```
 1  using System;
 2  using System.Drawing;
 3  using System.Text;
 4  using System.Windows.Forms;
 5
 6  namespace SortingComparisons
 7  {
 8      public partial class Form1 : Form
 9      {
10          DateTime dateTime = new DateTime();
11          Random random;
12          Timer timer;
13
14          const int trials = 256;
15
16          int initial, limdex, limits, median, partop;
17          int botind, number, pivot, topind, temp2;
18          int[] bottoms, tops;
19
20          void GenerateList(int[] A, int n)
21          {
22              dateTime = DateTime.Now;
23              random = new Random((int)dateTime.Ticks);
24              timer = new Timer();
25
26              for (int i = 0; i < n; i++)
27                  A[i] = random.Next(1, 10 * n);
28          }
29
30          int LeftNode(int i)
31          {
32              return 2 * i;
33          }
34
35          int RightNode(int i)
36          {
37              return 2 * i + 1;
38          }
39
40          int ParentNode(int i)
41          {
42              return i / 2;
43          }
44
45          void Heapify(int[] A, int i, int n)
46          {
47              int l = LeftNode(i);
48              int r = RightNode(i);
49              int largest;
50
51              if (l < n && A[l] > A[i])
52                  largest = l;
```

```
53
54              else
55                  largest = i;
56
57              if (r < n && A[r] > A[largest])
58                  largest = r;
59
60              if (largest != i)
61              {
62                  int x = A[i];
63
64                  A[i] = A[largest];
65                  A[largest] = x;
66
67                  Heapify(A, largest, n);
68              }
69          }
70
71          void BuildHeap(int[] A, int n)
72          {
73              for (int i = n / 2; i >= 0; i--)
74                  Heapify(A, i, n);
75          }
76
77          void HeapSort(int[] A, int n)
78          {
79              BuildHeap(A, n);
80
81              for (int i = n - 1; i >= 1; i--)
82              {
83                  int x = A[0];
84
85                  A[0] = A[i];
86                  A[i] = x;
87
88                  n = n - 1;
89
90                  Heapify(A, 0, n);
91              }
92          }
93
94          int Partition(int[] A, int p, int r)
95          {
96              int x = A[p];
97              int i = p - 1;
98              int j = r + 1;
99
100             while (true)
101             {
102
103                 do
104                 {
```

```
105                     j--;
106                 } while (A[j] > x);
107 
108                 do
109                 {
110                     i++;
111                 } while (A[i] < x);
112 
113                 if (i < j)
114                 {
115                     int z = A[i];
116 
117                     A[i] = A[j];
118                     A[j] = z;
119                 }
120 
121                 else
122                     return j;
123             }
124         }
125 
126         void QuickSort(int[] A, int p, int r)
127         {
128             int q;
129 
130             if (p < r)
131             {
132                 q = Partition(A, p, r);
133                 QuickSort(A, p, q);
134                 QuickSort(A, q + 1, r);
135             }
136         }
137 
138         void SingletonsSortGotos(int[] tosort, int n)
139         {
140             initial = 0;
141             limdex = 0;
142             limits = 100;
143             partop = 0;
144             botind = 0;
145             number = n - 1;
146             pivot = 0;
147             topind = 0;
148 
149             bottoms = new int[limits];
150             tops = new int[limits];
151 
152             for (int i = 0; i < limits; i++)
153                 bottoms[i] = tops[i] = 0;
154 
155             number = n - 1;
156 
```

```
157             goto SinkTest;
158
159         Split:
160
161             median = (partop + number) / 2;
162             pivot = tosort[median];
163             topind = partop;
164             botind = number;
165
166             if (tosort[partop] > pivot)
167             {
168                 tosort[median] = tosort[partop];
169                 tosort[partop] = pivot;
170                 pivot = tosort[median];
171             }
172
173             if (tosort[number] < pivot)
174             {
175                 tosort[median] = tosort[number];
176                 tosort[number] = pivot;
177                 pivot = tosort[median];
178
179                 if (tosort[partop] > pivot)
180                 {
181                     tosort[median] = tosort[partop];
182                     tosort[partop] = pivot;
183                     pivot = tosort[median];
184                 }
185             }
186
187         FindSmall: botind--;
188             if (tosort[botind] > pivot)
189                 goto FindSmall;
190
191             temp2 = tosort[botind];
192
193         FindLarge: topind++;
194             if (tosort[topind] < pivot)
195                 goto FindLarge;
196
197             if (topind <= botind)
198             {
199                 tosort[botind] = tosort[topind];
200                 tosort[topind] = temp2;
201                 goto FindSmall;
202             }
203
204
205             if (botind - partop < number - topind)
206             {
207                 tops[limdex] = partop;
208                 bottoms[limdex] = botind;
```

```
209                 partop = topind;
210             }
211
212             else
213             {
214                 tops[limdex] = topind;
215                 bottoms[limdex] = number;
216                 number = botind;
217             }
218
219             limdex++;
220
221         SinkTest:
222
223             if (number - partop > 10)
224                 goto Split;
225
226             if (initial == partop)
227             {
228                 if (partop < number)
229                     goto Split;
230             }
231
232             for (int i = partop + 1; i <= number; i++)
233             {
234                 pivot = tosort[i];
235                 topind = i - 1;
236
237                 if (tosort[topind] > pivot)
238                 {
239                     while (true)
240                     {
241                         tosort[topind + 1] = tosort[topind];
242                         topind--;
243
244                         if (tosort[topind] <= pivot)
245                             break;
246                     }
247
248                     tosort[topind + 1] = pivot;
249                 }
250             }
251
252             limdex--;
253
254             if (limdex >= 0)
255             {
256                 partop = tops[limdex];
257                 number = bottoms[limdex];
258                 goto SinkTest;
259             }
260         }
```

```
261
262         public Form1()
263         {
264             InitializeComponent();
265
266             comboBox1.Items.Add("Heap Sort");
267             comboBox1.Items.Add("Quick Sort");
268             comboBox1.Items.Add("Singleton's Sort");
269             comboBox1.Items.Add("All");
270
271             comboBox2.Items.Add("8");
272             comboBox2.Items.Add("16");
273             comboBox2.Items.Add("32");
274             comboBox2.Items.Add("64");
275
276             comboBox3.Items.Add("1000");
277             comboBox3.Items.Add("10000");
278             comboBox3.Items.Add("100000");
279             comboBox3.Items.Add("1000000");
280
281             comboBox1.SelectedIndex = 0;
282             comboBox2.SelectedIndex = 0;
283             comboBox3.SelectedIndex = 0;
284
285             textBox1.Font = new Font("Courier New", 8.0f, FontStyle.Bold);
286         }
287
288         String Format(double number)
289         {
290             String s = number.ToString("000.000");
291             StringBuilder sb = new StringBuilder();
292
293             for (int i = 0; i < s.Length; i++)
294             {
295                 if (s[i] == '0' && i < 2)
296                     sb.Append(' ');
297
298                 else
299                     sb.Append(s[i]);
300             }
301
302             return sb.ToString();
303         }
304
305         private void button1_Click(object sender, EventArgs e)
306         {
307             if (radioButton1.Checked)
308             {
309                 textBox1.Text = "";
310
311                 string algorithm = (string)comboBox1.SelectedItem;
312                 int index = comboBox2.SelectedIndex;
```

```
313                 int number = (int)Math.Pow(2, index + 3);
314                 int[] A0 = new int[number];
315                 int[] A1 = new int[number];
316                 int[] A2 = new int[number];
317                 int[] A3 = new int[number];
318
319                 GenerateList(A0, number);
320
321                 for (int i = 0; i < number; i++)
322                     A1[i] = A2[i] = A3[i] = A0[i];
323
324                 if (algorithm == "Heap Sort" ||
325                     algorithm == "All")
326                 {
327                     HeapSort(A1, number);
328                 }
329
330                 if (algorithm == "Quick Sort" ||
331                     algorithm == "All")
332                 {
333                     QuickSort(A2, 0, number - 1);
334                 }
335
336                 if (algorithm == "Singleton's Sort" ||
337                     algorithm == "All")
338                 {
339                     SingletonsSortGotos(A3, number);
340                 }
341
342                 for (int i = 0; i < number; i++)
343                 {
344                     textBox1.Text += (i + 1) + "\t";
345                     textBox1.Text += A0[i] + "\t";
346
347                     if (algorithm == "Heap Sort" ||
348                       algorithm == "All")
349                     {
350                         textBox1.Text += A1[i] + "\t";
351                     }
352
353                     if (algorithm == "Quick Sort" ||
354                       algorithm == "All")
355                     {
356                         textBox1.Text += A2[i] + "\t";
357                     }
358
359                     if (algorithm == "Singleton's Sort" ||
360                       algorithm == "All")
361                     {
362                         textBox1.Text += A3[i] + "\t";
363                     }
364
```

```
365                     textBox1.Text += "\r\n";
366                 }
367             }
368 
369             else if (radioButton2.Checked)
370             {
371                 textBox1.Text = "";
372 
373                 string algorithm = (string)comboBox1.SelectedItem;
374                 int index = comboBox3.SelectedIndex;
375                 int number = (int)Math.Pow(10, index + 3);
376                 int[] A0 = new int[number];
377                 int[] A1 = new int[number];
378                 int[] A2 = new int[number];
379                 int[] A3 = new int[number];
380 
381                 double[] heapSortTime = new double[trials];
382                 double[] quickSortTime = new double[trials];
383                 double[] singletonsSortTime = new double[trials];
384 
385                 for (int k = 0; k < trials; k++)
386                 {
387                     GenerateList(A0, number);
388 
389                     for (int i = 0; i < number; i++)
390                     {
391                         A1[i] = A0[i];
392                         A2[i] = A0[i];
393                         A3[i] = A0[i];
394                     }
395 
396                     DateTime start, finish;
397                     TimeSpan timeSpan;
398 
399                     if (algorithm == "Heap Sort" ||
400                         algorithm == "All")
401                     {
402                         start = DateTime.Now;
403 
404                         HeapSort(A1, number);
405 
406                         finish = DateTime.Now;
407                         timeSpan = finish.Subtract(start);
408 
409                         heapSortTime[k] = timeSpan.Milliseconds;
410                     }
411 
412                     if (algorithm == "Quick Sort" ||
413                         algorithm == "All")
414                     {
415                         start = DateTime.Now;
416 
```

```
417                         QuickSort(A2, 0, number - 1);
418
419                         finish = DateTime.Now;
420                         timeSpan = finish.Subtract(start);
421
422                         quickSortTime[k] = timeSpan.Milliseconds;
423                     }
424
425                     if (algorithm == "Singleton's Sort" ||
426                         algorithm == "All")
427                     {
428                         start = DateTime.Now;
429
430                         SingletonsSortGotos(A3, number);
431
432                         finish = DateTime.Now;
433                         timeSpan = finish.Subtract(start);
434
435                         singletonsSortTime[k] = timeSpan.Milliseconds;
436                     }
437                 }
438
439                 // calculate sums and means
440
441                 double heapSortTimeSum = 0;
442                 double quickSortTimeSum = 0;
443                 double singletonsSortTimeSum = 0;
444                 double heapSortTimeMean;
445                 double quickSortTimeMean;
446                 double singletonsSortTimeMean;
447
448                 for (int k = 0; k < trials; k++)
449                 {
450                     if (algorithm == "Heap Sort" ||
451                         algorithm == "All")
452                     {
453                         heapSortTimeSum += heapSortTime[k];
454                     }
455
456                     if (algorithm == "Quick Sort" ||
457                         algorithm == "All")
458                     {
459                         quickSortTimeSum += quickSortTime[k];
460                     }
461
462                     if (algorithm == "Singleton's Sort" ||
463                         algorithm == "All")
464                     {
465                         singletonsSortTimeSum += singletonsSortTime[k];
466                     }
467                 }
468
```

```
469                 heapSortTimeMean = heapSortTimeSum / trials;
470                 quickSortTimeMean = quickSortTimeSum / trials;
471                 singletonsSortTimeMean = singletonsSortTimeSum / trials;
472
473                 // calculate sample standard deviations
474
475                 double heapSortTimeSumSD = 0;
476                 double quickSortTimeSumSD = 0;
477                 double singletonsSortTimeSumSD = 0;
478
479                 for (int k = 0; k < trials; k++)
480                 {
481                     if (algorithm == "Heap Sort" ||
482                         algorithm == "All")
483                     {
484                         heapSortTimeSumSD += Math.Pow(
485                             heapSortTime[k] - heapSortTimeMean, 2.0);
486                     }
487
488                     if (algorithm == "Quick Sort" ||
489                         algorithm == "All")
490                     {
491                         quickSortTimeSumSD += Math.Pow(
492                             quickSortTime[k] - quickSortTimeMean, 2.0);
493                     }
494
495                     if (algorithm == "Singleton's Sort" ||
496                         algorithm == "All")
497                     {
498                         singletonsSortTimeSumSD += Math.Pow(
499                             singletonsSortTime[k] - singletonsSortTimeMean, 2.0);
500                     }
501                 }
502
503                 double NminusOne = 1.0 / (trials - 1);
504                 double heapSortTimeStdDev = Math.Sqrt(
505                     NminusOne * heapSortTimeSumSD);
506                 double quickSortTimeStdDev = Math.Sqrt(
507                     NminusOne * quickSortTimeSumSD);
508                 double singletonsSortTimeStdDev = Math.Sqrt(
509                     NminusOne * singletonsSortTimeSumSD);
510
511                 textBox1.Text += "# Trials = " + trials + "\r\n";
512                 textBox1.Text += "# Numbers = " + number + "\r\n";
513                 textBox1.Text += "Mean Times in Milliseconds\r\n\r\n";
514                 textBox1.Text += "Algorithm\t\t  Mean\t\t Std Dev\r\n\r\n";
515                 textBox1.Text += "Heap Sort\t\t";
516                 textBox1.Text += Format(heapSortTimeMean) + "\t\t";
517                 textBox1.Text += Format(heapSortTimeStdDev) + "\r\n";
518                 textBox1.Text += "Quick Sort\t\t";
519                 textBox1.Text += Format(quickSortTimeMean) + "\t\t";
520                 textBox1.Text += Format(quickSortTimeStdDev) + "\r\n";
```

```
521                 textBox1.Text += "Singleton's Sort\t";
522                 textBox1.Text += Format(singletonsSortTimeMean) + "\t\t";
523                 textBox1.Text += Format(singletonsSortTimeStdDev) + "\r\n\r\n";
524             }
525         }
526     }
527 }
```