Heapify Operations (heapifyMax and heapifyMin):

Time Complexity: O(log n)
These functions are called recursively to maintain the heap property of a subtree rooted with node i assuming subtrees are already heapified. Each call affects the children of a node and their children, leading to a logarithmic time complexity.

Building Heaps (buildMaxHeap and buildMinHeap):

Time Complexity: O(n)
Building a heap from an unsorted array is done in linear time. This is achieved by calling heapify for all non-leaf nodes in reverse level order.

Extracting N Largest/Smallest Elements (getNMaxElements and getNMinElements):

Time Complexity: O(N log n)
For each of the N elements extracted, a heapify operation (O(log n)) is performed. Therefore, the total complexity is O(N log n).

Calculate Average Rate (calculateAvgRate):

Time Complexity: O(n)
This function iterates over all elements to sum their values and then calculates the average, which requires linear time.

Calculate Differences (calculateDifferenceM):

Time Complexity: O(n)
Similar to calculateAvgRate, this function iterates over all elements to adjust their values based on the average rate.

Maximum Subsequence Sum (maxSubsequenceSum):

Time Complexity: O(n)
This algorithm is a variation of Kadane's algorithm, which finds the maximum sum contiguous subarray within a one-dimensional array of numbers. It runs in linear time.

Reading CSV Data (readCSV):

Time Complexity: O(n)
Assuming the file read operation itself is not the bottleneck, the time complexity is linear with respect to the number of lines in the CSV file.

Main Function Complexity:

The main function calls buildMaxHeap, buildMinHeap, getNMaxElements, getNMinElements, calculateAvgRate, calculateDifferenceM, and maxSubsequenceSum. The dominant terms in the overall complexity would be the heap building and the extraction of N elements, both O(n) and O(N log n) respectively.