CSE 13S Winter Quarter 2022
Assignment 3: Sorting: Putting your affairs in order

Description of the Program:

The goal of this assignment is to familiarize ourselves with different sorting algorithms and to ultimately determine which sorting algorithms are the most efficient. In this assignment we will be implementing quick sort, batcher sort, heap sort, and insert sort. To observe each of their efficiencies, we will be recording each method's number of elements, moves, and comparisons.

Files to be included in the "asgn1" directory

- insert.c and insert.h
    - insert.c is where we are to implement the insert sorting algorithm, and isert.h specifies the interface to insert.c
- batcher.c and batcher.h
    - batcher.c is where we are to implement the batcher sorting algorithm and batcher.h specifies the interface to batcher.c
- heap.c and heap.h
    - heap.c is where we are to implement the heap sorting algorithm and heap.h specifies the interface to heap.c
- quick.c and quick.h
    - quick.c is where we are to implement the quick sorting algorithm and quick.h specifies the interface to quick.c
- sorting.c
    - This is the code we are supposed to write to implement all of the sorting algorithms and contains the main function
- stats.c and stats.h
    - stats.c helps display the statistics and stats.h specifies the interface to stats.c
- sets.h
    - set.h specifies the interface for the set ADT
- Makefile
    - This is used to help compile our code.
- Makefile
    - This is also given to us and is used to help compile our code.
- README.md
    - This is a file written in markdown format to explain the purpose of the code and how to build it, in addition to any errors we found.
- DESIGN.pdf
    - Gives insight into some of the processes of coming up with the code, along with the purpose of the assignment.

Purpose:

- The purpose of this assignment is to employ four different sorting algorithms and ultimately compare their efficiency by keeping track of the number of moves and comparisons were used.

Layout/Structure:

- This assignment makes use of four separate c files for the four different sorting algorithms.
- Additionally, each sorting algorithm includes a header file for the statistics module, stats.c
    - #include “stats.h”
- Finally, all four sorting algorithms are linked to the main sorting.c program using header files once again.
- Additionally, we must include the set.h header file in order to use sets in as part of our command-line option code.

Insert:

- Insertion sort works by looking at a given value in an array, starting with the second value, indices 1.
    - Insertions sort takes this value and loops over its predecessors or all of the values that come before it.
    - Then, insertion sort places the value in the array to the right of the first value it is larger than, and if it is larger than its predecessor, it does not move.
- Pseudocode:
    - Pseudocode was provided in python in the assignment document.
    - The main differences include having three arguments when calling the function instead of just the array A[]
        - We include Stats *stats, and uint32_t n
            - Stats *stats gives us the pointer to update the stats.moves and stats.comparisons
            - The uint32_t n gives us the length of the array, rather than calling len(A)
        - Additionally, when assigning the variables, such as temp = A[i], we instead use the move operation to update the stats.moves for the program.
        - Similarly, instead of comparing temp and A[j - 1] in while loop, we use cmp instead to update the stats.comparisons for the program.
    - The use of moves and cmp are universal in all of the sorting algorithms.

Heap:

- Heap sort works first by creating a heap. A heap in this case is a binary tree and goes in descending value, meaning that the parent nodes are larger than the child nodes.
    - Next, the largest value of the heap is removed, and so the heap has to be fixed to maintain the properties of a binary tree with larger parent nodes.
    - Finally, we simply build the heap, remove the largest value using a decreasing for loop, and continuously fix the heap.
- Pseudocode:
    - The Pseudocode was provided in python
    - Once again we need to include the Stats *stats argument not only in just the heap_sort function, but more importantly in all the functions, like fix_heap, and build_heap.
        - This is because all of the comparing, moving, and swapping is done inside of the smaller functions, while the heap_sort function simply just calls the other functions.
    - Once again we use cmp in order to update the stats.comparisons.
    - However, we no lumber use stats.moves, as it is only used in insertion sort, and now use swap in order to update the stats.moves

Quicksort:

- Quicksort works by creating pivots where values smaller than the pivot and values larger than the pivot are separated.
    - This happens over and over again until the array is sorted
- Pseudocode:
    - The pseudocode was provided in python
    - We follow the same rules as before, including Stats *stats as an argument and using cmp and swap in order to keep track of the values

Batchersort:

- Batcher sort works essentially by creating a comparison network so that every value can be compared with one another easier.
    - Batcher sort is easily the most complex function
- Pseudocode:
    - Here we follow the same protocol, including Stats *stats as an argument for all functions and remembering to use cmp and swp to update statistics values.

Sorting:

- This is the first we are writing this assignment without any provided pseudocode.
- The function of this program is to employ all of the sorting algorithms

- Pseudocode:

```
- def sorting():
    - initialize stats
    - Set s = empty set
    - Set seed(13371453)
    - initialize size and elements = 100, 100
    - use getopt loop to get arguments:
        - if a:
            - run all sorting algorithms
        - if i:
            - run insertion sort
        - if q:
            - run quicksort
        - if b:
            - run batchersort
        - if h:
            - run heapsort
        - if H:
            - print(synopsis and system usage)
        - if n:
            - size = input
        - if r:
            - set seed(input)
        - if p:
            - elements = input
    - allocate memory to arrays A and a
    - for (i in range(size)):
        - A[i] = random() - bitmask
        - a[i] = A[i]
```

- Notes:
    - When getopt loop use case and edit the set
    - use set.h and member_set in order to ultimately run all of the sorting algorithms.
    - Make sure to reset the stats before running each of the sorting algorithms
    - Make sure to reset the array to an unsorted array, so that when running two or more sorting algorithms at once, you don’t pass a sorted array as the argument for a function.
    - print the name of the function, number of moves, and comparisons
    - print the first element number of elements in the array

- Comparisons and Moves:
    - comparisons and moves are tracked via the stats.c module as a struct
    - each comparison increments stats.comparisons by one
    - each move increments stats.moves by one
    - each swap increments stats.moves by three
        - swapping two elements of an array results in three moves
        - given values a and b
            - temp = a
            - a = b
            - b = temp
            - hence the three moves