Assignment 3 - Sorting

About this program:

This assignment utilizes the implementation of multiple sorting algorithms in order to understand them and get a feel for computational complexity.

Files:

Insert.c = implements Insertion Sort.
Insert.h = specifies the interface to insert.c
Heap.c = implements Heap Sort.
Heap.h = specifies the interface to heap.c
Quick.c = implements recursive Quicksort.
Quick.h = specifies the interface to quick.c
Set.h = implements and specifies the interface for the set ADT.
Stats.c = implements the statistics module.
Stats.h = specifies the interface to the statistics module.
Shell.c = implements Shell Sort.
Shell.h = specifies the interface to
Sorting.c = contains main()and may contain any other functions necessary to complete the assignment.

Pseudocode:

Insert.c : // This code was given by the python pseudocode and Professor Long's C code in lecture

1. Loop through the list
2. Save current element to a temporary value
3. Compare the current value with all previous values to see if they are all/ any are greater than the current value. If they are then move the previous value to the current value
4. Move the temporary value to the current value

Sorting.c: // some of this code was given through Eugene's section on 10/12 (comments on actual code for which lines were given)

1. Include all header files and define all the options
2. Use enum for all the sort types
3. While loop to parse through options accordingly
    a. Use insert set with enum names to record which sorts are wanted
    b. For seed and size, get input and save it with atoi(optarg)
    c. For help message, if help option is chosen then only print message and quit

4. Set random seed and declare array using dynamic memory allocation (from Eugene's section)
    a. Randomize array and apply bitmask for only 30 bits
5. Loop through set made from parsing through command line
    a. Check what sort is part of the set then do that sort

Heap.c: // This code was given by the python pseudocode

1. Max child
    a. Find max child node from the parent node by comparing array with referenced indexes
    b. Return accordingly
2. Fix heap
    a. Get the max child node from parent node
    b. Compare nodes and swap accordingly
        i. Fixes heap to fit restrictions of the max heap
3. Build heap
    a. Call on fix heap after moving largest element from the heap
4. Heap sort
    a. Build heap
    b. Move large element to end
    c. Fix the heap

Quick.c // This code was given by the python pseudocode

1. Partition
    a. Split the array and compare elements to the left and right of it, swap accordingly
2. Quick sorter
    a. Recursively get partition and call on self until the array is sorted
3. Quick sort
    a. Call on quick sorter

Shell.c // This code was given by the python pseudocode

1. Shell sort
    a. Get largest possible gap
    b. Loop through and create new gaps to use and compare elements of the array with
    c. Move to put in increasing order

Brainstorming/Process:

- Test harness
    - Switch in while loop to enter all options into a set

- Exit while loop when done parsing
    - Use eugene code for enums, check what enum value in set and do sort accordingly
        - Ex: for loop until x < num sorts (or 4)
            - If 0 in set, that means do insertion sort
            - If 1 in set, do heap sort
            - If 2 in set do quick
            - If 3 in set do shell
            - Otherwise continue

Errors/Bugs:

The -a option does not print the stats the same as when each sort is called on through the other options, the moves and compares vary as opposed to calling -i -s -e -q

Lab Document Notes

1. Insertion Sort
    a. Considers elements one at a time, placing them in correct ordered position
    b. Compares kth element with each of preceding elements in descending order
2. Shell Sort
    a. First sorts pairs of elements far away from each other, distance is called gap
    b. Keeps iterating until gap of one is used
3. Heapsort
    a. Max heap
        i. Parent node has value >= to value of children
    b. Min heap
        i. Parent node has value <= to value of children
    c. Building a heap
        i. Order array into max heap, largest element is first
    d. Fixing a heap
        i. Largest array elements repeatedly removed from top and placed at end of sorted array (if array in increasing order)
4. Quicksort
    a. Finds pivot in array and places elements either to left or right
5. Your Task
    a. Implementing sorting functions in C from given python
    b. Implementing test harness
        i. Array of pseudorandom elements
    c. Statistics on each sort
        i. Size of array, moves required, comparisons required