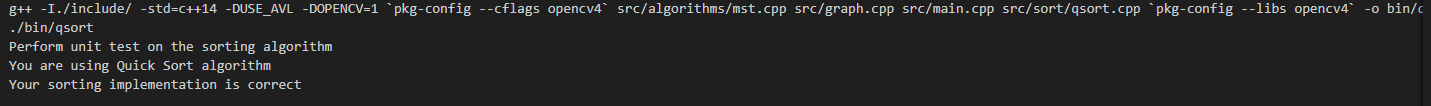
Blake Williams

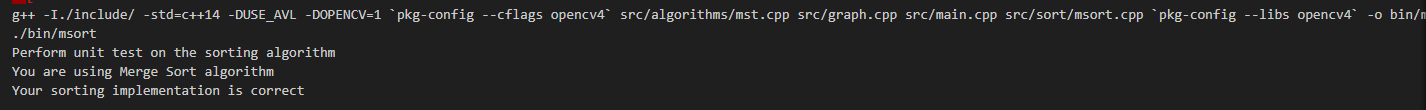
Algorithms HW4

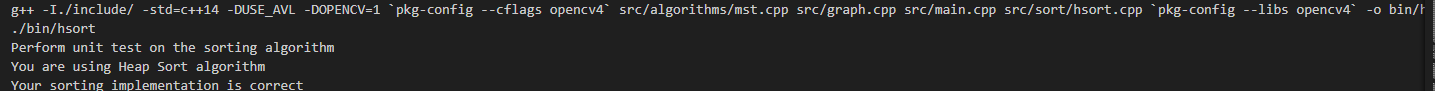
Student ID: 010974718

Date: 4/14/2024

This project focuses on developing 3 different sorting algorithms. These being Heap Sort, Merge Sort, and Quick Sort. Provided is the code from previous homework’s. First the implementation of hsort.cpp was as follows. 2 methods were created, the first called heapify(), which is a recursive function that ensures the subtree with root at index `i` is in the heap. We then calculate if the left child is larger than root, then do the same calculation for the right subtree. If the largest is not root, it is swapped, and heapify() is recursively called again. Then the sort() function was implemented, which first calculates the size of the segment of the array that needs to be sorted. After this the first loop begins building the heap from the array by starting on the halfway point of the array elements beyond the halfway point are already heaps. One the array is rearranged into a heap, the second loop extracts elements from the heap to sort the array. It moves the root of the heap to the end of the array, decreases the heap size by 1, and calls heapify() again. This repeats until all elements are extracted. Next to be implemented was mergeSort. Merge sort works by dividing the array, then sorting the smaller sub-arrays, then merging them back together. To do this in code 2 functions were implemented. A merge() function and a sort() function. The merge function created 2 temporary arrays which are filled with there respective halves of the array segment being merged. Then merges the 2 arrays back into the main array, finally it copies any remaining elements of left and right arrays into the main array. The sort() function works as follows. First we check to see if l<r or if the segment has more than one element. If it does then it returns immediately, else the sort() code runs. It calculates the middle index m, ensuring that the array can be divided evenly. Then Recursively calls the sort() function to reduce the array down to individual elements, then the merge() function is called which merges back into a single array. Finally the quicksort.cpp is implemented. First a pivot is chosen to be the last element in the array. Then a loop runs through the array, it checks if the current element is smaller than the pivot, if it is then swap position I and J. Once the end of this loop is reached the function is recursively called twice, the first sorts the left partition which contains elements less than the pivot, while the second sorts the right which contains elements greater than the pivot.







Check attached Williams\_test.txt to see a full compilation script.