the size 0 or 1 is considered to be already sorted.

JavaScript example: function qsort(arr) { if (arr.length == 0) return []; var left = [], right = [], pivot = arr[0]; for(i = 1; i < arr.length; i++) if(arr[i] < pivot)){ left.push(arr[i]); }else{ right. Push (arr[i]);} quickRecusriveFor(i);} result = qsort(left).concat(pivot, qsort(right)); return result; } The complexity order of this algorithm is $Q(n \cdot log\ n)$. Recursion quick sort multiplies fewer times compared with merge sort. If for unfavourable cases a slower performance is acceptable, then this will be chosen because for average and favourable cases this is faster than merge sort. (10)

IV. Technologies Used For The Implementation Of A System Of Animation For Sorting Algorithms

To implement a system of animation for sorting algorithms (installation or compatibility with the operating system) the easiest way is to use web technologies. The web programming languages used in developing my application are: HTML, LESS, JavaScript, jQuery and PHP.
For increased security and as simple a content management as possible it is recommended to use a CMS (a collection of procedures implemented in a web application). We may choose Drupal CMS that is the safest CMS, which is quite easy to use, being reliable, flexible and having efficient updates in terms of time.
We shall further briefly present each programming language to understand how our CMS works.
HTML stands for Hyper Text Markup Language and is a language for creating a document so that it can be published on the World Wide Web and viewed using a browser (Internet Explorer, Netscape Navigator, Mozilla Firefox, Opera, etc.).

The hypertext should be interpreted as a text that indicates a link to another web information, usually another web document that is identified by underlining or colour, to distinguish it from plain text. HTML is not a programming language, and that because HTML lacks the main attribute of any particular programming language, namely commands. One cannot create in HTML a sequence of actions, but there can only be described how the browser should display the document on the screen. Thus, HTML is just a description language containing elements that allows one to build web pages. HTML documents are exclusively of the text type (ASCII), therefore they can be created with any text editor. However, there have been developed specialized editors that allow editing in a kind of way of the type of WYSIWYG - What You See Is What You Get.Specialized HTML editors (ex. MS FrontPage or Macromedia DreamWeaver) allows the creation of web documents in a fast, easy, and therefore very effective way, just by pressing a few buttons or by resorting to several predefined functions.

HTML uses web-tags, also called tags, to describe documents, specific to each element described. The tags determine both the structure of the document and its layout. A tag is an identifier that provides the browser with document formatting instructions. To be definable, HTML tags are enclosed in angle brackets (<>). Tags are divided into: - even or pair tags – that imply the existence of a closing tag; - odd or singular tags – that require no closing.

LESS is a CSS pre-processor which extends its language by adding new features such as variables, combinations of style, functions and many other technical features that allows the CSS to be more structured, rational and extensible. We shall further briefly present the CSS in order to understand how LESS works.

CSS (Cascading Style Sheet) is a language that applies presentation (style, layout and, more recently, animation) to content languages, be it about HTML, XHTML, XML or SVG. In other words, if HTML tells the browser how information is structured (through items such as <h_1></h_1>, , <a> etc.), the CSS tells the same browser how to display the information so that it would *look good* to the eye (style, size, spacing, colour). CSS rules allow you to apply properties to HTML page elements. So, when you want to style an element within an HTML page, you identify that element and add properties related to colour, size, font, positioning etc.Usually, the CSS has the form: selector { property 1: value; property 2: value; property 3: value; } The selector identifies the page elements which, in turn, will be affected by the CSS. The selector may be represented by the name of the HTML element (body, div, p, h1, h2), or the attributes "class" or "id", as defined in the HTML page. The accolades contain the pairs "property: value", which are, in turn, separated by a semicolon ";".

There are several ways of entering styles into HTML.CSS inline introduction: - The easiest, but also problematic on the long term is the introduction of the style in the HTML tag using the attribute *style*. This attribute tells the browser to display a paragraph with a background colour ...., bold fonts, colour.... and a border to distinguish it from the other elements of at least 20 pixels.

As noted above, this method (as well as the following one ...) is problematic in the long run because when the number of html pages increases, if you want to make a change of style, you will have to take each html page in turn and make the changes. CSS introduction in the head of the page can be inserted into the element *head* of the HTML page using the <style></style> element.

Thus, the code: <!DOCTYPE html> <html> <head> <title>Stile CSS in a page html</title> <style type="text/css"> p { colour:white; background:blue; padding:5px; } </style> </head> <body> Text white written on blue background </body> </html>

The ultimate goal for Internet pages is to completely separate the information from the design. In other words, the information should be transmitted, even if the layout of the page does not make this information pleasing to the eye. The introduction of CSS in a page through an external file is the method closest to this philosophy.It is preferable that the style of pages are defined in an external file. It will have the extension .css extension and will be a text file (so any text editor can be used to open it). The file with the .css extension will contain only the CSS rules. For example, a file style.css will have : / * this is a comment * / p {colour: white; background: blue; padding: 5px; }

Including the style.css file is preferably done via the element <link> with the value of the attribute "href" the location of the file style.css. It is recommended to include the file through this tag in the element head: <DOCTYPE html> <html> <head> <title>Style CSS in a page html</title> <link rel="stylesheet" href="style.css" type="text/css"> </head> <body> ... </body> </html> If the file is found where you tell the HTML page that it is, the display rules will be observed by this page. **LESS** is used for CSS styling but with much more possibilities as compared to CSS. In fact, we could call it an extension, update or evolution of the CSS syntax. For an HTML file to be styled with LESS, we need a LESS engine that may be on JavaScript or PHP and introduce our style.less file in the HTML page in the same way as the style.css is introduced. These engines give us the possibility to compile the code so as to obtain the ready structured CSS. The variables in LESS are used to memorize properties that could be used repeatedly in some declarations or operations: - Declaring variables is done by: @<variable>: <property>; - Using it as declared, namely: @ <variable>; As in any programming language, we may introduce direct operations when declaring them.

Combinations of styles, best known as Mixins in LESS, are used to memorize combinations of styles in a particular name for its repeated use. The declaration is made: .<name> {<properties>}. It is used in the same way: bordered{ border-top: dotted 1px black; border-bottom: solid 2px black; } #menu a { colour: #111; bordered}. .Nesting rules allow writing the code in as brief and simple manner as possible, without much redundancy. They rely on the idea of writing elements as displayed in HTML. In CSS, this writing will be understood as: @media screen { .screen-colour { colour: green;} }; @media screen and (min-width: 768px) { .screen-colour { colour: red; } }; @media tv { .screen-colour { colour: black; }}.The LESS operations used are: +, -, *, /, and importing is done thus: @import "library"; // library.less @import "typo.css".

V. Javascript – Programming Language For Scripts

JavaScript is embedded directly in HTML pages with the following advantages: - JavaScript is an interpreted language (i.e., the script is executed directly, without preliminary compilation); - JavaScript gives HTML designers a programming tool; - JavaScript is a language with a very simple syntax and almost anyone can insert small sequences of code into HTML pages; JavaScript can dynamically insert text into an HTML page - A JavaScript statement like this: document.write ("<h1>" + name + "</h1>") can write a variable text into the HTML page; JavaScript can react to events - A JavaScript code can be designed to execute when something happens, such as when the page is fully charged or the user actuates an HTML element; - JavaScript can read and write HTML elements; - JavaScript can read and modify the content of HTML elements; - JavaScript can be used to validate data; - A JavaScript code can be used to validate data before sending them to the server. In this way the server does not operate further processing; - JavaScript can be used to detect the user's browser; - A JavaScript code can detect the type of browser and load a page specifically designed for that browser; - JavaScript can be used to create cookies - A JavaScript code can be used to store and retrieve information on the computer of the HTML page visitor; - A JavaScript code can be inserted into the head of the HTML file using the tag: <script type="text/javascript"> //cod < /script> or as file with the extension .js: <script type="text/javascript" src="nume_fisier.js">< /script>.

jQuery is the most popular JavaScript library functions that allows manipulation of the HTML code much faster and easier compared to the JavaScript functions that would have to be written manually. Its functions can be found on jQuery.com and can be used by anyone because they are licensed for free.

PHP is one of the most interesting technologies for websites and web programming, combining complex features with simplicity of use. PHP has become a leading tool for Web applications development. Unlike other tools for Web applications, such as Perl, PHP is a programming language easy for beginners, even for those who have never conducted any programming.

Unlike scripting languages such as JavaScript, PHP runs on the Web server, not on the Web navigator, therefore PHP can gain access to files, databases and other resources inaccessible to the JavaScript program. These constitute the richest sources of dynamic content that attract visitors. In addition, we can use the PHP language to generate the JavaScript code.The PHP code sequences may be included in an HTML file, then the Web server will identify this sequence of code, process it and finally generate the HTML code and finally replace the sequence of PHP code with the HTML code.

SQL (Structured Query Language) is currently one of the strongest structured query language for interrogating relational databases. It is a non-procedural and declarative language, because the user describes the data he wants to obtain, without having to set up ways to get to that data. It cannot be considered a programming or system language, but it is rather part of the language of applications, being oriented on sets. Very frequently, it is used in managing client/server databases, the client application being the one that generates SQL statements.

There are three basic methods on the implementation of the SQL language: 1. Direct Invocation is the introduction of instructions directly from the prompter; 2. Module Language uses procedures called from the application programs; 3. Embedded SQL contains instructions encapsulated in the program code.

SQL statements can be grouped into: - data definition instructions that allow describing the structure of DB; - instructions for handling the data: add, delete, modify records; - instructions for selecting the data that allow consultation of the DB; - transaction processing instructions; - instructions for cursor control; - instructions for data access control. The SQL language standardized by ISO does not use the formal terms of relation, attribute, tuple, but those of table, column, row.

Drupal is a modular open source content management system, a development framework for web applications and blogging engine. Drupal is one of the best CMS (Content Management Systems). After 2000, Drupal has gained in popularity thanks to the flexibility, adaptability, ease of management and operation, as well as a very active online community.Drupal is written in PHP, but installation, development and maintenance of a Drupal website does not usually require knowledge of PHP programming. Drupal works on a platform with a variety of operating systems such as Unix, Linux, BSD, Solaris, Windows, or Mac OS X. In addition to PHP, Drupal also needs a web server in order to operate, such as Apache or IIS, and a database engine such as MySQL.

VI. Aplication Development On Numerical Array Sorting Algorithms In Animated Examples

The application development implies completion of several steps: - Installing the Drupal; - Drupal setting; - Creating the Custom module for inserting and updating the array; - Writing the sorting algorithms in JavaScript and introducing them in the module; - Creating the Custom theme; - Styling the system; - Animating the algorithms.To **install** the Drupal we need its files that can be downloaded from drupal.org, an Apache server and MySQL engine. Installation takes place by unzipping files, locating them on the server, creating a database and declaring it in the file settings.php with all the access data. The given file can be found in

sites/default/default.settings.php, to which a copy is made, and after everything has been declared, it is renamed after the file specified above, namely settings.php. In the browser, upon opening the site, we choose the type of standard installation and wait for the installation of all the modules necessary to the core, and then we fill in all the required fields for the global settings of the site (email, global user, time zone, etc.). Once the installation is successful, there will be created pages for each algorithm separately. This is done by accessing the link in the content management menu then → add new content → basic page. We also take into account the menu that will allow us to navigate every algorithm separately when creating a page, and we will also create a link in our "main" menu, for the options below. To **set** and develop a module in Drupal it is necessary to create, from the start, the file <numele_modulului>.info. The file array.info contains the following lines: name = array; description = "Provides a form to insert array"; package = Sorting; core = 7.x; version = "7.x-1.0", where: name - the name of our module; description - the module description; package - the group of modules to which it belongs; core - the Drupal version for which this module works; version – the version of the module.

Once we save this file, we go to the administration menu modules, where we will look for the module used in the list of modules and we shall activate it. Basically, if our module is enabled, any PHP function properly inserted will have to be run by our PHP engine.There follows the creation of a file <filename>, modules where we will insert all the proposed functions which will enable us to add or update the array in a form with a text field. Next, there will be created two functions that will allow the construction of a form. The first will create the structure of the form, the second will save the form data after submit:

- function _array_form($form_state) { $form['array'] = array('#type' => 'textfield', '#default_value' => variable_get('array_content', ""),); $form['submit'] = array('#type' => 'submit', '#value' => 'Update',); return $form; } function_array_form_submit($form, &$form_state){variable_set('array_content', $form_ state ['input']['array']); } We see that in both functions we have the function variable_get function () and variable_set (). These functions are responsible for a database table called variable, which stores the global settings of the site, such as module, theme and structure settings. We have introduced the row in this table considering that it will be updated constantly, the action seeing it as a global setting.

There follows the **writing** of the sorting algorithms in JavaScript by creating the block that will display the array by creating a function that will store the array in a variable in the form of a well-structured HTML:

```
function _HTML_block_content() {$html_array = explode(",", variable_get('array_content', "")); $output = '<section class="array">'; $index = 0; foreach ($html_array as $value){ $output = $output . '<div index="' . $index . '" class="element">' . $value . '</div>'; $index++; } $output = $output . '</section>'; return $output; }.
```

The form and the array must be visible when we open the page, so we will resort to a Drupal API function that allows their display. In the function below, some displays of scripts are conditioned, each page having a corresponding script displayed in the folder js from the presented module. After displaying these blocks, which will be inserted into the theme areas and restricted from setting for visibility, the form will only appear on the front page and the block with the array will be visible on the other pages, except on the home page.

```
/**
 * Implements hook_block_view().
 */ function array_block_view($delta = '') { $block = array(); $path = drupal_ lookup_path('alias',current_path()); if($path=="bubble"){ $jspath = drupal_get_path('module', 'array') . '/js/bubble.js';}elseif($path=="insert"){ $jspath = drupal_get_path('module', 'array') . '/js/insert.js';} elseif($path=="merge"){ $jspath = drupal_get_path('module', 'array') . '/js/merge.js';} elseif($path=="quick") { $jspath = drupal_get_path('module', 'array') . '/js/quick.js';} elseif($path=="select"){ $jspath = drupal_get_path('module', 'array') . '/js/select.js';}switch ($delta) { case 'array_form': $block['subject'] = ''; $block['content'] = drupal_get_form('_array_form'); break; case 'HTML_array': $block['subject'] = ''; $block['content'] = array('#markup' => HTML_block_content(), '#attached' => array( 'js' => array( $jspath => array('weight' => 1000), ), ), ); break; }
 return $block; }
```

The Custom **theme** is a Bootstrap sub-theme. For its development, there was copied the folder bootstrap_subtheme from the bootstrap theme and, like with the modules, there was defined the file .info with the name of the theme, after which the information was entered inside it. Here, there were declared 4 regions that can then be used in the page.tpl.php template to display pages: regions[header] = Header regions[content] = Content regions[middle] = Middle regions[footer] = Footer. In the Middle region there was inserted the form and block with the proposed array and there were displayed the blocks on the appropriate pages. The stylization of the theme was made with LESS. To this effect, there were used the module LESS and the module Libraries, after which there was introduced the LESS engine in the folder /sites/all/libraries. Thus, our LESS engine should operate and recompile the code at every change made in LESS files.

Algorithm animation has been the most complicated process of developing our system. Besides the fact that all the functions *for* and *while* had to be rewritten in recursive functions, for the recursive algorithms

there were also used some unique methods compared to the algorithms with complexity n^2. In general, animation was created by modifying the content, swapping elements, adding and deleting certain classes in real time, operations that were set at a given time interval.For Bubble, the resulted Insert and Quick sort functions look like this:

Bubble:

```
(function ($) {
  $(document).ready(function () {
    //bubble_sort();
    bubble_sort();
  });
  delay = 500;
  function bubble_sort() {
    ( function bubbleSortWhileLoop() {
      swapped = false;
      ( function bubbleSortForLoop( i ) {
        setTimeout( function() {
          var test = parseInt($(".array div:eq("+ parseInt(i-1) +")").text()) > parseInt($(".array div:eq("+ parseInt(i) +")").text());
          $(".array .element").removeClass('red');
          $(".array .element").removeClass('green');
          $(".array div:eq("+ parseInt(i-1) +")").addClass('red');
          $(".array div:eq("+ parseInt(i) +")").addClass('red');
          setTimeout( function() {
            if( test ){
              $(".array div:eq("+ parseInt(i-1) +")").insertAfter($(".array div:eq("+ parseInt(i) +")"));
              $(".array div:eq("+ parseInt(i-1) +")").addClass('green');
              $(".array div:eq("+ parseInt(i) +")").addClass('green');
              swapped = true;
            }
            if ( ++i < parseInt($(".array .element").length) ) bubbleSortForLoop( i );
            else {
              if ( swapped ){bubbleSortWhileLoop();}
              else{
                $(".array .element").addClass('red');
                $(".array .element").addClass('green');
                return;
              };
            }
          }, test ? delay : 0 );
        }, delay );
      } )( 1 );
    } )();
  }
```

Insert:

```
(function ($) {
  $(document).ready(function () {
    insert_sort();
  });
  delay = 500;
  function insert_sort() {
    ( function insertSortForLoop(i) {
      d = i;
      ( function insertSortWhileLoop() {
        setTimeout( function() {
          var test = d > 0 && parseInt($(".array div:eq("+ parseInt(d-1) +")").text()) > parseInt($(".array div:eq("+ d +")").text())
          $(".array .element").removeClass('red');
          $(".array .element").removeClass('green');
          if(d > 0){
            $(".array div:eq("+ parseInt(d-1) +")").addClass('red');
            $(".array div:eq("+ parseInt(d) +")").addClass('red');
          }
          setTimeout( function() {
            if( test ){
              $(".array div:eq("+ parseInt(d-1) +")").insertAfter($(".array div:eq("+ parseInt(d) +")"));
              $(".array div:eq("+ parseInt(d-1) +")").addClass('green');
              $(".array div:eq("+ parseInt(d) +")").addClass('green');
              d = d-1;
              insertSortWhileLoop();
            }else{
              if ( ++i < parseInt($(".array .element").length) ){insertSortForLoop(i);}
              else{
                $(".array .element").addClass('red');
                $(".array .element").addClass('green');
                return;
              };
            }
          }, test ? delay : 0 );
        }, delay );
      } )();
    } )(1);
  }
})(jQuery);
```

Select:

```
delay = 500;
function select_sort() {
  ( function selectSortFirstForLoop(i) {
    pos_min = i;
    ( function selectSortSecondForLoop(j) {

      setTimeout( function() {
        var test = parseInt($(".array div:eq("+ j +")").text()) < parseInt($(".array div:eq("+ pos_min +")").text());
        $(".array div:eq("+ parseInt(pos_min+1) +")").addClass('red');

        setTimeout( function() {
          if( test ){
            pos_min=j;
            $(".array .element").removeClass('green');
            $(".array div:eq("+ pos_min +")").addClass('green');
          }
          $(".array div:eq("+ parseInt(j+1) +")").addClass('red');
          if ( ++j < parseInt($(".array .element").length) ){
            selectSortSecondForLoop(j);
          }else{
            if (pos_min != i){
              temp = $(".array div:eq("+ i +")").text();
              temp_index = $(".array div:eq("+ i +")").attr('index');
              $(".array div:eq("+ i +")").text($(".array div:eq("+ pos_min +")").text());
              $(".array div:eq("+ i +")").attr('index', $(".array div:eq("+ pos_min +")").attr('index'));
              $(".array div:eq("+ pos_min +")").text(temp);
              $(".array div:eq("+ pos_min +")").attr('index', temp_index);
            }
            if ( ++i < parseInt($(".array .element").length) - 1 ){
              selectSortFirstForLoop(i);
            }else
            {
              $(".array .element").addClass('red');
              $(".array .element").addClass('green');
              return;
            };
          }
        }, test ? delay : 0 );
      }, delay );
    } )(i+1);
    $(".array .element").removeClass('red');
    $(".array .element").removeClass('green');
  } )(0);
}
```

For the Merge sort algorithm there was used a multiplier procedure of the execution time so that each step may be seen with the naked eye. The problem was in the execution of the merge function on several arrays at the same time, in which case animation was not possible. But, if at each execution of the function the time is multiplied each time the function was run, then each step could be seen separately. If for the algorithms above the operations took place objectively on HTML elements, for this algorithm the information of each element is stored in an array which is then shared by the merge sort functions, and finally the animation function is introduced for each stage.

Merge:

```
(function ($) {
  /*MERGE*/
  $(document).ready(function (){
    var test = [];
    for (var i = 0; i < parseInt($(".array .element").length); i++) {
      test[i] = {
          index: $(".array div:eq("+ i +")").attr('index'),
          text: $(".array div:eq("+ i +")").text()
        };
    }

    mergeSort(test);
  });
  //delay=500;
  time = 1000;
  middle_time = time/2;
  function merge(left, right){
    var result  = [],
        il      = 0,
        ir      = 0;


    (function recursiveWhile(){
    time = time+1000;
    result = result.concat(left.slice(il)).concat(right.slice(ir));
    recursive_animation(result, time, middle_time);
    return result;
  }

  function mergeSort(items){
    // Terminal case: 0 or 1 item arrays don't need sorting
    if (items.length < 2) {
        return items;
    }

    var middle = Math.floor(items.length / 2),
        left    = items.slice(0, middle),
```

```
    time = time+1000;
    result = result.concat(left.slice(il)).concat(right.slice(ir));
    recursive_animation(result, time, middle_time);
    return result;
  }

  function mergeSort(items){
    // Terminal case: 0 or 1 item arrays don't need sorting
    if (items.length < 2) {
        return items;
    }

    var middle = Math.floor(items.length / 2),
        left    = items.slice(0, middle),
        right   = items.slice(middle);

    return merge(mergeSort(left), mergeSort(right));
  }

  function recursive_animation(result, time, middle_time){
    setTimeout(function (){
      $(".array .element").removeClass('red');
      $(".array .element").removeClass('green');
      for(var i = 0; i < result.length; i++){
        $(".array div[index='"+ parseInt(result[i].index) +"']").addClass('red');
      }
      setTimeout(function (){
        for(var i = 0; i < result.length; i++){
          if(i < result.length-1){
            $(".array div[index='"+ parseInt(result[i+1].index) +"']").insertAfter( $( ".array div[index='"+
          ]
          $(".array div[index='"+ parseInt(result[i+1].index) +"']").addClass('green');
          $(".array div[index='"+ parseInt(result[i].index) +"']").addClass('green');
        }
      }, middle_time);
    }, time);
  };
})(jQuery);
```

Select:

```
(function ($) {
  $(document).ready(function (){
    var test = [];
    for (var i = 0; i < parseInt($(".array .element").length); i++) {
      test[i] = {
          index: $(".array div:eq("+ i +")").attr('index'),
          text: $(".array div:eq("+ i +")").text()
        };
    }
    qsort(test);
  });

  time = 500;
  function qsort(arr) {
    if (arr.length == 0) return [];
    var left = [], right = [], pivot = arr[0];
    ( function quickRecusriveFor(i) {
      test = i < arr.length;
      if(test){
        if(parseInt(arr[i].text) < parseInt(pivot.text)){
          left.push(arr[i++]);
        }else{
          right.push(arr[i++]);
        }
        quickRecusriveFor(i);
      }else{
        return;
      };
    })(1);

    result = qsort(left).concat(pivot, qsort(right));
    animation_array(result);
    return result;
  }

  anim = [];
  function animation_array(result){
    anim.push(result);
    return anim;
  };
```

At quick sort, we had to make an array of arrays in which each step covered by the array was saved. Thus, another function was passed through that array to extract the necessary information for animation.

Quick:

```
function animate_qsort(anim){
  (function recursivefor(i){
    if(i < anim.length){
    setTimeout( function() {
      $(".array .element").removeClass('red');
      $(".array .element").removeClass('green');
      setTimeout( function() {
        for (r=0; r < anim[i].length; r++ ){
          $(".array div[index='"+ parseInt(anim[i][r].index) +"']").addClass('red');
        };
        $(".array div.red:first").addClass('green');
        setTimeout( function() {
          for (r=1; r < anim[i].length; r++ ){
            d=r;
            while(d > 0){
              if(parseInt($(".array div.red:eq("+ parseInt(d-1) +")").text()) > parseInt($(".array div.red:eq("+ d +")").text())){
                $(".array div.red:eq("+ parseInt(d) +")").insertBefore($(".array div.red:eq("+ parseInt(d-1) +")"));
              }else{
                $(".array div.red:eq("+ parseInt(d) +")").insertAfter($(".array div.red:eq("+ parseInt(d-1) +")"));
              }
              d--;
            }
          };
          i++;
          recursivefor(i);
        }, time );
      }, time );
    }, time * 2 );
    }else{
      return;
    }
  })(0)
}

$(document).ready(function(){
  animate_qsort(anim);
});
})(jQuery);
```

```
function animate_qsort(anim){
  (function recursivefor(i){
    if(i < anim.length){
    setTimeout( function() {
      $(".array .element").removeClass('red');
      $(".array .element").removeClass('green');
      setTimeout( function() {
        for (r=0; r < anim[i].length; r++ ){
          $(".array div[index='"+ parseInt(anim[i][r].index) +"']").addClass('red');
        };
        $(".array div.red:first").addClass('green');
        setTimeout( function() {
          for (r=1; r < anim[i].length; r++ ){
            d=r;
            while(d > 0){
              if(parseInt($(".array div.red:eq("+ parseInt(d-1) +")").text()) > parseInt($(".array div.red:eq("+ d +")").text())){
                $(".array div.red:eq("+ parseInt(d) +")").insertBefore($(".array div.red:eq("+ parseInt(d-1) +")"));
              }else{
                $(".array div.red:eq("+ parseInt(d) +")").insertAfter($(".array div.red:eq("+ parseInt(d-1) +")"));
              }
              d--;
            }
          };
          i++;
          recursivefor(i);
        }, time );
      }, time );
    }, time * 2 );
    }else{
      return;
    }
  })(0)
}

$(document).ready(function(){
  animate_qsort(anim);
});
})(jQuery);
```

VII. Conclusion

Animating arrays in real time is a rather complicated process, especially in terms of recursive functions. For the algorithms with complexity n^2, the animation process can be created simply by transforming the functions "while" and "for" in recursive functions, and by having such a structure, we can set a timer for our functions to run at the set time interval. Thus, the algorithms which do not work on the principle of "Divide et impera" can be easily animated.

Regarding the algorithms based on the principle of "Divide et impera", there should be found a solution to monopolize each amendment separately and represent it graphically. The most reasonable solution to such algorithms is, we believe, saving the steps for sorting the array into another array, in order to subsequently manipulate it with another function.

References

[1] D. Petcu, Parallel algorithms, *Printing UVT*, Timisoara, 1994.
[2] R. Andonie, I. Garbacea, *C ++ Algorithms fundamental perspective*, Libris Publishing House, Cluj-Napoca, 1995. .
[3] C. Lupu, The Importance of New Technologies in Learning Mathematics, *Social Media Academy in: Research and Teaching*, Pages: 197-200, 2013, http://apps.webofknowledge.com/full_record.do?.
[4] C. Lupu, The Efficiency of Computers with Maple Software in the Teaching and Learning of Plane Geometry, *Social Media Academy in: Research and Teaching*, Pages: 215-221, 2015. http:// apps. webofknowledge. com/full_ record.do? product= WOS&search_mode=GeneralSearch&qid=3&SID=Z1nSEtoQZRKMmNUqHjk&page=2&doc=19
[5] C. Lupu, The Role of the Computer in Learning Mathematics Through Numerical Methods, *Science Journal of Education*, Volume 4, Issue 2, Pag: 32-38, 2016. http://www.sciencepublishinggroup.com/journal/paperinfo?journalid=197&doi=10.11648/j.sjedu.20160402.13.
[6] Sorting algorithms, staff.cs.upt.ro/~gabia/TP/2008/L13-Sortari.html/.
[7] First Steps in CSS., Http://avenir.ro/primii-pasi-in-css-cum-introduc-css-in-paginile-html/.
[8] Sorting methods, http://profu.info/c-metode-de-sortare-metoda-bulelor-insertiei-selectiei-numararii/ .