

```
1  /*
2  280
3  SortingClassDemo
4  Elliot Shaw
5  */
6
7  #include <iostream>
8  #include <ctime>
9  #include <string>
10 using namespace std;
11
12 const int MAX_CHARS = 10;
13 const int MAX_STRINGS = 1000;
14
15 void cr(int n = 1) {
16     for (int i = 0; i < n; i++)
17         cout << endl;
18 } //cr
19
20 class Sorter {
21 private:
22     string workspace[MAX_STRINGS];
23     string randomString();
24 public:
25     Sorter();
26     void load();
27     void display();
28     void bubbleSort();
29     void selectionSort();
30     void insertionSort();
31     void shellSort();
32     void quickSort();
33     void quickSort(int, int);
34     int partition(int, int);
35     bool isSorted();
36 };
37
38 Sorter::Sorter() { //there's nothing to do
39 }
40
41 string Sorter::randomString() {
42     //pre: none
43     //post: return a string of MAX_CHARS random lowercase characters
44     string s = "";
45     for (int i = 0; i < MAX_CHARS; i++) {
46         char randomChar = char(rand() % 26 + 97);
47         s += randomChar;
48     }
49     return s;
```

```
50 } //randomString
51
52 void Sorter::load() {
53     //pre: none
54     //post: workspace is loaded with random strings
55     for (int i = 0; i < MAX_STRINGS; i++)
56         workspace[i] = randomString();
57 } //load
58
59 void Sorter::display() {
60     //pre: none
61     //post: display workspace elements, separated by blanks
62     //      for testing purposes with VERY small arrays
63     for (int i = 0; i < MAX_STRINGS; i++)
64         cout << workspace[i] << " ";
65     cout << endl;
66 } //display
67
68 void Sorter::bubbleSort() {
69     //pre: none
70     //post: the workspace array is sorted in ascending order
71     for (int pass = 0; pass < MAX_STRINGS - 1; pass++) {
72         for (int pos = 0; pos < MAX_STRINGS - pass - 1; pos++) {
73             if (workspace[pos] > workspace[pos + 1]) {
74                 string temp = workspace[pos];
75                 workspace[pos] = workspace[pos + 1];
76                 workspace[pos + 1] = temp;
77             }
78         }
79     }
80 } //bubbleSort
81
82 void Sorter::selectionSort() {
83     string temp;
84     for (int i = 0; i < MAX_STRINGS; i++) {
85         for (int j = i + 1; j < MAX_STRINGS; j++) {
86             if (workspace[i] > workspace[j]) {
87                 temp = workspace[j];
88                 workspace[j] = workspace[i];
89                 workspace[i] = temp;
90             }
91         }
92     }
93 }
94
95 void Sorter::insertionSort() {
96     for (int i = 1; i < MAX_STRINGS; i++) {
97         for (int j = 0; j < i; j++) {
98             if (workspace[j] > workspace[i]) {
```

```
109     }
110
111     void Sorter::shellSort() {
112         string temp;
113         int check;
114         for (int shell = MAX_STRINGS / 2; shell > 0; shell = shell / 2) {
115             for (int i = 0; i + shell < MAX_STRINGS; i++) {
116                 temp = workspace[i + shell];
117                 for (check = i; check >= 0; check = check - shell) {
118                     if (temp > workspace[check]) {
119                         workspace[check + shell] = temp;
120                         break;
121                     }
122                     else {
123                         workspace[check + shell] = workspace[check];
124                     }
125                 }
126             }
127         }
128     }
129
130     void Sorter::quickSort() {
131         quickSort(0, MAX_STRINGS - 1);
132     }
133
134     void Sorter::quickSort(int low, int high) {
135         if (low < high) {
136             int part = partition(low, high);
137
138             quickSort(low, part - 1);
139             quickSort(part + 1, high);
140         }
141     }
142
143     int Sorter::partition(int low, int high) {
144         string temp;
145         string pivot = workspace[high];
146         int i = low - 1;
147         for (int j = low; j <= high-1; j++) {
```

```
148     if (workspace[j] <= pivot) {
149         i++;
150         temp = workspace[j];
151         workspace[j] = workspace[i];
152         workspace[i] = temp;
153     }
154 }
155 temp = workspace[i + 1];
156 workspace[i + 1] = workspace[high];
157 workspace[high] = temp;
158 return(i + 1);
159 }
160
161 bool Sorter::isSorted() {
162     //pre: none
163     //post: returns true if the workspace is ascending sorted,
164     //      false otherwise
165     for (int i = 0; i < MAX_STRINGS - 1; i++)
166         if (workspace[i] > workspace[i + 1]) {
167             cout << i << endl;
168             return false;
169         }
170     return true;
171 } //isSorted
172
173 int main() {
174     srand(time(NULL));
175     clock_t start, stop;
176     Sorter s1; //use s1 with bubble sort
177     s1.load();
178     Sorter s2 = Sorter(s1); //use s2 with selection sort
179     Sorter s3 = Sorter(s1); //use s3 with insertion sort
180     Sorter s4 = Sorter(s1); //use s4 with shell sort
181     Sorter s5 = Sorter(s1); //use s5 with quick sort
182
183     //test bubble sort
184     start = clock();
185     s1.bubbleSort();
186     stop = clock();
187     //s1.display();
188     if (!s1.isSorted()) {
189         cout << "Error: bubble sort failed";
190         cr(2);
191         exit(EXIT_FAILURE);
192     }
193     else {
194         double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
195         cout << "Bubble sort time: " << elapsedTime << endl;
196     }
```

```
197     cr(2);
198
199     //test selection sort
200     start = clock();
201     s2.selectionSort();
202     stop = clock();
203     //s2.display();
204     if (!s2.isSorted()) {
205         cout << "Error: Selection sort failed";
206         cr(2);
207         exit(EXIT_FAILURE);
208     }
209     else {
210         double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
211         cout << "Selection sort time: " << elapsedTime << endl;
212     }
213     cr(2);
214
215     //test insertion sort
216     start = clock();
217     s3.insertionSort();
218     stop = clock();
219     //s2.display();
220     if (!s3.isSorted()) {
221         cout << "Error: insertion sort failed";
222         cr(2);
223         exit(EXIT_FAILURE);
224     }
225     else {
226         double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
227         cout << "insertion sort time: " << elapsedTime << endl;
228     }
229     cr(2);
230
231     //test shell sort
232     start = clock();
233     s4.shellSort();
234     stop = clock();
235     //s2.display();
236     if (!s4.isSorted()) {
237         cout << "Error: Shell sort failed";
238         cr(2);
239         exit(EXIT_FAILURE);
240     }
241     else {
242         double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
243         cout << "shell sort time: " << elapsedTime << endl;
244     }
245     cr(2);
```

```
246
247
248     //test quick sort
249     start = clock();
250     s5.quickSort();
251     stop = clock();
252     //s2.display();
253     if (!s5.isSorted()) {
254         cout << "Error: quick sort failed";
255         cr(2);
256         exit(EXIT_FAILURE);
257     }
258     else {
259         double elapsedTime = ((double)stop - start) / CLOCKS_PER_SEC;
260         cout << "quick sort time: " << elapsedTime << endl;
261     }
262     cr(2);
263
264
265 }
```