Assignment 3 – Sets and Sorting

Eric Lee

CSE 13S – Spring 2023

Purpose

This program implements different sorting algorithms to sort any list of numbers of any size. The algorithms it uses include insertion sort, shell sort, heapsort, quicksort, and batcher sort. The user chooses which algorithms they want to run using the command line options. They can also choose the random seed, array size, and number of elements printed.

How to Use the Program

The program runs from compiling and running the file ‘sorting.c’. The user has to use command line options in the terminal to specify what they want from the program. The following is a list of the each command and what they do:

- -a: runs all the algorithms
- -i: runs Insertion Sort
- -s: runs Shell Sort
- -h: runs Heap Sort
- -q: runs Quick Sort
- -b: runs Batcher Sort
- -r seed: sets the random seed which the user enters. The default seed is 13371453.
- -n size: sets the size of the array which the user enters. The default size is 100.
- -p elements: sets the amount of elements printed which the user enters. The default size is 100 and a size of 0 will print nothing.

Program Design

The program implements a different file for each sorting algorithm. The file for insertion sort is in ‘insert.c’. The other algorithm files include ‘shell.c’, ‘heap.c’, ‘quick.c’, and ‘batcher.c’ for shell sort, heap sort, quick sort, and batch sort, respectively. To compute operations like comparing, swapping, and moving variables, the functions are called from the file ‘stats.h’ instead of the normal functions in c. These special functions are used to keep track of each time the function is used to print the stats later. The file ‘gaps.h’ is used to get the size of the gap used in shell sort.

The file ‘sorting.c’ is the main file which calls all the other algorithm files. The command line options are implemented in this file. It also generates all the random numbers that are sorted using a user defined srandom seed.

Data Structures and Functions Descriptions

For insertion sort, a for loop is used with a while loop nested inside. A temporary variable is used to hold an element of the array to compare with the next one.

For shell sort, two nested for loops are used with a while loop nested in the inside one. A temp variable is also used to compare two elements, and the value of the gap is taken from ‘gap.h’.

For heap sort, the function ‘max_child’ is used to get the max value child node. The function ‘fix_heap’ resets the heap array according to the heap rules. The function ‘build_heap’ creates the heap array and ‘heap_sort’ uses a for loop to sort the array.

For quick sort, the function ‘partition’ sorts the array so that the greater values are to the right of the pivot index and the lesser values are on the left of the pivot. The function ‘quick_sort’ is used to call ‘quick_sorter’ which runs recursively until the array is sorted.

For batcher sort, the function ‘comparator’ compares and swaps two elements of the array. The function ‘batcher_sort’ sorts the array using two nested while loops and calling ‘comparator’ to check and swap elements.

Algorithms

Insertion sort works by taking one element in the array and comparing it with the next element until one is found that is lesser than the original one.

Shell sort works the same way as insertion except it doesn’t compare it to the next element but the one that is a certain distance or ‘gap’ away from the original.

Heap sort uses an array that is organized to be a heap, with the parent node value being larger than its two children node values. The largest element is taken to build the sorted array from greatest to least, with the heap being reset every time.

Quick sort divides the array into sub-arrays using a pivot point. All numbers greater than the pivot go to the right sub-array and the lesser numbers go to the left. To sort the sub-arrays correctly the program uses the function ‘partition’. It then divides the sub-arrays recursively until they are all sorted.

Batcher Sort uses a sorting network that connects the items in the array using comparators (or the function ‘comparator’ in this case). It sorts the even and odd sequences separately and combines them in the end.

Results

All my sorting functions work as expected, as well as ‘sorting.c’. The number moves and comparisons don’t match up exactly with the ‘sorting-arm’, but I’m pretty certain that it’s only because I may have used them in different places. The functions from ‘stats.c’ can be used almost everywhere in my code, so small differences here and there may have affected the total stats even though the function is essentially the same. I also couldn’t figure out how to generate a graph in time for this project.