```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include "sortlist.h"

struct Operations {
        int swaps;
        int comps;
} operations = {0, 0};

const struct Colors {
        char *yellow;
        char *red;
        char *green;
        char *none;
} colors = {"\033[0;43m", "\033[0;41m", "\033[0;42m", "\033[0m"};

int current_random = 0;
int verbose;
int list[LIST_SIZE];

void
prlist() {
        for (int i = 0; i < LIST_SIZE; i++)
                printf("%d ", list[i]);
}

void
prclist(int i1, int i2, char *c1, char *c2) {
        for (int i = 0; i < LIST_SIZE; i++) {
                printf("%s%d%s ",
                                i == i2 ? c2 : i == i1 ? c1 : "",
                                list[i],
                                colors.none);
        }
        putchar('\n');

}

void
disp(int i1, int i2, char *c1, char *c2, int swap) {
        if ((swap && SWAP_VERBOSE) || verbose) {
                prclist(i1, i2, c1, c2);
                usleep(MILLISECONDS * 1000);
        }
}

void
disp_swap(int i1, int i2) {
        operations.swaps++;
        disp(i1, i2, colors.green, colors.green, 1);
}

void
disp_comp(int i1, int i2) {
        operations.comps++;
        disp(i1, i2, colors.yellow, colors.red, 0);
}

void
swap(int i1, int i2) {
        int tmp = list[i1];
        list[i1] = list[i2];
        list[i2] = tmp;
        disp_swap(i1, i2);
}

#if BUBBLESORT
void
bubblesort() {
        int ordered = 1;
        do {
                ordered = 1;
                for (int i = 0; i < LIST_SIZE - 1; i++) {
                        disp_comp(i, i + 1);
                        if (list[i] > list[i + 1]) {
                                swap(i, i + 1);
                                ordered = 0;
                        }
                }
        } while (!ordered);
}
#endif

/*
 * Based on the Hoare partition scheme of quicksort
 * https://en.wikipedia.org/wiki/Quicksort#Hoare_partition_scheme
 */
#if QUICKSORT
int
partition(int start, int stop) {
        int pindex = (stop + start) / 2;
        int p = list[pindex];
        int left = start - 1;
        int right = stop + 1;
        while (1) {
                do {
                        left++;
                        disp_comp(left, pindex);
                } while (list[left] < p);
                do {
                        right--;
                        disp_comp(right, pindex);
                } while (list[right] > p);
                if (left >= right)
                        return right;
                swap(left, right);
        }
}

void
quicksort(int start, int stop) {
        if (start >= 0 && stop >= 0 && start < stop) {
                int p = partition(start, stop);
                quicksort(start, p);
                quicksort(p + 1, stop);
        }
}
#endif


// Based on the Fischer Yates Algorithm
// https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle#The_modern_algorithm
void
fill() {
        for (int i = 0; i < LIST_SIZE; i++)
                list[i] = i + 1;
}

#if RANDOM
void randomize() {
        srand(time(0));
        for (int i = LIST_SIZE - 1 ; i > 0; i--) {
                int j = rand() % (i + 1);
                swap(i, j);
        }
}

void
random() {
        printf("\n\nBegin Randomization.\n");
        current_random = 1;
        verbose = RAND_VERBOSE;
        randomize();
        current_random = 0;
#if LIST_VERBOSE
        printf("\nRandomized List:\t");
        prlist();
#endif
        printf("\n\n");
}
#endif

#if BUBBLESORT || QUICKSORT
void
sort() {
        operations.swaps = 0;
        printf("Begin Sorting.\n");
#if BUBBLESORT
                bubblesort();
#else
```

```c
                quicksort(0, LIST_SIZE - 1);
#endif
#if LIST_VERBOSE
        printf("Sorted List:\t");
        prlist();
#endif
        printf("\nFinished in %d swaps and %d comparisons (%d total operations).",
                        operations.swaps, operations.comps, operations.swaps + operations.comps);
}
#endif

int
main() {
        fill();
#if RANDOM
        random();
#endif
#if BUBBLESORT || QUICKSORT
        sort();
#endif
}

```