|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\index.html |

1 ﻿<!--

2 Dateiname : index.html

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 -->

8

9 <!DOCTYPE html>

10 <head>

11 <meta charset="utf-8">

12 </head>

13 <body>

14 <h2><a href="SortPage01.html">Neue Sortierverfahren Ansicht</a></h2>

15 <h2><a href="SortPage02.html">Alte Sortierverfahren Ansicht</a></h2>

16 <h2><a href="SortPage03.html">Heapsort Baumdarstellung</a></h2>

17 <h2><a href="SortPage04.html">Pracktische Nuntzung der Sortierverfahren</a></h2>

18 <h2><a href="SortPage05.html">Random Sort mit Zufällig entstehenden Bildern</a></h2>

19 </body>

20 </html>

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\SortPage01.html |

1 ﻿<!--

2 Dateiname : SortPage01.html

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 -->

8 <!DOCTYPE html>

9 <head>

10 <meta charset="utf-8">

11 <title>Neue Sortierverfahren Ansicht</title>

12 <link rel="stylesheet" href="css/sortStyle.css">

13 <script src="js/sort.js"></script>

14 </head>

15 <body>

16 <h1>Einstellungen</h1>

17 <div id="**settings**">

18 <div>

19 <div class="**dropdown** **orange**">

20 <select id="**arrayTyp**" class="**dropdown-select** **orange**">

21 <option value="0">zufällig</option>

22 <option value="1">aufsteigen</option>

23 <option value="2">absteigen</option>

24 <option value="3">zufällig 1-100</option>

25 </select>

26 </div>

27 <div class="**dropdown** **orange**" style="float**:** **right**">

28 <select id="**maxSteps**" class="**dropdown-select** **orange**">

29 <option value="254">254</option>

30 <option selected="" value="128">128</option>

31 <option value="64">64</option>

32 <option value="32">32</option>

33 <option value="16">16</option>

34 <option value="8">8</option>

35 <option value="4">4</option>

36

37 </select>

38 </div>

39 </div>

40 <div>

41 <button type="button" class="**button** **orange**" id="**startNewSort**" title="Startet alle Sortierverfahren mit einem neuen Array">Neues Array generieren</button>

42 <button type="button" style="float**:** **right**" class="**button** **orange**" id="**startSort**" title="Startet alle Sortierverfahren mit dem Aktuellen Array">Sorotierung Starten </button>

43 </div>

44 </div>

45 <h1>Sortierverfahren</h1>

46 <div id="**sortDesplayNew**">

47 <section id="**bubblesort**"><h2>Bubblesort</h2></section>

48 <section id="**insertionsort**"><h2>Insertionsort</h2></section>

49 <section id="**selectionsort**"><h2>Selectionsort</h2></section>

50 <section id="**shellsort**"><h2>Shellsort</h2></section>

51 <section id="**quicksort**"><h2>Quicksort</h2></section>

52 <section id="**heapsort**"><h2>Heapsort</h2></section>

53 <section id="**bitonicsorter**"><h2>Bitonic Sort</h2></section>

54 <section id="**oddeventranspositionsort**"><h2>Odd–even Sort</h2></section>

55 <section id="**oddevenmergesorter**"><h2>Odd-even Mergesort</h2></section>

56 <section id="**mergesort**"><h2>Mergesort</h2></section>

57 <section id="**bogosort**"><h2>Bogosort</h2></section>

58 </div>

59 <script>

60 **window.onload** **=** function **()**

61 **{**

62 **addToSortNew(**"#bubblesort"**,** **eSort.bubblesort);**

63 **addToSortNew(**"#selectionsort"**,** **eSort.selectionsort);**

64 **addToSortNew(**"#insertionsort"**,** **eSort.insertionsort);**

65 **addToSortNew(**"#shellsort"**,** **eSort.shellsort);**

66 **addToSortNew(**"#quicksort"**,** **eSort.quicksort);**

67 **addToSortNew(**"#heapsort"**,** **eSort.heapsorter);**

68 **addToSortNew(**"#bitonicsorter"**,** **eSort.bitonicsorter);**

69 **addToSortNew(**"#oddeventranspositionsort"**,** **eSort.oddeventranspositionsort);**

70 **addToSortNew(**"#oddevenmergesorter"**,** **eSort.oddevenmergesorter);**

71 **addToSortNew(**"#mergesort"**,** **eSort.mergesort);**

72 **addToSortNew(**"#bogosort"**,** **eSort.bogosort);**

73

74 **initSortNew(eArrayTyp.Random,** **128);**

75 **document.getElementById(**"startSort"**).onclick** **=** **startSortNew;**

76 **document.getElementById(**"startNewSort"**).onclick** **=** function **()** **{**

77 var **lArrayTyp** **=** **document.getElementById(**"arrayTyp"**).value** **\*** **1;**

78 var **lMaxSteps** **=** **document.getElementById(**"maxSteps"**).value** **\*** **1;**

79 **initSortNew(lArrayTyp,** **lMaxSteps);**

80 **};**

81 **};**

82 </script>

83

84 </body>

85 </html>

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\SortPage02.html |

1 ﻿<!--

2 Dateiname : SortPage02.html

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 -->

8 <!DOCTYPE html>

9 <head>

10 <meta charset="utf-8">

11 <title>Alte Sortierverfahren Ansicht</title>

12 <link rel="stylesheet" href="css/sortStyle.css">

13 <script src="js/sort.js"></script>

14 </head>

15 <body>

16 <h1>Einstellungen</h1>

17 <div id="**settings**">

18 <div>

19 <div class="**dropdown** **orange**">

20 <select id="**arrayTyp**" class="**dropdown-select** **orange**">

21 <option value="0">zufällig</option>

22 <option value="1">aufsteigen</option>

23 <option value="2">absteigen</option>

24 <option value="3">zufällig 1-100</option>

25 </select>

26 </div>

27 <div class="**dropdown** **orange**" style="float**:** **right**">

28 <select id="**maxSteps**" class="**dropdown-select** **orange**">

29 <option value="254">254</option>

30 <option selected="" value="128">128</option>

31 <option value="64">64</option>

32 <option value="32">32</option>

33 <option value="16">16</option>

34 <option value="8">8</option>

35 <option value="4">4</option>

36

37 </select>

38 </div>

39 </div>

40 <div>

41 <button type="button" class="**button** **orange**" id="**startNewSort**" title="Startet alle Sortierverfahren mit einem neuen Array">Neues Array generieren</button>

42 <button type="button" style="float**:** **right**" class="**button** **orange**" id="**startSort**" title="Startet alle Sortierverfahren mit dem Aktuellen Array">Sorotierung Starten </button>

43 </div>

44 </div>

45

46 <h1>Sortierverfahren</h1>

47 <div id="**sortDesplay**">

48 <div class="**SortArea**"><h2>Bubblesort</h2><section id="**bubblesort**"></section></div>

49 <div class="**SortArea**"><h2>Insertionsort</h2><section id="**insertionsort**"></section></div>

50 <div class="**SortArea**"><h2>Selectionsort</h2><section id="**selectionsort**"></section></div>

51 <div class="**SortArea**"><h2>Shellsort</h2><section id="**shellsort**"></section></div>

52 <div class="**SortArea**"><h2>Quicksort</h2><section id="**quicksort**"></section></div>

53 <div class="**SortArea**"><h2>Heapsort</h2><section id="**heapsort**"></section></div>

54 <div class="**SortArea**"><h2>Bitonic Sort</h2><section id="**bitonicsorter**"></section></div>

55 <div class="**SortArea**"><h2>Odd–even Sort</h2><section id="**oddeventranspositionsort**"></section></div>

56 <div class="**SortArea**"><h2>Odd-even Mergesort</h2><section id="**oddevenmergesorter**"></section></div>

57 <div class="**SortArea**"><h2>Mergesort</h2><section id="**mergesort**"></section></div>

58 <div class="**SortArea**"><h2>Bogosort</h2><section id="**bogosort**"></section></div>

59 </div>

60 <script>

61 **window.onload** **=** function **()**

62 **{**

63 **addToSortOld(**"#bubblesort"**,** **eSort.bubblesort);**

64 **addToSortOld(**"#selectionsort"**,** **eSort.selectionsort);**

65 **addToSortOld(**"#insertionsort"**,** **eSort.insertionsort);**

66 **addToSortOld(**"#shellsort"**,** **eSort.shellsort);**

67 **addToSortOld(**"#quicksort"**,** **eSort.quicksort);**

68 **addToSortOld(**"#heapsort"**,** **eSort.heapsorter);**

69 **addToSortOld(**"#bitonicsorter"**,** **eSort.bitonicsorter);**

70 **addToSortOld(**"#oddeventranspositionsort"**,** **eSort.oddeventranspositionsort);**

71 **addToSortOld(**"#oddevenmergesorter"**,** **eSort.oddevenmergesorter);**

72 **addToSortOld(**"#mergesort"**,** **eSort.mergesort);**

73 **addToSortOld(**"#bogosort"**,** **eSort.bogosort);**

74

75 **initSortOld(eArrayTyp.Random,** **128);**

76 **document.getElementById(**"startSort"**).onclick** **=** **startSortOld;**

77 **document.getElementById(**"startNewSort"**).onclick** **=** function **()** **{**

78 var **lArrayTyp** **=** **document.getElementById(**"arrayTyp"**).value** **\*** **1;**

79 var **lMaxSteps** **=** **document.getElementById(**"maxSteps"**).value** **\*** **1;**

80 **initSortOld(lArrayTyp,** **lMaxSteps);**

81 **};**

82 **};**

83 </script>

84

85 </body>

86 </html>

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\SortPage03.html |

1 ﻿<!--

2 Dateiname : SortPage03.html

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 -->

8 <!DOCTYPE html>

9 <head>

10 <meta charset="utf-8">

11 <title>Heapsort Baumdarstellung</title>

12 <link rel="stylesheet" href="css/sortStyle.css">

13 <script src="js/sort.js"></script>

14 </head>

15 <body>

16 <div id="**heapsortTree**"></div>

17 <script>

18 window**.onload** **=** **function** **()**

19 **{**

20 **initHeapsortTree(**"#heapsortTree"**);**

21 **};**

22 </script>

23 </body>

24 </html>

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\SortPage04.html |

1 ﻿<!--

2 Dateiname : SortPage04.html

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 -->

8 <!DOCTYPE html>

9 <html>

10 <head>

11 <meta charset="utf-8">

12 <title>User Sort</title>

13 <link rel="stylesheet" href="css/sortStyle.css">

14 <script src="js/sort.js"></script>

15 </head>

16 <body>

17 <div><h2>BubbleSort</h2><section id="**bubblesort**"></section></div>

18 <div><h2>Selectionsort</h2><section id="**selectionsort**"></section></div>

19 <div><h2>Insertionsort</h2><section id="**insertionsort**"></section></div>

20 <div><h2>Shellsort</h2><section id="**shellsort**"></section></div>

21 <div><h2>Quicksort</h2><section id="**quicksort**"></section></div>

22 <div><h2>MergeSorter</h2><section id="**mergesort**"></section></div>

23 <script>

24 window**.onload** **=** **function** **()** **{**

25

26 **addToUserSort(**"bubblesort"**,** eSort**.bubblesort);**

27 **addToUserSort(**"selectionsort"**,** eSort**.selectionsort);**

28 **addToUserSort(**"insertionsort"**,** eSort**.insertionsort);**

29 **addToUserSort(**"shellsort"**,** eSort**.shellsort);**

30 **addToUserSort(**"quicksort"**,** eSort**.quicksort);**

31 **addToUserSort(**"mergesort"**,** eSort**.mergesort);**

32

33 **initUserSort();**

34 **};**

35 </script>

36 </body>

37 </html>

38

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\SortPage05.html |

1 ﻿<!--

2 Dateiname : SortPage05.html

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 -->

8 <!DOCTYPE html>

9 <head>

10 <meta charset="utf-8">

11 <title>Random Sort mit Zufällig entstehenden Bildern</title>

12 <link rel="stylesheet" href="css/sortStyle.css">

13 <script src="js/sort.js"></script>

14 </head>

15 <body>

16 <h1>Einstellungen</h1>

17 <div id="**settings**">

18 <div class="**dropdown** **orange**" style="float**:** **left;** margin-top**:** **0px;** margin-right**:** **5px;**" >

19 <select id="**maxSteps**" class="**dropdown-select** **orange**">

20 <option value="30">30</option>

21 <option value="25">25</option>

22 <option selected="" value="20">20</option>

23 <option value="15">15</option>

24 <option value="10">10</option>

25 <option value="5">5</option>

26

27 </select>

28 </div>

29 <button type="button" class="**button** **orange**" id="**startSort**" title="Startet ">Start</button>

30 <button type="button" class="**button** **orange**" id="**stopSort**" title="Startet ">Stop</button>

31 </div>

32 <h1>Random Sort</h1>

33 <div id="**sortDesplay**">

34 <div class="**SortArea**"><section id="**randomsort1**"></section> </div>

35 <div class="**SortArea**"><section id="**randomsort2**"></section> </div>

36 <div class="**SortArea**"><section id="**randomsort3**"></section> </div>

37 <div class="**SortArea**"><section id="**randomsort4**"></section> </div>

38

39 <div class="**SortArea**"><section id="**randomsort5**"></section> </div>

40 <div class="**SortArea**"><section id="**randomsort6**"></section> </div>

41 <div class="**SortArea**"><section id="**randomsort7**"></section> </div>

42 <div class="**SortArea**"><section id="**randomsort8**"></section> </div>

43

44 <div class="**SortArea**"><section id="**randomsort9**"></section> </div>

45 <div class="**SortArea**"><section id="**randomsort10**"></section> </div>

46 <div class="**SortArea**"><section id="**randomsort11**"></section> </div>

47 <div class="**SortArea**"><section id="**randomsort12**"></section> </div>

48

49 </div>

50 <script>

51 **window.onload** **=** function **()**

52 **{**

53 **addToSortRandom(**"#randomsort1"**,** **1);**

54 **addToSortRandom(**"#randomsort2"**,** **2);**

55 **addToSortRandom(**"#randomsort3"**,** **3);**

56 **addToSortRandom(**"#randomsort4"**,** **4);**

57 **addToSortRandom(**"#randomsort5"**,** **5);**

58 **addToSortRandom(**"#randomsort6"**,** **6);**

59 **addToSortRandom(**"#randomsort7"**,** **7);**

60 **addToSortRandom(**"#randomsort8"**,** **8);**

61 **addToSortRandom(**"#randomsort9"**,** **9);**

62 **addToSortRandom(**"#randomsort10"**,** **10);**

63 **addToSortRandom(**"#randomsort11"**,** **11);**

64 **addToSortRandom(**"#randomsort12"**,** **12);**

65 var **lWidth** **=** **200;**

66 var **lStep** **=** **20;**

67 **initSortRandom(lWidth,** **lStep);**

68 **document.getElementById(**"startSort"**).onclick** **=** function **()** **{**

69 **lStep** **=** **document.getElementById(**"maxSteps"**).value** **\*** **1;**

70 **stopSortRandom();**

71 **initSortRandom(lWidth,** **lStep);**

72 **startSortRandom();**

73 **};**

74 **document.getElementById(**"stopSort"**).onclick** **=** **stopSortRandom;**

75

76 **};**

77 </script>

78 </body>

79 </html>

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort.js |

1 /\*

2 Dateiname : sort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 /\*

10 \* ArrayTyp Enum

11 \* **@type** int

12 \*/

13 var **eArrayTyp** = {

14 Random: 0,

15 Up: 1,

16 Down: 2,

17 RandomOneToHundred: 3

18 };

19 /\*

20 \* Sort Enum

21 \* **@type** type

22 \*/

23 var **eSort** = {

24 bubblesort: 0,

25 selectionsort: 1,

26 insertionsort: 2,

27 shellsort: 3,

28 quicksort: 4,

29 heapsorter: 5,

30 mergesort: 6,

31 bitonicsorter: 7,

32 oddeventranspositionsort: 8,

33 oddevenmergesorter: 9,

34 bogosort: 10

35 };

36

37 /\*

38 \* Läde alle benötigten JS Datei nach

39 \* sodass sie nicht in der HTML gesetzt werden müssen

40 \* **@returns** {undefined}

41 \*/

42 function **includeAll**() {

43 //D3js für die verarbeitung

44 include('js/d3/d3.min.js');

45

46 //Sortier Anzeigen

47 include('js/animate/animate-sort-new.js');

48 include('js/animate/animate-sort-old.js');

49 include('js/animate/animate-randomsort.js');

50 include('js/animate/animate-heapsort.js');

51 include('js/animate/animate-usersort.js');

52

53 //Sortierverfahren

54 include('js/sort/bubblesort.js');

55 include('js/sort/selectionsort.js');

56 include('js/sort/insertionsort.js');

57 include('js/sort/shellsort.js');

58 include('js/sort/quicksort.js');

59 include('js/sort/heapsort.js');

60 include('js/sort/mergesort.js');

61 include('js/sort/bitonicsort.js');

62 include('js/sort/oddeventranspositionsort.js');

63 include('js/sort/oddevenmergesort.js');

64 include('js/sort/bogosort.js');

65

66 include('js/randomsort.js');

67

68 }

69 /\*

70 \* Schreibt die JS Datei an die position

71 \* an der die sort.js Datei ausgeführt wird

72 \* **@param** {string} pFile

73 \* **@returns** {undefined}

74 \*/

75 function **include**(pFile) {

76 **document**.write('<script type="text/javascript" src="' + pFile + '"></script>');

77 }

78 includeAll();

79

80 /\*

81 \* Erzeugt die entsprechende Klasse

82 \* **@param** {type} pSortID

83 \* **@returns** {Bogosort|SelectionSort|Quicksort|ShellSort|BubbleSort|OddEvenTranspositionSort|MergeSorter|BitonicSorter|InsertionSort|Heapsort|OddEvenMergeSorter}

84 \*/

85 function **getSortClass**(pSortID) {

86 switch (pSortID) {

87 case **eSort**.bubblesort:

88 return new BubbleSort();

89 break;

90 case **eSort**.selectionsort:

91 return new SelectionSort();

92 break;

93 case **eSort**.insertionsort:

94 return new InsertionSort();

95 break;

96 case **eSort**.shellsort:

97 return new ShellSort();

98 break;

99 case **eSort**.quicksort:

100 return new Quicksort();

101 break;

102 case **eSort**.heapsorter:

103 return new Heapsort();

104 break;

105 case **eSort**.mergesort:

106 return new MergeSorter();

107 break;

108 case **eSort**.bitonicsorter:

109 return new BitonicSorter();

110 break;

111 case **eSort**.oddeventranspositionsort:

112 return new OddEvenTranspositionSort();

113 break;

114 case **eSort**.oddevenmergesorter:

115 return new OddEvenMergeSorter();

116 break;

117 case **eSort**.bogosort:

118 return new Bogosort();

119 break;

120 }

121 }

122

123 /\*

124 \* Erstellung eines Array für die Sortierung

125 \* **@param** {int} pArrayTyp

126 \* siehe eArrayTyp

127 \* **@param** {int} pMaxSteps

128 \* Sollte immer ein Vierfaches von 2 sein zum Beispiel: 64 oder 128

129 \* Anderefals Laufen die Sortieverfahren bitonicsorter und oddevenmergesorter nicht zufrieden stellend

130 \* **@returns** {lArray}

131 \*/

132 function **creatArray**(pArrayTyp, pMaxSteps) {

133 var lArray = **d3**.range(pMaxSteps);

134 switch (pArrayTyp) {

135 case **eArrayTyp**.Random:

136 return shuffleRandom(lArray.slice());

137 break;

138 case **eArrayTyp**.Up:

139 return lArray.slice();

140 break;

141 case **eArrayTyp**.Down:

142 return lArray.slice().sort(function(a, b){return b-a});

143 break;

144 case **eArrayTyp**.RandomOneToHundred:

145 return randomOneToHundred(lArray.slice());

146 break;

147 }

148 return lArray.slice();

149 }

150 /\*

151 \* Mischt das Array zufällig durch

152 \* **@param** {type} pArray

153 \* **@returns** {pArray}

154 \*/

155 function **shuffleRandom**(pArray) {

156 var i = pArray.length, j, t;

157 while (--i > 0) {

158 j = ~~(**Math**.random() \* (i + 1));

159 t = pArray[j];

160 pArray[j] = pArray[i];

161 pArray[i] = t;

162 }

163 return pArray;

164 }

165 /\*

166 \* Setzt neue Eintrage in das Array

167 \* zwischen 1 und max 100

168 \* **@param** {type} pArray

169 \* **@returns** {pArray}

170 \*/

171 function **randomOneToHundred**(pArray) {

172 var i = pArray.length, j;

173 var lmax = **Math**.min(100,i);

174 while (--i > 0) {

175 j = ~~(**Math**.random() \* (lmax + 1));

176 pArray[i] = j;

177 }

178 return pArray;

179 }

180

181 /\*

182 \* Benutzungs Hild für die Anzeige der Neuen Sortier Anzeige

183 \* initSortNew

184 \* Generiert die Auszugebene Klasse mit dem enstrechendem Array

185 \*

186 \* addToSortNew

187 \* Fügt einen Sortierverfahren hinzu

188 \*

189 \* startSortNew

190 \* Startet alle hinzugefügten Sortiervefarhen mit dem aktuellen Array

191 \*/

192 var **SortNewDic** = {};

193 var **SortNew** = null;

194 function **initSortNew**(pArrayTyp, pMaxSteps) {

195 if (**SortNew** !== null) {

196 for (var key in **SortNewDic**) {

197 **d3**.select(key).select("svg").remove();

198 }

199 }

200 **SortNew** = new SortingAnimationNew(creatArray(pArrayTyp, pMaxSteps),pMaxSteps);

201 for (var key in **SortNewDic**) {

202 **d3**.select(key).on("click", function () {

203 var pSortAreaID = "#" + this.id;

204 **d3**.select(pSortAreaID).select("svg").remove();

205 **SortNew**.add(**SortNewDic**[pSortAreaID].sortAll, pSortAreaID);

206 });

207 **SortNew**.add(**SortNewDic**[key].sortAll, key);

208 }

209 **SortNew**.reset();

210 **SortNew**.start(1000);

211 }

212 function **addToSortNew**(pSortAreaID, pSortID) {

213 **SortNewDic**[pSortAreaID] = getSortClass(pSortID);

214 }

215 function **startSortNew**() {

216 if (**SortNew** !== null) {

217 for (var key in **SortNewDic**) {

218 **d3**.select(key).select("svg").remove();

219 **SortNew**.add(**SortNewDic**[key].sortAll, key);

220 }

221 **SortNew**.start(1000);

222 }

223 }

224

225 /\*

226 \* Benutzungs Hild für die Anzeige der Alten Sortier Anzeige

227 \* initSortOld

228 \* Generiert die Auszugebene Klasse mit dem enstrechendem Array

229 \*

230 \* addToSortOld

231 \* Fügt einen Sortierverfahren hinzu

232 \*

233 \* startSortOld

234 \* Startet alle hinzugefügten Sortiervefarhen mit dem aktuellen Array

235 \*/

236 var **SortOldDic** = {};

237 var **SortOld** = null;

238 function **initSortOld**(pArrayTyp, pMaxSteps) {

239 if (**SortOld** !== null) {

240 for (var key in **SortOldDic**) {

241 **d3**.select(key).select("svg").remove();

242 }

243 }

244 **SortOld** = new SortingAnimationOld(creatArray(pArrayTyp, pMaxSteps),pMaxSteps);

245 for (var key in **SortOldDic**) {

246 **d3**.select(key).on("click", function () {

247 var pSortAreaID = "#" + this.id;

248 **d3**.select(pSortAreaID).select("svg").remove();

249 **SortOld**.add(**SortOldDic**[pSortAreaID].sortAll, pSortAreaID);

250 });

251 **SortOld**.add(**SortOldDic**[key].sortAll, key);

252 }

253 **SortOld**.reset();

254 **SortOld**.start(1000);

255 }

256 function **addToSortOld**(pSortAreaID, pSortID) {

257 **SortOldDic**[pSortAreaID] = getSortClass(pSortID);

258 }

259 function **startSortOld**() {

260 if (**SortOld** !== null) {

261 for (var key in **SortOldDic**) {

262 **d3**.select(key).select("svg").remove();

263 **SortOld**.add(**SortOldDic**[key].sortAll, key);

264 }

265 **SortOld**.start(1000);

266 }

267 }

268

269 /\*

270 \* Heapsort Baumdarstellung

271 \*

272 \* initHeapsortTree

273 \* Generiert die Auszugebene Klasse

274 \*/

275 var **HeapsortTree** = null;

276 function **initHeapsortTree**(pSortAreaID){

277 **HeapsortTree** = new heapsortTree(pSortAreaID);

278 }

279

280 /\*

281 \* Usersort Benutzerdarstellung

282 \*

283 \* initUsersort

284 \* Generiert die Auszugebene Klasse

285 \*/

286 var **UserSortDic** = {};

287 var **UserSortAll** = null;

288 function **initUserSort**() {

289 for (var key in **UserSortDic**) {

290 new UserSorting(**UserSortDic**[key], key);

291 }

292 }

293 function **addToUserSort**(pSortAreaID, pSortID) {

294 **UserSortDic**[pSortAreaID] = getSortClass(pSortID);

295 }

296

297 /\*

298 \* Random Sort Züfallige Bild Erzeugung

299 \*

300 \* initRandomSort

301 \* Generiert die Auszugebene Klasse

302 \*/

303 var **SortRandomDic** = {};

304 var **SortRandom** = null;

305 function **initSortRandom**(pWidth,pStep) {

306 if (**SortRandom** !== null) {

307 for (var key in **SortRandomDic**) {

308 **d3**.select(key).select("svg").remove();

309 }

310 }

311 **SortRandom** = new SortingAnimationRandom(pWidth,pStep,creatRGB((pStep\*pStep)-1));

312 for (var key in **SortRandomDic**) {

313 **d3**.select(key).on("click", function () {

314 var pSortAreaID = "#" + this.id;

315 **d3**.select(pSortAreaID).select("svg").remove();

316 **SortRandom**.add(**SortRandomDic**[pSortAreaID].sort, pSortAreaID);

317 **SortRandom**.restart();

318 });

319 **SortRandom**.add(**SortRandomDic**[key].sort, key);

320 }

321 **SortRandom**.reset();

322 **SortRandom**.start(1000);

323 }

324

325 function **addToSortRandom**(pSortAreaID, pSortID) {

326 **SortRandomDic**[pSortAreaID] = new RandomSort(pSortID);

327 }

328 function **startSortRandom**() {

329 if (**SortRandom**!== null) {

330 for (var key in **SortRandomDic**) {

331 **d3**.select(key).select("svg").remove();

332 **SortRandom**.add(**SortRandomDic**[key].sort, key);

333 }

334 **SortRandom**.start(1000);

335 }

336 }

337 function **stopSortRandom**() {

338 if (**SortRandom**!== null) {

339 **SortRandom**.reset();

340 }

341 }

342

343

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\animate\animate-heapsort.js |

1 /\*

2 Dateiname : animate-heapsort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 /\*

10 \* Erzuegt ein Bereich der alle Ausgabe

11 \* und eingabefelder besitzt die benötigt werden

12 \* **@param** {type} pTarget

13 \* **@returns** {heapsortTree}

14 \*/

15 function **heapsortTree**(pTarget) {

16

17 //Variablen

18 var target = pTarget;

19

20 var startHeapArray = [];

21 var heapArray = [];

22 var endHeapArray = [];

23 var maxArray = 13;

24 var leafFocus = 0;

25 var svgW = 620;

26 var svgH = 320;

27 var vRad = 18;

28 var treeFin = false;

29 var isHeap = false;

30 var **tree** = {cx: svgW / 2, cy: 30, w: 60, h: 70};

31 tree.**vis** = {v: 0, v2: 0, l: '?', **p**: {x: tree.cx, y: tree.cy}, c: []};

32 tree.size = 1;

33

34 /\*

35 \* Initialisiert einen neuen Baum mit neuem Array

36 \* **@returns** {undefined}

37 \*/

38 **initTree** = function () {

39 createNewArray();

40 tree.vis = {v: 0, v2: 0, l: heapArray[0], p: {x: tree.cx, y: tree.cy}, c: []};

41 tree.size = 1;

42 treeFin = false;

43 drawStartArrayDiv();

44 drawEndArrayDiv();

45 drawTree();

46 setFocus(0);

47 drawInfoText("Zunächst muss der Baum erstellt werden");

48 };

49 /\*

50 \* Wenn der Baum noch nicht fertiggestellt wurde

51 \* Wird er mit einer leichten Verzögerung zu Ende erstellt

52 \* **@returns** {undefined}

53 \*/

54 **finAutoTree** = function () {

55 if (treeFin)

56 return;

57 function **addWithTimeout**() {

58 if (tree.size < heapArray.length) {

59 addLeaf(getNextLeaf(-1));

60 setTimeout(function () {

61 addWithTimeout();

62 }, 500);

63 }

64 else {

65 treeFin = true;

66 setFocus(0);

67 drawInfoText("Der Baum ist nun erstellt, nun zur Heap Eigenschaft");

68 }

69 }

70 addWithTimeout();

71 };

72 /\*

73 \* es wird geschaut ob der Baum schon seine Heap eigenschaft hat

74 \* wenn nicht wird die Heap eigenschaft umgesetzt

75 \* mit eine leichten verzögerung

76 \* **@returns** {undefined}

77 \*/

78 **finAutoHeap** = function () {

79 leafFocus = 0;

80 function **heapWithTimeout**() {

81 if ((leafFocus >= 0)) {

82 setFocus(leafFocus);

83 if (isLeafHeap(leafFocus, (leafFocus \* 2) + 1) === 1) {

84 changeLeaf(leafFocus, (leafFocus \* 2) + 1);

85 }

86 else if (isLeafHeap(leafFocus, (leafFocus + 1) \* 2) === 1) {

87 changeLeaf(leafFocus, (leafFocus + 1) \* 2);

88 }

89 leafFocus = hasHeap();

90 setTimeout(function () {

91 heapWithTimeout();

92 }, 1000);

93 }

94 else {

95 isHeap = true;

96 setFocus(0);

97 }

98 }

99 heapWithTimeout();

100 };

101 /\*

102 \* Löste den gesamten Baum von anfang an oder mitten drin an gestartet

103 \* **@returns** {undefined}

104 \*/

105 **finAutoFullTree** = function () {

106 function **addWithTimeout**() {

107 if (tree.size < heapArray.length) {

108 addLeaf(getNextLeaf(-1));

109 setTimeout(function () {

110 addWithTimeout();

111 }, 500);

112 }

113 else {

114 treeFin = true;

115 setFocus(0);

116 drawInfoText("Der Baum ist nun erstellt, nun zur Heap Eigenschaft");

117 leafFocus = 0;

118 setTimeout(function () {

119 heapWithTimeout();

120 }, 2000);

121 }

122 }

123 function **heapWithTimeout**() {

124 if ((leafFocus >= 0)) {

125 setFocus(leafFocus);

126 if (isLeafHeap(leafFocus, (leafFocus \* 2) + 1) === 1) {

127 changeLeaf(leafFocus, (leafFocus \* 2) + 1);

128 }

129 else if (isLeafHeap(leafFocus, (leafFocus + 1) \* 2) === 1) {

130 changeLeaf(leafFocus, (leafFocus + 1) \* 2);

131 }

132 leafFocus = hasHeap();

133 setTimeout(function () {

134 heapWithTimeout();

135 }, 1000);

136 }

137 else {

138 isHeap = true;

139 setFocus(0);

140 setTimeout(function () {

141 setToEndArray();

142 }, 1000);

143 if (tree.size !== 0) {

144 setTimeout(function () {

145 heapWithTimeout();

146 }, 1000);

147 }

148 }

149 }

150 if (treeFin) {

151 leafFocus = 0;

152 heapWithTimeout();

153 }

154 else

155 addWithTimeout();

156 };

157 /\*

158 \* Erstellt ein neues Array welches mit zufallszahlen gefühlt wird

159 \* **@returns** {undefined}

160 \*/

161 **createNewArray** = function () {

162 heapArray = [];

163 endHeapArray = [];

164 var lMax = **Math**.floor((**Math**.random() \* (maxArray - 7)) + 8);

165 for (var i = 0; i < lMax; i++) {

166 heapArray.push(**Math**.floor((**Math**.random() \* lMax) + 1));

167 endHeapArray.push(-1);

168 }

169 startHeapArray = heapArray.slice();

170 };

171 /\*

172 \* Setzt die Variable des Stams in das end Array

173 \* und setzt das die Variable des letzte Blattes an die Stelle des Stams

174 \* **@returns** {undefined}

175 \*/

176 **setToEndArray** = function () {

177 endHeapArray[tree.size - 1] = getLeaf(0);

178 drawEndArrayDiv();

179 tree.vis.l = getLeaf(tree.size - 1);

180 removeLeaf(tree.size - 1);

181 if (tree.size === 0)

182 drawInfoText("Der Baum wurde vollständig gelöst");

183 else

184 drawInfoText("Heap Eigenschaft erneut erreichen");

185 setFocus(0);

186 };

187 /\*

188 \* Füght einen neues Blat hinzu

189 \* und nimmt die ensprechende näschte Astgabelung in focus

190 \* **@param** {type} leafId

191 \* **@returns** {undefined}

192 \*/

193 **addLeaf** = function (pLeafId) {

194 function **add**(t) {

195 if (t.v === pLeafId) {

196 t.c.push({v: tree.size++, v2: tree.size, l: heapArray[tree.size - 1], **p**: {}, c: []});

197 return;

198 }

199 t.c.forEach(add);

200 }

201 add(tree.vis);

202 reposition(tree.vis);

203 redrawTree();

204 setFocus(getNextLeaf(-1));

205 };

206 /\*

207 \* Gibt die Variable des entsprechenden Blattes zurück

208 \* **@param** {int} leafId

209 \* **@returns** {Number}

210 \*/

211 **getLeaf** = function (pLeafId) {

212 var lReturn = -1;

213 function **get**(t) {

214 if (lReturn !== -1)

215 return;

216 if (t.v === pLeafId) {

217 lReturn = t.l;

218 return;

219 }

220 t.c.forEach(get);

221 }

222 get(tree.vis);

223 return lReturn;

224 };

225 /\*

226 \* Entfernt das ensprechende Blatt

227 \* Wenn keine Blatter mehr da sind wird einen Leeres Blatt stehen gelassen

228 \* **@param** {type} pLeafId

229 \* **@returns** {Number}

230 \*/

231 **removeLeaf** = function (pLeafId) {

232 var lReturn = -2;

233 function **remove**(t) {

234 if (lReturn !== -2)

235 return;

236 if (t.c.length === 0)

237 return;

238 if (t.c.length === 1) {

239 if (t.c[0].v === pLeafId) {

240 t.c.splice(0, 1);

241 lReturn = 1;

242 return;

243 }

244 }

245 if (t.c.length === 2) {

246 if (t.c[0].v === pLeafId) {

247 t.c.splice(0, 1);

248 lReturn = 1;

249 return;

250 }

251 if (t.c[1].v === pLeafId) {

252 t.c.splice(1, 2);

253 lReturn = 1;

254 return;

255 }

256 }

257 t.c.forEach(remove);

258 }

259 remove(tree.vis);

260 if (tree.size === 1) {

261 tree.vis = {v: 0, v2: 0, l: '', p: {x: tree.cx, y: tree.cy}, c: []};

262 lReturn = 1;

263 }

264 if (lReturn === 1) {

265 tree.size--;

266 reposition(tree.vis);

267 drawTree();

268 }

269 return lReturn;

270 };

271 /\*

272 \* Vertausch zwei entsprechende Blätter

273 \* um die Heap eigenschaft zu gewährleisten

274 \* **@param** {type} pLeafIdFrom

275 \* **@param** {type} pLafIdTo

276 \* **@returns** {undefined}

277 \*/

278 **changeLeaf** = function (pLeafIdFrom, pLafIdTo) {

279 function **change**(t) {

280 if (t.v === pLeafIdFrom) {

281 var v2 = t.v2;

282 var l = t.l;

283 if (t.c[0].v === pLafIdTo) {

284 t.v2 = t.c[0].v2;

285 t.c[0].v2 = v2;

286 t.l = t.c[0].l;

287 t.c[0].l = l;

288 }

289 else {

290 t.v2 = t.c[1].v2;

291 t.c[1].v2 = v2;

292 t.l = t.c[1].l;

293 t.c[1].l = l;

294 }

295 return;

296 }

297 t.c.forEach(change);

298 }

299 change(tree.vis);

300 redrawTree();

301 };

302 /\*

303 \* Gibt die nächste Astgabelung zurück

304 \* des entsprechenden Blattes

305 \* **@param** {type} pLeafId

306 \* **@returns** {unresolved}

307 \*/

308 **getNextLeaf** = function (pLeafId) {

309 if (pLeafId > -1)

310 return parseInt((pLeafId - 1) / 2, 10);

311 else

312 return parseInt((tree.size - 1) / 2, 10);

313 };

314 /\*

315 \* Zählt alle Blätter die an der Astgableung dran sind

316 \* **@param** {type} pLeaf

317 \* **@returns** {Number}

318 \*/

319 **getLeafCount** = function (pBranch) {

320 if (pBranch.c.length === 0)

321 return 1;

322 else

323 return pBranch.c.map(getLeafCount).reduce(function (a, b) {

324 return a + b;

325 });

326 };

327 /\*

328 \* Sortiert den Baum ab den entsprechenden Astgabelung um

329 \* **@param** {type} pBranch

330 \* **@returns** {undefined}

331 \*/

332 **reposition** = function (pBranch) {

333 var lC = getLeafCount(pBranch), left = pBranch.p.x - tree.w \* (lC - 1) / 2;

334 pBranch.c.forEach(function (d) {

335 var w = tree.w \* getLeafCount(d);

336 left += w;

337 d.**p** = {x: left - (w + tree.w) / 2, y: pBranch.p.y + tree.h};

338 reposition(d);

339 });

340 };

341 /\*

342 \* Klick Funktion der Blätter

343 \* **@param** {type} pLeafId

344 \* **@returns** {undefined}

345 \*/

346 **clickLeaf** = function (pLeafId) {

347 if (tree.size >= heapArray.length || treeFin) {

348 if (leafFocus === pLeafId) {

349 if (isLeafHeap(leafFocus, pLeafId) === 1) {

350 setFocus(leafFocus + 1);

351 }

352 else

353 drawInfoText("Heap Eigenschaft ist nicht erreicht");

354 }

355 else {

356 if (leafFocus === getNextLeaf(pLeafId)) {

357 if (isLeafHeap(leafFocus, pLeafId) === 1) {

358 changeLeaf(leafFocus, pLeafId);

359 setFocus(leafFocus + 1);

360 }

361 else

362 drawInfoText("Heap Eigenschaft ist nicht erreicht");

363 }

364 else {

365 if (isHeap) {

366 setToEndArray();

367 }

368 }

369 }

370 }

371 else {

372 if (pLeafId !== getNextLeaf(-1)) {

373 drawInfoText("Der Baum ist noch nicht fertig");

374 }

375 else {

376 addLeaf(pLeafId);

377 if (tree.size === heapArray.length) {

378 treeFin = true;

379 setFocus(0);

380 drawInfoText("Der Baum ist nun erstellt, nun zur Heap Eigenschaft");

381 }

382 }

383 }

384 };

385 /\*

386 \* Überprüft ob der gesamte Baum die Heap Eigenschaft besitzt

387 \* Wenn gibt er -1 zurück

388 \* wenn nicht die Blatt ID an der es keine Heap Eigenschaft gibt

389 \* **@returns** {t.v|Number}

390 \*/

391 **hasHeap** = function () {

392 var lReturn = -2;

393 function **heap**(t) {

394 if (lReturn !== -2)

395 return;

396 if (t.c.length === 0)

397 return;

398 if (t.c.length === 1) {

399 if (t.l < t.c[0].l) {

400 lReturn = t.v;

401 return;

402 }

403 }

404 if (t.c.length === 2) {

405 if (t.l < t.c[0].l || t.l < t.c[1].l) {

406 lReturn = t.v;

407 return;

408 }

409 }

410 t.c.forEach(heap);

411 }

412 heap(tree.vis);

413 if (lReturn === -2)

414 lReturn = -1;

415 return lReturn;

416 };

417 /\*

418 \* Überprüft ob die beiden Blätter die Heap Eigenschaft besitzen

419 \* wenn nicht -1

420 \* **@param** {int} pLeafIdFrom

421 \* **@param** {int} pLeafIdTo

422 \* **@returns** {Number}

423 \*/

424 **isLeafHeap** = function (pLeafIdFrom, pLeafIdTo) {

425 var lReturn = -2;

426 function **isHeap**(t) {

427 if (t.v === pLeafIdFrom) {

428 if (lReturn !== -2)

429 return;

430 if (t.c.length === 0)

431 return;

432 if (t.v === pLeafIdTo) {

433 if (t.c.length === 1) {

434 if (t.l < t.c[0].l)

435 {

436 lReturn = 0;

437 return;

438 }

439 }

440 if (t.c.length === 2) {

441 if (t.l < t.c[0].l || t.l < t.c[1].l) {

442 lReturn = 0;

443 return;

444 }

445 }

446 lReturn = 1;

447 return;

448 }

449 else {

450 if (t.c.length === 1) {

451 if (t.c[0].v === pLeafIdTo && t.l < t.c[0].l) {

452 lReturn = 1;

453 return;

454 }

455 }

456 else {

457 if (t.c[0].v === pLeafIdTo && t.l < t.c[0].l && t.c[1].l <= t.c[0].l) {

458 lReturn = 1;

459 return;

460 }

461 if (t.c[1].v === pLeafIdTo && t.l < t.c[1].l && t.c[0].l <= t.c[1].l) {

462 lReturn = 1;

463 return;

464 }

465 }

466 lReturn = 0;

467 return;

468 }

469 }

470 t.c.forEach(isHeap);

471 }

472 isHeap(tree.vis);

473 if (lReturn === -2)

474 lReturn = -1;

475 return lReturn;

476 };

477 /\*

478 \* Setzt das ensprechende Blatt in den Focus

479 \* **@param** {type} pLeafId

480 \* **@returns** {undefined}

481 \*/

482 **setFocus** = function (pLeafId) {

483 if (hasHeap() === -1)

484 pLeafId = -1;

485 else if (isLeafHeap(pLeafId, pLeafId) === -1 && treeFin)

486 pLeafId = hasHeap();

487 leafFocus = pLeafId;

488 if (pLeafId === -1) {

489 if (tree.size > 0) {

490 drawInfoText("Heap Eigenschaft erreicht, Wurzel entfernen");

491 pLeafId = 0;

492 }

493 isHeap = true;

494 }

495 else

496 isHeap = false;

497

498 var circles = **d3**.select("#g\_circles").selectAll('circle').data(getVertices());

499 circles.attr('style', function (d) {

500 if (d.v === pLeafId) {

501 drawFocusArea(pLeafId);

502 return 'stroke:steelblue;stroke-width:4px;';

503 }

504 else

505 return '';

506 });

507 };

508 /\*

509 \* Gibt die Positionen der einzelenen Kreise zurück die angezeigt werden sollen

510 \* für den Baum

511 \* **@returns** {Array}

512 \*/

513 **getVertices** = function () {

514 var v = [];

515 function **getVert**(t, f) {

516 v.push({v: t.v, v2: t.v2, l: t.l, p: t.p, f: f});

517 t.c.forEach(function (d) {

518 return getVert(d, {v: t.v, v2: t.v2, p: t.p});

519 });

520 }

521

522 getVert(tree.vis, {});

523 return v.sort(function (a, b) {

524 return a.v2 - b.v2;

525 });

526 };

527 /\*

528 \* Gibt die Positionen der Verbindungslinien zurück

529 \* die angezeigt werden sollen für den Baum

530 \* **@returns** {Array}

531 \*/

532 **getEdges** = function () {

533 var e = [];

534 function **getEd**(t) {

535 t.c.forEach(function (d) {

536 e.push({v1: t.v, l1: t.l, p1: t.p, v2: d.v, l2: d.l, p2: d.p});

537 });

538 t.c.forEach(getEd);

539 }

540 getEd(tree.vis);

541 return e.sort(function (a, b) {

542 return a.v2 - b.v2;

543 });

544 };

545 /\*

546 \* Gibt die Positionen der Verbindungslinien

547 \* für den Focusierten Bereich zurück

548 \* **@param** {type} pLeafId

549 \* **@returns** {heapsortTree.getFocusArea.e2|Array}

550 \*/

551 **getFocusArea** = function (pLeafId) {

552 var e = [];

553 function **getEd**(t) {

554 t.c.forEach(function (d) {

555 if (t.v === pLeafId)

556 e.push({v1: t.v, l1: t.l, p1: t.p, v2: d.v, l2: d.l, p2: d.p});

557 });

558 t.c.forEach(getEd);

559 }

560 getEd(tree.vis);

561 var e2 = [];

562 if (e.length === 0)

563 return e2;

564 var lLeft = e[0];

565 var lRight = e[0];

566 if (e.length > 1)

567 lRight = e[1];

568 var lSpeas = 25;

569 e2.push({**p1**: {x: lLeft.p2.x - lSpeas, y: lLeft.p2.y + lSpeas}, **p2**: {x: lRight.p2.x + lSpeas, y: lRight.p2.y + lSpeas}});

570 e2.push({**p1**: {x: lRight.p2.x + lSpeas, y: lRight.p2.y + lSpeas}, **p2**: {x: lRight.p2.x + lSpeas, y: lRight.p1.y - lSpeas}});

571 e2.push({**p1**: {x: lRight.p2.x + lSpeas, y: lRight.p1.y - lSpeas}, **p2**: {x: lLeft.p2.x - lSpeas, y: lLeft.p1.y - lSpeas}});

572 e2.push({**p1**: {x: lLeft.p2.x - lSpeas, y: lLeft.p1.y - lSpeas}, **p2**: {x: lLeft.p2.x - lSpeas, y: lLeft.p2.y + lSpeas}});

573 return e2;

574 };

575 /\*

576 \* Gibt die Positionen der einzelenen Kreise zurück die angezeigt werden sollen

577 \* für das Start Array und das End Array

578 \*/

579 **getPosition** = function (pStart) {

580 var lSteep = svgW / startHeapArray.length;

581 var lPositions = [];

582

583 if (pStart) {

584 for (var i = 0; i < startHeapArray.length; i++) {

585 lPositions.push({v: i, l: startHeapArray[i], up: 1, **p**: {x: lSteep \* i + 30, y: 20}, **p2**: {x: lSteep \* i + 30, y: 20}});

586 }

587 }

588 else {

589 for (var i = 0; i < endHeapArray.length; i++) {

590 lPositions.push({v: i, l: endHeapArray[i], up: 0, **p**: {x: lSteep \* i + 30, y: 20}, **p2**: {x: lSteep \* i + 30, y: 20}});

591 }

592 }

593 return lPositions;

594 };

595 /\*

596 \* Zeichnet den Baum neu ohne Animation

597 \* **@returns** {undefined}

598 \*/

599 **drawTree** = function () {

600 //Löscht die alten Elemente

601 **d3**.select("#g\_lines").remove();

602 **d3**.select("#g\_circles").remove();

603 **d3**.select("#g\_labels").remove();

604 //Lines für die verbindungen

605 **d3**.select("#treesvg").append('g').attr('id', 'g\_lines').selectAll('line').data(getEdges()).enter().append('line')

606 .attr('x1', function (d) {

607 return d.p1.x;

608 })

609 .attr('y1', function (d) {

610 return d.p1.y;

611 })

612 .attr('x2', function (d) {

613 return d.p2.x;

614 })

615 .attr('y2', function (d) {

616 return d.p2.y;

617 });

618 //Kreise für die Blätter

619 **d3**.select("#treesvg").append('g').attr('id', 'g\_circles').selectAll('circle').data(getVertices()).enter().append('circle')

620 .attr('cx', function (d) {

621 return d.p.x;

622 })

623 .attr('cy', function (d) {

624 return d.p.y;

625 })

626 .attr('r', vRad)

627 .on('click', function (d) {

628 return clickLeaf(d.v);

629 });

630 //Nummerierung der Kreise

631 **d3**.select("#treesvg").append('g').attr('id', 'g\_labels').selectAll('text').data(getVertices()).enter().append('text')

632 .attr('x', function (d) {

633 return d.p.x;

634 })

635 .attr('y', function (d) {

636 return d.p.y + 5;

637 })

638 .text(function (d) {

639 return d.l;

640 })

641 .on('click', function (d) {

642 return clickLeaf(d.v);

643 });

644 };

645 /\*

646 \* Zeichnet den Baum mitrtels Animation um

647 \* **@returns** {undefined}

648 \*/

649 **redrawTree** = function () {

650 //Verbindungsstriche des Baums

651 var edges = **d3**.select("#g\_lines").selectAll('line').data(getEdges());

652 edges.transition().duration(500)

653 .attr('x1', function (d) {

654 return d.p1.x;

655 })

656 .attr('y1', function (d) {

657 return d.p1.y;

658 })

659 .attr('x2', function (d) {

660 return d.p2.x;

661 })

662 .attr('y2', function (d) {

663 return d.p2.y;

664 });

665 edges.enter().append('line')

666 .attr('x1', function (d) {

667 return d.p1.x;

668 })

669 .attr('y1', function (d) {

670 return d.p1.y;

671 })

672 .attr('x2', function (d) {

673 return d.p1.x;

674 })

675 .attr('y2', function (d) {

676 return d.p1.y;

677 })

678 .transition().duration(500)

679 .attr('x2', function (d) {

680 return d.p2.x;

681 })

682 .attr('y2', function (d) {

683 return d.p2.y;

684 });

685 //Variablen Kreise des Baums

686 var circles = **d3**.select("#g\_circles").selectAll('circle').data(getVertices());

687 circles.transition().duration(500)

688 .attr('cx', function (d) {

689 return d.p.x;

690 })

691 .attr('cy', function (d) {

692 return d.p.y;

693 });

694 circles.enter().append('circle')

695 .attr('cx', function (d) {

696 return d.f.p.x;

697 })

698 .attr('cy', function (d) {

699 return d.f.p.y;

700 })

701 .attr('r', vRad)

702 .on('click', function (d) {

703 return clickLeaf(d.v);

704 })

705 .transition().duration(500)

706 .attr('cx', function (d) {

707 return d.p.x;

708 })

709 .attr('cy', function (d) {

710 return d.p.y;

711 });

712 //Variablen Texte der Kreise des Baums

713 var labels = **d3**.select("#g\_labels").selectAll('text').data(getVertices());

714 labels.transition().duration(500)

715 .text(function (d) {

716 return d.l;

717 })

718 .attr('x', function (d) {

719 return d.p.x;

720 })

721 .attr('y', function (d) {

722 return d.p.y + 5;

723 });

724 labels.enter().append('text')

725 .attr('x', function (d) {

726 return d.f.p.x;

727 })

728 .attr('y', function (d) {

729 return d.f.p.y + 5;

730 })

731 .text(function (d) {

732 return d.l;

733 })

734 .on('click', function (d) {

735 return clickLeaf(d.v);

736 })

737 .transition().duration(500)

738 .attr('x', function (d) {

739 return d.p.x;

740 })

741 .attr('y', function (d) {

742 return d.p.y + 5;

743 });

744 };

745 /\*

746 \* Zeichnet den Fokussierten Bereich der zurzeit relevant ist

747 \* **@param** {type} pLeafId

748 \* **@returns** {undefined}

749 \*/

750 **drawFocusArea** = function (pLeafId) {

751 //Fokussierter Bereich mittels Linien

752 **d3**.select("#g\_linesArea").remove();

753 if (treeFin) {

754 **d3**.select("#treesvg").append('g').attr('id', 'g\_linesArea').selectAll('line').data(getFocusArea(pLeafId)).enter().append('line')

755 .attr('style', 'stroke:steelblue; stroke-width:3px')

756 .attr('stroke-dasharray', '3,3')

757 .attr('x1', function (d) {

758 return d.p1.x;

759 })

760 .attr('y1', function (d) {

761 return d.p1.y;

762 })

763 .attr('x2', function (d) {

764 return d.p2.x;

765 })

766 .attr('y2', function (d) {

767 return d.p2.y;

768 });

769 }

770 };

771 /\*

772 \* Zeichnet die Info Ausgabe in den Baum Bereich

773 \* **@param** {type} pText

774 \* **@returns** {undefined}

775 \*/

776 **drawInfoText** = function (pText) {

777 **d3**.select("#g\_labelInfo").remove();

778 **d3**.select("#treesvg").append('g').attr('id', 'g\_labelInfo').append('text')

779 .attr('x', 20)

780 .attr('y', 300)

781 .text(pText);

782 };

783 /\*

784 \* Zeichnet das Start Array welches zum Sortieren genutzt wird

785 \* **@returns** {undefined}

786 \*/

787 **drawStartArrayDiv** = function () {

788 var SortDiv = **d3**.select("#startArraydiv");

789 //Löscht die alten Elemente

790 SortDiv.select("#startArraySvg").select("#g\_circles2").remove();

791 SortDiv.select("#startArraySvg").select("#g\_labels2").remove();

792

793 //Kreise

794 SortDiv.select("#startArraySvg").append('g').attr('id', 'g\_circles2').selectAll('circle').data(getPosition(true)).enter().append('circle')

795 .attr('cx', function (d) {

796 return d.p.x;

797 })

798 .attr('cy', function (d) {

799 return d.p.y;

800 })

801 .attr('r', vRad);

802

803 //Nummerierung der Kreise

804 SortDiv.select("#startArraySvg").append('g').attr('id', 'g\_labels2').selectAll('text').data(getPosition(true)).enter().append('text')

805 .attr('x', function (d) {

806 return d.p.x;

807 })

808 .attr('y', function (d) {

809 return d.p.y + 5;

810 })

811 .text(function (d) {

812 return d.l;

813 });

814 };

815 /\*

816 \* Zeichnet das Ergebnis Array welches beim Sortieren entsteht

817 \* **@returns** {undefined}

818 \*/

819 **drawEndArrayDiv** = function () {

820 var SortDiv = **d3**.select("#endArraydiv");

821 //Löscht die alten Elemente

822 SortDiv.select("#endArraySvg").select("#g\_circles2").remove();

823 SortDiv.select("#endArraySvg").select("#g\_labels2").remove();

824

825 //Kreise

826 SortDiv.select("#endArraySvg").append('g').attr('id', 'g\_circles2').selectAll('circle').data(getPosition(false)).enter().append('circle')

827 .attr('cx', function (d) {

828 return d.p.x;

829 })

830 .attr('cy', function (d) {

831 return d.p.y;

832 })

833 .attr('r', vRad);

834

835 //Nummerierung der Kreise

836 SortDiv.select("#endArraySvg").append('g').attr('id', 'g\_labels2').selectAll('text').data(getPosition(false)).enter().append('text')

837 .attr('x', function (d) {

838 return d.p.x;

839 })

840 .attr('y', function (d) {

841 return d.p.y + 5;

842 })

843 .text(function (d) {

844 if (d.l !== -1)

845 return d.l;

846 else

847 return "";

848 });

849 };

850 /\*

851 \* Initialisiert die entsprechenden Bereiche der Ausgaben und Eingabe möglichkeiten der Buttons

852 \*/

853 **initHeapSort** = function () {

854 //Start Array Anzeige

855 **d3**.select(target).append("div").attr('id', 'startArraydiv').append("svg").attr("width", svgW + 10).attr("height", 40).attr('id', 'startArraySvg');

856

857 //Baum

858 **d3**.select(target).append("svg").attr("width", svgW + 10).attr("height", svgH).attr('id', 'treesvg');

859

860 //End Array Anzeige

861 **d3**.select(target).append("div").attr('id', 'endArraydiv').append("svg").attr("width", svgW + 10).attr("height", 40).attr('id', 'endArraySvg');

862

863

864 //Button

865 **d3**.select(target).append("div").attr('id', 'buttondiv');

866

867 **d3**.select("#buttondiv").append("button").attr('type', 'button').attr('class', 'button orange').attr('style', 'width:250px').text('Neuer Versuch')

868 .on('click', function (d) {

869 return initTree();

870 });

871 **d3**.select("#buttondiv").append("button").attr('type', 'button').attr('class', 'button orange').attr('style', 'width:250px; float:right').text('Baum Erstellen')

872 .on('click', function (d) {

873 return finAutoTree();

874 });

875 **d3**.select("#buttondiv").append("button").attr('type', 'button').attr('class', 'button orange').attr('style', 'width:250px; margin-top: 10px;').text('Heap Lösen')

876 .on('click', function (d) {

877 return finAutoHeap();

878 });

879 **d3**.select("#buttondiv").append("button").attr('type', 'button').attr('class', 'button orange').attr('style', 'width:250px; margin-top: 10px; float:right').text('Baum Lösen')

880 .on('click', function (d) {

881 return finAutoFullTree();

882 });

883

884 initTree();

885 };

886 initHeapSort();

887 return this;

888 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\animate\animate-randomsort.js |

1 /\*

2 Dateiname : animate-randomsort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 /\*

10 \* Erstellt ein Array welches alle Grundfarben enthält

11 \* die anschlissen Zufallig vertauscht werden

12 \* **@param** {int} pMax

13 \* **@returns** {creatRGB.lArray|Array}

14 \*/

15 function **creatRGB**(pMax) {

16 var lArray = [];

17 var lMode = 1;

18 var R, G, B;

19 R = 255;

20 G = 0;

21 B = 0;

22 var s = 1530.0 / pMax;

23 var i = -1;

24 do {

25 switch (lMode) {

26 case 1:

27 B += s;

28 if (B >= 255)

29 lMode++;

30 break;

31 case 2:

32 R -= s;

33 if (R <= 0)

34 lMode++;

35 break;

36 case 3:

37 G += s;

38 if (G >= 255)

39 lMode++;

40 break;

41 case 4:

42 B -= s;

43 if (B <= 0)

44 lMode++;

45 break;

46 case 5:

47 R += s;

48 if (R >= 255)

49 lMode++;

50 break;

51 case 6:

52 G -= s;

53 if (G <= 0)

54 lMode = 1;

55 break;

56 }

57 i++;

58 if (R < 0)

59 R = 0;

60 if (R > 255)

61 R = 255;

62 if (G < 0)

63 G = 0;

64 if (G > 255)

65 G = 255;

66 if (B < 0)

67 B = 0;

68 if (B > 255)

69 B = 255;

70 lArray.push({i: i, **v**: {r: R, g: G, b: B, color: "rgb(" + R + "," + G + "," + B + ")"}});

71 } while (i < pMax);

72

73 var i = lArray.length, j, v;

74 while (--i > 0) {

75 j = ~~(**Math**.random() \* (i + 1));

76 v = lArray[j].v;

77 lArray[j].v = lArray[i].v;

78 lArray[i].v = v;

79 }

80 return lArray;

81 }

82 /\*

83 \*

84 \* **@param** {type} pWidth

85 \* **@param** {type} pSteps

86 \* **@param** {type} pArray

87 \* **@returns** {SortingAnimationRandom.ret}

88 \*/

89 function **SortingAnimationRandom**(pWidth, pSteps, pArray) {

90 var **ret** = {};

91 var **margin** = {top: 10, right: 10, bottom: 10, left: 10};

92 var steps = pSteps;

93 var widthHeight = pWidth;

94 var z = (widthHeight) / steps;

95 var **srcData** = pArray;

96 var allActions = [];

97 var rects = [];

98 var timer\_off = true;

99

100 /\*

101 \* Kopier Funktion da Array.slice() nicht funktionier

102 \* wenn das Array Ojekte enthält

103 \* **@returns** {Array|SortingAnimationRandom.copyArray.lCopyArray}

104 \*/

105 function **copyArray**() {

106 var lCopyArray = [];

107 srcData.forEach(function (entry) {

108 lCopyArray.push({i: entry.i, **v**: {r: entry.v.r, g: entry.v.g, b: entry.v.b, color: entry.v.color}});

109 });

110 return lCopyArray;

111 }

112 /\*

113 \* Erzeugt aus dem Letzten stand des zu sortierenden Array eine neuen stand

114 \* sodass Random sort weiter sortiert wird bis unter 100 änderungen sind

115 \* dannn wird die Sortierung gestopt

116 \* **@param** {type} pObjekt

117 \* **@param** {type} delay

118 \* **@returns** {undefined}

119 \*/

120 function **startNew**(pObjekt, delay) {

121 var actions = pObjekt.sortingfunction(pObjekt.data, steps).reverse();

122 if (actions.length < 100)

123 pObjekt.stop = true;

124 else

125 pObjekt.actions = actions;

126 }

127 /\*

128 \* Leert das gesamte Target sodass es nicht zu fehlern kommt

129 \* **@param** {type} pTarget

130 \* **@returns** {Boolean}

131 \*/

132 function **clear**(pTarget) {

133 for (var i = 0; i < allActions.length; i++) {

134 var actionTarget = allActions[i];

135 if (pTarget === actionTarget.Target) {

136 allActions.splice(i, 1);

137 return true;

138 }

139 }

140 return false;

141 }

142

143 /\*

144 \* Startet die Ausgabe von Random Sort solange bis es gestopt wird

145 \* **@param** {type} delay

146 \* **@returns** {undefined}

147 \*/

148 ret.**start** = function (delay) {

149 timer\_off = false;

150 **d3**.timer(function () {

151 for (**l** = 0; **l** < 5; **l**++) {

152 for (var i = 0; i < allActions.length; i++) {

153 var action = allActions[i].actions.pop();

154 var rect = rects[i];

155 if (action)

156 {

157 switch (action.type) {

158 case "swap":

159 {

160 var t = rect[0][action.i];

161 rect[0][action.i] = rect[0][action.j];

162 rect[0][action.j] = t;

163 rect.attr("transform", function (d, i) {

164 return "translate(" + (i % steps) \* z + "," + **Math**.floor(i / steps) \* z + ")";

165 });

166 break;

167 }

168 }

169 }

170 }

171 }

172 for (var i = 0; i < allActions.length; i++) {

173 if (allActions[i].actions.length === 0 && allActions[i].stop === false) {

174 startNew(allActions[i]);

175 i--;

176 }

177 }

178 return timer\_off;

179 });

180 };

181 /\*

182 \* Resetet die Ausgabe bevor man die Function Start ausführt.

183 \* **@returns** {undefined}

184 \*/

185 ret.**reset** = function () {

186 allActions = [];

187 rects = [];

188 timer\_off = true;

189 };

190 /\*

191 \* Startet den Sortierung wenn sie ganz aus ist

192 \* **@returns** {undefined}

193 \*/

194 ret.**restart** = function () {

195 if (timer\_off)

196 ret.start(0);

197 };

198 /\*

199 \* Hizufügen einer Instanz für RandomSort

200 \*/

201 ret.**add** = function (sortingfunction, target) {

202 clear(target);

203 var data = copyArray();

204 var svg = **d3**.select(target).append("svg").attr("width", widthHeight + margin.left + margin.right).attr("height", widthHeight + margin.top + margin.bottom)

205 .append("g").attr("transform", "translate(" + margin.left + "," + margin.top + ")");

206 var rect = svg.selectAll("rect")

207 .data(data)

208 .enter().append("rect")

209 .attr("transform", function (d) {

210 var lX = **Math**.floor(widthHeight / z);

211 return "translate(" + (d.i % lX) \* z + "," + **Math**.floor(d.i / lX) \* z + ")";

212 })

213 .attr("width", z)

214 .attr("height", z)

215 .style("fill", function (d) {

216 return **d3**.rgb(d.v.color);

217 });

218 // sort the list, then reverse the stack of operations so we can animate chronologically from the start

219 var actions = sortingfunction(data, steps).reverse();

220

221 // push our actions and reference to our lines to the animator

222 allActions.push({

223 actions: actions,

224 sortingfunction: sortingfunction,

225 data: data,

226 target: target,

227 stop: false

228 });

229 rects.push(rect);

230 };

231 return ret;

232 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\animate\animate-sort-new.js |

1 /\*

2 Dateiname : animate-sort-new.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 /\*

10 \*

11 \* **@param** {int[]} pArray

12 \* Das zu sortierende Array

13 \*

14 \* **@param** {int} pMaxSteps

15 \* Anzahl der enthaltenden Werte

16 \*

17 \* **@returns** {SortingAnimationNew.ret}

18 \*/

19 function **SortingAnimationNew**(pArray, pMaxSteps) {

20 var **ret** = {};

21 var allActions = [];

22 var lines = [];

23 var infos = [];

24

25 var **srcData** = pArray;

26 var maxSteps = pMaxSteps;

27

28 var **margin** = {top: 10, right: 10, bottom: 10, left: 10};

29 var width = 800 - margin.left - margin.right;

30 var height = 50 - margin.top - margin.bottom;

31 var index = **d3**.range(maxSteps);

32 var x = **d3**.scale.ordinal().domain(index).rangePoints([0, width]);

33 var a = **d3**.scale.linear().domain([0, maxSteps - 1]).range([height / 5, height]);

34 var color = **d3**.scale.quantize().domain([height / 10, height])

35 .range(["rgb(255,245,235)", "rgb(254,230,206)", "rgb(253,208,162)", "rgb(253,174,107)", "rgb(253,141,60)", "rgb(241,105,19)", "rgb(217,72,1)", "rgb(166,54,3)", "rgb(127,39,4)"]);

36

37 /\*

38 \* Startet die Ausgabe der Sortierten Stacks

39 \* **@param** {type} delay

40 \* das delay gibt an wie lange gewartet werden soll bis es gestartet werden soll nach dem Start ausgeführt wurden ist

41 \*

42 \* **@returns** {undefined}

43 \*/

44 ret.**start** = function (delay) {

45 setTimeout(

46 function () {

47 setInterval(function () {

48 for (var i = 0; i < allActions.length; i++) {

49 var action = allActions[i].actions.pop();

50 var line = lines[i];

51 var info = infos[i];

52 if (action)

53 switch (action.type) {

54 case "partition":

55 {

56 info.style("stroke", function (d, i) {

57 return i == action.pivot ? "red" : null;

58 });

59 info.style("opacity", function (d, i) {

60 return i == action.pivot ? 1 : 0;

61 });

62 break;

63 }

64 case "swap":

65 {

66 var t = line[0][action.i];

67 line[0][action.i] = line[0][action.j];

68 line[0][action.j] = t;

69 line.attr("transform", function (d, i) {

70 return "translate(" + x(i) + ")";

71 });

72 line.style("stroke", function (d, i) {

73 return color(a(d));

74 });

75

76 info.style("opacity", function (d, i) {

77 return i == action.i || i == action.j ? 1 : 0;

78 });

79 info.style("stroke", function (d, i) {

80 return color(a(d));

81 });

82 break;

83 }

84 case "done":

85 {

86 info.style("opacity", function (d, i) {

87 return 0;

88 });

89 break;

90 }

91 }

92 }

93 }, 1);

94 }, delay);

95 };

96

97 /\*

98 \* Resetet die Ausgabe bevor man die Function Start ausführt.

99 \* **@returns** {undefined}

100 \*/

101 ret.**reset** = function () {

102 allActions = [];

103 lines = [];

104 infos = [];

105 };

106

107 /\*

108 \* Hinzufügen des zu Sortierenden Animation

109 \*

110 \* **@param** {function} sortingfunction

111 \* Sortierfunktion die das gesamte Array Sortiert

112 \* und die Einzelen Schritte als Stack zurück gibt

113 \*

114 \* **@param** {string} target

115 \* ID des **<section>** Tags "#target"

116 \*

117 \* **@returns** {undefined}

118 \*/

119 ret.**add** = function (sortingfunction, target) {

120 // Erzeugung einer Kopie des genutzten Arrays

121 var data = srcData.slice();

122

123 // Erstellung der SVG Zeichnung in HTML Tag <section> mit der ID der target Variable

124 var svg = **d3**.select(target).append("svg")

125 .attr("width", width + margin.left + margin.right)

126 .attr("height", height + margin.top + margin.bottom)

127 .append("g")

128 .attr("transform", "translate(" + margin.left + "," + (margin.top + height) + ")");

129

130 // Zeichnen aller Linien in der SVG Zeichnung

131 var line = svg.selectAll("line").data(data).enter().append("line")

132 .attr("index", function (d, i) {

133 return "i" + i;

134 })

135 .style("stroke", function (d) {

136 return color(a(d));

137 })

138 .attr("x2", function (d) {

139 return 0;

140 })

141 .attr("y2", function (d) {

142 return -a(d);

143 })

144 .attr("transform", function (d, i) {

145 return "translate(" + x(i) + ")";

146 });

147

148 // Zeichnen aller Info Linien in der SVG Zeichnung

149 var info = svg.selectAll("g").data(data).enter().append("svg:g").append("svg:line")

150 .attr("x1", function (d) {

151 return 0;

152 })

153 .attr("y1", function (d) {

154 return -height - 10;

155 })

156 .attr("x2", function (d) {

157 return 0;

158 })

159 .attr("y2", function (d) {

160 return -height - 5;

161 })

162 .style("stroke", function (d) {

163 return "pink";

164 })

165 .style("opacity", function (d) {

166 return 0;

167 })

168 .attr("transform", function (d, i) {

169 return "translate(" + x(i) + ")";

170 });

171

172 //Sortierung des Arrays und Speicherung des Stacks in der Variable actions, sodass die Sortierung animiert werden kann

173 var actions = sortingfunction(data).reverse();

174 allActions.push({actions: actions});

175 lines.push(line);

176 infos.push(info);

177 };

178 return ret;

179 }

180

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\animate\animate-sort-old.js |

1 /\*

2 Dateiname : animate-sort-old.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 //Sortierte Animation eines Array auf der Sich von X und Y Achse

10 /\*

11 \*

12 \* **@param** {int[]} pArray

13 \* Das zu sortierende Array

14 \*

15 \* **@param** {int} pMaxSteps

16 \* Anzahl der enthaltenden Werte

17 \*

18 \* **@returns** {SortingAnimationNew.ret}

19 \*/

20 function **SortingAnimationOld**(pArray, pMaxSteps) {

21 //Variablen

22 var **ret** = {};

23 var allActions = [];

24 var rects = [];

25

26 var maxSteps = pMaxSteps;

27 var **srcData** = pArray;

28

29 var **margin** = {top: 10, right: 10, bottom: 10, left: 10};

30 var widthHeight = 200 - margin.left - margin.right;

31 var z = (widthHeight - margin.left - margin.right) / maxSteps;

32

33 /\*

34 \* Startet die Ausgabe der Sortierten Stacks

35 \* **@param** {type} delay

36 \* das delay gibt an wie lange gewartet werden soll bis es gestartet werden soll nach dem Start ausgeführt wurden ist

37 \*

38 \* **@returns** {undefined}

39 \*/

40 ret.**start** = function (delay) {

41 setTimeout(

42 function () {

43 setInterval(function () {

44 for (var i = 0; i < allActions.length; i++) {

45 var action = allActions[i].actions.pop();

46 var rect = rects[i];

47 if (action)

48 {

49 switch (action.type) {

50 case "swap":

51 {

52 var t = rect[0][action.i];

53 rect[0][action.i] = rect[0][action.j];

54 rect[0][action.j] = t;

55 rect.attr("transform", function (d, i) {

56 return "translate(" + (i \* z) + "," + ((maxSteps - d) \* z) + ")";

57 });

58 break;

59 }

60 }

61 }

62 }

63 }, 1);

64 }, delay);

65 };

66 /\*

67 \* Resetet die Ausgabe bevor man die Function Start ausführt.

68 \* **@returns** {undefined}

69 \*/

70 ret.**reset** = function () {

71 allActions = [];

72 rects = [];

73 };

74

75 /\*

76 \* Hinzufügen des zu Sortierenden Animation

77 \*

78 \* sortingfunction

79 \* Sortierfunktion die das gesamte Array Sortiert

80 \* und die Einzelen Schritte als Stack zurück gibt

81 \*

82 \* target (String) "#target"

83 \* ID des **<section>** Tags

84 \*/

85 ret.**add** = function (sortingfunction, target) {

86 //Erzeugung einer Kopie des genutzten Arrays

87 var data = srcData.slice();

88

89 //Erstellung der SVG Zeichnung in HTML Tag <section> mit der ID der target Variable

90 var svg = **d3**.select(target).append("svg")

91 .attr("width", widthHeight + margin.left + margin.right)

92 .attr("height", widthHeight + margin.top + margin.bottom)

93 .append("g")

94 .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

95

96 //Zeichnen aller Rechtecke in der SVG Zeichnung

97 var rect = svg.selectAll("rect")

98 .data(data)

99 .enter().append("rect")

100 .attr("transform", function (d, i) {

101 return "translate(" + (i \* z) + "," + ((maxSteps - d) \* z) + ")";

102 })

103 .attr("width", z)

104 .attr("height", z)

105 .style("fill", "black");

106 //Sortierung des Arrays und Speicherung des Stacks in der Variable actions, sodass die Sortierung animiert werden kann

107 var actions = sortingfunction(data, maxSteps).reverse();

108 allActions.push({actions: actions});

109 rects.push(rect);

110 };

111 return ret;

112 }

113

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\animate\animate-usersort.js |

1 /\*

2 Dateiname : animate-usersort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9

10 function **UserSorting**(sortingfunction, target) {

11 this.**SortingFunction** = sortingfunction;

12 this.**Target** = target;

13

14 var svgW = 598, svgH = 100, vRad = 18;

15

16

17 var maxArray = 8;

18 var \_SteepFin = false;

19 this.**SortInfo** = {

20 "**Array**": [],

21 "**ArrayUp**": [],

22 "**Left**": -1,

23 "**Right**": -1,

24 "**UpLeft**": -1,

25 "**UpRight**": -1,

26 "**click**": -1,

27 "**Mode**": 0,

28 "**Update**": true

29 };

30 var \_Steep;

31

32 this.**startNew** = function () {

33 this.createNewArray();

34 this.SortingFunction.init(this.SortingFunction);

35 \_SteepFin = false;

36 if (this.SortingFunction.LeftStart == -2) {

37 this.SortInfo.**Left** = Math.floor((this.SortInfo.**Array**.length / 2) - 1);

38 this.SortInfo.**Right** = this.SortInfo.**Array**.length;

39 }

40 else {

41 this.SortInfo.**Left** = this.SortingFunction.LeftStart;

42 }

43 this.SortInfo.**Right** = this.SortingFunction.RightStart;

44 this.SortInfo.**click** = -1;

45 this.SortInfo.**UpLeft** = -1;

46 this.SortInfo.**UpRight** = -1;

47 this.SortInfo.**Mode** = 0;

48 this.SortInfo.**Update** = true;

49 \_Steep = 0;

50 updateAll(this);

51 };

52 this.**createNewArray** = function ()

53 {

54 this.SortInfo.**Array** = [];

55 this.SortInfo.**ArrayUp** = [];

56 var lMax = 8;

57 for (var i = 0; i < lMax; i++) {

58 this.SortInfo.**Array**.push(Math.floor((Math.random() \* lMax) + 1));

59 this.SortInfo.**ArrayUp**.push(-1);

60 }

61 };

62

63 this.**clickArray** = function (Id) {

64 this.SortInfo.**click** = Id;

65 this.SortInfo = this.SortingFunction.onClick(this.SortInfo, true);

66 if (this.SortInfo.**Update**)

67 updateAll(this);

68 else

69 redraw(this);

70 };

71 this.**clickArrayUp** = function (Id) {

72 this.SortInfo.**click** = Id;

73 this.SortInfo = this.SortingFunction.onClick(this.SortInfo, false);

74 if (this.SortInfo.**Update**)

75 updateAll(this);

76 else

77 redraw(this);

78 };

79

80

81 this.**getPosition** = function (all) {

82

83 var lArray = this.SortInfo.**Array**;

84 var lSteep = svgW / lArray.length;

85 var lPositions = [];

86 for (var i = 0; i < lArray.length; i++) {

87 if (lArray[i] !== -1 || all)

88 lPositions.push({**v**: i, **l**: lArray[i], **up**: 0, **p**: {**x**: lSteep \* i + 30, **y**: 60}, **p2**: {**x**: lSteep \* i + 30, **y**: 60}});

89 }

90 lArray = this.SortInfo.**ArrayUp**;

91 for (var i = 0; i < lArray.length; i++) {

92 if (lArray[i] !== -1)

93 lPositions.push({**v**: i, **l**: lArray[i], **up**: 1, **p**: {**x**: lSteep \* i + 30, **y**: 20}, **p2**: {**x**: lSteep \* i + 30, **y**: 20}});

94 }

95

96 return lPositions;

97 };

98

99 **updateAll** = function (pThis)

100 {

101 var SortDiv = d3.select("#usersort" + pThis.Target);

102

103 //Schreibt den nächsten Schritt für den User

104 SortDiv.select("#info").select('p').text(pThis.SortingFunction.updateMode(pThis.SortInfo.Mode));

105

106 //Löscht die alten Elemente

107 SortDiv.select("#sortsvg").select("#g\_circles").remove();

108 SortDiv.select("#sortsvg").select("#g\_labels").remove();

109

110 //Kreise

111 SortDiv.select("#sortsvg").append('g').attr('id', 'g\_circles').selectAll('circle').data(pThis.getPosition(true)).enter().append('circle').attr('cx', function (d) {

112 return d.p.x;

113 }).attr('cy', function (d) {

114 return d.p.y;

115 }).attr('r', vRad)

116 .on('click', function (d) {

117 if (d.up === 0)

118 return pThis.clickArray(d.v);

119 else

120 return pThis.clickArrayUp(d.v);

121 }).attr('style', function (d) {

122

123 if (d.v === pThis.SortInfo.Left && pThis.SortInfo.Left >= 0 && d.up === 0)

124 return 'stroke:green;stroke-width:4px;';

125 else if (d.v === pThis.SortInfo.Right && pThis.SortInfo.Right >= 0 && d.up === 0)

126 return 'stroke:red;stroke-width:4px;';

127 else if (d.v === pThis.SortInfo.UpLeft && pThis.SortInfo.UpLeft >= 0 && d.up === 1)

128 return 'stroke:green;stroke-width:4px;';

129 else if (d.v === pThis.SortInfo.UpRight && pThis.SortInfo.UpRight >= 0 && d.up === 1)

130 return 'stroke:red;stroke-width:4px;';

131 else

132 return 'steelblue;stroke-width:2px;';

133

134 });

135

136 //Nummerierung der Kreise

137 SortDiv.select("#sortsvg").append('g').attr('id', 'g\_labels').selectAll('text').data(pThis.getPosition(false)).enter().append('text').attr('x', function (d) {

138 return d.p.x;

139 }).attr('y', function (d) {

140 return d.p.y + 5;

141 }).text(function (d) {

142 return d.l;

143 }).on('click', function (d) {

144 if (d.up === 0)

145 return pThis.clickArray(d.v);

146 else

147 return pThis.clickArrayUp(d.v);

148 });

149 };

150 **redraw** = function (pThis) {

151 var SortDiv = d3.select("#usersort" + pThis.Target);

152 var circles = SortDiv.select("#sortsvg").select("#g\_circles").selectAll('circle').data(pThis.getPosition(true));

153

154 circles.transition().duration(500).attr('cx', function (d) {

155 return d.p.x;

156 }).attr('cy', function (d) {

157 return d.p.y;

158 }).attr('style', function (d) {

159

160 if (d.v === pThis.SortInfo.Left && pThis.SortInfo.Left >= 0 && d.up === 0)

161 return 'stroke:green;stroke-width:4px;';

162 else if (d.v === pThis.SortInfo.Right && pThis.SortInfo.Right >= 0 && d.up === 0)

163 return 'stroke:red;stroke-width:4px;';

164 else if (d.v === pThis.SortInfo.UpLeft && pThis.SortInfo.UpLeft >= 0 && d.up === 1)

165 return 'stroke:green;stroke-width:4px;';

166 else if (d.v === pThis.SortInfo.UpRight && pThis.SortInfo.UpRight >= 0 && d.up === 1)

167 return 'stroke:red;stroke-width:4px;';

168 else

169 return 'steelblue;stroke-width:2px;';

170

171 });

172

173 circles.enter().append('circle').attr('cx', function (d) {

174 return d.p2.x;

175 }).attr('cy', function (d) {

176 return d.p2.y;

177 }).attr('r', vRad).on('click', function (d) {

178 if (d.up === 0)

179 return pThis.clickArray(d.v);

180 else

181 return pThis.clickArrayUp(d.v);

182 }).transition().duration(500).attr('cx', function (d) {

183 return d.p.x;

184 }).attr('cy', function (d) {

185 return d.p.y;

186 }).attr('style', function (d) {

187

188 if (d.v === pThis.SortInfo.Left && pThis.SortInfo.Left >= 0 && d.up === 0)

189 return 'stroke:green;stroke-width:4px;';

190 else if (d.v === pThis.SortInfo.Right && pThis.SortInfo.Right >= 0 && d.up === 0)

191 return 'stroke:red;stroke-width:4px;';

192 else if (d.v === pThis.SortInfo.UpLeft && pThis.SortInfo.UpLeft >= 0 && d.up === 1)

193 return 'stroke:green;stroke-width:4px;';

194 else if (d.v === pThis.SortInfo.UpRight && pThis.SortInfo.UpRight >= 0 && d.up === 1)

195 return 'stroke:red;stroke-width:4px;';

196 else

197 return 'steelblue;stroke-width:2px;';

198

199 });

200

201 var labels = SortDiv.select("#sortsvg").select("#g\_labels").selectAll('text').data(pThis.getPosition(false));

202 labels.text(function (d) {

203 return d.l;

204 }).transition().duration(500).attr('x', function (d) {

205 return d.p.x;

206 }).attr('y', function (d) {

207 return d.p.y + 5;

208 });

209

210 labels.enter().append('text').attr('x', function (d) {

211 return d.p2.x;

212 }).attr('y', function (d) {

213 return d.p2.y + 5;

214 }).text(function (d) {

215 return d.l;

216 }).on('click', function (d) {

217 if (d.up === 0)

218 return pThis.clickArray(d.v);

219 else

220 return pThis.clickArrayUp(d.v);

221 }).transition().duration(500).attr('x', function (d) {

222 return d.p.x;

223 }).attr('y', function (d) {

224 return d.p.y + 5;

225 });

226 };

227

228 **initialize** = function (pThis) {

229 d3.select("#" + pThis.Target).append("div").attr('id', 'usersort' + pThis.Target);

230

231 d3.select("#usersort" + pThis.Target).append("svg").attr("width", svgW).attr("height", svgH).attr('id', 'sortsvg').attr('class', 'mybuttondiv');

232 d3.select("#usersort" + pThis.Target).append("div").attr('id', 'info').append("p").attr('class', 'mybuttondiv');

233

234 d3.select("#usersort" + pThis.Target).append("div").attr('id', 'buttondiv' + pThis.Target).attr('class', 'mybuttondiv');

235 d3.select("#buttondiv" + pThis.Target).append("button").attr('type', 'button').attr('class', 'button orange').text('Neuer Versuch').on('click', function (d) {

236 return pThis.startNew();

237 });

238 pThis.startNew();

239 };

240 initialize(this);

241 return this;

242 }

243

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\bitonicsort.js |

1 /\*

2 Dateiname : bitonicsort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny L�ttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **BitonicSorter**() {

10 var **\_Actions**, **\_Array**, ASCENDING = true, DESCENDING = false;

11

12 /\*

13 \* Startet die Bitonic Sortierung

14 \*/

15 this.**sortAll** = function (array) {

16 \_Actions = [];

17 \_Array = array.slice();

18 bitonicSort(0, \_Array.length, ASCENDING);

19 \_Actions.push({**type**: "done", "**done**": 0});

20 return \_Actions;

21 };

22

23 /\*

24 \* Tauscht den Inhalt der 2 übergebenen Parameter und merkt sich die Aktion zur späteren Darstellung

25 \*/

26 function **swap**(i, j) {

27 var t = \_Array[i];

28 \_Array[i] = \_Array[j];

29 \_Array[j] = t;

30 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

31 }

32

33 /\*

34 \* Implementiert Bitonic Sort

35 \*/

36 function **bitonicSort**(lo, n, dir) {

37 if (n > 1) {

38 var m = n >>1 ;

39 bitonicSort(lo, m, !dir);

40 bitonicSort(lo + m, n - m, dir);

41 bitonicMerge(lo, n, dir);

42 }

43 }

44

45 /\*

46 \* Implementiert die Merge Funktion für Bitonic Sort

47 \*/

48 function **bitonicMerge**(lo, n, dir) {

49 if (n > 1) {

50 var m = greatestPowerOfTwoLessThan(n);

51 for (var i = lo; i < lo + n-m; i++)

52 compare(i, i + m, dir);

53 bitonicMerge(lo, m, dir);

54 bitonicMerge(lo + m, n - m, dir);

55 }

56 }

57

58 /\*

59 \* Vergleicht die Argumente und tauscht wenn die Reihenfolge korrekt ist

60 \*/

61 function **compare**(i, j, dir) {

62 if (dir === (\_Array[i] > \_Array[j]))

63 swap(i, j);

64 }

65

66 /\*

67 \* Berechnet den gr��ten Faktor von 2 kleiner n

68 \*/

69 function **greatestPowerOfTwoLessThan**(n)

70 {

71 var k = 1;

72 while(k < n)

73 k = k << 1;

74 return k >> 1;

75 }

76

77 /\*

78 \* initialisiert ben�tigte Membervariablen für Usersort

79 \*/

80 this.**init** = function (pTarget) {

81 };

82

83 return this;

84 }

85

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\bogosort.js |

1 /\*

2 Dateiname : bogosort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **Bogosort**() {

10 var **\_Actions**, **\_Array**, n;

11

12 /\*

13 \* Startet die Bogosort Sortierung

14 \*/

15 this.**sortAll** = function (array) {

16 \_Actions = [];

17 \_Array = array.slice();

18 n = \_Array.length;

19 start();

20 \_Actions.push({**type**: "done", "**done**": 0});

21 return \_Actions;

22 };

23

24 /\*

25 \* F�llt das Array mit Zufallszahlen

26 \*/

27 function **shuffle**() {

28 var i = \_Array.length, j, t;

29 while (--i > 0) {

30 j = (2\*(Math.random() \* (i + 1)))>>1;

31 t = \_Array[j];

32 \_Array[j] = \_Array[i];

33 \_Array[i] = t;

34 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

35 }

36 return \_Array;

37 }

38

39 /\*

40 \* Kontrolliert ob ein Array korrekt sortiert ist

41 \*/

42 function **inOrder**() {

43 for (var i = 1; i < \_Array.length; i++) {

44 if (\_Array[i] < \_Array[i - 1]) {

45 return false;

46 }

47 }

48 return true;

49 }

50

51 /\*

52 \* Implementiert Bogosort

53 \*/

54 function **start**() {

55 var tries = 0;

56 while (!inOrder()) {

57 if (tries > 100)

58 return;

59 tries++;

60 shuffle();

61 }

62 }

63

64 /\*

65 \* initialisiert benötigte Membervariablen für Usersort

66 \*/

67 this.**init** = function (pTarget) {

68 };

69

70 return this;

71 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\bubblesort.js |

1 /\*

2 Dateiname : bubblesort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **BubbleSort**() {

10 /\*\*

11 \* Visual Sort

12 \*/

13

14 var **\_Actions**, **\_Array**;

15

16 /\*

17 \* Startet die BubbleSort Sortierung

18 \*/

19 this.**sortAll** = function (array) {

20 \_Actions = [];

21 \_Array = array.slice();

22 bubblesort();

23 \_Actions.push({**type**: "done", "**done**": 0});

24 return \_Actions;

25 };

26

27 /\*

28 \* Tauscht den Inhalt der 2 übergebenen Parameter und merkt sich die Aktion zur späteren Darstellung

29 \*/

30 function **swap**(i, j) {

31 var t = \_Array[i];

32 \_Array[i] = \_Array[j];

33 \_Array[j] = t;

34 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

35 }

36

37 /\*

38 \* Implementierung von Bubblesort

39 \*/

40 function **bubblesort**() {

41 var swapped = true;

42 var i = 0;

43 while (swapped) {

44 swapped = false;

45 for (i = 0; i < \_Array.length - 1; i++) {

46 if (\_Array[i] > \_Array[i + 1]) {

47 swap(i, i + 1);

48 swapped = true;

49 }

50 }

51 }

52 }

53

54 /\*\*

55 \* User Sort

56 \*/

57

58 this.**LeftStart** = 0;

59 this.**RightStart** = 1;

60

61 /\*

62 \* initialisiert benötigte Membervariablen für Usersort

63 \*/

64 this.**init** = function (pTarget) {

65 pTarget.**LeftStart** = 0;

66 pTarget.**RightStart** = 1;

67 };

68

69 /\*

70 \* gibt die Textbeschreibung für den aktuellen Schritt zurück

71 \*/

72 this.**updateMode** = function (pMode) {

73 switch (pMode) {

74 case 0:

75 return "Wenn das 1. Vergleichselement (Grün) größer ist als das 2. Vergleichselement (Rot) dann klicke Grün ansonsten Rot";

76 case 3:

77 return "Das Array ist sortiert";

78 }

79 };

80

81 /\*

82 \* Verarbeitet die onClick Events entsprechend des Implementierten Algorithmus

83 \*/

84 this.**onClick** = function (pSortInfo, pOriginal) {

85 switch (pSortInfo.**Mode**) {

86 case 0:

87 // Wenn auf grün geklickt wurde

88 if (pSortInfo.click === pSortInfo.**Left**) {

89 // und der Linke Wert größer als der Rechte ist

90 if (pSortInfo.Array[pSortInfo.**Left**] > pSortInfo.Array[pSortInfo.**Right**]) {

91 // tausche die beiden Werte

92 var t = pSortInfo.Array[pSortInfo.**Left**];

93 pSortInfo.Array[pSortInfo.**Left**] = pSortInfo.Array[pSortInfo.**Right**];

94 pSortInfo.Array[pSortInfo.**Right**] = t;

95 // und verschiebe die beiden Markierungen um jeweils 1 nach Rechts

96 pSortInfo.**Right**++;

97 pSortInfo.**Left**++;

98 // und schreibe in Up das etwas sortiert wurde

99 pSortInfo.**Up** = -1;

100 }

101 }

102 // Wenn auf rot geklickt wurde

103 else if (pSortInfo.click === pSortInfo.**Right**) {

104 // Und der linke Wert kleiner gleich dem Rechten ist

105 if (pSortInfo.Array[pSortInfo.**Left**] <= pSortInfo.Array[pSortInfo.**Right**]) {

106 // verschiebe die beiden Markierungen um jeweils 1 nach Rechts

107 pSortInfo.**Right**++;

108 pSortInfo.**Left**++;

109 }

110 }

111 // Wenn wir mit dem roten Element ganz Rechts angekommen sind

112 if (pSortInfo.**Right** === pSortInfo.Array.length) {

113 // Und nicht getauscht wurde

114 if (pSortInfo.**Up** === -2) {

115 // Wechsel zum nächsten Modus

116 pSortInfo.**Left** = -1;

117 pSortInfo.**Right** = -1;

118 pSortInfo.**Mode** = 3;

119 }

120 else {

121 // grün ganz nach Links

122 // rot rechts daneben

123 pSortInfo.**Left** = 0;

124 pSortInfo.**Right** = 1;

125 // es wurde getauscht, also noch einmal durchgehen

126 pSortInfo.**Up** = -2;

127 }

128 }

129 break;

130 }

131 return pSortInfo;

132 };

133 return this;

134 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\heapsort.js |

1 /\*

2 Dateiname : heapsort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **Heapsort**() {

10 var **\_Actions**, **\_Array**, n;

11

12 /\*

13 \* Startet die Heapsort Sortierung

14 \*/

15 this.**sortAll** = function (array) {

16 \_Actions = [];

17 \_Array = array.slice();

18 n = \_Array.length;

19 start();

20 \_Actions.push({**type**: "done", "**done**": 0});

21 return \_Actions;

22 };

23

24 /\*

25 \* Tauscht den Inhalt der 2 übergebenen Parameter und merkt sich die Aktion zur späteren Darstellung

26 \*/

27 function **swap**(i, j) {

28 var t = \_Array[i];

29 \_Array[i] = \_Array[j];

30 \_Array[j] = t;

31 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

32 }

33

34 /\*

35 \* Implementiert Heapsort

36 \*/

37 function **start**() {

38 buildheap();

39 while (n > 1) {

40 n--;

41 swap(0, n);

42 downheap(0);

43 }

44 }

45

46 /\*

47 \* buildheap und downheap erstellen den Heap

48 \*/

49 function **buildheap**() {

50 for (var v = (n >>1 ) - 1; v >= 0; v--) {

51 downheap(v);

52 }

53 }

54

55 function **downheap**(v) {

56 var w = 2 \* v + 1; // erster Nachfolger von v

57 while (w < n) {

58 if (w + 1 < n) // gibt es einen zweiten Nachfolger?

59 if (\_Array[w + 1] > \_Array[w])

60 w++;

61 // w ist der Nachfolger von v mit maximaler Markierung

62 if (\_Array[v] >= \_Array[w])

63 return; // v hat die Heap-Eigenschaft

64 // sonst

65 swap(v, w); // vertausche Markierungen von v und w

66 v = w; // fahre mit v=w fort

67 w = 2 \* v + 1;

68 }

69 }

70

71 /\*

72 \* initialisiert ben�tigte Membervariablen für Usersort

73 \*/

74 this.**init** = function (pTarget) {

75 };

76

77 return this;

78 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\insertionsort.js |

1 /\*

2 Dateiname : insertionsort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **InsertionSort**() {

10 /\*\*

11 \* Visual Sort

12 \*/

13

14 var **\_Actions**, **\_Array**;

15

16 /\*

17 \* Startet die Insertion Sortierung

18 \*/

19 this.**sortAll** = function (array) {

20 \_Actions = [];

21 \_Array = array.slice();

22 insertionsort();

23 \_Actions.push({**type**: "done", "**done**": 0});

24 return \_Actions;

25 };

26

27 /\*

28 \* Tauscht den Inhalt der 2 übergebenen Parameter und merkt sich die Aktion zur späteren Darstellung

29 \*/

30 function **swap**(i, j) {

31 var t = \_Array[i];

32 \_Array[i] = \_Array[j];

33 \_Array[j] = t;

34 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

35 }

36

37 /\*

38 \* Implementierung von Insertionsort

39 \*/

40 function **insertionsort**() {

41 var i, j;

42 for (i = 1; i < \_Array.length; i++) {

43 j = i;

44 while (j > 0 && \_Array[j - 1] > \_Array[j]) {

45 swap(j, j - 1);

46 j--;

47 }

48 }

49 }

50

51 /\*

52 \* User Sort

53 \*/

54 this.**LeftStart** = 0;

55 this.**RightStart** = 1;

56

57 /\*

58 \* initialisiert benötigte Membervariablen für Usersort

59 \*/

60 this.**init** = function (pTarget) {

61 pTarget.**LeftStart** = 0;

62 pTarget.**RightStart** = 1;

63 };

64

65 /\*

66 \* gibt die Textbeschreibung für den aktuellen Schritt zurück

67 \*/

68 this.**updateMode** = function (pMode) {

69 switch (pMode) {

70 case 0:

71 return "1. Vergleichselement(Grün) anklicken";

72 case 1:

73 return "Grün Fokussieren";

74 case 2:

75 return "Ist das Linke Element größer als Grün? Wenn ja anklicken ansonsten auf den Leeren Kreis klicken.";

76 case 3:

77 return "Das Array ist sortiert.";

78 }

79 };

80

81 /\*

82 \* Verarbeitet die onClick Events entsprechend des Implementierten Algorithmus

83 \*/

84 this.**onClick** = function (pSortInfo,pOriginal) {

85 switch (pSortInfo.**Mode**) {

86 case 0:

87 // Wenn auf grün geklickt wurde

88 if (pSortInfo.click === pSortInfo.**Left**) {

89 // Gehe zur aktuellen Position

90 pSortInfo.**Left** = pSortInfo.**Right**;

91 // Und wechsel in den Modus

92 pSortInfo.**Mode**++;

93 }

94 break;

95 case 1:

96 // Wenn auf grün und rot geklickt wurde

97 if (pSortInfo.click === pSortInfo.**Left** && pSortInfo.click === pSortInfo.**Right**) {

98 // Verschiebe den aktuellen in das obere Array

99 pSortInfo.ArrayUp[pSortInfo.click] = pSortInfo.Array[pSortInfo.click];

100 // den unteren nicht mehr Anzeigen

101 pSortInfo.Array[pSortInfo.click] = -1;

102

103 // den oberen Wert auch grün Zeichnen und den Modus wechseln

104 pSortInfo.**UpLeft** = pSortInfo.**Left**;

105 pSortInfo.**Mode**++;

106 pSortInfo.**Update** = true;

107 }

108 break;

109 case 2:

110 // Wenn links von den grünen Kreis geklickt wurde

111 if (pSortInfo.**Left** - 1 === pSortInfo.click) {

112 // Und der geklickte Wert größer als das Array ist

113 if (pSortInfo.Array[pSortInfo.click] > pSortInfo.ArrayUp[pSortInfo.**Left**]) {

114 // dann schiebe den oberen Wert zur geklickten Position

115 pSortInfo.ArrayUp[pSortInfo.click] = pSortInfo.ArrayUp[pSortInfo.**Left**];

116 pSortInfo.ArrayUp[pSortInfo.**Left**] = -1;

117

118 // und den unteren zur vorherigen Position

119 pSortInfo.Array[pSortInfo.**Left**] = pSortInfo.Array[pSortInfo.click];

120 pSortInfo.Array[pSortInfo.click] = -1;

121 // den grünen Kreis um eins nach links verschieben

122 pSortInfo.**Left**--;

123 // Im oberen Array den grünen Kreis auf die gleiche Position wie den unteren Kreis schieben

124 pSortInfo.**UpLeft** = pSortInfo.**Left**;

125 pSortInfo.**Update** = false;

126 }

127 }

128 // Wenn auf den grünen Kreis geklickt wurde

129 else if (pSortInfo.**Left** === pSortInfo.click) {

130 var Set = false;

131 // und wir bei der ersten Position sind

132 if (pSortInfo.click === 0)

133 Set = true;

134 // oder links von uns ein kleinerer Wert ist

135 else if (pSortInfo.Array[pSortInfo.click - 1] <= pSortInfo.ArrayUp[pSortInfo.**Left**])

136 Set = true;

137 if (Set) {

138 // Verschiebe den oberen grünen Wert wieder in das untere Array

139 pSortInfo.Array[pSortInfo.click] = pSortInfo.ArrayUp[pSortInfo.**Left**];

140 // die obere Zahl nicht mehr anzeigen

141 pSortInfo.ArrayUp[pSortInfo.**Left**] = -1;

142 // den roten Kreis um einen nach Rechts verschieben

143 pSortInfo.**Right**++;

144 // den oberen grünen Kreis nicht mehr anzeigen

145 pSortInfo.**UpLeft** = -1;

146 if (pSortInfo.**Right** === pSortInfo.Array.length) {

147 // wenn der rote Kreis am Array Ende angekommen ist, ist das Array sortiert

148 // und es wird der Modus gewechselt

149 pSortInfo.**Left** = -1;

150 pSortInfo.**Right** = -1;

151 pSortInfo.**Mode** = 3;

152 }

153 else

154 // ansonsten wechsel zum ersten Modus

155 pSortInfo.**Mode** = 0;

156 }

157 pSortInfo.**Update** = true;

158 }

159 break;

160 }

161 return pSortInfo;

162 };

163 return this;

164 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\mergesort.js |

1 /\*

2 Dateiname : mergesort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **MergeSorter**() {

10 /\*\*

11 \* Visual Sort

12 \*/

13 var **\_Actions**, **\_ArrayA**, \_ArrayB;

14

15 /\*

16 \* Startet die Mergesort Sortierung

17 \*/

18 this.**sortAll** = function (array) {

19 \_Actions = [];

20 \_ArrayA = array.slice();

21 \_ArrayB = array.slice();

22 mergesort(0, \_ArrayA.length - 1);

23 \_Actions.push({**type**: "done", "**done**": 0});

24 return \_Actions;

25 };

26

27 /\*

28 \* Implementierung von Mergesort

29 \*/

30 function **mergesort**(lo, hi) {

31 if (lo < hi) {

32 var m = (lo + hi)>>1 ;

33 if (m >= 1) {

34 mergesort(lo, m);

35 mergesort(m + 1, hi);

36 }

37 var lArrayOld = \_ArrayA.slice();

38 merge(lo, m, hi);

39 var lArrayNew = \_ArrayA.slice();

40 findChanges(lArrayOld, lArrayNew);

41 }

42 }

43

44

45 // einfache Merge-Variante a

46 function **merge**(lo, m, hi) {

47 var i, j, k;

48 // beide H�lften von a in Hilfsarray b kopieren

49 for (i = lo; i <= hi; i++)

50 \_ArrayB[i] = \_ArrayA[i];

51

52 i = lo;

53 j = m + 1;

54 k = lo;

55 // jeweils das n�chstgr��te Element zur�ckkopieren

56 while (i <= m && j <= hi) {

57 if (\_ArrayB[i] <= \_ArrayB[j])

58 \_ArrayA[k++] = \_ArrayB[i++];

59 else

60 \_ArrayA[k++] = \_ArrayB[j++];

61 }

62 // Rest der vorderen H�lfte falls vorhanden zur�ckkopieren

63 while (i <= m)

64 \_ArrayA[k++] = \_ArrayB[i++];

65 }

66 function **findChanges**(arrayOld, arrayNew) {

67 for (var i = 0; i < arrayNew.length; i++) {

68 for (var j = 0; j < arrayOld.length; j++) {

69 if (arrayNew[i] === arrayOld[j]) {

70 if (i !== j) {

71 var t = arrayOld[i];

72 arrayOld[i] = arrayOld[j];

73 arrayOld[j] = t;

74 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

75 }

76 break;

77 }

78 }

79 }

80 }

81

82 /\*\*

83 \* User Sort

84 \*/

85

86 this.**LeftStart** = 0;

87 this.**RightStart** = -1;

88

89 this.**GlobalPosition** = 0;

90 this.**HelpArraySize** = 1;

91 this.**ArrayLeftPosition** = 0;

92 this.**ArrayRightPosition** = 0;

93

94 /\*

95 \* initialisiert benötigte Membervariablen für Usersort

96 \*/

97 this.**init** = function (pTarget) {

98 pTarget.**LeftStart** = 0;

99 pTarget.**RightStart** = -1;

100

101 pTarget.**GlobalPosition** = 0;

102 pTarget.**HelpArraySize** = 1;

103 pTarget.**ArrayLeftPosition** = 0;

104 pTarget.**ArrayRightPosition** = 0;

105 };

106

107 /\*

108 \* gibt die Textbeschreibung für den aktuellen Schritt zurück

109 \*/

110 this.**updateMode** = function (pMode) {

111 switch (pMode) {

112 case 0:

113 return "Grün anklicken um Array ins Hilfsarray zu kopieren";

114 case 1:

115 return "Merge Größe " + this.**HelpArraySize** + ", Array zusammen kopieren, dafür den größeren der Markierten Werte anklicken";

116 case 2:

117 return "Hilfs Array löschen";

118 case 3:

119 return "Das Array ist sortiert";

120 }

121 };

122

123 /\*

124 \* Verarbeitet die onClick Events entsprechend des Implementierten Algorithmus

125 \*/

126 this.**onClick** = function (pSortInfo, pOriginal) {

127 switch (pSortInfo.**Mode**) {

128 case 0:

129 // Wenn auf grün geklickt wurde

130 if (pSortInfo.click === pSortInfo.**Left**) {

131 // verschiebe das untere Array in das obere

132 for (var i = 0; i < pSortInfo.Array.length; i++) {

133 pSortInfo.ArrayUp[i] = pSortInfo.Array[i];

134 pSortInfo.Array[i] = -1;

135 }

136 // Lösche die rote Markierung

137 pSortInfo.**Right** = -1;

138 // setze die grüne Markierung nach unten und oben Links

139 pSortInfo.**Left** = 0;

140 pSortInfo.**UpLeft** = 0;

141 // und die obere rote an die Position des 2. Arrays

142 pSortInfo.**UpRight** = pSortInfo.**Left** + this.**HelpArraySize**;

143 // Wechsel den Modus

144 pSortInfo.**Mode** = 1;

145 }

146 break;

147 case 1:

148 if (!pOriginal) {

149 // Wenn oben auf grün geklickt wurde

150 if (pSortInfo.click === pSortInfo.**UpLeft**) {

151 // Vergleiche den Wert der oberen roten Markierung mit dem Wert der oberen grünen Markierung

152 if (pSortInfo.ArrayUp[pSortInfo.**UpLeft**] <= pSortInfo.ArrayUp[pSortInfo.**UpRight**])

153 {

154 // wenn er kleiner gleich ist, verschiebe den oberen grünen Wert in

155 // die untere grüne Markierung und lösche die obere grüne Markierung

156 pSortInfo.Array[pSortInfo.**Left**] = pSortInfo.ArrayUp[pSortInfo.**UpLeft**];

157 pSortInfo.ArrayUp[pSortInfo.**UpLeft**] = -1;

158 // erhöhe die Position im ersten Hilfsarray

159 this.**ArrayLeftPosition**++;

160 // wenn das Ende vom Hilfsarray nicht erreicht ist

161 if (this.**ArrayLeftPosition** < this.**HelpArraySize**) {

162 // verschiebe beide grünen Markierungen um eins nach rechts

163 pSortInfo.**UpLeft**++;

164 pSortInfo.**Left**++;

165 }

166 else {

167 // wenn das Ende erreicht wurde, lösche die obere grüne Markierung

168 pSortInfo.**UpLeft** = -1;

169 // und verschiebe nur die untere einen nach Rechts

170 pSortInfo.**Left**++;

171 }

172 }

173 // Wenn auf grün geklickt wurde und das rechte Array leer ist

174 // (Die Position im 2. Array ist über/gleich der Arraygrenze

175 else if (this.**ArrayRightPosition** >= this.**HelpArraySize**)

176 {

177 // verschiebe den oberen Wert in den unteren

178 pSortInfo.Array[pSortInfo.**Left**] = pSortInfo.ArrayUp[pSortInfo.**UpLeft**];

179 // lösche die obere grüne Markierung

180 pSortInfo.ArrayUp[pSortInfo.**UpLeft**] = -1;

181 // erhöhe die Postion im Hilfsarray

182 this.**ArrayLeftPosition**++;

183 if (this.**ArrayLeftPosition** < this.**HelpArraySize**) {

184 // wenn das Ende im Hilfsarray noch nicht erreicht ist

185 // verschiebe beide grüne Markierungen um eins nach rechts

186 pSortInfo.**UpLeft**++;

187 pSortInfo.**Left**++;

188 }

189 else {

190 // wenn das Ende erreicht ist, lösche die obere grüne Markierung

191 pSortInfo.**UpLeft** = -1;

192 }

193 }

194 }

195 // Wenn oben auf rot geklickt wurde

196 else if (pSortInfo.click === pSortInfo.**UpRight**) {

197 // Vergleiche den Wert der oberen grünen Markierung mit der oberen Roten

198 if (pSortInfo.ArrayUp[pSortInfo.**UpLeft**] >= pSortInfo.ArrayUp[pSortInfo.**UpRight**])

199 {

200 // wenn er größer gleich ist, verschiebe den oberen roten Wert

201 // in den unteren grünen und lösche die obere grüne Markierung

202 pSortInfo.Array[pSortInfo.**Left**] = pSortInfo.ArrayUp[pSortInfo.**UpRight**];

203 pSortInfo.ArrayUp[pSortInfo.**UpRight**] = -1;

204 // erhöhe die Position im zweiten Hilfsarray

205 this.**ArrayRightPosition**++;

206 // wenn das Ende vom Hilfarray nicht erreicht ist

207 if (this.**ArrayRightPosition** < this.**HelpArraySize**) {

208 // verschiebe die obere rote und die untere grüne Markierung einen nach rechts

209 pSortInfo.**UpRight**++;

210 pSortInfo.**Left**++;

211 }

212 else {

213 // wenn das Ende erreicht wurde, lösche die obere rote Markierung

214 pSortInfo.**UpRight** = -1;

215 // und verschiebe nur die untere einen nach Rechts

216 pSortInfo.**Left**++;

217 }

218 }

219 // Wenn auf rot geklickt wurde und das linke Array leer ist

220 // (Die Position im 1. Array ist über/gleich der Arraygrenze

221 else if (this.**ArrayLeftPosition** >= this.**HelpArraySize**)

222 {

223 // verschiebe den oberen Wert in den unteren

224 pSortInfo.Array[pSortInfo.**Left**] = pSortInfo.ArrayUp[pSortInfo.**UpRight**];

225 // lösche die obere rote Markierung

226 pSortInfo.ArrayUp[pSortInfo.**UpRight**] = -1;

227 // erhöhe die Position im 1. Hilfsarray

228 this.**ArrayRightPosition**++;

229 if (this.**ArrayRightPosition** < this.**HelpArraySize**) {

230 //wenn das Ende im Hilfsarray noch nicht erreicht ist

231 // verschiebe die obere rote und die untere grüne Markierung um 1 nach rechts

232 pSortInfo.**UpRight**++;

233 pSortInfo.**Left**++;

234 }

235 else {

236 // wenn das Ende erreicht ist, lösche die obere rote Markierung

237 pSortInfo.**UpRight** = -1;

238 }

239 }

240 }

241 }

242 // Haben beide Hilfsarrays ihr Ende erreicht?

243 else if (this.**ArrayLeftPosition** >= this.**HelpArraySize** && this.**ArrayRightPosition** >= this.**HelpArraySize**) {

244 // Wenn ja die Position wieder auf 0 setzen

245 this.**ArrayLeftPosition** = 0;

246 this.**ArrayRightPosition** = 0;

247 // Merge größe der Arrays erhöhen?

248 if (this.**GlobalPosition** + this.**HelpArraySize** \* 2 >= pSortInfo.Array.length)

249 {

250 //ja, also den Modus wechseln, die Globale Position zurücksetzen

251 // und die größe der Arrays verdoppeln

252 this.**GlobalPosition** = 0;

253 this.**HelpArraySize** \*= 2;

254 pSortInfo.**Mode** = 0;

255 }

256 else

257 {

258 //nein, also ist das Array noch groß genug um beide Arrays nach rechte zu verschieben

259 this.**GlobalPosition** += 2 \* this.**HelpArraySize**;

260 }

261 // Die Markierungen wieder zum Anfang zurücksetzen

262 pSortInfo.**UpLeft** = this.**GlobalPosition**;

263 pSortInfo.**UpRight** = this.**GlobalPosition** + this.**HelpArraySize**;

264 pSortInfo.**Left** = this.**GlobalPosition**;

265 }

266 if (this.**HelpArraySize** === pSortInfo.Array.length) {

267 // Das Ende wurde erreicht, es ist alles sortiert

268 pSortInfo.**Mode** = 2;

269 }

270 break;

271 case 2:

272 // Es wurde alles sortiert, das obere Array löschen

273 for (var i = 0; i < pSortInfo.Array.length; i++) {

274 if (pSortInfo.ArrayUp[i] >= 0) {

275 pSortInfo.Array[pSortInfo.**Left**] = pSortInfo.ArrayUp[i];

276 pSortInfo.ArrayUp[i] = -1;

277 pSortInfo.**Left**++;

278 }

279 }

280 // und die Werte zurücksetzen

281 this.**Left** = 0;

282 this.**Right** = -1;

283 this.**GlobalPosition** = 0;

284 this.**HelpArraySize** = 1;

285 this.**ArrayLeftPosition** = 0;

286 this.**ArrayRightPosition** = 0;

287 pSortInfo.**Mode** = 3;

288 break;

289 }

290 return pSortInfo;

291 };

292

293 return this;

294 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\oddevenmergesort.js |

1 /\*

2 Dateiname : oddevenmergesort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **OddEvenMergeSorter**() {

10 var **\_Actions**, **\_Array**;

11

12 /\*

13 \* Startet die Oddeven Merge Sortierung

14 \*/

15 this.**sortAll** = function (array) {

16 \_Actions = [];

17 \_Array = array.slice();

18 oddEvenMergeSort(0, \_Array.length);

19 \_Actions.push({**type**: "done", "**done**": 0});

20 return \_Actions;

21 };

22

23 /\*

24 \* Tauscht den Inhalt der 2 übergebenen Parameter und merkt sich die Aktion zur späteren Darstellung

25 \*/

26 function **swap**(i, j) {

27 var t = \_Array[i];

28 \_Array[i] = \_Array[j];

29 \_Array[j] = t;

30 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

31 }

32

33 /\*

34 \* Implementierung von Oddeven Merge Sort

35 \*/

36 function **oddEvenMergeSort**(lo, n) {

37 if (n > 1) {

38 var m = n >>1 ;

39 oddEvenMergeSort(lo, m);

40 oddEvenMergeSort(lo + m, m);

41 oddEvenMerge(lo, n, 1);

42 }

43 }

44

45 /\*

46 \* Implementierung der Merge Funktion

47 \*/

48 function **oddEvenMerge**(lo, n, r) {

49 var m = r \* 2;

50 if (m < n) {

51 oddEvenMerge(lo, n, m); // gerade Teilfolge

52 oddEvenMerge(lo + r, n, m); // ungerade Teilfolge

53 for (var i = lo + r; i + r < lo + n; i += m) {

54 if (\_Array[i] > \_Array[i + r])

55 swap(i, i + r);

56 }

57 }

58 else {

59 if (\_Array[lo] > \_Array[lo + r])

60 swap(lo, lo + r);

61 }

62 }

63

64 /\*

65 \* initialisiert ben�tigte Membervariablen für Usersort

66 \*/

67 this.**init** = function (pTarget) {

68 };

69

70 return this;

71 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\oddeventranspositionsort.js |

1 /\*

2 Dateiname : oddeventranspositionsort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **OddEvenTranspositionSort**() {

10 var **\_Actions**, **\_Array**;

11

12 /\*

13 \* Startet die Oddeven Sortierung

14 \*/

15 this.**sortAll** = function (array) {

16 \_Actions = [];

17 \_Array = array.slice();

18 sort();

19 \_Actions.push({**type**: "done", "**done**": 0});

20 return \_Actions;

21 };

22

23 /\*

24 \* Tauscht den Inhalt der 2 üergebenen Parameter und merkt sich die Aktion zur späteren Darstellung

25 \*/

26 function **swap**(i, j) {

27 var t = \_Array[i];

28 \_Array[i] = \_Array[j];

29 \_Array[j] = t;

30 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

31 }

32

33 /\*

34 \* Implementierung von Oddeven Sort

35 \*/

36 function **sort**() {

37 var sorted = false;

38 while (!sorted) {

39 sorted = true;

40 for (var i = 1; i < \_Array.length - 1; i += 2) {

41 if (\_Array[i] > \_Array[i + 1]) {

42 swap(i, i + 1);

43 sorted = false;

44 }

45 }

46 for (var i = 0; i < \_Array.length - 1; i += 2) {

47 if (\_Array[i] > \_Array[i + 1]) {

48 swap(i, i + 1);

49 sorted = false;

50 }

51 }

52 }

53 }

54

55 /\*

56 \* initialisiert ben�tigte Membervariablen für Usersort

57 \*/

58 this.**init** = function (pTarget) {

59 };

60

61 return this;

62 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\quicksort.js |

1 /\*

2 Dateiname : quicksort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **Quicksort**() {

10 var **\_Actions**, **\_Array**;

11

12 /\*

13 \* Startet die BubbleSort Sortierung

14 \*/

15 this.**sortAll** = function (array) {

16 \_Actions = [];

17 \_Array = array.slice();

18 recurse(0, \_Array.length - 1);

19 \_Actions.push({**type**: "done", "**done**": 0});

20 return \_Actions;

21 };

22

23 /\*

24 \* Tauscht den Inhalt der 2 übergebenen Parameter und merkt sich die Aktion zur späteren Darstellung

25 \*/

26 function **swap**(i, j) {

27 var t = \_Array[i];

28 \_Array[i] = \_Array[j];

29 \_Array[j] = t;

30 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

31 }

32

33 /\*

34 \* Implementierung von Quicksort

35 \*/

36 function **recurse**(left, right) {

37 var i = left, j = right;

38 // Vergleichselement x

39 var x = \_Array[ (left + right)>> 1 ];

40 \_Actions.push({**type**: "partition", **pivot**: x});

41 // Aufteilung

42 while (i <= j) {

43 while (\_Array[i] < x)

44 i++;

45 while (\_Array[j] > x)

46 j--;

47 if (i <= j)

48 {

49 swap(i, j);

50 i++;

51 j--;

52 }

53 }

54 // Rekursion

55 if (left < j)

56 recurse(left, j);

57 if (i < right)

58 recurse(i, right);

59 }

60 /\*\*

61 \* User Sort

62 \*/

63

64 this.**LeftStart** = -2;

65 this.**RightStart** = -1;

66 // untere Grenze der Rekursion

67 this.**Low** = 0;

68 // obere Grenze der Rekursion

69 this.**High** = -1;

70 this.**CompareElement** = 0;

71 this.**SwapElementOne** = 0;

72 this.**SwapElementTwo** = 0;

73 this.**RecursionArray** = [];

74

75 /\*

76 \* initialisiert benötigte Membervariablen für Usersort

77 \*/

78 this.**init** = function (pTarget) {

79 pTarget.**LeftStart** = -2;

80 pTarget.**RightStart** = -1;

81 pTarget.**Low** = 0;

82 pTarget.**High** = -1;

83 pTarget.**CompareElement** = 0;

84 pTarget.**SwapElementOne** = 0;

85 pTarget.**SwapElementTwo** = 0;

86 pTarget.**RecursionArray** = [];

87 };

88

89 /\*

90 \* gibt die Textbeschreibung für den aktuellen Schritt zurück

91 \*/

92 this.**updateMode** = function (pMode) {

93 switch (pMode) {

94 case 0:

95 return "Wähle das Mittlere Element(Grün) der Folge als Vergleichselement.";

96 case 1:

97 return "Suche von Links(Grün) das erste Element welches größer gleich das Vergleichselement ist.";

98 case 2:

99 return "Suche von Rechts(Rot) das erste Element welches kleiner gleich das Vergleichselement ist.";

100 case 3:

101 return "Tausche diese beiden Elemente.";

102 case 4:

103 return "Neue Grenzen auswählen."

104 case 5:

105 return "Das Array ist sortiert.";

106 }

107 };

108

109 /\*

110 \* Verarbeitet die onClick Events entsprechend des Implementierten Algorithmus

111 \*/

112 this.**onClick** = function (pSortInfo, pOriginal) {

113 switch (pSortInfo.**Mode**) {

114 case 0:

115 // Zum Anfang wird this.High mit -1 initialisiert

116 // High auf die Arraygrenze

117 if (this.**High** === -1){

118 this.**High** = pSortInfo.Array.length - 1;

119 }

120 // Wenn auf grün geklickt wurde

121 if (pSortInfo.click === pSortInfo.**Left**) {

122 // Setze beide Tauschelemente und das Vergleichselement zurück

123 this.**SwapElementOne** = -1;

124 this.**SwapElementTwo** = -1;

125 this.**CompareElement** = pSortInfo.Array[pSortInfo.**Left**];

126 // Setze die aktuellen grenzen der Rekursion

127 // und wechsel den Modus

128 pSortInfo.**Left** = this.**Low**;

129 pSortInfo.**Right** = this.**High**;

130 pSortInfo.**Mode**++;

131 }

132 break;

133

134 case 1:

135 // Zuerst wird von links das Element gesucht

136 if (pSortInfo.click === pSortInfo.**Left**) {

137 // Wenn auf grün geklickt wurde und der Wert kleiner als das Vergleichselement ist

138 if (pSortInfo.Array[pSortInfo.**Left**] < this.**CompareElement**) {

139 // Verschiebe die Markierung um einen nach Rechts

140 pSortInfo.**Left**++;

141 if (pSortInfo.**Left** > pSortInfo.**Right**) {

142 // Wenn wir auf über die rote Markierung sind

143 // müssen wir die Grenzen neu machen

144 // Modus wechseln

145 pSortInfo.**Mode** = 4;

146 }

147 }

148 else {

149 // Wenn der Wert größer ist haben wir das erste Tauschelement gefunden

150 this.**SwapElementOne** = pSortInfo.**Left**;

151 // Wechselt den Modus

152 pSortInfo.**Mode**++;

153 }

154 }

155 break;

156

157 case 2:

158 // Danach wird von rechts das nächste Element gesucht

159 // Wenn auf die rote Umrandung geklickt wurde

160 if (pSortInfo.click === pSortInfo.**Right**) {

161 // Und der rote Wert über dem Vergleichselement liegt

162 if (pSortInfo.Array[pSortInfo.**Right**] > this.**CompareElement**) {

163 // dann verschiebe die rote Umrandung ein Element nach links

164 pSortInfo.**Right**--;

165 if (pSortInfo.**Left** > pSortInfo.**Right**) {

166 // wenn dabei das grüne überschritten wird, müssen die Grenzen neu bestimmt werden

167 // daher Modus wechseln

168 pSortInfo.**Mode** = 4;

169 }

170 }

171 else {

172 // liegt der rote Wert unter dem Vergleichselement wurde das 2. Tauschelement

173 this.**SwapElementTwo** = pSortInfo.**Right**;

174 // wechselt den Modus

175 pSortInfo.**Mode**++;

176 }

177 }

178 break;

179

180 case 3:

181 // Wenn auf rot oder grün geklickt wurde

182 if (pSortInfo.click === pSortInfo.**Left** || pSortInfo.click === pSortInfo.**Right**) {

183 //Tausche die beiden Elemente

184 var t = pSortInfo.Array[this.**SwapElementOne**];

185 pSortInfo.Array[this.**SwapElementOne**] = pSortInfo.Array[this.**SwapElementTwo**];

186 pSortInfo.Array[this.**SwapElementTwo**] = t;

187 // verschiebe grün einen nach rechts

188 pSortInfo.**Left**++;

189 // verschiebe rot einen nach links

190 pSortInfo.**Right**--;

191 if (pSortInfo.**Left** <= pSortInfo.**Right**) {

192 // Wenn links dabei unter bzw. auf rechts bleibt

193 // neues element suchen, also den Modus wechseln

194 pSortInfo.**Mode** = 1;

195 }

196 else {

197 // kommt links dabei über rechts müssen die Grenzen neu berechnet werden

198 pSortInfo.**Mode** = 4;

199 }

200 }

201 break;

202

203 case 4:

204 // Wenn auf grün oder rot geklickt wurde

205 if (pSortInfo.click === pSortInfo.**Left** || pSortInfo.click === pSortInfo.**Right**) {

206 // und grün unter der oberen Grenze liegt

207 if (pSortInfo.**Left** < this.**High**) {

208 // speicherere die Grenzen um eine spätere Rekursion durchzuführen

209 // Hinweis: Bei Quicksort werden finden 2 Rekursive aufrufe statt.

210 // den ersten Aufruf arbeiten wir direkt ab. Für den 2. Speichern wir

211 // hier die Parameter zwischen und arbeiten diese ab wenn wir im ersten

212 // Auruf nicht mehr tiefer können

213 this.**RecursionArray**.unshift(this.**High**);

214 this.**RecursionArray**.unshift(pSortInfo.**Left**);

215 }

216 // wenn rot über der unteren Grenze liegt

217 if (this.**Low** < pSortInfo.**Right**)

218 {

219 //setzen wir die Parameter für eine weitere Rekursion

220 this.**Low** = this.**Low**;

221 this.**High** = pSortInfo.**Right**;

222 pSortInfo.**Left** = (this.**Low** + this.**High**) >> 1;

223 pSortInfo.**Right** = -1;

224 this.**SwapElementOne** = -1;

225 this.**SwapElementTwo** = -1;

226 pSortInfo.**Mode** = 0;

227 }

228 else {

229 // Die ersten Rekursionsaufrufe wurden alle abgearbeitet

230 // nun arbeiten wir die den 2. aufruf ab.

231 if(this.**RecursionArray**.length != 0){

232 this.**Low** = this.**RecursionArray**[0];

233 this.**High** = this.**RecursionArray**[1];

234 // 2 Elemente aus dem Array entfernen

235 this.**RecursionArray**.shift();

236 this.**RecursionArray**.shift();

237 pSortInfo.**Left** = (this.**Low** + this.**High**) >> 1;

238 pSortInfo.**Right** = -1;

239 this.**SwapElementOne** = -1;

240 this.**SwapElementTwo** = -1;

241 pSortInfo.**Mode** = 0;

242 }

243 else {

244 // Der Stack für den 2. Aufruf ist leer

245 // Wir sind am Ende angekommen und das Array ist fertig sortiert

246 pSortInfo.**Mode** = 5;

247 pSortInfo.**Left** = -1;

248 pSortInfo.**Right** = -1;

249

250 }

251 }

252 }

253 break;

254

255 case 5:

256 //Fertig sortiert

257 break;

258 }

259 return pSortInfo;

260 };

261 return this;

262 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\selectionsort.js |

1 /\*

2 Dateiname : selectionsort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **SelectionSort**() {

10 /\*\*

11 \* Visual Sort

12 \*/

13 var **\_Actions**, **\_Array**;

14

15 /\*

16 \* Startet die Insertion Sortierung

17 \*/

18 this.**sortAll** = function (array) {

19 \_Actions = [];

20 \_Array = array.slice();

21 selectionsort();

22 \_Actions.push({**type**: "done", "**done**": 0});

23 return \_Actions;

24 };

25

26 /\*

27 \* Tauscht den Inhalt der 2 übergebenen Parameter und merkt sich die Aktion zur späteren Darstellung

28 \*/

29 function **swap**(i, j) {

30 var t = \_Array[i];

31 \_Array[i] = \_Array[j];

32 \_Array[j] = t;

33 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

34 }

35

36 /\*

37 \* Implementierung von Selectionsort

38 \*/

39 function **selectionsort**() {

40 var i = 0, j = 0;

41 for (i = 0; i < \_Array.length - 1; i++) {

42 for (j = i + 1; j < \_Array.length; j++) {

43 if (\_Array[i] > \_Array[j]) {

44 swap(i, j);

45 }

46 }

47 }

48 }

49

50 /\*

51 \* User Sort

52 \*/

53 this.**LeftStart** = 0;

54 this.**RightStart** = -1;

55

56 /\*

57 \* initialisiert benötigte Membervariablen für Usersort

58 \*/

59 this.**init** = function (pTarget) {

60 pTarget.**LeftStart** = 0;

61 pTarget.**RightStart** = -1;

62 };

63

64 /\*

65 \* gibt die Textbeschreibung für den aktuellen Schritt zurück

66 \*/

67 this.**updateMode** = function (pMode) {

68 switch (pMode) {

69 case 0:

70 return "Auf das Grüne Element klicken.";

71 case 1:

72 return "Den oberen Wert(Grün) anklicken wenn er kleiner gleich als der Untere(Rot) ist.";

73 case 2:

74 return "Fertig";

75 }

76 };

77

78 /\*

79 \* Verarbeitet die onClick Events entsprechend des Implementierten Algorithmus

80 \*/

81 this.**onClick** = function (pSortInfo, pOriginal) {

82 switch (pSortInfo.**Mode**) {

83 case 0:

84 // Wenn auf grün geklickt wurde

85 if (pSortInfo.click === pSortInfo.**Left**) {

86 // setze den roten Kreis rechts von grün

87 pSortInfo.**Right** = pSortInfo.**Left** + 1;

88 // und den oberen roten über den unteren roten Kreis

89 pSortInfo.**UpLeft** = pSortInfo.**Right**;

90 // in den oberen roten Kreis wird der Wert von dem geklickten geschrieben

91 pSortInfo.ArrayUp[pSortInfo.**Right**] = pSortInfo.Array[pSortInfo.click];

92 // wechsel den Modus

93 pSortInfo.**Mode**++;

94 pSortInfo.**Update** = true;

95 }

96 break;

97 case 1:

98 // Wenn auf Rot geklickt wurde

99 if (pSortInfo.click === pSortInfo.**Right**) {

100 pSortInfo.**Update** = false;

101 if (pOriginal) {

102 // Und der Wert im oberen Grünen Kreis über den im unteren roten Kreis liegt

103 if (pSortInfo.ArrayUp[pSortInfo.**UpLeft**] > pSortInfo.Array[pSortInfo.**Right**])

104 {

105 // schreibe den Wert vom roten Kreis in den grünen

106 pSortInfo.Array[pSortInfo.**Left**] = pSortInfo.Array[pSortInfo.**Right**];

107 // in den roten Wert den von den oberen Grünen

108 pSortInfo.Array[pSortInfo.**Right**] = pSortInfo.ArrayUp[pSortInfo.**UpLeft**];

109 // den oberen grünen Wert um 1 nach rechts verschieben

110 pSortInfo.ArrayUp[pSortInfo.**UpLeft** + 1] = pSortInfo.Array[pSortInfo.**Left**];

111 // und den alten Wert nicht mehr anzeigen

112 pSortInfo.ArrayUp[pSortInfo.**UpLeft**] = -1;

113 // den oberen grünen und den unteren roten Kreis nach rechts verschieben

114 pSortInfo.**UpLeft**++;

115 pSortInfo.**Right**++;

116 }

117 }

118 else {

119 // Wenn der obere Grüne kleiner gleich dem unteren Roten ist

120 if (pSortInfo.ArrayUp[pSortInfo.**UpLeft**] <= pSortInfo.Array[pSortInfo.**Right**]) {

121 // dann verschiebe den Wert und die grüne Markierung um 1 nach Rechts

122 pSortInfo.ArrayUp[pSortInfo.**UpLeft** + 1] = pSortInfo.ArrayUp[pSortInfo.**UpLeft**];

123 pSortInfo.ArrayUp[pSortInfo.**UpLeft**] = -1;

124 pSortInfo.**UpLeft**++;

125 // genauso die Rote Markierung

126 pSortInfo.**Right**++;

127 }

128 }

129 // Wenn die rote Umrandung das Array Ende erreicht hat, fang nochmal von vorne an

130 if (pSortInfo.**Right** === pSortInfo.Array.length) {

131 pSortInfo.**Left**++;

132 pSortInfo.**Right** = -1;

133 pSortInfo.**Mode** = 0;

134 pSortInfo.**Update** = true;

135 }

136 // Wenn die grüne Umrandung das Array Ende erreicht hat, beende die Sortierung

137 if (pSortInfo.**Left** === pSortInfo.Array.length-1) {

138 pSortInfo.**Left** = -1;

139 pSortInfo.**Mode** = 2;

140 pSortInfo.**Update** = true;

141 }

142 }

143 break;

144 }

145 return pSortInfo;

146 };

147 return this;

148 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\sort\shellsort.js |

1 /\*

2 Dateiname : shellsort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny L�ttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **ShellSort**() {

10 var **\_Actions**, **\_Array**;

11

12 /\*

13 \* Startet die Shellsort Sortierung

14 \*/

15 this.**sortAll** = function (array) {

16 \_Actions = [];

17 \_Array = array.slice();

18 shellsort();

19 \_Actions.push({**type**: "done", "**done**": 0});

20 return \_Actions;

21 };

22

23 /\*

24 \* Tauscht den Inhalt der 2 übergebenen Parameter und merkt sich die Aktion zur späteren Darstellung

25 \*/

26 function **swap**(i, j) {

27 var t = \_Array[i];

28 \_Array[i] = \_Array[j];

29 \_Array[j] = t;

30 \_Actions.push({**type**: "swap", **i**: i, **j**: j});

31 }

32 function **shellsort**() {

33 var i, j;

34 var m = (\_Array.length)>>1 ;

35 while (m > 0) {

36 for (i = m; i < \_Array.length; i++) {

37 j = i;

38 while (j >= m && \_Array[j - m] > \_Array[j]) {

39 swap(j, j - m);

40 j = j - m;

41 }

42 }

43 m = ~~(m / 2);

44 }

45 }

46

47 /\*

48 \* User Sort

49 \*/

50

51 this.**LeftStart** = 0;

52 this.**RightStart** = 1;

53

54 this.**cols** = [4, 2, 1];

55 this.**LeftStart** = 0;

56 this.**RightStart** = -1;

57 this.**i** = 0;

58 this.**j** = 0;

59 this.**k** = 0;

60 this.**h** = 0;

61 this.**t** = 0;

62

63 /\*

64 \* initialisiert benötigte Membervariablen für Usersort

65 \*/

66 this.**init** = function (pTarget) {

67 pTarget.**LeftStart** = 0;

68 pTarget.**RightStart** = 1;

69 pTarget.**cols** = [4, 2, 1];

70 pTarget.**LeftStart** = 0;

71 pTarget.**RightStart** = -1;

72 pTarget.**i** = 0;

73 pTarget.**j** = 0;

74 pTarget.**k** = 0;

75 pTarget.**h** = 0;

76 pTarget.**t** = 0;

77 };

78

79 /\*

80 \* gibt die Textbeschreibung für den aktuellen Schritt zurück

81 \*/

82 this.**updateMode** = function (pMode) {

83 switch (pMode) {

84 case 0:

85 return "Spalten Anzahl auf " + String(this.**cols**[0]) + " setzen.";

86 case 1:

87 return "In der " + String(this.**i** % this.**cols**[0] + 1) + ". Spalte mit dem nächsten Element(Rot) vergleichen.";

88 case 2:

89 return "Wenn das 1. Element (Grün) größer ist als das 2. Element (Rot) dann klicke Grün ansonsten Rot.";

90 case 3:

91 return "Das Array ist sortiert.";

92 }

93 };

94

95 /\*

96 \* Verarbeitet die onClick Events entsprechend des Implementierten Algorithmus

97 \*/

98 this.**onClick** = function (pSortInfo, pOriginal) {

99 switch (pSortInfo.**Mode**) {

100 case 0:

101 // Wenn auf grün geklickt wurde

102 if (pSortInfo.click === pSortInfo.**Left**) {

103 // die colum Breite setzen

104 this.**h** = this.**cols**[0];

105 this.**i** = this.**cols**[0];

106 // Die grüne und rote Umrandung setzen

107 pSortInfo.**Left** = this.**i** - this.**h**;

108 pSortInfo.**Right** = this.**i**;

109 // Modus wechseln

110 pSortInfo.**Mode**++;

111 }

112 break;

113

114 case 1:

115 // Wenn auf grün oder Rot geklickt wurde

116 if (pSortInfo.click === pSortInfo.**Left** || pSortInfo.click === pSortInfo.**Right**) {

117 if (this.**i** < pSortInfo.Array.length) {

118 // Und wir noch nicht am Ende angekommen sind

119 this.**j** = this.**i**;

120 this.**t** = pSortInfo.Array[this.**j**];

121 pSortInfo.**Left** = this.**i** - this.**h**;

122 pSortInfo.**Right** = this.**j**;

123

124 if (this.**j** >= this.**h** && pSortInfo.Array[this.**j** - this.**h**] > this.**t**)

125 {

126 // Es muss ein Objekt in der Colum getauscht werden

127 // hierfür den Modus wechseln

128 pSortInfo.**Mode**++;

129 }

130 else {

131 // Kein Objekt in der Colum gefunden

132 // in die nächste Colum wechseln

133 this.**i** += 1;

134 pSortInfo.**Left**++;

135 pSortInfo.**Right**++;

136 }

137 }

138 else {

139 this.**cols**.shift();

140 if (this.**cols**.length == 0) {

141 // Keine weiteren colums

142 // Sind am Ende angekommen

143 pSortInfo.**Left** = -1;

144 pSortInfo.**Right** = -1;

145 pSortInfo.**Mode** = 3;

146 }

147 else {

148 // Es sind noch colums vorhanden

149 // von vorne anfangen

150 pSortInfo.**Mode** = 0;

151 }

152 }

153 }

154 break;

155

156 case 2:

157 // Wenn auf rot oder grün geklickt wurde

158 if (pSortInfo.click === pSortInfo.**Left** || pSortInfo.click === pSortInfo.**Right**) {

159 // Und das nächste Element in der Colum kleiner ist als der Vorgänger

160 if (this.**j** >= this.**h** && pSortInfo.Array[this.**j** - this.**h**] > this.**t**) {

161 //dann tausche die Werte

162 pSortInfo.Array[this.**j**] = pSortInfo.Array[this.**j** - this.**h**];

163 this.**j** = this.**j** - this.**h**;

164 pSortInfo.Array[this.**j**] = this.**t**;

165

166 // um einen klick zu sparen fragen wir hier die bedingung nochmal ab

167 if (this.**j** >= this.**h** && pSortInfo.Array[this.**j** - this.**h**] > this.**t**) {

168 // und gehen ein Element in der Colum zurück wenn sie erfüllt ist

169 pSortInfo.**Left** -= this.**h**;

170 pSortInfo.**Right** -= this.**h**;

171 }

172 else {

173 // oder wechseln in in die nächste Colum wenn sie nicht erfüllt ist

174 pSortInfo.**Mode** = 1;

175 pSortInfo.**Left**++;

176 pSortInfo.**Right**++;

177 this.**i** += 1;

178 }

179 }

180 else {

181 // Wenn das Element nicht kleiner ist dann wechsel in die nächste Column

182 pSortInfo.**Mode** = 1;

183 pSortInfo.**Left**++;

184 pSortInfo.**Right**++;

185 this.**i** += 1;

186 }

187 }

188 break;

189

190 case 3:

191 break;

192 }

193 return pSortInfo;

194 };

195 return this;

196 }

|  |
| --- |
| Z:\studium\ProjektSS15\public\_html\js\randomsort.js |

1 /\*

2 Dateiname : randomsort.js

3 Erstellt : 2015

4 Authoren : Peter Steensen (550020) and Denny Lüttschwager (343709)

5 Projekt : JavaScript anstelle Java-Applet

6 Professor : Prof. Lang

7 \*/

8

9 function **RandomSort**(pSortArtID) {

10 var **\_Actions**, \_Array, \_Step,\_Lenght;

11 var SortArtID = pSortArtID;

12 var \_Mode, \_ShowRGB;

13

14

15 /\*\*

16 \* mode

17 \* 0 = 12 Uhr

18 \* 1 = 1:30 Uhr

19 \* 2 = 3 Uhr

20 \* 3 = 4:30 Uhr

21 \* 4 = 6 Uhr

22 \* 5 = 7:30 Uhr

23 \* 6 = 9 Uhr

24 \* 7 = 10:30 Uhr

25 \*

26 \* showRGB

27 \* 0 = R

28 \* 1 = G

29 \* 2 = B

30 \* **@type** Array

31 \*/

32

33 var SortArt = [{mode:[0,1,2],showRGB:[0,1,2]},

34 {mode:[7,4,2],showRGB:[0,1,2]},

35 {mode:[0,2,5],showRGB:[0,1,2]},

36 {mode:[4,0,4],showRGB:[0,1,2]},

37 {mode:[1,4,2],showRGB:[0,1,2]},

38

39 {mode:[2,5,6],showRGB:[0,1,2]},

40 {mode:[6,1,4],showRGB:[0,1,2]},

41 {mode:[7,2,5],showRGB:[0,1,2]},

42 {mode:[1,3,0],showRGB:[0,1,2]},

43

44 {mode:[6,1,4],showRGB:[0,1,2]},

45 {mode:[4,6,2],showRGB:[0,1,2]},

46 {mode:[5,1,7],showRGB:[0,1,2]},

47 {mode:[2,4,6],showRGB:[0,1,2]},

48 ];

49

50

51

52 this.**sort** = function (pArray,pStep) {

53 \_Actions = [];

54 \_Array = pArray.slice();

55 \_Step = pStep;

56 \_Lenght = (pStep \*pStep ) -1;

57 \_Mode = SortArt[SortArtID].mode;

58 \_ShowRGB = SortArt[SortArtID].showRGB;

59

60 randomsort();

61 return \_Actions;

62 };

63 function **swap**(i, j) {

64 var v = \_Array[j].v;

65 \_Array[j].v = \_Array[i].v;

66 \_Array[i].v = v;

67 \_Actions.push({type: "swap", i: i, j: j});

68 }

69 function **random**(pMax) {

70 return **Math**.floor(**Math**.random() \* pMax);

71 }

72 function **randomsort**() {

73 //Wiederhole

74 var Steps = 0;

75 var StepsMax = 1000;

76 var lRow, lCol, lMode, lShowRGB, lOption;

77 do {

78 //w�hle einen zuf�lligen Bildpunkt (row, col)

79 lRow = random(\_Step);

80 lCol = random(\_Step);

81 lOption = random(3);

82 lMode = \_Mode[lOption];

83 lShowRGB = \_ShowRGB[lOption];

84

85 show(lRow, lCol, lMode, lShowRGB);

86 Steps++;

87 }

88 while (Steps <= StepsMax);

89 }

90 function **show**(pRow, pCol, pMode, pShowRGB) {

91 var lIndex1, lIndex2;

92 lIndex1 = pRow \* \_Step + pCol;

93 switch (pMode) {

94 case 0:

95 lIndex2 = (pRow - 1) \* \_Step + pCol;

96 break;

97 case 1:

98 lIndex2 = (pRow - 1) \* \_Step + pCol + 1;

99 break;

100 case 2:

101 lIndex2 = pRow \* \_Step + pCol + 1;

102 break;

103 case 3:

104 lIndex2 = (pRow+1) \* \_Step + pCol + 1;

105 break;

106 case 4:

107 lIndex2 = (pRow + 1) \* \_Step + pCol;

108 break;

109 case 5:

110 lIndex2 = (pRow + 1) \* \_Step + pCol - 1;

111 break;

112 case 6:

113 lIndex2 = pRow \* \_Step + pCol - 1;

114 break;

115 case 7:

116 lIndex2 = (pRow -1) \* \_Step + pCol - 1;

117 break;

118 }

119 if(!(0 <= lIndex2 && lIndex2 < \_Lenght))

120 return ;

121 switch (pShowRGB) {

122 case 0:

123 if (\_Array[lIndex1].v.r > \_Array[lIndex2].v.r)

124 swap(lIndex1, lIndex2);

125 break;

126 case 1:

127 if (\_Array[lIndex1].v.g > \_Array[lIndex2].v.g)

128 swap(lIndex1, lIndex2);

129 break;

130 case 2:

131 if (\_Array[lIndex1].v.b > \_Array[lIndex2].v.b)

132 swap(lIndex1, lIndex2);

133 break;

134 }

135 }

136 return this;

137 }