Tim Kraemer

Assignment 3 Design Document

The purpose of the program:

The purpose of this program is to test out different sorting algorithms, and compare them with different parameters such as size of array to be sorted, seed for randomness of array values, etc to see which sorting algorithms fare better than the others depending on these set parameters. This program also is a way to improve our understanding and implementation of pointers, structures, and ADTs.

How it works:

How the program will work is somewhat the same as our last assignment. The program will ask the user for input on what they want the program to do. This includes: running any test(s), changing the seed for the random values in the array, changing the length of the array, changing how many elements you want printed, or a helpful help message on what options are available. The output will be which sort(s) you wanted printed, the statistics of the sort (how many comparisons and moves there were), as well as the first elements that you want printed (could be 0). The way this works in the test harness is we use the function getopt to look for specific input (and specific arguments for some of the inputs). If it sees one or more of the specific inputs, it will go to its corresponding switch case, which will then flip a corresponding bit value from 0 to 1 in our set. Later in our code, after declaring and allocating our array and our stats structure and checking to make sure the seed input is valid,we have if statements that checks which bits are flipped in our set, and will print accordingly based on which bits were flipped. I also wrote functions above our main function that facilitate the printing, array filling, and reseeding of our random function in our if statements to make it much more readable and clean.

-a	Prints all sorts
-e	Prints heap sort
-i	Prints insertion sort
-s	Prints shell sort
-q	Prints quicksort
-r	Choose the seed for the random values in our array
-n	Choose the size of the array
-p	Choose the amount of elements to be printed
-h	Prints help message

The implementation:

First implement and test each of the sorting algorithms so that understanding on how it works is clear. Next, create the test harness, such that it is easier to test out your sorting algorithms without doing the testing in its own C file (afterwards, the test harness will have user input with Sets implemented, functions for print neatness and readability, as well as array and stats structure memory allocation). Next, create a makefile so cleaning, formatting, and compiling is a breeze. Next, implement the functions of the stats structure, such that you replace the comparisons and swaps in the sorting algorithms with the functions of the structure.

Insertion Sort

Shell Sort

Heap Sort

Quick Sort

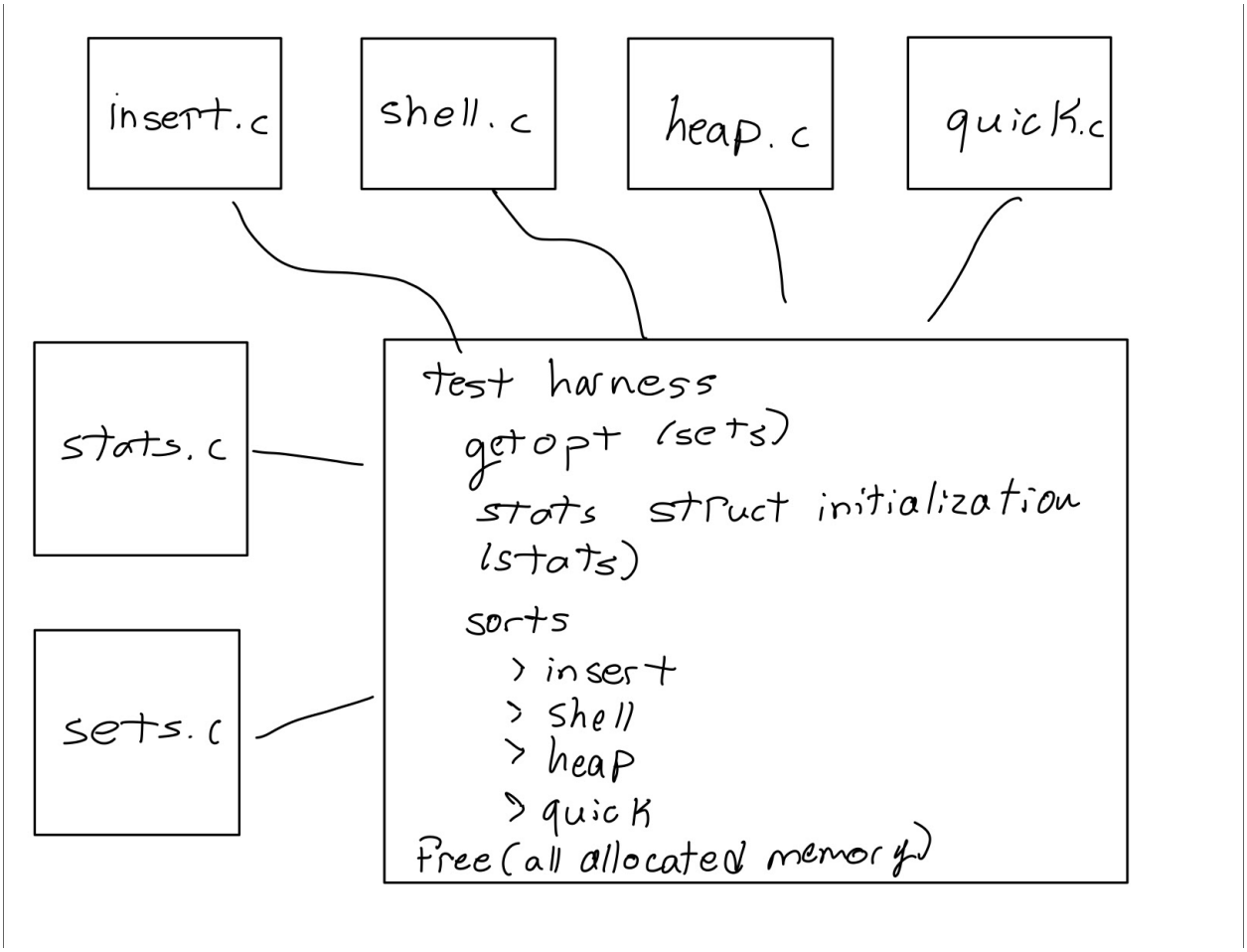
For these we will use the pseudocode provided to us in the assignment document.

Test Harness:

- Create print functions to make the file neater
- Getopt with the sets (binary operations), flip a specific bit if a certain input
- Initialize the array, seed random, reseed and re-enter values in array after every sort (was done in our print functions for neatness)
- Perform sort based on input (use set function and if statements to check specific input)
- Print out columns of sorted array

In this assignment, we allocate memory for our declared stats structure, as well as the declared array to be sorted (since we want to be able to use the max amount of space given to us in memory if so necessary).

A helpful representation on what c files are needed for our test harness, and the main portions of our test harness:



Pseudocode

Insertion:

```
for i from 1 to length A
        J = i
        Temp = A[i]
        While j greater than 0 and temp less than A[j-1]
                A[j] = a[j-1]
                J -=1
        A[j] = temp
```

Shell:

```
Define gaps function
        For i in range (log function ) to 0 with -1 steps
        yield (3^i - 1) // 2

^this can be integrated into the main sorting function

Define shell_sort function
        For gap gaps(length of a)
                For i in range gap to length a
                        J = i
                        Temp = a[i]
                        While j >= gap and temp < a[j - gap]
                                A[j] = a[j-gap]
                                J -= gap
                        A[j] = temp
```

Heapsort:

```
Define max child function args: array, first int, last int
        Left = 2 * first
        Right = left + 1
        If right <= last and a[right - 1] > a[left - 1]
                Return right
        Return left

Define fix heap function args: array, first int, last int
        Found = false
        Mother = first
        Great = max child(array, mother, last)
        While mother <= last // 2 and !(found)
                If a[mother - 1] < a[great - 1]
                        A[mother - 1] = a[great - 1]
                        A[great - 1] = a[mother - 1] (use a temp variable)
                        Mother = great
                        Great = max child(array, mother, last)
                Else
                        found= true
Def build_heap(array, first int, last int)
        For father in range from last // 2 to first - 1 in steps -1
                Fix heap(array, father, last)

Define heap_sort(array)
```

```
    First = 1
    Last = length of array
    build _heap(array, first, last)
    For leaf in range from last to first in steps of -1
        A[first - 1] = a[leaf - 1]
        A[leaf - 1] = a[first - 1]
        fix_heap(array, first, leaf - 1)
```

Quick sort:

```
Define partition function(array, integer lo, integer hi)
    I = lo-1
    For j in range(lo, hi)
        If a[j - 1] < a[hi - 1]
            I += 1
            A[i - 1] = a[j - 1]
            A[j - 1] = a[i - 1]
    a[i] = a[hi - 1]
    A[hi - 1] = a[1]
    Return i + 1

Define quick sorter(array, integer lo, integer hi)
    If lo < hi
        P = partition(a, lo, hi)
        quick_sorter(a, lo, p-1)
        quick_sorter(a, p+1, hi)

Define quick_sort function(array)
    quick_sorter(array, 1, length of array)
```