Algorithm design for assignment **AS2\_PS8\_BI\_168**

* We have designed a main class “Solver”, it has methods to
  + read input
  + solve the input data
  + output the solution to a file
* The main class “Solver” is declared inside main.py file along with all the supporting lambda functions.
* For testing the class “Solver” the following lines of code can be used
  + solver = Solver('inputPS8.txt')
  + solver.solve()
  + solver.solve(method='advance')
  + solver.output()
* Class “Solver” methods
  + \_\_init\_\_( input\_file, output\_file='outputPS8.txt'): This method is called at the time of class initialization, input\_file is a positional parameter and on initialization of the class data is read from the file and gets preserved to be used further. Output file parameter is optional parameter and the value supplied here is remembered.
  + input(file=None) : this method parses data from the supplied text file and attaches it to the instance as “\_data” attribute. Data can be checked using “data” property of any instance of a class. This method is called on class initialization with the “input\_file” parameter supplied at class initialization.
  + solve(method='linear'): This method returns the ”day to buy”, “day to sale” and “profit” values after running the select method to analyse data attached to the instance. Supplying “method” parameter with “linear” will call the iterative method to get the solution, Divide and conquer method can be called by supplying “advance” as “method” parameter.
  + output(self, output\_file:str='outputPS8.txt'): This method is used to write output to a text file, it calls self.solve() internally.
* Complexity of iterative solution (linear): we were able to write a function “solve\_linear(data)” with time and space complexity of Θ(n). Our algorithm is based on human nature, if we find something for a price lower then what we have ever seen we buy it, if we get better selling price then the current we sell it. But we keep track of the best buy and sell information along with the running transactions. Only if the current transaction is better then the best one we replace the best solution. Using this idea we were able to get rid of the nested loop solution (comparing every price with other price) which we have implemented as mehtod “\_solve\_tester()” which has a time complexity of O(n^2). “\_solve\_tester()” method is commented but we used it in testing to check our linear and divide and conquer methods against it for deviation.
* Complexity of divide and conquer solution (advance): we were able to write a function “solve\_advance(data)” with time complexity Θ(nLogn) and space complexity of Θ(n). The algorithm first gets the differences of daily prices by scanning them once (x = arr[i]-arr[i-1]). Then the daily differences are used to find a Maximum Subarray Sum using Divide and Conquer algorithm. Divide and merge is done using these three possible conditions:
  + The price minima and maxima are both on left side.
  + The price minima and maxima are both on right side.
  + The price minima is on left side and the maxima is on the right side
* Run time analysis of the algorithms:
  + Reading input from file and cleaning it to make a list of lists, with time and space complexity of Θ(n).
  + Solving data using iterative (linear) method, with time and space complexity of Θ(n).
  + Solving data using divide and conquer solution (advance), with time complexity Θ(nLogn) and space complexity of Θ(n).
  + Writing output to file, with complexity of Θ(1).
  + **Overall runtime complexity for getting output:**
    - **For linear solution Θ(n) + Θ(n) + Θ(1)**
    - **For divide and conquer solution Θ(n) + Θ(nLogn) + Θ(1)**
* We have included basic testing script “run\_test.py”, it checks the algorithms with random inputs against the traditional iterative method. Before using this file the testing methods in the class “Solver” need to be uncommented from the main.py file.