```cpp
#include "Sorting.h"

void Sorting::InsertionSort(double a[], int n) {
    for (int j = 1; j < n; j++)
    {
        double key = a[j];
        int i = j - 1;

        while (i >= 0 && a[i] > key)
        {
            a[i + 1] = a[i];
            i--;
        }

        a[i + 1] = key;
    }
}

void Sorting::SelectionSort(double a[], int n) {
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (a[i] > a[j])
            {
                double t = a[i];

                a[i] = a[j];
                a[j] = t;
            }
        }
    }
}

int Sorting::Parent(int i) {
    return i / 2;
}

int Sorting::Left(int i) {
    return 2 * i;
}

int Sorting::Right(int i) {
    return 2 * i + 1;
}

void Sorting::Heapify(double a[], int i, int n, int& heapSize) {
    int l = Left(i);
    int r = Right(i);
    int largest = -1;

    if (l < heapSize && a[l] > a[i])
```

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

```
105     {
106         if (a[i] < x)
107         {
108             t = a[i];
109             a[i] = a[storeIndex];
110             a[storeIndex++] = t;
111         }
112     }
113
114     t = a[storeIndex];
115     a[storeIndex] = a[hi];
116     a[hi] = t;
117
118     return storeIndex;
119 }
120
121 void Sorting::DoQuickSort(double a[], int n, int p, int r) {
122     if (p < r)
123     {
124         int q = Partition(a, n, p, r);
125
126         DoQuickSort(a, n, p, q - 1);
127         DoQuickSort(a, n, q + 1, r);
128     }
129 }
130
131 void Sorting::QuickSort(double a[], int n) {
132     DoQuickSort(a, n, 0, n - 1);
133 }
134
135 void Sorting::DoSingletonsSort(double a[], int ii, int jj) {
136     int m = 0;
137     int i = ii;
138     int j = jj;
139     int iu[50] = { 0 };
140     int il[50] = { 0 };
141     int ij = 0, l = 1, k = 0;
142     double t = 0.0, tt = 0.0;
143     Label1:
144         if (i >= j)
145             goto Label8;
146         goto Label2;
147     Label2:
148         k = i;
149         ij = (j + i) / 2;
150         t = a[ij];
151         if (a[i] <= t)
152             goto Label3;
153         a[ij] = a[i];
154         a[i] = t;
155         t = a[ij];
156         goto Label3;
```

```
157     Label3:
158         l = j;
159         if (a[j] >= t)
160             goto Label5;
161         a[ij] = a[j];
162         a[j] = t;
163         t = a[ij];
164         if (a[i] <= t)
165             goto Label5;
166         a[ij] = a[i];
167         a[i] = t;
168         t = a[ij];
169         goto Label5;
170     Label4:
171         a[l] = a[k];
172         a[k] = tt;
173         goto Label5;
174     Label5:
175         l--;
176         if (a[l] > t)
177             goto Label5;
178         tt = a[l];
179         goto Label6;
180     Label6:
181         k++;
182         if (a[k] < t)
183             goto Label6;
184         if (k <= l)
185             goto Label4;
186         if (l - i <= j - k)
187             goto Label7;
188         il[m] = i;
189         iu[m] = l;
190         i = k;
191         m++;
192         goto Label9;
193     Label7:
194         il[m] = k;
195         iu[m] = j;
196         j = l;
197         m++;
198         goto Label9;
199     Label8:
200         m--;
201         if (m == -1)
202             return;
203         i = il[m];
204         j = iu[m];
205         goto Label9;
206     Label9:
207         if (j - i > 10)
208             goto Label2;
```

```
209         if (i == ii)
210             goto Label1;
211         i--;
212         goto Label10;
213     Label10:
214         i++;
215         if (i == j)
216             goto Label8;
217         t = a[i + 1];
218         if (a[i] <= t)
219             goto Label10;
220         k = i;
221         goto Label11;
222     Label11:
223         a[k + 1] = a[k];
224         k--;
225         if (t < a[k])
226             goto Label11;
227         a[k + 1] = t;
228         goto Label10;
229 }
230
231 void Sorting::SingletonsSort(double a[], int n) {
232     DoSingletonsSort(a, 0, n - 1);
233 }
234
235 void Sorting::SiftUp(double a[], int i, int n) {
236     double copy = a[i];
237     int j;
238
239     while (true)
240     {
241         j = i + i;
242
243         if (j <= n)
244         {
245             if (j < n)
246             {
247                 if (a[j + 1] > a[j])
248                     j++;
249             }
250
251             if (a[j] > copy)
252             {
253                 a[i] = a[j];
254                 i = j;
255                 continue;
256             }
257         }
258
259         break;
260     }
```

```
261
262     a[i] = copy;
263 }
264
265 void Sorting::TreeSort3(double a[], int n)
266 {
267     for (int i = n / 2; i >= 1; i--)
268         SiftUp(a, i, n - 1);
269
270     for (int i = n - 1; i >= 1; i--)
271     {
272         SiftUp(a, 0, i);
273
274         double t = a[0];
275
276         a[0] = a[i];
277         a[i] = t;
278     }
279 }
```