```
#include <iostream>
  1
  2
    #include <fstream>
  3
  4
    using namespace std;
  5
    ifstream dataIn("infile.txt");
  6
  7
    ofstream dataOut("outfile.txt");
  8
  9
    void compare(int, int, int);
 10 void quickSort(int a[], int first, int last,int& swaps, int &compare);
     int pivot(int a[], int first, int last, int& swaps , int &compare);
 11
 12
    void swap(int& a, int& b, int& swaps);
 13
 14
 15
    void heapify(int[], int , int);
 16
    void heapSort(int[], int, int& , int& );
 17
 18
    int main()
 19
 20
         int arr[100], bubble[100], Quick[100], heap[100], amount;
 2.1
         string type;
 2.2
 23
         while (!dataIn.eof()){
 2.4
         dataIn>>amount;
         dataIn>>type;
 25
 26
         dataOut<<endl<<amount<<" "<<type<<" in order."<<endl;
 27
         for(int i = 0 ; i < amount; i++){</pre>
 28
 2.9
             dataIn>>arr[i];
 30
             bubble[i] = Quick[i] = heap[i] = arr[i];
             dataOut<<bubble[i]<<" ";</pre>
 31
 32
         }
 33
         dataOut<<"\n\n";</pre>
 34
 35
           // Bubble Sort Starts Here
          int temp, compareBubble= 0, interchangeBubble = 0;
 36
          for(int i=0; i<amount; i++){</pre>
 37
 38
             for(int j=0; j<amount-1; j++){
 39
 40
                  compareBubble ++;
 41
                  if(bubble[j]>bubble[j+1]){
 42
 43
                      interchangeBubble++;
 44
                      temp=bubble[j];
 45
                      bubble[j]=bubble[j+1];
 46
                      bubble[j+1]=temp;
 47
 48
 49
 50
         dataOut<<"Bubble Sort Used :: "<<interchangeBubble<<" interchange : " <<</pre>
compareBubble<<" comparisons"<<endl;</pre>
 51
         for(int i = 0 ; i < amount; i++){</pre>
 52
           dataOut<<bubble[i]<<" ";</pre>
 53
 54
 55
          int interchangeQuick = 0, compareQuick = 0;
 56
         quickSort(Quick, 0, amount, interchangeQuick, compareQuick);
 57
         dataOut<<"\n\nQuick Sort Used :: "<<interchangeQuick<<" interchange : " <<</pre>
compareQuick<<" comparisons"<<endl;</pre>
 58
         for(int i = 0 ; i < amount; i++){
 59
           dataOut<<Quick[i]<<" ";</pre>
 60
 61
 62
         int interchangeHeap= 0, compareHeap = 0;
 63
         heapSort(heap, amount, interchangeHeap, compareHeap);
 64
         dataOut<<"\n\nHeap Sort Used :: "<<interchangeHeap<<" interchange : " <<
```

```
compareHeap<<" comparisons"<<endl;</pre>
           for(int i = 0 ; i < amount; i++){</pre>
 65
 66
            dataOut<<heap[i]<<" ";</pre>
 67
 68
           dataOut<<"\n\nINTERCHANGES :: ";</pre>
 69
 70
         compare(interchangeBubble, interchangeQuick, interchangeHeap);
 71
         dataOut<<"\nCOMPARISONS :: ";</pre>
 72
         compare(compareBubble, compareQuick, compareHeap);
 73
          dataOut<<"\n\n\n\n\n";</pre>
 74
      }
 75
 76
         return 0;
 77
     void compare(int bubble, int quick, int heap){
 78
 79
 80
          if(bubble>quick && bubble > heap){
 81
 82
              if(quick > heap){
 83
                  dataOut<<"Bubble had the most, then Quick , then heap";</pre>
 84
 85
 86
              else if(quick < heap){</pre>
                  dataOut<<"Bubble had the most, then heap, then Quick";</pre>
 87
 88
 89
              else{
 90
                  dataOut<<"Bubble had the most, then heap and Quick as equal";</pre>
              }
 91
 92
93
 94
         else if(quick>bubble && quick > heap){
 95
 96
              if(bubble > heap){
 97
 98
                  dataOut<<"Quick had the most, then Bubble , then heap.";</pre>
99
100
              else if(bubble < heap){</pre>
                  dataOut<<"Quick had the most, then heap, then Bubble.";</pre>
101
102
103
              else{
                  dataOut<<"Quick had the most, then heap and Bubble as equal.";</pre>
104
105
106
107
108
         else{
109
110
              if(bubble > quick){
111
112
                  dataOut<<"Heap had the most, then Bubble , then Quick.";</pre>
113
114
              else if(bubble < quick){</pre>
115
                  dataOut<<"Heap had the most, then Quick, then Bubble.";</pre>
116
117
              else {
118
                  dataOut<<"Heap had the most, then Quick and Bubble as equal.";
119
120
          }
     }
121
122
123
     void quickSort( int a[], int first, int last, int& swaps, int& compare )
124
125
126
          int pivotElement;
127
         int counter = 0;
128
          if(first < last)</pre>
129
```

```
130
            pivotElement = pivot(a, first, last, swaps,compare);
131
            quickSort(a, first, pivotElement-1, swaps, compare);
132
            quickSort(a, pivotElement+1, last,swaps,compare);
133
134
         }
135
136
    }
137
138
139
     * Find and return the index of pivot element.
     * @param a - The array.
140
141
     * @param first - The start of the sequence.
142
     * @param last - The end of the sequence.
143
     * @return - the pivot element
144
    * /
145
    int pivot(int a[], int first, int last, int& swaps, int& compare)
146
147
         int p = first;
148
         int pivotElement = a[first];
149
150
         for(int i = first+1 ; i \le last ; i++)
151
152
             /* If you want to sort the list in the other order, change "<=" to ">" */
153
             compare++;
154
             if(a[i] <= pivotElement)</pre>
155
156
                 p++;
157
                 swap(a[i], a[p],swaps);
158
159
         }
160
161
         swap(a[p], a[first],swaps);
162
163
         return p;
164
     }
165
166
167
     * Swap the parameters.
168
      * @param a - The first parameter.
169
     * @param b - The second parameter.
170
     * /
171
172
    void swap(int& a, int& b, int& swaps)
173
174
         swaps++;
175
         int temp = a;
176
         a = b;
177
         b = temp;
178
     }
179
180
     * Swap the parameters without a temp variable.
181
182
     * Warning! Prone to overflow/underflow.
183
      * @param a - The first parameter.
184
     * @param b - The second parameter.
185
    * /
186
    void swapNoTemp(int& a, int& b)
187
188
         a -= b;
         b += ai// b gets the original value of a
189
190
         a = (b - a)i// a gets the original value of b
191
     }
192
193
     //**************
194
195
```

```
196
    void heapify(int arr[], int n, int i, int& swaps, int & compare)
197
198
         int largest = i; // Initialize largest as root
199
         int 1 = 2*i + 1; // left = 2*i + 1
         int r = 2*i + 2; // right = 2*i + 2
200
201
202
         // If left child is larger than root
203
204
         if (1 < n && arr[1] > arr[largest])
205
             largest = 1;
206
207
         // If right child is larger than largest so far
208
         if (r < n && arr[r] > arr[largest])
209
             largest = r;
210
211
         // If largest is not root
212
         compare++;
213
         if (largest != i)
214
215
             swap(arr[i], arr[largest], swaps);
216
217
             // Recursively heapify the affected sub-tree
218
             heapify(arr, n, largest, swaps, compare);
219
         }
220
     }
221
222
    // main function to do heap sort
223
    void heapSort(int arr[], int n, int& swaps , int& compare)
224
225
         // Build heap (rearrange array)
226
         for (int i = n / 2 - 1; i >= 0; i--)
             heapify(arr, n, i, swaps, compare);
227
228
229
         // One by one extract an element from heap
230
         for (int i=n-1; i>=0; i--)
231
         {
232
             // Move current root to end
             swap(arr[0], arr[i], swaps);
233
234
             // call max heapify on the reduced heap
235
236
             heapify(arr, i, 0 , swaps , compare);
         }
237
     }
238
```