When 10% Matters: A Tale of Two Mindsets Through the Eyes of a Young Investor

A comparative analysis of Dynamic Programming and the Greedy Algorithm to model "patient" versus "impulsive" investment strategies.

Maheswara Bayu Kaindra - 13523015

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: kaindramaheswara11@gmail.com , 13523015@std.stei.itb.ac.id

***Abstract*—For a student with a limited capital, investing can be a double-edged sword. Investing provides a good opportunity to force students to save. On the other hand, investment also has big risks, where losing 10% can change their expenses for daily life. This paper analyzes how to invest (especially in the Indonesian Stock market) by comparing two opposing mindsets: impulsive and patient. The impulsive mindset is represented by the Greedy Algorithm that prioritized short-term profits. The patient mindset is modeled by the Dynamic Programming Approach, which identifies the most optimal series of transactions over a given period of time.**

Keywords—Dynamic Programming; Greedy; Risk; Indonesian Stock Market

I. Introduction

In the last year or two, stock market investing has become a popular topic among the younger generation, particularly university students. A major reason for this trend is technology. With mobile trading apps, it is now easier to start investing, even with a small amount of money. Their motivations of investing vary, ranging from long-term saving, growing their capital by reinvesting dividends, and some to profit from buying and selling stocks. *[1]*

However, it's important to note that university students operate on a "tight" budget. They frequently sacrifice their daily spending money to invest in the market. Under this condition, investing in stocks poses a significant risk, as even a 10% loss can have a major impact on their day-to-day life. This is why making wise investment decisions is crucial. This paper analyzes the issue from two different perspectives: the patient investor and the greedy one. *[2]*

For this study, the patient investor will be modeled by the Dynamic Programming algorithm, where decisions are made by analyzing all historical transaction data. Meanwhile, the greedy investor will be modeled by the Greedy Algorithm, which operates by targeting a certain price at which to buy or sell a stock. This research will focus on the Indonesian stock market and will be implemented using the Python programming language. This research is expected to provide a quantitative, data-driven basis for the importance of patience and discipline in investing. Furthermore, this paper demonstrates a real-world application of Greedy and Dynamic Programming Algorithms. *[3]*

II. Theoretical Background

Stocks and Indonesian Stock Market

Definition – A stock is a financial instrument that represents a small fraction of ownership (equity) in a company. When someone buys a stock, they are essentially purchasing a small piece of that corporation. Investors buy stocks with the expectation of earning a return, either through capital appreciation (increase in the stock's price) or through dividends (distributions of the company's profits).

Factors That Determine Stock Prices – A stock's price in the market is determined by the principle of supply and demand. This price fluctuates due to a variety of factors, including company-specific events (like earnings reports) and macroeconomic conditions. Essentially, a stock's price reflects what investors believe a company is worth and what its future prospects are.

Indonesian Stock Market – To understand the dynamics of the Indonesian stock market, it's essential to recognize the structure of the Indonesian Composite Stock Price Index (IHSG). IHSG is a market-capitalization-weighted index. This means that the index's movement is not influenced by all Indonesian stocks. Instead, it's heavily dominated by the companies with the largest market values.

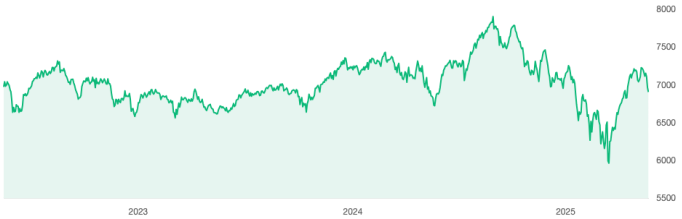

Figure 2.1. Indonesian Composite Stock Price Index (IHSG) Performance, three-year Period (June 2022 - June 2025)
Source : StockBit

Based on the market capitalization data from the first quarter of 2024, the five stocks with the most significant influence to the IHSG's movement are as follows:

TABLE 2.1: 50 BIGGEST MARKET CAPITALIZATION - IHSG, FEB 2024

No.	**Stocks**	
	Code - Listed Stocks	*%*
1.	BBCA - Bank Central Asia	10.31
2.	BBRI - Bank Rakyat Indonesia	7.68
3.	BREN - Barito Renewables E.	6.93
4.	BYAN - Bayan Resources	5.56
5.	BMRI - Bank Mandiri	5.53

Source: Bursa Efek Indonesia

This research focuses on those 5 stocks.

Greedy Algorithm

The Greedy Algorithm is a simple, straightforward, and widely used method for solving optimization problems. There are two types of optimization:

a) Maximization.
b) Minimization.

By definition, the Greedy Algorithm is a method that solves problems by making the best possible choice at each stage, without considering future consequences. It operates on the hope that by repeatedly choosing the local optimum, it'll arrive at the global uptimum (Munir, 2024).

Terminology – There are some keywords that are essential for describing this algorithm:

a) Candidate Set: The set of candidates from which a selection is made at each stage.
b) Solution Set: The set of candidates that have already been chosen.
c) Solution Function: Determines if the solution set constitutes a complete solution.
d) Selection Function: Chooses the next candidate based on a spesific greedy heuristic.
e) Feasibility Function: Checks if a selected candidate can be added to the current solution set without violating constraints.
f) Objective Function: The function to be maximized of minimized.

Algorithm–The Greedy algorithm is usually implemented by these 3 steps:

a) Initialization: Create an empty solution set (*S*) assuming that a candidate set (*C*) already exists.

b) Process: As long as the candidate set (*C*) is not empty and a complete solution hasn't been found:

i) Select the best available candidate *x* from *C* (based on the greedy heuristic).
ii) Remove *x* from *C*.
iii) Check if *x* is feasible to be added to *S*.
iv) If it is feasible, add *x* do the solution set *S*.

c) Result: If *S* represents a complete solution, return *S*. Otherwise, report that no solution was found.

Advantages – There are several advantages to using the Greedy Algorithm:

a) Efficient: Greedy algorithm are often more efficient and faster than other methods like exhaustive search. While other algorithms can have exponential time complexity, the Greedy Algorithm runs in polinomial time, making it much faster for large inputs.

b) Useful for approximation: while finding an optimal solution is too slow (or computationally expensive), the Greedy Algorithm can be used to quickly find a relatively good solution.

Disadvantages – There are also several disadvantages to using the Greedy Algorithm:

a) Not Always Optimal: The Greedy Algorithm does not guarantee a globally optimal solution; That is because it does not consider all possibilities exhaustively.

b) Strategy dependent: The success of the algorithm depends heavily on choosing the right greedy heuristic. For some problems, finding a heuristic that always produces the best solution is almost impossible.

Dynamic Programming

Definition – The Dynamic Programming algorithm is a problem-solving method by breadking down the solution into a series of stages so that the solution can be viewed as a series of interrelated decisions (Munir, 2024). This method builds a solution by examining and remembering the outcomes of many different decision sequences.

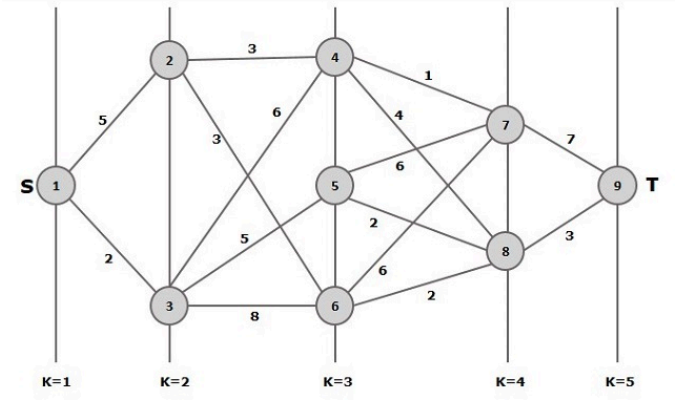

Figure 2.1. A multi-staged Graph
Source:
https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_multistage_graph.htm

In this context, the problem can be represented as a multi-stage graph (as shown in the figure above), where:

a) *K* denotes the stage number.
b) Each *node* represents a state (decision to make).

c) Each $edge$ represents the "local" cost of making a particular decision.

This algorithm follows the Principle of Optimality: for an overall solution to be considered optimal, every sub-solution within it (such as the segment up to stage K) must also be optimal.

Algorithm – The following are the steps to solve a problem using Dynamic Programming:

a) Characterize the Structure of an Optimal Solution:
The first step is to define the problem's structure. This involves identifying the stages (K), the states within each stage, and the decisions to be made.

b) Recursively Define the Value of an Optimal Solution:
The second step is to formulate a recursive relationship that defines how the value of an optimal solution (at one stage) depends on the values of optimal solutions from previous stages.
i) Base case: $f_1(s) = c_{x1,\,s}$
ii) Recursive case: $f_k(s) = min(f_{k-1}(x_k) + c_{xk,\,s}$

c) Compute the value of an optimal solution:
The third step is to calculate the value of the optimal solution using a state table. This computation can be done forward (from $K = 1$ to $K = n$) or backward (from $K = n$ to $K = 1$).

TABLE 2.1: STATE TABLE

x_k	**Optimal Solution**	
	$f_k(x_k)$	x_k *
x_{k1}	Cost for x_{k1}	…
x_{k2}	Cost for x_{k2}	…

*Note: x_k * represents the minimum cost for x_k*

d) Construct the Optimal Solution:
This final step involves tracing back through the state tables to find the actual sequence of decisions that yields the optimal solution.

III. IMPLEMENTATION DESIGN

The Greedy Algorithm

In this research, the greedy algorithm determines its daily action based on a trend-following rule relative to a short-term moving average. The logic, representing the "greedy trader", is as follows:

1) *SMA (Simple Moving Average):* contains a 5-day moving average of the stock's price.
2) *Buy case:* If the current price drops below this average, the algorithm invests its entire cash balance to buy the stock.
3) *Sell case:* If the current price rises above this average, it sells its entire holding.

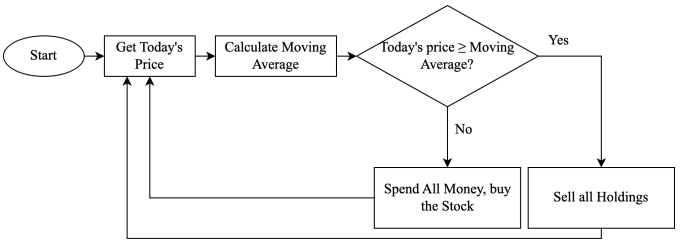

Fig 3.1. The Greedy Algorithm Approach Flowchart
Source: Made with draw.io

The Dynamic Programming Algorithm

The Dynamic Programming approach works by analyzing the best action to take for every iteration (day). Overall, there are four possible actions that can be taken, as follows:

1) *Hold Cash:* do nothing while holding cash.
2) *Sell Stock:* sell the entire stock holding.
3) *Hold Stock:* do nothing while holding the stock.
4) *Buy Stock:* purchase stock with all available cash.

In this approach, all transactions are "all-in", meaning the entire cash balance is used for a purchase, and the entire stock position is liquidated in a sale.

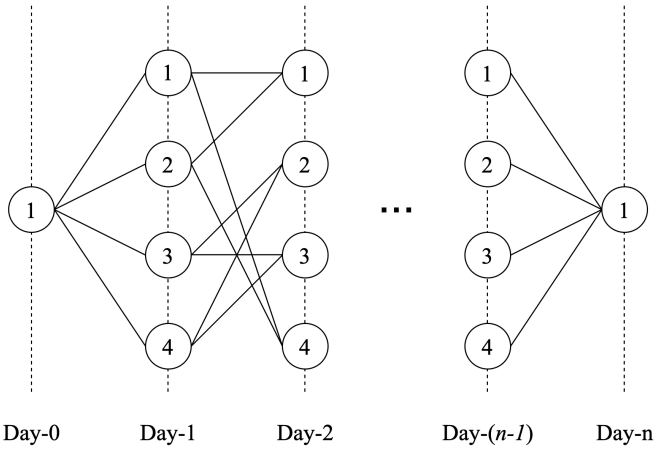

Fig 3.2. Multi-Staged Graph for Stock/Cash Actions
Source: Made with draw.io

In this implementation, the base case is represented by Day-0, where cash is equal to the initial capital (IDR 10,000,000 is used). For the next iterations, profit maximization is applied as follows:

a) Base Case:

$$cash_0 = initial\ capital,$$
$$hold_0 = -\infty$$

b) Recursive Case:

$$cash_k = max(cash_{k-1},\ hold_{k-1} \times \frac{price_k}{price_{k-1}})$$
$$hold_k = max(hold_{k-1} \times \frac{price_k}{price_{k-1}},\ cash_{k-1})$$

Where $cash_k$ represents the cash on hand at the end of day-k.

Therefore, this algorithm cannot be used as a practical (in real life) strategy, but rather as a benchmark representing the biggest possible return (a.k.a an idealized example of a trader with perfect foresight).

The Overall Program

In general, tha application consists of a main program in the root folder that calls functions from modules located in the src directory. The modules within the src directory are as follows:

1) *Data Downloader*
   a) Prompts the user for a stock ticker symbol as input.
   b) Downloads the last two years of historical stock data, counting back from the current date.
   c) Converts the data into .csv format and saves it in the data directory.
2) *Data Manager*
   a) Retrieves the CSV from the data directory.
   b) Generates the necessary plots and graphs, converts them into .png format, and saves them tot he results directory.
3) *Greedy and Dynamic Programming Algorithms*
   a) Contains the implementation of the core trading logic.

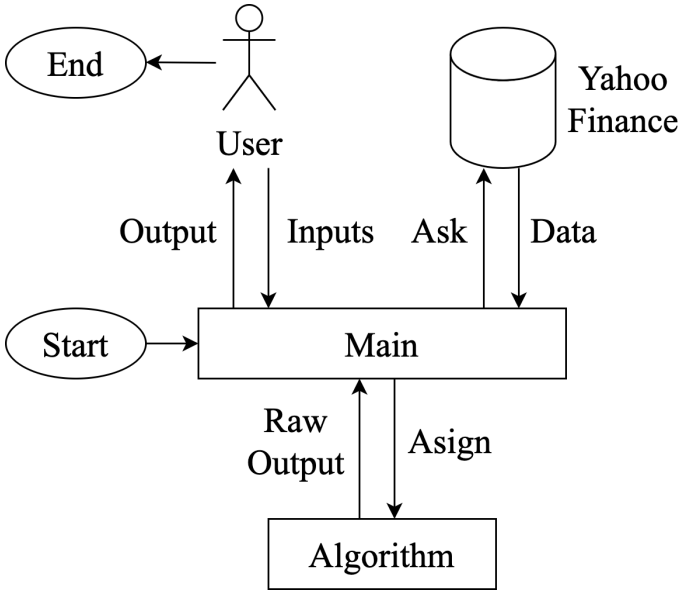

Fig 3.3. Overall Program Data-flow Diagram
Source: Made with draw.io

The program prompts the user to enter some stock ticker symbols and, in return, generates the following outputs:

1) Summary of the investment returns (.png and .csv).
2) Graphs illustrating the portfolio's equity performance over time (.png)..

IV. Program Implementation

Repository Structure and Link

The entire project is housed in this GitHub repository, which has the following general structure:

```
two-mindsets-investing-simulator/
├── data/
│   └── BBCA.csv, BBRI.csv, etc.
├── results/
├── src/
│   ├── __init__.py
│   ├── data_manager.py
│   ├── greedy_simulator.py
│   ├── dp_simulator.py
│   └── visualizer.py
├── main.py
├── requirements.txt
└── README.md
```

Greedy Algorithm Implementation

The following is the implementation of the Greedy Algorithm according to the specified design.

```python
def greedy_simulator(stock_data,
initial_capital=10000000, sma_window=5):
   data             = stock_data.copy()
   data['SMA']      =
data['Close'].rolling(window=sma_window).mean()
   cash             = initial_capital
   shares           = 0
   portfolio_values = []

   # Loop: Trading Simulation (Iterate Daily)
   for i in range(len(data)):

       # Handle: if SMA is NaN, skip the iteration
(for the first few days)
       if np.isnan(data['SMA'].iloc[i]):
           portfolio_values.append(initial_capital)
           continue

       price_today = data['Close'].iloc[i]
       sma_today   = data['SMA'].iloc[i]

       # Buy: If price is above SMA and we don't
hold the stock, buy shares
       if price_today > sma_today and shares == 0:
           # Buy: Go all in
           shares_to_buy = cash // price_today
           cost          = shares_to_buy *
price_today
           cash         -= cost
           shares       += shares_to_buy
           print(f"{data.index[i].date()}: Bought
{shares_to_buy} shares at {price_today:.2f}, Cash
left: {cash:.2f}")
```

```
        # Sell: If price is below SMA and we hold the
stock, sell shares
        elif price_today < sma_today and shares > 0:
            # Sell: Go all out
            sale_value = shares * price_today
            cash       += sale_value
            print(f"{data.index[i].date()}: Sold
{shares} shares at {price_today:.2f}, Cash now:
{cash:.2f}")
            shares     = 0

        current_portfolio_value = cash + (shares *
price_today)

portfolio_values.append(current_portfolio_value)

    print(f"Final Portfolio Value:
{portfolio_values[-1]:.2f}")
    return pd.Series(portfolio_values,
index=data.index)
    return summary_df
```

Dynamic Programming Implementation

The following is the implementation of the Dynamic Programming Algorithm according to the specified design.

```
def dynamic_programming_simulator(stock_data,
initial_capital=10000000):

    prices = stock_data['Close'].to_numpy()
    n = len(prices)
    if n < 2:
        return pd.Series([initial_capital] * n,
index=stock_data.index)

    cash = np.zeros(n)
    hold = np.zeros(n)

    cash[0] = initial_capital
    hold[0] = 0

    for i in range(1, n):
        price = prices[i]
        prev_price = prices[i-1]
        cash_stay = cash[i-1]
        cash_sell = 0
        if hold[i-1] > 0:
            cash_sell = hold[i-1] * price /
prev_price

        cash[i] = max(cash_stay, cash_sell)

        hold_keep = 0
        if hold[i-1] > 0:
            hold_keep = hold[i-1] * price /
prev_price

        hold_buy = cash[i-1]
        hold[i] = max(hold_keep, hold_buy)

    # Work backwards to find the actual buy/sell
sequence
```

Cont:

```
transactions = []
    i = n - 1

    if cash[n-1] > hold[n-1]:
        current_state = 'cash'
    else:
        current_state = 'hold'
        transactions.append(('sell', n-1,
prices[n-1]))
    while i > 0:
        price = prices[i]
        prev_price = prices[i-1]
        if current_state == 'cash':
            if hold[i-1] > 0:
                cash_from_sell = hold[i-1] * price /
prev_price
                if abs(cash[i] - cash_from_sell) <
abs(cash[i] - cash[i-1]):
                    transactions.append(('sell', i,
price))
                    current_state = 'hold'
        else:
            if abs(hold[i] - cash[i-1]) < 1e-6:
                transactions.append(('buy', i,
price))
                current_state = 'cash'
        i -= 1
    transactions.reverse()

    # SIMULATE ACTUAL TRADING
    current_cash = initial_capital
    current_shares = 0
    portfolio_values = [initial_capital]

    transaction_idx = 0

    for i in range(1, n):
        price = prices[i]
        date_str =
stock_data.index[i].strftime('%Y-%m-%d')

        if transaction_idx < len(transactions) and
transactions[transaction_idx][1] == i:
            action, day, transaction_price =
transactions[transaction_idx]

            if action == 'buy' and current_shares ==
0:
                # Buy as many shares as possible
                shares_to_buy = current_cash // price
                if shares_to_buy > 0:
                    cost = shares_to_buy * price
                    current_cash -= cost
                    current_shares += shares_to_buy
                    print(f"{date_str}: Bought
{shares_to_buy:.0f} shares at {price:.2f}, Cash
left: {current_cash:.2f}")

            elif action == 'sell' and current_shares
> 0:
```

Cont:

```
sale_value = current_shares * price
                current_cash += sale_value
                print(f"{date_str}: Sold
{current_shares:.0f} shares at {price:.2f}, Cash
now: {current_cash:.2f}")
                current_shares = 0

            transaction_idx += 1
        portfolio_value = current_cash +
(current_shares * price)
        portfolio_values.append(portfolio_value)

    # Sell any remaining shares at the end
    if current_shares > 0:
        final_price = prices[-1]
        sale_value = current_shares * final_price
        current_cash += sale_value
        date_str =
stock_data.index[-1].strftime('%Y-%m-%d')
        print(f"{date_str}: Final sale - Sold
{current_shares:.0f} shares at {final_price:.2f},
Final cash: {current_cash:.2f}")
        portfolio_values[-1] = current_cash

    final_value = portfolio_values[-1]
    print(f"Final Portfolio Value:
{final_value:,.0f}")
    print(f"Total Return: {((final_value /
initial_capital) - 1) * 100:.2f}%")

    return pd.Series(portfolio_values,
index=stock_data.index)
```

Other supporting implementations (other part of the program) can be found in this GitHub repository.

V. Comparing The Performace of th Greedy and Patient Trader Strategies

After conducting a 730-day trading simulation using the Greedy and Dynamic Programming approach on the five most influential stocks on IHSG (BBCA, BBRI, BREN, BYAN, and BMRI), the analysis of these results is presented below.

The Greedy Model

The Greedy Algorithm, which in this research models an impulsive investor's mindset, demonstrated highly variable performance that was heavily dependent on the price movement characteristics of each stock. Based on the portofolio growth comparison graph (the graph below), it can be concluded that the Greedy Algorithm did not consistently generate profits across all five stocks.

a) *Highly Profitable on Stocks with strong trends:*
The strategy successfully recorded its most significant gains on stocks exhibiting strong directional trends such as BYAN and BREN.

b) *Not optimal on Sideways-Trending stocks:*
For the large banking stocks like BBCA, BBRI, and BMRI, the strategy yielded far lower returns, with portfolios often stanating or even ending with a minor loss.

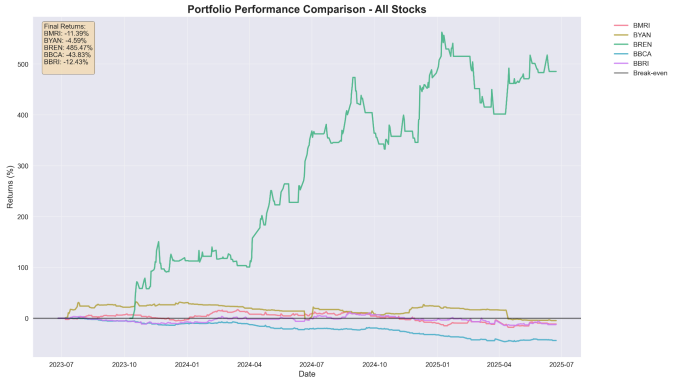

Fig 5.1. Portofolio Performance using the Greedy algorithm
Source: Kaindra

The Dynamic Programming Model

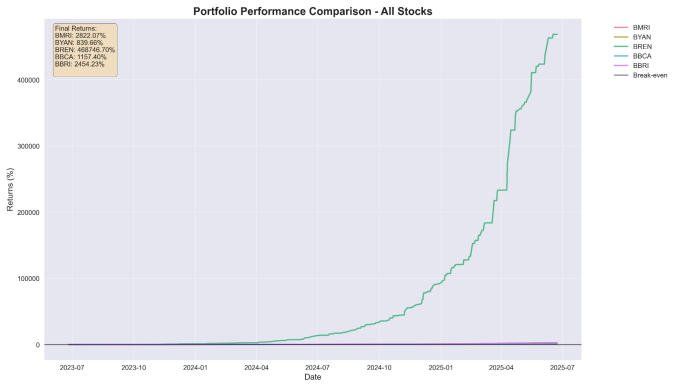

Fig 5.2. Portofolio Performance using the Dynamic Programming algorithm
Source: Kaindra

Overall, the Dynamic Programming Model, representing the "patient trader" exhibits a more analytical trading character. It makes decisions by considering its existing cash and stock holdings from the previous day and calculating the maximum potential value it can achieve, whether held as cash or stock. The main characteristics of this "patient trader" can be summarized as follows:

a) *Performance in Difficult Markets:*
The Dynamic Programming model consistently managed to turn volatile or sideways market conditions into a profit.

b) *Transaction Frequency and Precision:*
In contrast to the Greedy model, the Dynamic Programming approach is more "quiet". It executes only the optimal transaction after analyzing the entire dataset to find the best course of action within the overall trend.

c) *Basis for Decision Making:*
Unlike the Greedy model, the DP model's decisions

are global and optimal, based on an analysis of the complete historical data. This makes to model impractical for real-world applications, as it's impossible to know future prices.

Therefore, this model cannot function as a predictive tool but rather as a theoretical benchmark to measure the maximum potential that exists within the market.

Conclusion

These findings offer a powerful, data-driven takeaway for the younger generation, particularly university students: the temptation to chase quick profits can be a double-edged sword. This is demonstrated by the Greedy model, in which the "greedy trader" often achieved sub-optimal results, with any profits being largely attributable to luck.

While in the real world we'll never have the perfect foresight of the Dynamic Programming model, it offers a valuable principle. The discipline of not overreacting to market shocks, performing in-depth analysis before acting, and focusing on well-considered, long-term decisions proves to be the foundation of a much safer and more profitable investment strategy.

When a 10% loss can significantly impact one's lfie, this research shws that success in investing is not determined by the frequency of transactions, but rather by the quality of one's investment decisions. Patience, in this context, is an invaluable asset for an investor.

https://www.youtube.com/@MaheswaraKaindraSTD

Include link of your video on YouTube in this section.

Acknowledgment (indonesian)

Penulis mengucapkan terima kasih kepada pihak-pihak yang telah membantu penulis selama melaksanakan perkuliahan IF2211 Strategi Algoritma pada tahun ajaran 2024/2025 (para Dosen, rekan kerja, hingga teman belajar) dan serta mereka yang telah memberikan saran berharga selama proses penulisan makalah ini.

Penulis menyadari bahwa makalah ini masih jauh dari sempurna. Oleh karena itu, repositori dari penelitian ini bersifat terbuka bagi siapa saja yang ingin melakukan eksplorasi atau penyempurnaan lebih lanjut.

Penulis berharap penelitian ini dapat memberi semangat dan pengetahuab baru bagi mahasiswa dan pelajar, dengan membuktikan bahwa konsep-konsep teoretis dari strategi algoritma dapat digunakan secara praktis untuk memodelkan dan menganalisis dilema finansial dalam kehidupan sehari-hari, terutama dalam berinvestasi.

References

[1] Munir, R. (2025). *Algoritma Greedy (Bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf. Program Studi Teknik Informatika, Institut Teknologi Bandung. Diakses 20 Juni 2025.

[2] Munir, R. (2025). *Program Dinamis (Bagian 1)* https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf. Program Studi Teknik Informatika, Institut Teknologi Bandung. Diakses 20 Juni 2025.

[3] Tutorials Point. (n.d.). *Design and analysis of algorithms - Multistage graph*. https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_multistage_graph.htm. Diakses 25 Juni 2025.

[4]

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Juni 2025

Maheswara Bayu Kaindra (13523015)