Alex Asch

Assignment 3 Design

Description of program

This program consists of 11 files which together make up the necessary code for sorting and the necessary interfaces for the sorting algorithms. The types of algorithms to be implemented are insert, heap, shell, and quick sorts. The program will also record the relative efficiencies of these methods by recording the size of the array, the number of moves required, and the number of comparisons.

Files Included

- insert.c
    - The implementation of insertion sort.
- insert.h
    - The interface of insert.c.
    - Given in the resources repo. (© Professor Long 2021)
- heap.c
    - implements heap sort.
- heap.h
    - The interface of heap sort.
    - Given in the resources repo. (© Professor Long 2021)
- quick.c
    - Implements recursive quicksort.
- quick.h
    - Specified interface to quick.c.
    - Given in the resources repo. (© Professor Long 2021)
- set.h
    - Implements and specifies the interface for the set.
    - Given in the resources repo.
- stats.c
    - Implements statistics module
    - Given in resources repo. (© Professor Long 2021)
- stats.h
    - Specifies the interface to the statistics module.
    - Given in the resources repo. (© Professor Long 2021)
- shell.c
    - Implements Shell sort.
- shell.h
    - Specifies interface to shell.c.
    - Given in the resources repo (© Professor Long 2021).
- sorting.c
    - Contains main() and may contain any other functions necessary for testing.
    - This is the test harness for the rest of the code.

Pseudocode/Notes

- **All sorting algorithm code inspired by Professor Long's python code in the assignment document.**

Insertion sort

- Iterate over the given array starting at the second element.
    - Set a placeholder variable to the current index
    - Set a placeholder variable to the current indexed array element.
    - Iterate through the elements in the array that are less than the placeholder index while the element to the left of the current element is less than the temporary element.
        - While this is true, set the current element equal to the element to its left.
        - Decrement the current element pointer by one.
        - What this does in essence is, while the value we are iterating for is greater than the value we are at, shift the entire array rightwards by one (starting by moving to where our original element was) until we find a place where the element to the left is less than the current element. Then we set our current value to the temp element.
    - Set the element that was iterated to the temporary value.

Shell Sort

- This sorting method uses a gap between two values to compare a value with the value that is gap length away.
- We will start with a gap of n^3/2, calculated when n is the length of the array we are sorting., and each subsequent gap is decreased via the given pattern.
    - In this case, the gaps are computed as (3^k-1)/2, with the largest k being log(2n+3)/log(3). Which when substituted back gives us the largest gap as (n-1).
    - Using this we will make an array of gaps from the largest down to gap 1.
    - **Note:**instead of making an array, we can iteratively calculate the gaps after the first one. Calculate the first gap outside of the loop, then iterate and change the gap.
- Now, with that array in place, we will iterate using a loop with the above gap value from max to 1.
    - Then we will iterate through all of the elements in the range from gap to the length of the array to sort.
        - Set a temporary variable equal to the current index of the sorting array, j = i.
        - Set a temporary value to the current element of the array at i.
        - While j is greater than the gap variable, and the temporary value is less than the element that is gap distance to the left of j.
            - The element at j is equal to the element the gap distance away.
            - Decrement j by gap
        - A[j], the element at the final j value is equal to the temporary variable.

HeapSort

- This sorting type uses a data structure known as a heap to sort the data.
    - The heap is a special binary tree.
    - We will be using a max heap.
        - This is a binary tree where the parent node is always greater than or equal to the child nodes.
        - The heap is represented as an array, for which for an index k, the children are 2k and 2k+1.
    - Heapsort has two separate parts of the sort.

        - Building a heap.
            - The first routine is to take an array and build a heap that follows the rules of a heap.
        - Fixing the heap.
            - The second route is to sort the array.
            - We remove from the top of the heap and place at the end of the sorted array.
            - After that we n.eed to fix the heap to make it obey the constraints of a heap
    - The first function heap_sort() contains the calls to the rest of the function.
        - set the first element to 1.
        - set the last size of the array.
        - Call build heap with the array and the above first and last values.
            - build_heap().
                - This pseudo assumes one based indexing, indexes will be incremented and decremented as needed.
                - This function takes an array, and two ints.
                - The integers are the first and last indices of the array.
                - It will loop over the first half of the array, and call fix_heap on each element that it iterates over.
                - This will build a binary heap to begin heapsort with.
            - fix_heap()
                - This is the heap maintenance function, it will make sure that the heap follows the constraints given, in our case, it maintains it as a max heap.
                - This function takes the array, the parentindex(1) which will be decremented in heap_sort(), and the last index, which is the length of the array.
                - Find the max child of the given mother value, set that equal to a placeholder variable.
                - While the index of the mother variable is less than halfway through the array(aka still has child values), and the mother value is less than its max child, swap the values of mother and child, set the mother index counter to that of the max child, and set the placeholder variable to the largest child value.
                - In essence, what this does is takes an unordered heap, and starting at the element it is given, will check if it has children greater than it, swaps the value with the greatest child, if the now swapped child variable has children it will repeat this process until there are no more children, or the parent is greater than both the children.
            - max_child()
                - This returns the largest child of a given parent.

        - It will evaluate with left as the default case, and right as the checking case, since there is a possibility that the parent does not have a rightchild.
    - heap_sort()
        - This will use the build heap and fix heap functions to heap sort the array.
        - It will set the first and last variables as the length of the array.
        - It will then build heap, to make a max heap.
        - After this, it will iterate through the elements from the largest to smallest,
            - It will swap the current element at the front, with the current iteration.
            - It then fixes the heap, using the first element as the starting parent.
        - This iteration, in summary, takes the top parent in a max heap, and moves it to the end of the heap. Which puts the value in the correct index.
        - It then fixes the heap, which is now one smaller, as the last value is set.
        - It repeats this until there is no more heap, and a sorted array in its place
        - It is important to note that both heap and the sorted array are the same array, just divided at a point.

Quick sort

- Recursive sorting algorithm that makes use of partitions.
- quick_sorter(A: list, lo: int, hi:int)
    - if lo < hi (this makes sure that the function has at least two values to compare).
    - Call the partition function on the given array, and store the returned value as p.
    - Call self twice, once on the range from low to p-1, and once on the range from p+1, to the high value.
    - This in essence, creates two partitions and then continues to partition the array recursively until each partition is a length of 2.
- partition()
    - This takes an array, a low value, and a high value.
        - This begins by setting i to the index one less than the first element of the array.
        - Iterate from low to high.
            - If the current element of the iteration is less than the high value, increment i by one, and swap the value at i with the value of the current element.
        - After iterating, swap the value at i with the value at high (this is the pivot).
        - Return the value one greater than i.

Sorting.c

- This is the testing harness for sorting algorithms implemented above.
- This takes command line arguments, and uses a set to track the given arguments.
- This will loop through the given command line arguments, and insert into the set the elements that are included (that aren't elements that specify values after args.
- Implementing a method inspired by code posted by Professor Long on looping through a list of function pointers.
- Create an enumerated type with all of the possible command line arguments as the enums.
- While in list of command line arguments
    - if argument -x is in the list
        - Add x to the set of arguments
    - if argument -x num is in the list
        - Take the appropriate action for this ie. set the variable for printing amount.
        - In here, also check if it is a valid value, and use a default value if not.
- Loop through the enumerated type and check and call all the relevant functions with properly implemented arrays.
    - How to do this.
        - Loop through enumerated types.
            - If the value of the enum is found in the set.
            - Allocate space for an array of the input length.
            - Loop through the array allocated and set each number to a random bitmasked int
            - Call a sorting function with that array.
            - Free the allocated memory.