**Technical Solution**

This section contains all my codes, there are over 2000 lines of codes and many functions and procedures. These algorithms and methods are all of varying complexity. To make it easier to identify the more complex algorithms I will list below the key functions that are used in the code. One can consult the document design section to discover more about particular functions or procedures where the pseudocode is provided for some, but the basic steps and explanations are shown for all. Please find the list of key functions and procedure below:

1. ***AICorrBtnn***
   1. This is a procedure that calculates all the valid correlation coefficients of all the stocks that have been pulled. It then proceeds and assign the corresponding values to a record, which is then stored in an array. The values are: both stock indexes, the correlation coefficients, and the start and end date of where the correlation coefficient was calculated. It lastly sorts the lists from highest positive to highest negative by calling the PerformMerge procedure.
2. ***ShowAiBttn***
   1. This is the button that calls AICorrBttn which allows this procedure to show all correlated stocks correlation coefficients from highest positive to highest negative. It also shows the suggested portfolio which is based on the amount of stocks that the user desires.
3. ***ListAvgBttn***
   1. This is procedure that lists all the buy positions of all the stocks that you have pulled. The reason this is an impressive algorithm is because it uses a 3d array, which is an array of an array of an array. The first tier of the array indexes the stock. The 2nd tier of the array has a length of 2 and indexes the high or low averages. The 3rd tier of array actually contains the values of the average for that array and for the specific high or low period.
4. ***MovingAverage2Bttn***
   1. This procedure calls the procedure MakeArrayList, and plots the values on the graph. I believe it shows the interaction with the chart object very well.
5. ***PullDatBttn***
   1. This procedure takes the input of the filename and initially tries to find a .txt file where then the data is all formatted for the program. By calling the constructor of TStock for .txt files. However this procedure also sets up the request to pull the data from the API and then it stores the JSON values in a data source object. The procedure then gets the individual values out of this data source and stores them in a string array which from there it can call the constructor of TStock with the overloaded live data parameters (The procedure name was meant to have said PullDataBttn not PullDatBttn).
6. ***Correlation***
   1. This function actually calculates the correlation coefficient between the 2 stocks passed in the parameters. It does this by calling a series of functions and storing their values such as means and standard deviation and covariance. Additionally the code also works out the overlapping time frame between the stocks by calling another function which allows the program to determine if they are not aligned and if so by how much and then gives details about what needs to happens.
7. ***PerformMerge***
   1. This is start of the merge sort. This is a procedure that recursively calls itself to split up the array that allows them for comparing, in the ‘Merge’ procedure. It also calls the ‘Merge’ algorithm.
8. ***Merge***
   1. This is the algorithm that compares and sorts. The split up array. This algorithm combined with ‘PerformMerge’ is a complex merging algorithm which allows the array of correlation coefficient to be sorted from 0🡪Max index, goes from highest negative coefficient to highest positive coefficient. One difference between this algorithm and another merge sort algorithm is that this has to swap every value in the record not just the correlation coefficient.
9. ***Constructor for TStock***
   1. This constructor creates the formatting for all the different values in the a single data point. It is overloaded so that the parameters that it takes in can be in the live data form or the .txt form. Which it deals with each appropriately. It call the constructor of TData to format individual data points.
10. ***Constructor for TData***
    1. This constructor creates the formatting for all the different values in the a single data point. It breaks up the line of data, putting it in the correct format. This constructor is also overloaded so that it can do it correctly dependent on if its formatting live data or the .txt data.