Assignment 3: Design Doc

Madison Ormsby

January 27, 2022

1 Description

In this assignment, there will be four sorting algorithms that will be created using a python pseudo-program from the assignment doc. Using these four sorting algorithms, they will be implemented into another C file that will input pseudo-random arrays to test the sorting algorithms out. As the sorting algorithms work, another program will conduct some research by gathering the size of the array accepted into each algorithm, the number of moves required to finish the sort, and the number of comparisons needed. All of the sorts will be fed the same array to make it "fair" for them. The test program, called `sorting.c` will use a `getopt()` command to accept many command line options such as which sorting algorithms to enable and the size of the array that the algorithms will sort. This getopt loop will use a `set` to make sure no arguments are repeated, ensuring that the arguments may be input in whatever combination is needed. As the sorting algorithms work, statistics will be collected on them such as how many compares were used to reach a perfectly sorted set. These statistics will be stored in stats.c and will be output in a printf statement along with other statistics such as how many elements there are and which sort is being used.

2 Files Included

batcher.c A C program that will implement Batcher Sort to be used in `sorting.c`.

batcher.h A header file that has the interface for `batcher.c`.

insert.c A C program that will implement Insertion Sort to be used in `sorting.c`.

insert.h A header file that contains the interface for `insert.c`.

heap.c A C program that will implement Heap Sort to be used in `sorting.c`.

heap.h A header file that contains the interface for `heap.c`.

quick.c A C program that will implement Quick sort to be used in `sorting.c`.

quick.h A header file that contains the interface for `quick.c`.

set.h A file that uses boolean algebra to track which command-line options are used when `sorting.c` is run.

stats.c A C program that will be used to track the number of moves and comparisons. It will also contain extra functions to compare, swap, move, and reset elements.

stats.h A header file that contains the interface for `stats.c`.

sorting.c The main C function containing the program to randomly generate an array of numbers and have all four sorting algorithms test their capabilities along with printing out the results (SEE END OF LECTURE 01.21.2022)

Makefile The makefile used to build, clean, and format the C programs and header files shown above. It should compile `sorting.c` into an executable function.

README.md A text file using markdown formatting to briefly explain the program and the Makefile along with listing all possible parameters the program will accept.

DESIGN.pdf A pdf file written in LaTeXwith a detailed plan on coding the assignment along with a list of items to be included in the final submission.

WRITEUP.pdf A pdf file written in LaTeXcontaining an analysis of the code along with a couple of graphs to display data gathered. A brief self reflection on the commands chosen along with what I feel I learned most from this assignment will also be included.

3 Pseudocode

batcher.c

```
define a function "bitlength" (takes an int):
    len = 0
    bita = int
    for bita != 0:
            bita shifted right by 1
            len += 1
    return int len

define a function "comparator" (takes an array and two ints):
    compare if list[a] is -ge list[b] using cmp:
            swap list[a] and list[b]
    returns void

define a function "batchersort" (takes an array list):
    if the array length is 0, break
    set n to the length of the array (using a global var from sorting)
    t = bitlength(n)
    p = 1 shifted to the left by (t - 1)
    while p is -gt 0;
            d = p
            r = 0
            q = 1 shifted to the left by (t - 1) (NOTE NOT == P)
            while d is -gt 0:
                    while i (iteration var) is -le n - d:
                            if i AND p == r use comparator(array, i, i + d)
                    d = q - p
                    q = q shifted right by 1
                    r = p
            p = p shifted right by 1
    return void
```

insert.c

```
define a function "insert" (takes an array):
    while i -le length of array
            j = i
```

```
        use move to set i to A[i]
        while j -gt 0 and temp -lt A[j - 1]:
            use move to set A[j] to A[j - 1]
            j -= 1
        A[j] = temp
    return void
```

heap.c

```
  define a function "maxchild" (takes an array, int first, and int last)
    set left equal to 2 * first and right equal to left + 1
    if right is -le last AND use cmp to see if A[right -1] -gt A[left -1]
    return right
    else return left

  define a function "fixheap" (takes an array, int first, int last)
    set found to False, mother to first, and great to maxchild(A, mother, last)
    while mother is -ge to last divided by two and rounded down, AND found is False
        use cmp to see if the position of mother - 1 in the array is -lt the position of great - 1
            switch the values
            let mother = great
            let great be the maxchild of the array
        else set found to True
    return void
  define a function "buildheap" (takes an array, int first, int last)
    for father in range of floor(last/2) to (first - 1) with an increment of -1
        fixheap with array, father, and last

  define a function "heapsort" (takes an array)
    set first equal to 1
    set last to the length of the array (global var)
    build the heap using buildheap(A, first, last)
    for leaf in range last to first with an increment of -1;
        swap A[first -1] and A[leaf -1] using stats.c
        fix the heap using fixheap(A, first, leaf -1)
```

quick.c

```
  define a function "partition" (Takes an array, int low, and int high)
    set i to low - 1
    for j in range of low to high with increment 1
        use cmp() to see if array[j - 1] is less than array[i - 1];
            add 1 to i
            swap A[i - 1] and A[j - 1]
    swap A[i] and A[high - 1]
    return i + 1

  define a function "quicksorter" (takes an array, int low, and int high)
    if low is -lt high
        set p to partition(array, low, high)
        recursively use quicksorter(array, low, p - 1)
```

```
            recursively use quicksorter(array, p + 1, high)

    define a function "quicksort" (takes an array):
        use quicksorter(array, 1, length of array)
```

sorting.c

```
    define a function "help" (takes void)
        prints out a summary and uses along with all options

    create a function to print out the sorted array

    define a function "main" (takes int argc, char **argv)
        int seed = 13371453
        int size = 100
        int elements = 100
        int opt = 0
        function pointer var that accepts(array)
        create a while loop to take getopt args "ahbiqr:n:p:H"
                use a switch (opt) to take args
                case "a":
                        insert h, b, i, and q into set
                case "h":
                        insert h into set
                case "b":
                        insert b into set
                case "i":
                        insert i into set
                case "q":
                        insert q into set
                case "r":
                        set seed to optarg
                case "n":
                        set size to optarg
                case "p":
                        set elements to optarg
                default:
                        print help function
        dynamically allocate memory to an array
        seed the random number generator with seed
        for i in range 0 to size incrementing i by 1;
                add random numbers to the array
        if set contains i:
                create a copy of the array (arrayi)
                printf "Insertion Sort, %d elements, %d moves, %d compares"
                insertionsort(arrayi)
                print out the sorted array in columns of 5 with a loop
        repeat for h b and q
        free the array
```