Thomas McLinden

10/20/24

CMPSC 463

Professor Janghoon Yang

# [Project 1 (Stock Market Analyzer)](https://github.com/thomasmclinden/CMPSC363-Project1)

**Description:**

This project was designed to analyze arrays to determine averages, detect anomalies (outliers), and visually display the datapoints. This project utilizes merge sort as its sorting algorithm, due to its preferable average time complexity of *O(nlogn)* and *matplotlib.pyplot* to draw graphs using the input array. The closest-pair algorithm has also been used to detect anomalies in the data sets.

**Structure of Code:**

* *merge(arr, left, mid, right):*

This algorithm receives an array and recursively subdivides it until each element is separated. Each element is then compared to each other before being merged together into a sorted array. This specific sorting algorithm was chosen, due to its fast average time complexity of *O(nlogn)*.

* *maxSubArray(arr):*

This algorithm utilizes Kadane’s algorithm to find maximum sum of a contiguous subarray. It achieves this by going through each item of the list, creating a sum of each number encountered whilst storing the highest number encountered in the list. The sum will reset if the algorithm encounters a 0 in the list. After completion, the algorithm returns the largest sum value.

* *findAvg(arr):*

This is a simple function that calculates the average value of the array. This is achieved through the use of the *sum* function and returning its quotient after dividing it by the size of the array.

* *printArray(arr):*

This is another simple function that prints the contents of the array. This works by using the *map* function to convert each integer into a string and *join* to concatenate the strings together, using a space as the separator for each. The function then prints this value.

* *sort\_points\_by\_x(points):*

This function takes a list of points and arranges them in ascending order by their x coordinates using the lambda function.

* *sort\_points\_by\_y(points):*

This function takes a list of points and arranges them in ascending order by their y coordinates using the lambda function.

* *dist(p1, p2):*

This function calculates the distance between two points using the Euclidean distance formula.

* *brute\_force(points):*

This algorithm uses a for loop and the *dist(p1, p2)* function to check the distances between every point to find the shortest distance.

* *strip\_closest(strip, size, d):*

This algorithm narrows the search for the closest pairs by limiting comparisons based on their y coordinates. This is accomplished by checking if the distance between two y coordinates is less than the *min\_distance.* If the difference condition is met, their distance is calculated using the *dist(p1, p2)* function and the *min\_distance* is updated.

* *closest\_util(px, py):*

This function divides the set of points, recursively finds the closest pairs in the divided subsets, and checks for pairs across the dividing line, returning the smallest distance.

* *closest(points):*

This function initializes data for the closest pair search by sorting the points before passing them to the *closest\_util(px, py)* function.

* *plotData(arr, anomalies, max\_profit\_period):*

This algorithm utilizes *matplotlib* to display the data on a graph. It begins by initializing the size of the graph. The elements of the list are plotted on a graph, with the anomalies being plotted in red to contrast from the normal points. The algorithm finishes by adding the titles and labels to the graph.

With all these functions and algorithms, a list containing data points is passed as an argument into each algorithm and the required data is printed in the output window and displayed on the graph.

**Instructions:**

Using the application is very simple. The user needs to edit the list with the with data points, install *matplotlib.pyplot,* and run the application. The list is named as *arr* on line 146.

**Verification of code functionality (Demonstration):**

In this demonstration, the list named *arr* is initialized, with each element representing the price of a stock index at the end of a trading day.

A screen shot of a computer code

Description automatically generated

Once the application runs, each function is called, and their return values are printed. The console will list the stock prices in the list, the stock prices in ascending order after being sorted with the merge-sort algorithm, the maximum subarray value, the average stock price, their coordinates sorted by their x and y values, and the smallest distance between two points.

A screenshot of a computer

Description automatically generated

Through the use of *matplotlib.pyplot,* the data is plotted on a visual graph.

A graph with a line and a point

Description automatically generated

Each data point is listed on the line graph, with the anomalies being marked in red, and the period of maximum profit being highlighted in green.

**Discussion of Findings:**

While the application has all of the basic features to analyze stock market trends, these are only the beginning steps. The application doesn’t have the necessary features to be used by any major market. In its current phase, the application needs to have each value inserted manually and can only calculate these values for one index at a time, which would automatically make the application unusable in a real-world scenario. Furthermore, there is no algorithm to convert currency values when analyzing foreign stock indexes, which would need to be updated regularly with conversion rates fluctuating. The application also only tracks values at the end of each trading day, where more efficient stock market analyzers such as *Google Finance* update the stock values every few minutes. While the application has the first steps necessary to analyze stock indexes, it has a long way to go before it can be used by stock traders or private equity firms.