Assignment 3: Sorting

Aniket Pratap

January 26, 2022

Abstract

the goal of this assignment is to create and perform multiple sorting algorithms using the provided python pseudocode. The algorithms are insertion, heap, quick, and batcher sort. However, due to the quality of life features of python, one cannot exactly copy the code. For insertion sort, the python code only takes a list as the input. C doesn't have a built-in length function, for example, so one must input the length of the list as a parameter. Not only that, but the struct *stats also needs to be an argument in the function insertion sort. The pointer to the stats struct is used to count the number of moves or comparisons a sort makes. A comparison is defined whenever a number in an array is compared to another number, and moves are when a variable is moved from one spot to another. In this case, I can use the cmp() function in the struct to check if temp is less than A[j -1] in the insertion function. This function adds 1 to the comparison variable and it returns a value of 1, -1, or 0 based on the comparison it does. The move function can be used as followed. If A[j - 1] is being copied to A[j], that's 1 move. Then, when the temp is going into A[j], that's another move. The stats for this assignment are comparisons and moves, along with the size of the array. The array size can be read using getopt using user input, but the default size is 100. Not only that, but I have to create pseudorandom numbers using the random function. This can be done by using the time as an input, similar to Collatz.c. Finally, I'm supposed to use sets to track which command-line options were specified. I can do this by first creating an empty set, then inputing a 1 if the value, which is connected to a bit, is wanted from the user. I can use malloc() to create a spot in memory for the random array that the user wants. Malloc allocates memory, but after use, I need ot deallocate it using free. User input will be as follows: -a activates all sorting, -h does heap, -b is batcher, -i is an insertion, -q is quicksort, -r sets the seed of the pseudo-random numbers, -n sets the array size, and -p prints out the elements of the sorted array–the default being 100. My error handling could be using switch cases to take in user input. If the user wants to print out the number of elements more than the array, I will print out the whole list.