To test the algorithm functioned correctly I used a test sweep. The first test was a one-dimensional array of positive integers in randomized order. I used this test case to troubleshoot any problems with the algorithm followed with several print statements to show me what the value of multiple variables was (see second commit for printed out statements). Once the algorithm properly sorted the first test case all the print statements were commented out and deleted in the following commit. I used the second test case to see what would happen if the method sort of sort was given a sorted array in theory This should have been “best case” because of the array already being sorted. However, this was not the case because there was no function to check if it was already sorted. What the method did was just “sort it” again. Leading to conclude that the algorithm I made does not have a best-case or worse case. The third test case was a one-dimensional array of negative integers. I wanted to test if it calculated negative integers correctly and sorted them as intended. The fourth test case was a combination of the first case and the third case given a set of real numbers can it sort them correctly. The fifth is an empty array it never computes or compares anything because the first while loop prevents any sorting on an array with a length of zero. The last test case was an array with three duplicates the reason I tested this was because the was the sorting gis don is in pairs of twos but having three duplicate values means one must be sorted in a different position if implemented correctly. I tested the limits of this algorithm and its time-complexity with Big O notation as O(n^2) and ΄Ω(n^2) because you will never check and sort an element more than once in the array and if given a sorted array it will still "sort it" because I never implemented a function to check if the array was sorted. The true time-complexity analysis as represented by the (n + 1)(n / 2). I calculate that the cost of the first for loop is n (n is the number of elements in the array) then n-1 than n-2..... till it reaches 1 because every single time it is comparing the max value one less time. Why? Because after a single pass we officially sort a single element and we do not compare that element more than once. This sequence can also be denoted as (n + 1) the second part of the algorithm is the while loop is n which denotes the number of elements in the array. (if statement are constant) Personally, this can be faster by having two pointers to check which element is the largest (this can be done with three additional if statement).