Team Name: LED (Leonid Emma Dalila)  
[Project GitHub Repo](https://github.com/Dalilaportal04/InvestmentPortfolio) | [Link to Video Demo](https://youtu.be/gPJb5SKRoV8)

# Team Members

Leonid Cherevko: [leonid-cherevko](https://github.com/leonid-cherevko)

Dalila Portal: [Dalilaportal04](https://github.com/Dalilaportal04)

Emma Baumgartner: [emmabaumgartner](https://github.com/emmabaumgartner)

# Extended and Refined Proposal

### Problem:

Keeping track of your portfolio effectively is a challenge, especially when trying to monitor the performance of purchased stocks against current market prices.

### Motivation:

It is essential to understand how individual stocks in a portfolio are performing relative to the market to make informed financial decisions. Comparing purchase prices with current stock prices helps investors evaluate gains or losses and determine future actions.

Features:

* Users can add stocks to their portfolio by entering stock tickers and purchase prices.
* The system calculates the delta (difference) between the user’s purchase price and the most recent stock price.
* Stocks in the portfolio are ranked based on these deltas for easy evaluation of performance.

### Description of Data:

Due to dataset constraints, we used randomly generated stock tickers and prices to simulate a larger dataset. Ideally, the YFinance API would have been used to fetch real stock data; however, its limited dataset of ~10,000 stocks was insufficient to meet our project's requirements.

### Tools/Languages/APIs/Libraries Used:

* **Development Tools**: CLion
* **Languages**: Python, C++, HTML, CSS, JavaScript
* **Libraries**: nlohmann JSON library for parsing JSON files

### Algorithms Implemented:

* **BTree**: Used for storing and organizing all stock data efficiently.
* **Max Heap**: Implemented to organize and rank the deltas in stock prices, enabling quick access to the best-performing stocks.

### Distribution of Responsibility:

* **Leonid Cherevko**: Explored API options, ultimately generated synthetic data, and implemented the BTree structure.
* **Emma Baumgartner**: Developed the intermediary components to connect the front end with the back end and implemented the Max Heap for delta ranking.
* **Dalila Portal**: Designed and implemented the front-end interface, handling user input and displaying results

# Analysis

During the completion of the project, the website design was changed from the initial visualization. With the time restrictions, the initial website design was very complicated and would've taken too much time on top of the implementation of the data structures and computations our program also has. We decided to go with a table for the visualization of the data due to limitations for visualizing in C++. One of the biggest changes was that we started the project using a B+ tree, but in the end, we decided to use a B tree instead. This decision was made with the intent to simplify our code and to simplify the access of our data in the heap.

Our major functions relate to the implementation of the heap data structure and the B tree data structure. Heap has the functions heapify(), top(), extractmax(), insertstock(), deletestock(), printheap(), and buildheap(). Of these functions, one of the major functions that is used in the main function of our program is heapify() which has a time complexity of O(logn). This function is a recursive function and the depth of the heap is logn, which is well the worst case time complexity is O(logn). Another major function of the heap is insertStock() which inserts the delta of the B tree data into the heap and has a time complexity of O(nlogn) - O(n) for the resizing of the array and O(logn) for insertion into the heap. Extractmax(), deletestock(), and printheap() were used for debugging but were not used for the implementation of the actual project. Buildheap() has a time complexity of O(n) and makes a call to heapify. B tree has the functions insert(), searchByTicker(), loadDataFromFile(), inOrderTraversal(), insertNonFull(), splitChild(), and search(). Insert() has a time complexity of O(d \* o) where d is the depth of the tree and o is the order (degree of the B tree). SearchByTicker() is o(d) where d is the depth of the tree. InOrderTraversal() is O(n) where n is the amount of keys in the tree that must be traversed. InsertNonFull is also O(d) where d is the depth of the tree. SplitChild() is O(o) where o is the maximum degree of the B tree. The heap and B tree are data structures for different tasks, which is why in our project we do not use them for the same task, but instead use them for different tasks. The heap is used for the deltas, so that the greatest delta (the data the user would want) is fast to extract. The B tree is used for organizing all potential stock data. For some general comparisons, insertions into both are O(logn) generally. Seaching in a B tree is faster, as the heap is instantiated as an array which has a search time of O(n). Traversal for both is O(n).

# Reflection

As a group, our overall experience of the project was positive. The group atmosphere for the project was wonderful for debugging and sharing ideas on how to implement things. Everyone was helpful by contributing their own knowledge and understanding about mistakes.

Overall, we did come across many difficulties we did not consider when we were brainstorming the project. We would say project control in the beginning was a bit rocky with getting our repository configured. Connection between the backend and frontend was difficult. Setting up the coding environment was also difficult, due to issues with our computers and proper pathing to find Python and set up the environment for the frontend to work.

We would change the workflow by not overcomplicating the project. We overestimated our ability to create our vision within the time constraints of the project, especially when it comes to creating a website for sleekly visualizing everything. We would also be organized with all our files and documentation of what each file is supposed to be used for.

Dalila learned HTML and the art of website creation and discovered that she has a passion for frontend. Leo learned how to work with APIs such as the yfinance, as well as how to effectively implement both B and B+ trees. Emma learned about working with APIs in Python (yfinance) as well as the connection between frontend and backend, and how to create heaps and input data into it from a different data structure.