

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <time.h>
5 #include "sortlist.h"
6
7 struct Operations {
8     int swaps;
9     int comps;
10 } operations = {0, 0};
11
12 const struct Colors {
13     char *yellow;
14     char *red;
15     char *green;
16     char *none;
17 } colors = {"\033[0;43m", "\033[0;41m", "\033[0;42m", "\033[0m"};
18
19 int current_random = 0;
20 int verbose;
21 int list[LIST_SIZE];
22
23 void
24 prlist() {
25     for (int i = 0; i < LIST_SIZE; i++)
26         printf("%d ", list[i]);
27 }
28
29 void
30 prclist(int i1, int i2, char *c1, char *c2) {
31     for (int i = 0; i < LIST_SIZE; i++) {
32         printf("%s%d%s ",
33             i == i2 ? c2 : i == i1 ? c1 : "",
34             list[i],
35             colors.none);
36     }
37     putchar('\n');
38 }
39
40
41 void
42 disp(int i1, int i2, char *c1, char *c2, int swap) {
43     if ((swap && SWAP_VERBOSE) || verbose) {
44         prclist(i1, i2, c1, c2);
45         usleep(MILLISECONDS * 1000);
46     }
47 }
48
49 void
50 disp_swap(int i1, int i2) {
51     operations.swaps++;
52     disp(i1, i2, colors.green, colors.green, 1);
53 }
54
55 void
56 disp_comp(int i1, int i2) {
57     operations.comps++;
58     disp(i1, i2, colors.yellow, colors.red, 0);
59 }
60
61 void
62 swap(int i1, int i2) {
63     int tmp = list[i1];
64     list[i1] = list[i2];
65     list[i2] = tmp;
66     disp_swap(i1, i2);
67 }
68
69 #if BUBBLESORT
70 void
71 bubblesort() {
72     int ordered = 1;
73     do {
74         ordered = 1;
75         for (int i = 0; i < LIST_SIZE - 1; i++) {
76             disp_comp(i, i + 1);
77             if (list[i] > list[i + 1]) {
78                 swap(i, i + 1);
79                 ordered = 0;
80             }
81         }

```

```

82     } while (!ordered);
83 }
84 #endif
85
86 /*
87  * Based on the Hoare partition scheme of quicksort
88  * https://en.wikipedia.org/wiki/Quicksort#Hoare\_partition\_scheme
89  */
90 #if QUICKSORT
91 int
92 partition(int start, int stop) {
93     int pindex = (stop + start) / 2;
94     int p = list[pindex];
95     int left = start - 1;
96     int right = stop + 1;
97     while (1) {
98         do {
99             left++;
100             disp_comp(left, pindex);
101         } while (list[left] < p);
102         do {
103             right--;
104             disp_comp(right, pindex);
105         } while (list[right] > p);
106         if (left >= right)
107             return right;
108         swap(left, right);
109     }
110 }
111
112 void
113 quicksort(int start, int stop) {
114     if (start >= 0 && stop >= 0 && start < stop) {
115         int p = partition(start, stop);
116         quicksort(start, p);
117         quicksort(p + 1, stop);
118     }
119 }
120 #endif
121
122
123 // Based on the Fischer Yates Algorithm
124 // https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates\_shuffle#The\_modern\_algorithm
125 void
126 fill() {
127     for (int i = 0; i < LIST_SIZE; i++)
128         list[i] = i + 1;
129 }
130
131 #if RANDOM
132 void randomize() {
133     srand(time(0));
134     for (int i = LIST_SIZE - 1; i > 0; i--) {
135         int j = rand() % (i + 1);
136         swap(i, j);
137     }
138 }
139
140 void
141 random() {
142     printf("\n\nBegin Randomization.\n");
143     current_random = 1;
144     verbose = RAND_VERBOSE;
145     randomize();
146     current_random = 0;
147 #if LIST_VERBOSE
148     printf("\nRandomized List:\t");
149     prlist();
150 #endif
151     printf("\n\n");
152 }
153 #endif
154
155 #if BUBBLESORT || QUICKSORT
156 void
157 sort() {
158     operations.swaps = 0;
159     printf("Begin Sorting.\n");
160 #if BUBBLESORT
161     bubblesort();
162 #else

```

```
163         quicksort(0, LIST_SIZE - 1);
164 #endif
165 #if LIST_VERBOSE
166     printf("Sorted List:\t");
167     prlist();
168 #endif
169     printf("\nFinished in %d swaps and %d comparisons (%d total operations).",
170           operations.swaps, operations.comps, operations.swaps + operations.comps);
171 }
172 #endif
173
174 int
175 main() {
176     fill();
177     #if RANDOM
178         random();
179     #endif
180     #if BUBBLESORT || QUICKSORT
181         sort();
182     #endif
183 }
184
```