Assignment 4: Implementing sorting techniques and gathering statistics from them

James Contini
10.23.22

bubble_sort():

Inside this function a loop iterates for as many times as there are elements in the array. Nested in this loop, another loop iterates from the last element backwards to whatever element the outer loop is on. If the element of the array at index *–inner loop's loop variable number–* is lesser than the element before it then the stats function *swap()* is called to swap their positions so that the greater number is closer to the end of the array while the lesser is moved backward. This loop continues swapping until the smallest number is at the beginning. The outer loop breaks and the program finishes only when an inner loop runs completely without making a single swap meaning the array is in order. The *swap()* function also marks three moves per swap for the moves, temp storage, replacement, and replacement, so this makes statistic gathering quite easy.

next_gap() from shell _sort():

This function works by returning a value which is 5//11 of the input, unless the input is two or less in which case the function will return one.

shell_sort():

For each gap calculated in *next_gap()* iterate through the array by it starting at the gap number. Then compare the element at the index of the gap number with that index minus gap. Continue to compare the numbers at the opposite end of the gap all the way through. If the elements on opposite sides of the gap are in the wrong spots keep switching by one gap backwards until it's on the gap that it should stay on. This will occur for each gap until it's sorted.

Test-harness control flow *sorting.c*:

The test harness will use get_opt to parse the command line options and then the data structure of sets to keep track of what functions have been called by the command. This eliminates the issue of calling a function twice if -a is called.

heap_sort():

Essentially, *heap_sort()* works by employing five helper functions, *l_child(), r_child(), parent(), down_heap()* and *up_heap*. All the child and parent functions do is return index values which correspond to the parent or child of a given index by their mathematically described rules. The left child of a parent is equal to two times the parent index and the right child is two times the parents index plus one. The parent of a child found by floor dividing a child's index by two. *up_heap*, swaps the child with the parent so long as the child is smaller than the parent. *down_heap* swaps the parent with the smaller child so long as the parent is smaller than the smaller child. And does this until the parent is no longer smaller than the smaller child. And lastly, *build_heap* employs *up_heap* to correctly populate a newly allocated memory with the sorting arrays numbers. Lastly, in a for loop *heap_sort* moves the first element of the already structured heap to the same spot in our original array, then it takes the last element of heap and puts it first and then *down_heap*s it so that the next smallest element appears next. Then upon the next iteration of the for loop the next smallest element is in *heap[0]*.

quick_sort():

Callocate an array of equal size to the main array. Go through the main array and move values greater than the arr[last] which is the pivot to the right side and values less than it to the left. Copy that array to the main array and replace the zero or zeros in the middle of the array with the pivot number. Call Quicksort again on the start value to the value just before the pivot and from just after the last pivot to the end of the array and recurse. Once the array length becomes less than eight, call Shellsort on the array bit to finish it up.

Statistics:

Each sorting function will get a struct of stats data type to keep track of that function's operation statistics and aid in sorting. During a swap operation *swap()*, from stats.c should be called. This function takes for parameters, the memory address of the *stats struct* recording the statistics for this sort as well as the addresses of the two variables in the array that are being swapped. Luckily for us, the *swap()* function not only keeps track of the number of moves but actually swaps them in the array.

Write-up scripts:

To print my plots I used modulo and read line in bash to extract the comparisons and the moves from the ./sorting. Once I had these values in a .dat file I could write them to gnuplot and make my graphs.