This section proposes a pivot-selection technique based on the values of every element in the array to

split the array into relatively equal halves for each recursive call which in turn reduces the number of

recursive calls and the overall execution time of QuickSort. The proposed technique aims to ensure

equal splitting for the array. Therefore, it drives the worst case behavior to be O(n log n). The proposed

technique also verifies an already sorted array or sub array which is done while comparing the elements

of the array to the pivot. If the array is already sorted, it will not be processed any further which

transforms the O(n2) complexity into the best case behavior of the algorithm; i.e. O(n). The proposed

technique operates as follows; at first, the pivot value is chosen to be the value of the rightmost element

of the array. Each element value will be compared with the pivot value and two counters are utilized to

count the number of elements with values smaller than the pivot versus the number of elements with

values larger than the pivot; namely, CountLess and CountLarger, respectively. The sum of the values of

the elements smaller than the pivot and the sum of those larger than the pivot are stored in SumLess and

SumLarger, respectively. These variables are then used to calculate the next pivots for the recursive calls.

The integer average of the values smaller than the pivot is passed as the pivot value of the recursive call

for the left sub array. Likewise, the integer average of the values larger than the pivot is passed as the

pivot value of the recursive call for the right sub array. This pivot selection technique helps in

successively splitting the array into nearly equal halves which in turn improves the efficiency of the

QuickSort algorithm. A Boolean variable is utilized by the algorithm to recognize an already sorted array

or sub array which reduces the number of recursive calls. Along with the reduction in recursive calls, the

proposed technique converts the worst-case scenario for the classical QuickSort algorithm into a best

case scenario with Θ(n) runtime. The modified algorithm MQuickSort is provided in Figure 2.

