Assignment 3 DESIGN.pdf

Victor Nguyen

January 24, 2022

1 Description of Program:

This program will generate a pseudo random sequence of numbers and put them into an array. Afterwards, the user will be allowed to specify which sorting algorithm to use on the array. The user may also specify a variety of other things such as:

- r seed: Set the random seed for the pseudo random number generator. Default seed is 13371453.
- n size: Set the size of the array. Default size is 100.
- p elements: Set the number of elements to display. Default size is 100.

The user may also specify what sorting algorithm to use on a specified array. The sorting algorithms that this program allows are:

- Insertion Sort
- Heap Sort
- Quick Sort
- Batcher Sort

2 Files to be included in directory "asgn3":

1. sorting.c
    - Source file that contains main() and is responsible for printing out the sorting algorithms.
2. insert.c
    - Source code that contains the implementation of insertion sort.
    - Code derived from asgn3.pdf.
3. insert.h
    - Header file that contains the function declaration for insert.c. Useful for sharing between other source files.
    - File was given by the professor.

4. heap.c
    - Source code that contains the implementation of heap sort.
    - Code derived from asgn3.pdf.
5. heap.h
    - Header file that contains the function declaration for heap.c. Useful for sharing between other source files.
    - File was given by professor.
6. quick.c
    - Source code that contains the implementation of quick sort.
    - Code derived from asgn3.pdf.
7. quick.h
    - Header file that contains the function declaration for heap.h. Useful for sharing between other source files.
    - File was given by professor.
8. batcher.c
    - Source code that contains the implementation of batcher sort.
    - Code derived from asgn3.pdf.
9. batcher.h
    - Header file that contains the function declaration for batcher.c. Useful for sharing between other source files.
    - File was given by professor.
10. stats.c
    - Source code for the struct stat that is used to count the number of comparisons and moves used for specific sorting algorithms.
    - File was given by the professor.
11. stats.h
    - Header file that contains the struct and other functions used by stats.c. Useful for sharing between other source files.
    - File was given by professor.
12. set.h
    - Header file that contains the functions for set operations. Useful for sharing between other source files.

- File was given by professor.

13. test.c
    - Source file that I used to test random things.
14. Makefile
    - File that formats the program into clang-format.
    - File also compiles sorting.c and test.c.
    - Cleans files that were generated during the compilation process.
    - File also contains scan-build and valgrind to check for any errors/memory leaks.
15. README.md
    - A text file that's in Markdown format that describes how you would build/run the program.
16. DESIGN.pdf
    - Describes the design and thought process of the main program.

3 Pseudocode / Structure for sorting.c:

```
I needed to implement a usage page for when the user inputs nothing
or incorrectly. To do this, I created a function called usage that takes in
no arguments. The main point of this function is to display text to the user
about specific parameters and how to run the program correctly.


Then, get the user input on the sorting algorithm that should be used (user may
also specify all or some of the algorithms) and store them into the set.

See if the user specified the length of the array, number of elements to print,
and seed to store for later use.

Set the set s to be an empty set, initialize the seed, the length n, and the
element count p.

Create an array that contains the sorting algorithm names.

Allocate space on the heap so that we can store the random
values onto the heap location.

Create a function array so that we can call a specific
sorting algorithm to use later.

Iterate over the set:
```

```
    If a sorting algorithm is in the set:
        Initialize/reset the random seed
        Initialize the struct
        Run the function using the function array and pass it the struct name
            and the required parameters
        Print the function name, length of array, no. of moves, and no. of
            comparisons.
        Print out the numbers in the array.
        Reinitialize the p counter to 0.
Free the heap
Return 0
```

3.1 NOTE ABOUT THE PSEUDOCODE:

- I skipped over explaining the pseudocode implementations of the individual sorting algorithms because the code was given in python inside the assignment 3 pdf. I do want to mention that I added structs as an argument for the functions inside the sorting algorithms during the translation of the code. I also used the stats.c functions to keep track of the comparisons and moves that the sorting algorithm used.
- The program should have default parameters for n (length), p (elements), and r (seed). Those of which should be 100, 100, and 13371453.
- The program must include a getopt to take in the user's command line arguments.
- Sets were used to enable/disable specific algorithms to be executed. This was one of the requirements for this assignment.
- The array of names is useful for selecting and displaying the correct sorting algorithm name during execution.
- Allocating space onto the heap allows us to compute array sizes that are required for this assignment.
- The function array will allow us to call functions in a more generalized way, reducing the amount of code needed and makes things look cleaner.
- We need to initialize a random seed so that we can generate pseudo random numbers for the array. We can then use a specific algorithm to sort the array. Note: We should create a function for this and should use this to reinitialize arrays if multiple sorting algorithms are to be used. This is so that we don't try to sort an already sorted array, or sort a completely new array. We should also mask the number so that it isn't above 30 bits.
- The struct is used to collect data on the number of moves and comparisons we've done during a sorting algorithm. This serves the purpose of resetting the struct if multiple sorting algorithms were called (so that they don't overlap/increment).
- Printing out the numbers inside the array should only be a width of 13 and should be displayed with 5 columns. Also if the user specified the amount of number to print out, we should have an

if statement to check if we've reached that number and stop printing numbers. Reinitialize the p counter after every sorting algorithm.

- Free the space used on the heap so that we free up unused memory.

4 Error Handling:

Some problems and solutions I found while coding.

1. If the user ever inputs anything incorrectly, or nothing at all I will display a usage page.
2. Some invalid entries like putting in strings will cause the program to not print out the desired results. Strings will cause the program to print out only the names of the insertions but will contain 0 comparisons, 0 moves, and no numbers inside the array.

5 Credit:

- For the sorting algorithms, I followed the python code (that was provided in the assignment 3 pdf) and translated it to C.
- I attended Eugene's section on January 21st. He helped me understand the sorting algorithms, how to use structs, how to use sets, and usage/application of malloc/free.
- Toomai in discord provided a snippet of code on how to compute the binary length of a number. Although I didn't use his function, I thought I might mention this since the code I wrote to find the binary length of a number looks similar to his. This function can be found in the batcher.c file.
- Audrey's example design pdf helped me structure my own design pdf. This can be found on this quarters cse13s discord server in the asgn-1-faq-n-tips channel.
- I used my asgn2 integrate.c from asgn2 as reference and inspiration to implementing my switch cases, and setting up function pointers.
- I've copied the Synopsis help page for the ./sorting executable. This was given through the resources repository in assignment 3.
- Ben's tutorial on how to compile and link multiple files together was really helpful. I also followed his scan-build guide on the hilalmorrar.com website.
- I followed a stack overflow on how to use valgrind. I then implemented it into my makefile. Link is: stackoverflow.com/questions/5134891/how-do-i-use-valgrind-to-find-memory-leaks
- I've used my own Makefile from asgn2 as reference to build the makefile for this assignment.
- On discord, the user Gecko10000#7137 and RixinLi#6370 gave me an idea to use a function array so that I can call my sorting algorithms in a more generalized way.