Sculptural Sorts

An Artistic Investigation of Sorting Algorithms

David E. Shere

April 3, 2013

The modern society in which we live is increasingly reliant on technology for its day to day functions. In spite of this, many people are still ignorant of even the basics of how the devices they rely on work; for many they are magical boxes handed down by the priests of technology. In recent years this divide between those who understand technology and the general population has grown. The success of tablet computers and personal computers which leave the user with little or no access to the system internals seems to indicate that most people don't want to know more. However, it is the openness of earlier popular platforms that gave users the opportunity to discover the world being closed off. Without this openness the barriers for entry into the world of technological understanding will continue to grow.

Most of the focus on correcting this problem is directed at providing resources for people to learn how to program. Projects like Computing At School[1], Arduino[2], and Raspberry Pi[3] are forming to provide resources to increase the understanding of technology. While I applaud their efforts and believe they make a true difference, I believe that the integration of computer science concepts into society at large requires their visibility outside of their traditional realm. This is the motivation behind my algorithmic sculptures. By making artwork that puts the concepts our technology is built on front and center these building blocks are exposed to an audience that would otherwise ignore them. In this way they help to shed light seemingly esoteric topics.

[1]

[1] http://www.computingatschool.org.uk/

[2] http://arduino.cc/

[3] http://raspberrypi.org/

Listing 1: Main Program [main.c]

```
#include <stdlib.h>

#include "sorts/sorts.h"
#include "shift.h"
#include "util.h"

/* Available color states */
enum color_type {
    GREEN,
    CYAN,
    BLUE
};

/* Function executed for each step of the sorting algorithm
   .
 * This is used to change the state of the LEDs of the
    sculpture.
 *      uint8_t a[] -- The current state of the array.
 *      uint8_t len -- The length of the array.
 *      uint8_t pos -- The position of the sorting cursor.
 */
void display_sort_state(
        uint8_t a[], uint8_t len, uint8_t pos) {

    SHIFT_BEGIN();

    // Shift out the color array state.
    for (uint8_t i = 0; i < len; i++) {
        switch (a[i]) {
            case GREEN:
                SHIFT_OUT(ON);
                SHIFT_OUT(OFF);
                break;
            case CYAN:
                SHIFT_OUT(ON);
                SHIFT_OUT(ON);
                break;
            case BLUE:
                SHIFT_OUT(OFF);
                SHIFT_OUT(ON);
                break;
        }
    }
```

```
    // Shift out the cursor position.
    for (uint8_t i = 0; i < len; i++)
        SHIFT_OUT(i == pos ? ON : OFF);

    SHIFT_END();

    // This delay will need tweaking.
    for (uint32_t i = 0; i < 1000000000UL; i++);

}

uint8_t a[8];
int main(void) {
    SHIFT_INIT();
    for(;;) {       // Randomize and sort forever.
        for (uint8_t i = 0; i < LENGTH(a); i++) {
            a[i] = random() % 3;
        }
        SORT_APPLY(ALGORITHM, a, display_sort_state);
    }
}
```

Listing 2: LED Shift Register Defines [shift.h]

```
#ifndef SHIFT_H
#define SHIFT_H
#include <avr/io.h>

/* Pin and port values for the shift registers */
#define CFG_SHIFT_DDR   DDRB
#define CFG_SHIFT_SER   PB0
#define CFG_SHIFT_RCLK  PB1
#define CFG_SHIFT_SRCLK PB2
#define CFG_SHIFT_PORT  PORTB

/* Initialize the pins used for the shift registers. */
#define SHIFT_INIT() \
    CFG_SHIFT_DDR |= _BV(CFG_SHIFT_RCLK) \
                  |  _BV(CFG_SHIFT_SRCLK) \
                  |  _BV(CFG_SHIFT_SER)

/* Shift a bit state out to the shift registers */
#define SHIFT_OUT(bit) \
    do { \
        CFG_SHIFT_PORT &= ~_BV(CFG_SHIFT_SRCLK); \
```

```
        if (bit) \
            CFG_SHIFT_PORT |= _BV(CFG_SHIFT_SER); \
        else \
            CFG_SHIFT_PORT &= ~_BV(CFG_SHIFT_SER); \
        CFG_SHIFT_PORT |= _BV(CFG_SHIFT_SRCLK); \
    } while (0)

/* Begin a shift out process.
 * Run this before using shift_bit_out.
 */
#define SHIFT_BEGIN() (CFG_SHIFT_PORT &= ~_BV(
   CFG_SHIFT_RCLK))

/* End a shift out process.
 * Run this after the last call of shift_bit_out.
 */
#define SHIFT_END()    (CFG_SHIFT_PORT |= _BV(
   CFG_SHIFT_RCLK))

#endif
```

Listing 3: Utility Defines [util.h]

```
#ifndef UTIL_H
#define UTIL_H
#include <stdint.h>

#define ON 1
#define OFF 0

/* Length macro for uint8_t arrays */
#define LENGTH(a) (sizeof(a)/sizeof(uint8_t))

/* Expands to a sort function name with an apply function.
 *     NAME -- the sort's name (ie. bubble or selection).
 *     A    -- the array to sort.
 *     FN   -- the function to apply.
 */
#define SORT_APPLY(NAME, A, FN) (SORT_FN(NAME)(A, LENGTH(a)
   , FN))
#define SORT_FN(str) (str ## _sort_apply)

#endif
```

Listing 4: Sort Declarations [sorts/sorts.h]

```
#ifndef SORT_H
#define SORT_H
#include <stdint.h>

/* Type signature for functions that can be applied to the
   sort. */
typedef void (*apply_fn)(uint8_t*, uint8_t, uint8_t);

/* Sort functions with an applied function that will run
 * for each step of the sort.
 *      uint8_t a[] -- The array to sort.
 *      uint8_t len -- the length of the array.
 *      apply_fn fn -- The display function for the sort.
 */

void bubble_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn);

void selection_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn);

void insertion_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn);

void shell_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn);

void heap_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn);

#endif
```

Listing 5: Bubble Sort [sorts/bubble.c]

```
#include "sorts.h"
#include "../util.h"

void bubble_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn) {

    uint8_t swapped;
    uint8_t i, tmp;
    for (;;) {
```

```
        swapped = 0;
        for (i = 0; i < len-1; i++) {
            fn(a, len, i);
            if (a[i+1] < a[i]) {
                tmp = a[i];
                a[i] = a[i+1];
                a[i+1] = tmp;
                swapped = 1;
            }
        }
        fn(a, len, i);
        if (!swapped)
            break;
    }
}
```

Listing 6: Heap Sort [sorts/heap.c]

```
#include "sorts.h"

volatile int8_t cursor_last = -1;

void sift_down(
        uint8_t a[], uint8_t len,
        uint8_t start, uint8_t end, apply_fn fn) {

    uint8_t root = start;
    uint8_t tmp, child, swap;
    if (start != cursor_last) {
        fn(a, len, start);
        cursor_last = start;
    }

    while ((root << 1) + 1 <= end) {
        child = (root << 1) + 1;
        swap = root;
        if (a[swap] < a[child])
            swap = child;
        if (child + 1 <= end && a[swap] < a[child+1])
            swap = child + 1;
        if (swap != root) {
            tmp = a[swap];
            a[swap] = a[root];
            a[root] = tmp;
            if (swap != cursor_last) {
```

```
                fn(a, len, swap);
                cursor_last = swap;
            }
            root = swap;
        } else
            return;
    }
}

void heap_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn) {

    uint8_t end, tmp, start;
    start = (len - 2) >> 1;
    while (start > 0) {
        sift_down(a, len, start--, len - 1, fn);
    }
    sift_down(a, len, start, len - 1, fn);

    end = len - 1;
    while (end > 0) {
        if (cursor_last != 0) {
            fn(a, len, 0);
            cursor_last = 0;
        }
        tmp = a[end];
        a[end] = a[0];
        a[0] = tmp;
        if (cursor_last != end) {
            fn(a, len, end);
            cursor_last = end;
        }
        sift_down(a, len, 0, --end, fn);
    }
}
```

Listing 7: Selection Sort [sorts/selection.c]

```
#include "sorts.h"
#include "../util.h"

void selection_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn) {

    uint8_t i, j, k, t;
```

```
    for (i = 0; i < len; i++) {
        k = i;
        fn(a, len, i);
        for (j = i + 1; j < len; j++) {
            fn(a, len, j);
            if (a[j] < a[k]) {
                k = j;
            }
        }
        if (k != i) {
            if (k != j-1)
                fn(a, len, k);
            t = a[i];
            a[i] = a[k];
            a[k] = t;
        }
        if (i+1 != len)
            fn(a, len, i);
        else
            fn(a, len, -1);
    }
}
```

Listing 8: Insertion Sort [sorts/insertion.c]

```
#include "sorts.h"
#include "../util.h"

void insertion_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn) {

    uint8_t i, j, t;
    for (i = 1; i < len; i++) {
        fn(a, len, i);
        for (j = i; j > 0 && a[j] < a[j-1]; j--) {
            t = a[j];
            a[j] = a[j-1];
            a[j-1] = t;
            fn(a, len, j-1);
        }
    }

}
```

Listing 9: Shell Sort [sorts/shell.c]

```
#include "sorts.h"

void shell_sort_apply(
        uint8_t a[], uint8_t len, apply_fn fn) {

    uint8_t i, j, tmp;
    uint8_t h = 1;
    while (h < len / 3)
        h = 3 * h + 1;

    while (h >= 1) {
        for (i = h; i < len; i++) {
            fn(a, len, i);
            for (j = i; j >= h && a[j] < a[j-h]; j -= h) {
                tmp = a[j];
                a[j] = a[j-h];
                a[j-h] = tmp;
                fn(a, len, j-h);
            }
        }
        h /= 3;
    }

}
```

References

[1] Wikipedia, *Sorting algorithm — Wikipedia, the free encyclopedia*, 2012. [Online; accessed 4-December-2012].