**Algorithm Selection\_Sort (A, N):**

for i = 0 to i < N-1 do

min = i

if i=0 then

min\_temp=N

for j = i+1 to j < N do

if A[j] < A[min] then

min = j

if min=min\_temp then

exit loop

End if

End for

temp = A[min]

A[min] = A[i]

A[i] = temp

Min\_temp=temp+1

End for

The changes I have made to the algorithm are highlighted in red. The first change I have added is a new variable called min\_temp. This will be set to N on the first cycle of the loop so it does not trigger the if statement in the second loop. Once the first loop has completed a cycle, it sets min\_temp to min+1. The loop the starts the next cycle and skips the if statement because I !=0 an continues into the second for loop. It scans through all values and finds the minimum and then the if statement checks if the new minimum is one greater than the previous minimum. If it is, it skips checking the rest of the numbers as this is definitely the next smallest minimum and the algorithm continues until the list is fully sorted.

1,2,4,3:

Original algorithm: It checks all numbers each time and finds that one is the smallest, then it knocks that off, finds that 2 is the smallest, knocks that off, finds that 3 is the smallest swaps it with four and knocks that off, then finally checks that four is the smallest and finishes and returns the sorted numbers.

Changed Algorithm: The algorithm runs the exact same as before however, it does not run all the way through the numbers after the first pass. I.e after it finds out one is the lowest, it skips checking 3 and 4 on the next pass because it knows 2 is the lowest.

**Algorithm Bubble\_Sort (A, N)**:

for i = 0 to i < N-1 do

for j = 0 to j < N-1 do

if A[j]<A[j+1] then

min\_value=A[j]

End if

If min\_value <A[j] and min\_value=A[j-1] or min\_value=A[0] then

J++

End if

if A[j] > A[j+1] then

temp = A[j]

A[j] = A[j+1]

A[j+1] = temp

End if

End for

End for

The changes I have made to the algorithm are highlighted in red. I have added 2 if statements into the algorithm. The first checks through the array to find the smallest number and sets it to min\_value. The second if statement checks if the min\_value is in position j-1 in the array and if the number above it is greater than the min\_value and if both these conditions are met it increments j and moves on to check the two values above it.

1,2,4,3:

Original: The original algorithm will check one and two and will see that they are already ordered, it will check two and four and check they are ok also, it will finally check 3 and four and swap them around before returning to the beginning to check that they are all ordered properly.

Changed Algorithm: Works the exact same as the original, however, it finds out what the minimum value is (1) and checks if its in the first position or the position below A[j] and if the number above the minimum (2) is bigger which it is. It then increments the counter so that it checks (2) and (4) and it makes no change. It then takes (2) as the minimum value and checks if (4) is bigger which it is. It then increments the counter and checks (4) against (3) and then swaps them as (4) is bigger than (3). The loop goes back around and takes (3) as the minimum, checks against (4) and makes no swap. The array is now fully sorted. The big difference is that it doesn’t return to the first value when checking the array as it knows it is the minimum value already.

**Big O**

**Iterative GCD:** O(1). This is because the algorithm contains no loops and the run time of the algorithm will not change greatly with respect to the size of the elements.

**Tree Drawing Algorithim:** O(N^2). This is because the algorithm contains a for loop within a for loop and the run time of the algorithm will change greatly with respect to the number of elements.

**Rock, Paper, Scissors:** O(1). This is because the algorithm contains no loops and the run time of the algorithm will not change greatly with respect to the size of the elements.