Assignment 3 DESIGN.pdf

Roman Luo

February 5, 2023

1 Description of Program

1. Implement Shell Sort, Batcher Sort (Batcher's method), Heapsort, and recursive Quicksort in C language based on the provided Python pseudocode. The interfaces for these sorting algorithms will be given in the header files shell.h, batcher.h, heap.h, and quick.h and should not be modified.
2. Create a test harness in the file sorting.c to test the performance of the sorting algorithms by creating an array of pseudorandom elements and testing each of the sorts.
3. Gather statistics about each sort and its performance, including the size of the array, the number of moves required, and the number of comparisons required. Note that a comparison is counted only when two array elements are compared.

2 Files to be included in directory "asgn2":

- Major files
    1. batcher.c implements Batcher Sort.
    2. batcher.h specifies the interface to batcher.c.
    3. shell.c implements Shell Sort.
    4. shell.h specifies the interface to shell.c.
    5. gaps.h provides a gap sequence to be used by Shell sort.
    6. heap.c implements Heap Sort.
    7. heap.h specifies the interface to heap.c.
    8. quick.c implements recursive Quicksort.
    9. quick.h specifies the interface to quick.c.
    10. set.c implements bit-wise Set operations.
    11. set.h specifies the interface to set.c.
    12. stats.c implements the statistics module.
    13. stats.h specifies the interface to the statistics module.
    14. sorting.c contains main() and may contain any other functions necessary to complete the assignment.
- Minor files
    1. Makefile
    2. README.md
    3. DESIGN.pdf
    4. WRITEUP.pdf

3 Pseudocode / Sorting Functions:

3.1 shell.c

```
Void Function shell_sort(Stats, arr, length)
    gap_length = size of gap sequence
    for k = 0 to gap_length - 1
        gap = gap sequence at index k
        for i = gap to length - 1
            j = i
            temp = arr[i]
            while j >= gap and cmp,arr[j - gap] > temp
                arr[j] = arr[j - gap]
                j = j - gap
            end while
            arr[j] = temp
        end for
    end for
end procedure
```

3.2 batcher.c

```
Void Function comparator(Stats, x, y)
    IF cmp(Stats, x, y) = 1 THEN
        swap(Stats, x, y)


Void Function batcher_sort(Stats, A, n)
    IF n = 0 THEN
        RETURN

    t = 0
    WHILE (1 << t) < n DO
        t = t + 1

    p = 1 << (t - 1)

    WHILE p > 0 DO
        q = 1 << (t - 1)
        r = 0
        d = p

        WHILE d > 0 DO
            FOR i = 0 TO n - d - 1 DO
                IF (i & p) = r THEN
                    comparator(Stats, A[i], A[i + d])

            d = q - p
            q = q >> 1
            r = p
        p = p >> 1
```

3.3 heap.c

```
Void Function max_child(Stats *stats, uint32_t *A, uint32_t first, uint32_t last):
    left = 2 * first
```

```
    right = left + 1
    if (right <= last) and (cmp(stats, A[right - 1], A[left - 1]) == 1):
        return right
    return left

Void Function fix_heap(Stats *stats, uint32_t *A, uint32_t first, uint32_t last):
    found = false
    mother = first
    great = max_child(stats, A, mother, last)
    while (mother <= last / 2) and (not found):
        if (cmp(stats, A[mother - 1], A[great - 1]) == -1):
            swap(stats, &A[mother - 1], &A[great - 1])
            mother = great
            great = max_child(stats, A, mother, last)
        else:
            found = true

Void Function build_heap(Stats *stats, uint32_t *A, uint32_t first, uint32_t last):
    for (father = last / 2; father >= first; father--):
        fix_heap(stats, A, father, last)

Void Function heap_sort(Stats *stats, uint32_t *A, uint32_t n):
    first = 1
    last = n
    build_heap(stats, A, first, last)
    for (leaf = last; leaf >= first; leaf--):
        swap(stats, &A[first - 1], &A[leaf - 1])
        fix_heap(stats, A, first, leaf - 1)
```

3.4 quick.c

```
Void Function partition(Stats *stats, uint32_t *A, uint32_t lo, uint32_t hi)
    i = lo - 1
    for j = lo to hi - 1
        if cmp(stats, A[j - 1], A[hi - 1]) == -1
            i = i + 1
            swap(stats, &A[i - 1], &A[j - 1])
    swap(stats, &A[i], &A[hi - 1])
    return i + 1

Void Function quick_sorter(Stats *stats, uint32_t *A, uint32_t lo, uint32_t hi)
    if lo < hi
        p = partition(stats, A, lo, hi)
        quick_sorter(stats, A, lo, p - 1)
        quick_sorter(stats, A, p + 1, hi)

Void Function quick_sort(Stats *stats, uint32_t *A, uint32_t n)
    quick_sorter(stats, A, 1, n)
```

4 Pseudocode / set.c

4.1 set.c

```
Set set_empty(void) return 0
```

```
Set set_universal(void) return 32 bits fill with 1s;

Set set_insert(Set s, uint8_t x) return s or (1 << x);

Set set_remove(Set s, uint8_t x) return s and inverse(1 << x);

bool set_member(Set s, uint8_t x) return s and (1 << x);

Set set_union(Set s, Set t) return s or t;

Set set_intersect(Set s, Set t) return s and t;

Set set_difference(Set s, Set t) return s and inverse t;

Set set_complement(Set s) return inverse s;
```

5 Pseudocode / sorting / test harness:

5.1 sorting.c

```
    set Set that take options
    set Stats for each sorting method
    set array to NULL for each sorting method
    set char help message
    set defualt seed
    set defualt elements
    set defualt print numbers

    define a void function that print test data, with arguments:
    char my_program_name[], double my_value, char test_name[],
    double system_value, int terms, bool *s_check, bool *flag
        if the corresponding flag is true {
            print corresponding data based on structure given

            print using a for loop with specific style
        }

    define a void function that copy array

    //for plot, define another functions for reverse array

    Use int main(int argc, char **argv)
        set int opt = 0;

        while ((opt = getopt(argc, argv, OPTIONS)) != -1)
            set a switch with (opt)
            case s
                check if it is the only command, if so, print help message.
                break
            for each case, use the set_insert function to add the command to
            the set.

        if there is no command following, print help message

        use malloc to allocate memory for deafualt random array
```

```
the same for each sorting arrays
then use the copy fucntion to copy the default random array
for each sorting arrays

use set_member to check if a command is used:
a: do all sorting by calling each sort functions
else:
based on what is in the set, do specifc sortings

use the print test data function correspondingly for each of the test

free all the allocated memories
return 0;
```

6 Pseudocode / Makefile:

6.1 Makefile

- CC = clang must be specified.
- CFLAGS = -Wall -Wextra -Werror -Wpedantic must be specified.
- make must build the mathlib-test executable, as should make all and make mathlib-test.
- make clean must remove all files that are compiler generated.
- make format should format all your source code, including the header files.