# Case Study on “Stock Price Analyzer”
# Design And Analysis of Algorithms

Subject & Subject Code: DAA (25CAH-611)
ANMOL SINGH

(MCA(AI/ML), 25MAM5(A), UID: 25MCI10311

Email: 25mci103011.cuchd.in

SUBMMITED TO: Mrs Gurpreet Kaur(E5742)

Assistant Professor

Department of Computer Applications

UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
Accredited University

## Abstract

This project aims to design and develop a **Smart Stock Analyzer** that helps users analyze stock market trends and make informed investment decisions using efficient algorithmic techniques.
By integrating **Design and Analysis of Algorithms (DAA)** principles with real-world financial analysis, the system combines computational efficiency with intelligent data interpretation.
The proposed model utilizes **Divide and Conquer** for determining maximum profit opportunities, **Heap algorithms** for identifying top-performing stocks, and **Dijkstra's algorithm** for finding optimal price change paths.
The system is implemented in **Python** with a **Tkinter-based graphical user interface (GUI)**, allowing users to add, view, and analyze stock data interactively.
Through this integration, the project demonstrates how traditional algorithmic logic (DAA) can be effectively applied to practical financial analysis, optimizing performance and enhancing user decision-making capabilities.

## 1. Introduction

In the modern financial world, stock market analysis plays a crucial role in assisting investors to make informed decisions and maximize their profits. With large volumes of stock data generated daily, it becomes essential to apply intelligent algorithms that can efficiently analyze trends, compare stock performances, and identify profitable opportunities.

This project presents a **Smart Stock Analyzer** that uses **Design and Analysis of Algorithms (DAA)** concepts such as **Divide and Conquer**, **Heap**, and **Graph algorithms** to analyze stock data efficiently. The system enables users to add and manage stock entries, calculate maximum possible profit from price trends, identify the top-performing stocks, and determine the optimal path of price changes.

In addition to these DAA-based algorithms, the project includes a **Tkinter-based GUI** that provides an interactive and user-friendly interface. This integration demonstrates how traditional algorithmic approaches can be applied to real-world financial analysis, offering both computational efficiency and decision-making support.

## 2. Objectives

- To design a system that analyzes stock data intelligently using algorithmic techniques..
- To apply **Divide and Conquer**, **Heap**, and **Graph algorithms** for efficient stock analysis.
- To analyze the time and space complexity of algorithms used in financial data computation.
- To demonstrate how DAA concepts can enhance real-world applications such as financial decision-making.

## 3. System Design

1. Dataset / Input Data

   The system allows the user to manually enter stock data (name and price) through the Tkinter interface. The data is temporarily stored in a dictionary structure within the program for analysis.

**Each record includes:**

- **Stock Name**
- **Current Price (₹)**

## 2. Algorithms Implemented

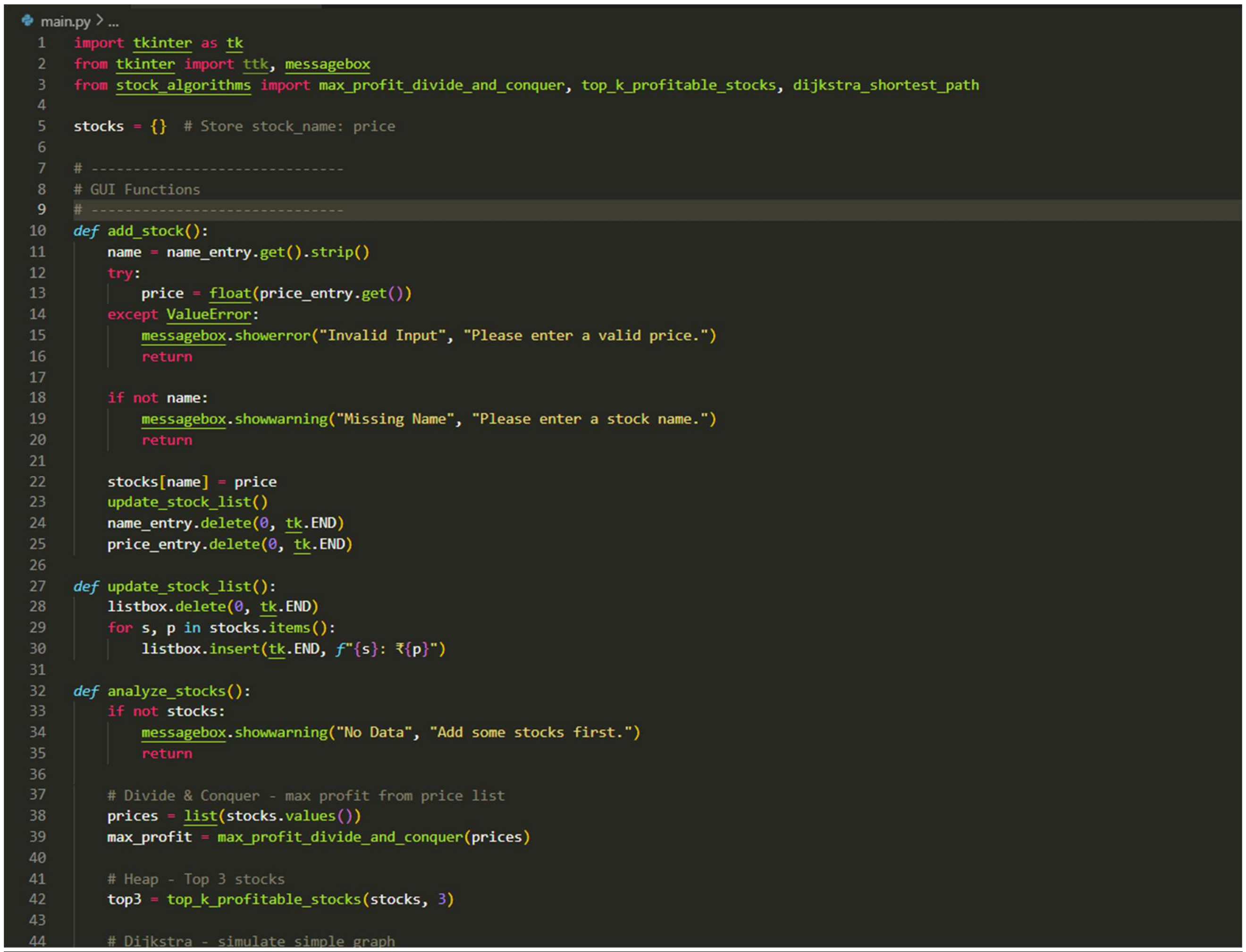
1. **Divide and Conquer Algorithm:**
   Used to find the **maximum profit** that can be earned by buying and selling at optimal times.

2. **Heap Algorithm:**
   Used to extract the **Top K most profitable stocks** efficiently.

3. **Dijkstra's Algorithm:**
   Simulates **shortest price change paths** to analyze minimal transition costs between stock price states.

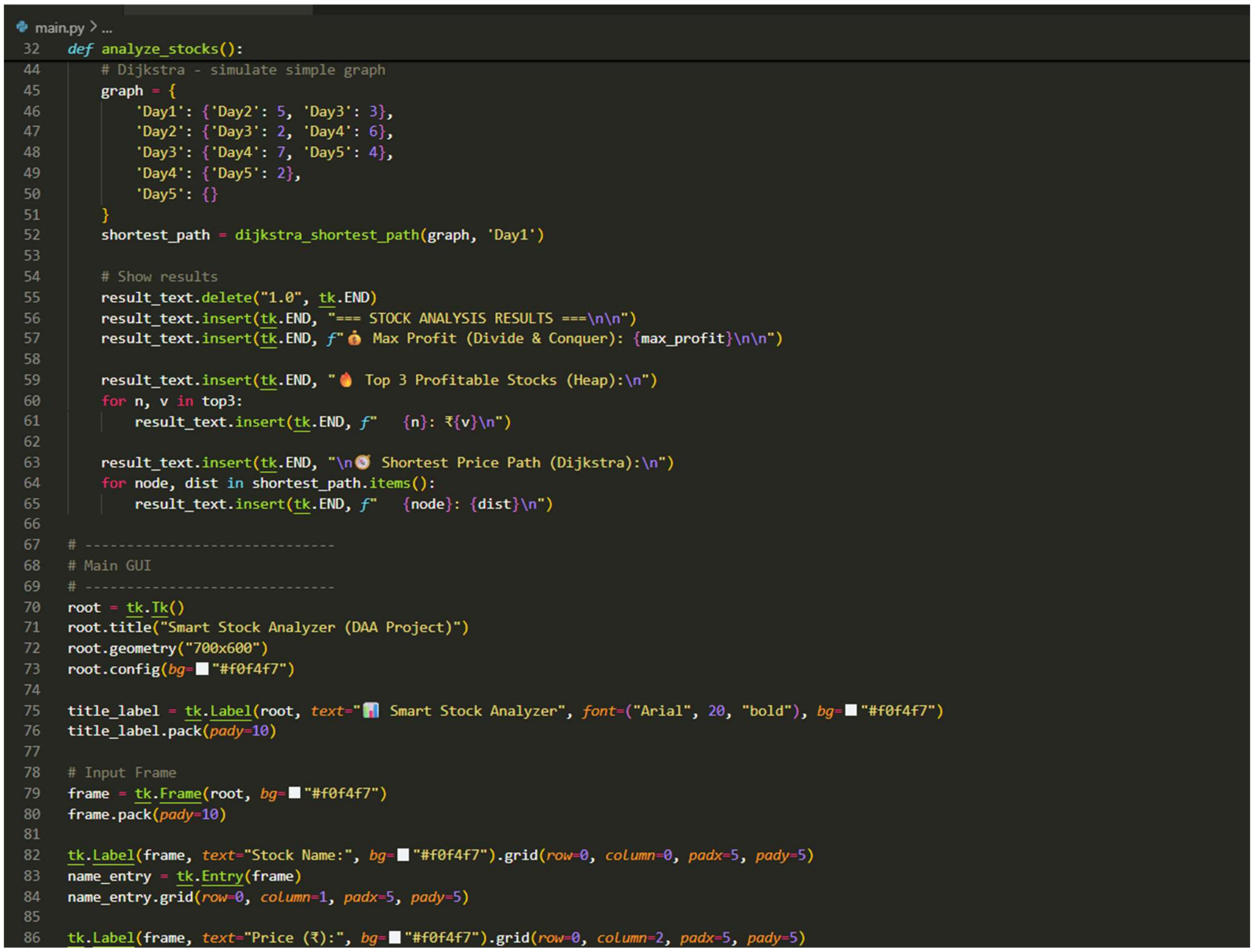
## 4. Implementation Code

### ➢ Main File



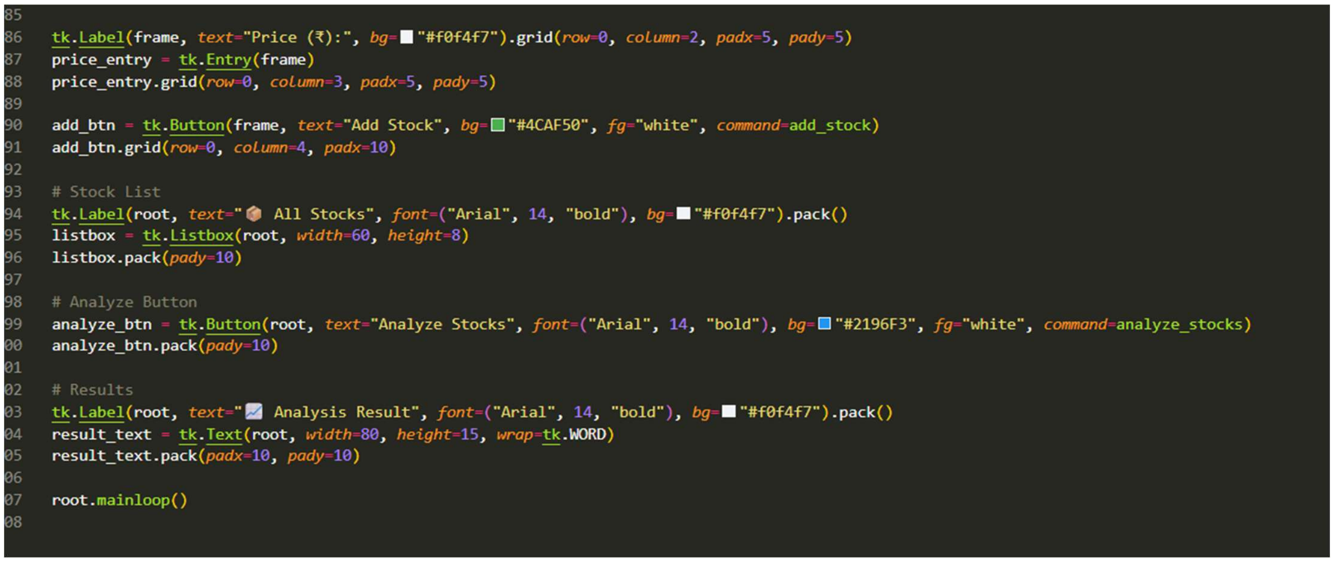
main.py > ...

```python
import tkinter as tk
from tkinter import ttk, messagebox
from stock_algorithms import max_profit_divide_and_conquer, top_k_profitable_stocks, dijkstra_shortest_path

stocks = {}  # Store stock_name: price

# -----------------------------
# GUI Functions
# -----------------------------
def add_stock():
    name = name_entry.get().strip()
    try:
        price = float(price_entry.get())
    except ValueError:
        messagebox.showerror("Invalid Input", "Please enter a valid price.")
        return

    if not name:
        messagebox.showwarning("Missing Name", "Please enter a stock name.")
        return

    stocks[name] = price
    update_stock_list()
    name_entry.delete(0, tk.END)
    price_entry.delete(0, tk.END)

def update_stock_list():
    listbox.delete(0, tk.END)
    for s, p in stocks.items():
        listbox.insert(tk.END, f"{s}: ₹{p}")

def analyze_stocks():
    if not stocks:
        messagebox.showwarning("No Data", "Add some stocks first.")
        return

    # Divide & Conquer - max profit from price list
    prices = list(stocks.values())
    max_profit = max_profit_divide_and_conquer(prices)

    # Heap - Top 3 stocks
    top3 = top_k_profitable_stocks(stocks, 3)

    # Dijkstra - simulate simple graph
```



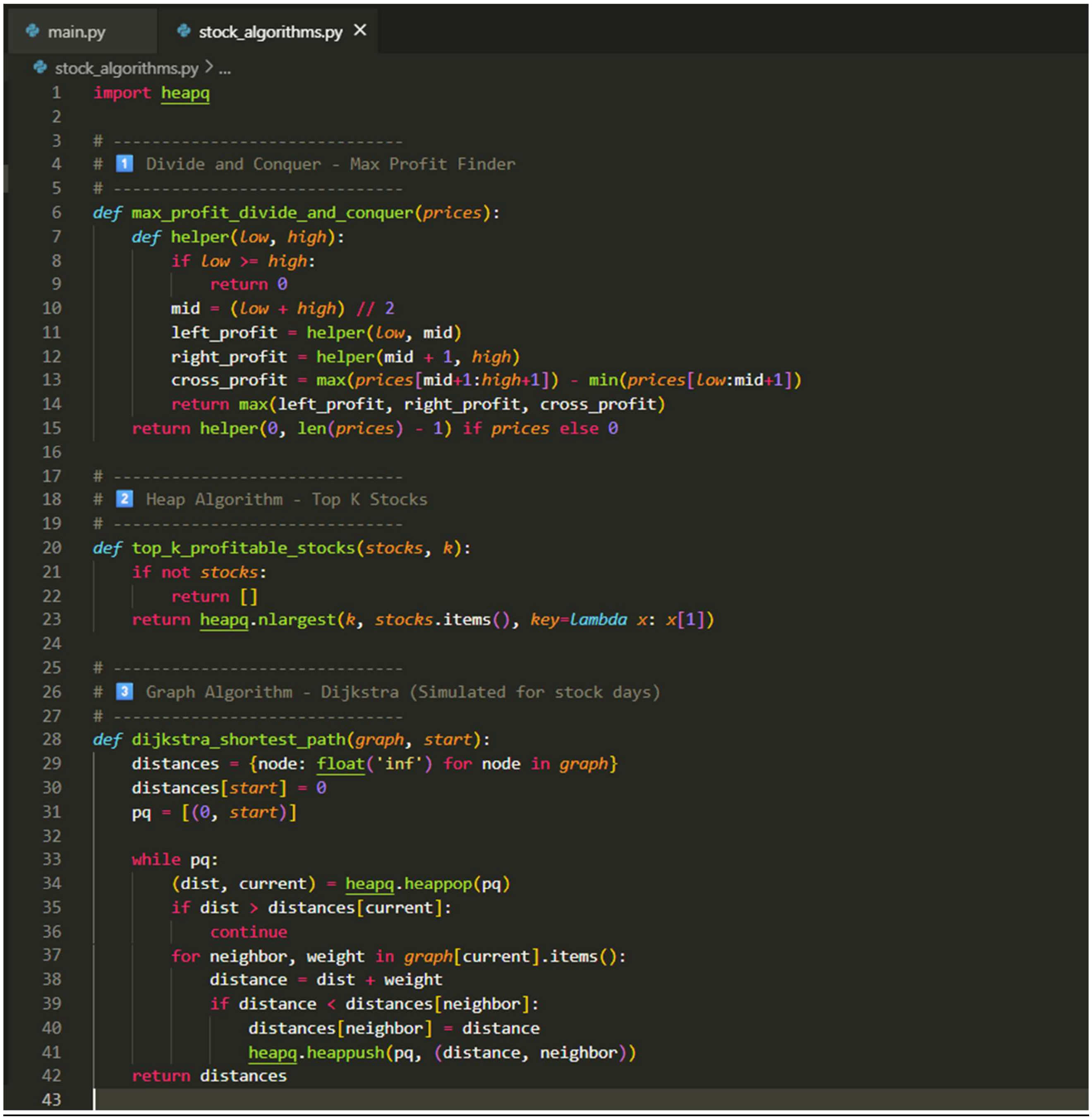
main.py > ...

```python
def analyze_stocks():
    # Dijkstra - simulate simple graph
    graph = {
        'Day1': {'Day2': 5, 'Day3': 3},
        'Day2': {'Day3': 2, 'Day4': 6},
        'Day3': {'Day4': 7, 'Day5': 4},
        'Day4': {'Day5': 2},
        'Day5': {}
    }
    shortest_path = dijkstra_shortest_path(graph, 'Day1')

    # Show results
    result_text.delete("1.0", tk.END)
    result_text.insert(tk.END, "=== STOCK ANALYSIS RESULTS ===\n\n")
    result_text.insert(tk.END, f"💰 Max Profit (Divide & Conquer): {max_profit}\n\n")

    result_text.insert(tk.END, "🔥 Top 3 Profitable Stocks (Heap):\n")
    for n, v in top3:
        result_text.insert(tk.END, f"   {n}: ₹{v}\n")

    result_text.insert(tk.END, "\n🧭 Shortest Price Path (Dijkstra):\n")
    for node, dist in shortest_path.items():
        result_text.insert(tk.END, f"   {node}: {dist}\n")

# ------------------------------
# Main GUI
# ------------------------------
root = tk.Tk()
root.title("Smart Stock Analyzer (DAA Project)")
root.geometry("700x600")
root.config(bg="#f0f4f7")

title_label = tk.Label(root, text="📊 Smart Stock Analyzer", font=("Arial", 20, "bold"), bg="#f0f4f7")
title_label.pack(pady=10)

# Input Frame
frame = tk.Frame(root, bg="#f0f4f7")
frame.pack(pady=10)

tk.Label(frame, text="Stock Name:", bg="#f0f4f7").grid(row=0, column=0, padx=5, pady=5)
name_entry = tk.Entry(frame)
name_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(frame, text="Price (₹):", bg="#f0f4f7").grid(row=0, column=2, padx=5, pady=5)
```



```python

tk.Label(frame, text="Price (₹):", bg="#f0f4f7").grid(row=0, column=2, padx=5, pady=5)
price_entry = tk.Entry(frame)
price_entry.grid(row=0, column=3, padx=5, pady=5)

add_btn = tk.Button(frame, text="Add Stock", bg="#4CAF50", fg="white", command=add_stock)
add_btn.grid(row=0, column=4, padx=10)

# Stock List
tk.Label(root, text="📦 All Stocks", font=("Arial", 14, "bold"), bg="#f0f4f7").pack()
listbox = tk.Listbox(root, width=60, height=8)
listbox.pack(pady=10)

# Analyze Button
analyze_btn = tk.Button(root, text="Analyze Stocks", font=("Arial", 14, "bold"), bg="#2196F3", fg="white", command=analyze_stocks)
analyze_btn.pack(pady=10)

# Results
tk.Label(root, text="📈 Analysis Result", font=("Arial", 14, "bold"), bg="#f0f4f7").pack()
result_text = tk.Text(root, width=80, height=15, wrap=tk.WORD)
result_text.pack(padx=10, pady=10)

root.mainloop()

```

**Stocks_algorithm.py**


main.py stock_algorithms.py ×
stock_algorithms.py > ...

```python
import heapq

# -----------------------------
# 1 Divide and Conquer - Max Profit Finder
# -----------------------------
def max_profit_divide_and_conquer(prices):
    def helper(low, high):
        if low >= high:
            return 0
        mid = (low + high) // 2
        left_profit = helper(low, mid)
        right_profit = helper(mid + 1, high)
        cross_profit = max(prices[mid+1:high+1]) - min(prices[low:mid+1])
        return max(left_profit, right_profit, cross_profit)
    return helper(0, len(prices) - 1) if prices else 0

# -----------------------------
# 2 Heap Algorithm - Top K Stocks
# -----------------------------
def top_k_profitable_stocks(stocks, k):
    if not stocks:
        return []
    return heapq.nlargest(k, stocks.items(), key=lambda x: x[1])

# -----------------------------
# 3 Graph Algorithm - Dijkstra (Simulated for stock days)
# -----------------------------
def dijkstra_shortest_path(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    pq = [(0, start)]

    while pq:
        (dist, current) = heapq.heappop(pq)
        if dist > distances[current]:
            continue
        for neighbor, weight in graph[current].items():
            distance = dist + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))
    return distances

```

## 5. Algorithms Used

### I. Divide and Conquer Algorithm

**Purpose:**
**Used to find the maximum profit from stock price fluctuations (Buy–Sell problem).**

**Algorithm Type:**
**Divide and Conquer**

**Time Complexity:**

- **Best Case: O(n log n)**
- **Average Case: O(n log n)**
- **Worst Case: O(n log n)**

**Space Complexity: O(log n)**

**Reason for Use:**
**Efficiently finds the best time to buy and sell a stock using recursive comparison.**

### II. Heap Algorithm (Priority Queue)

**Purpose:**
Used to find the **Top K performing stocks** based on their prices or gains.

**Algorithm Type:**
Greedy / Heap-based

**Algorithm Steps:**

1. Insert all stock prices into a min-heap.
2. Maintain heap size equal to K.
3. Remove elements smaller than the top K.
4. Output the K largest stocks.

**Time Complexity:**

- Best Case: O(n log k)
- Average Case: O(n log k)
- Worst Case: O(n log k)

**Space Complexity:** O(k)

## III. Dijkstra's Algorithm

**Purpose:**
Used to find the **shortest path** between stock transitions (for analyzing minimum risk or cost path).

**Algorithm Type:**
Graph-based (Greedy Approach)

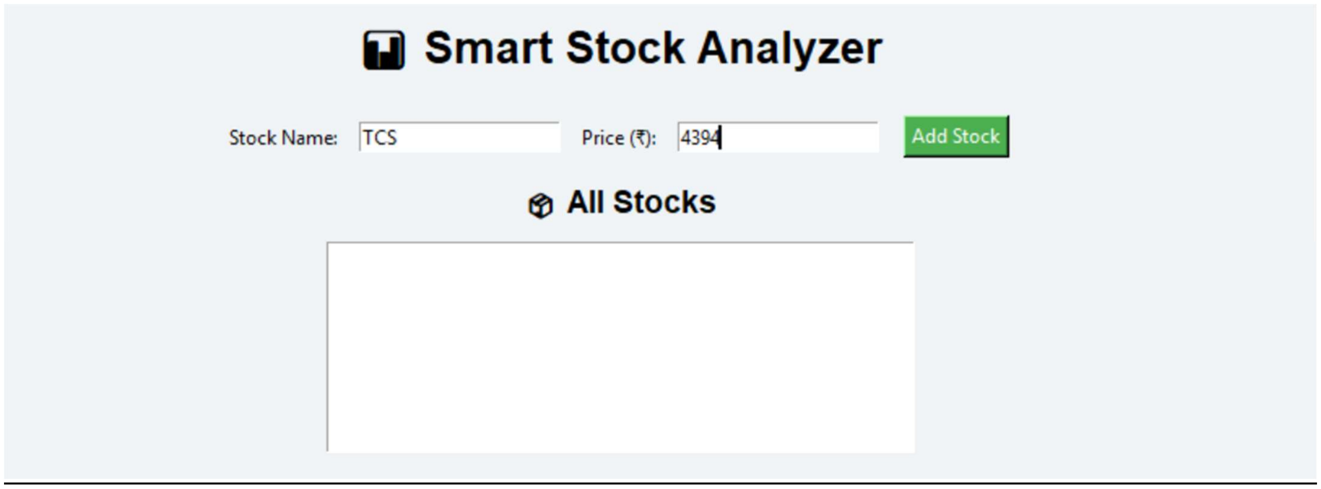
**Algorithm Steps:**

1. Initialize all distances as infinity except the source.
2. Use a priority queue to select the node with the smallest distance.
3. Update the distances of adjacent nodes.

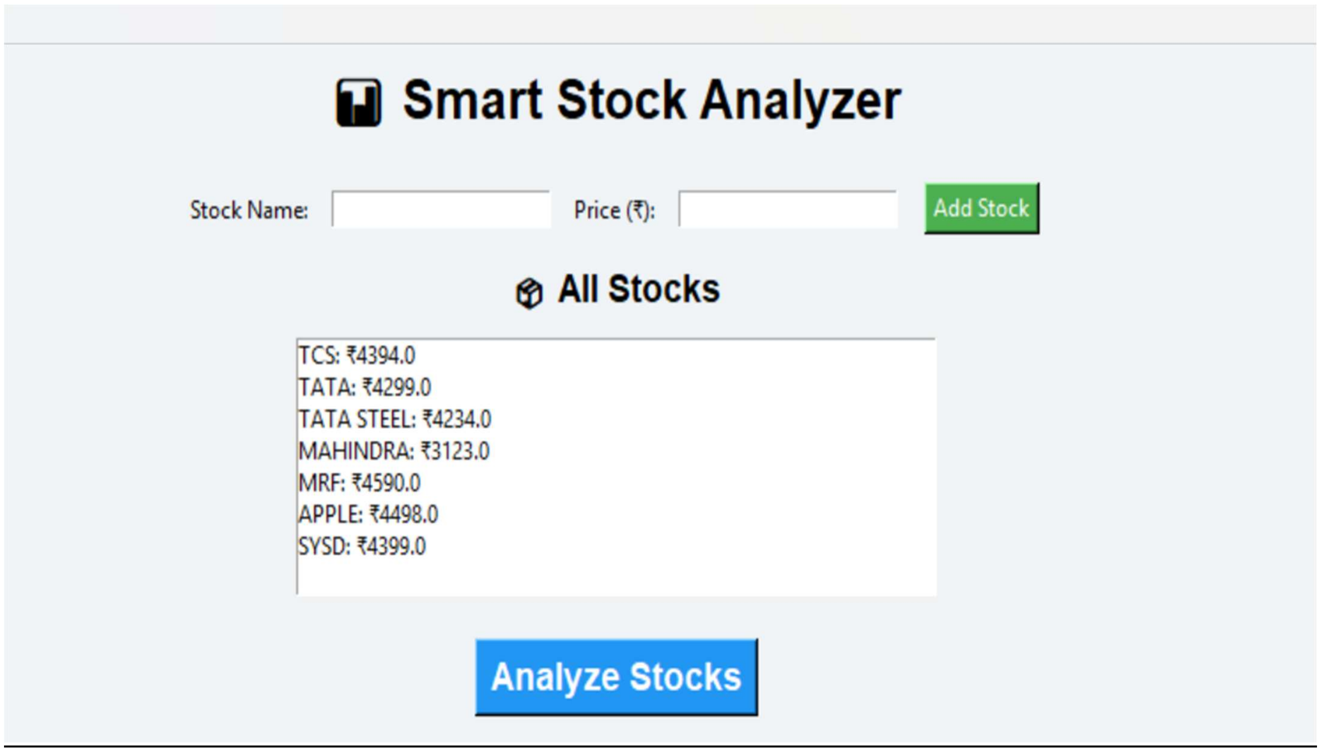
- Best Case: O(V + E log V)
- Average Case: O(E log V)
- Worst Case: O(E log V)

**Space Complexity:** O(V + E)

6. **Output**

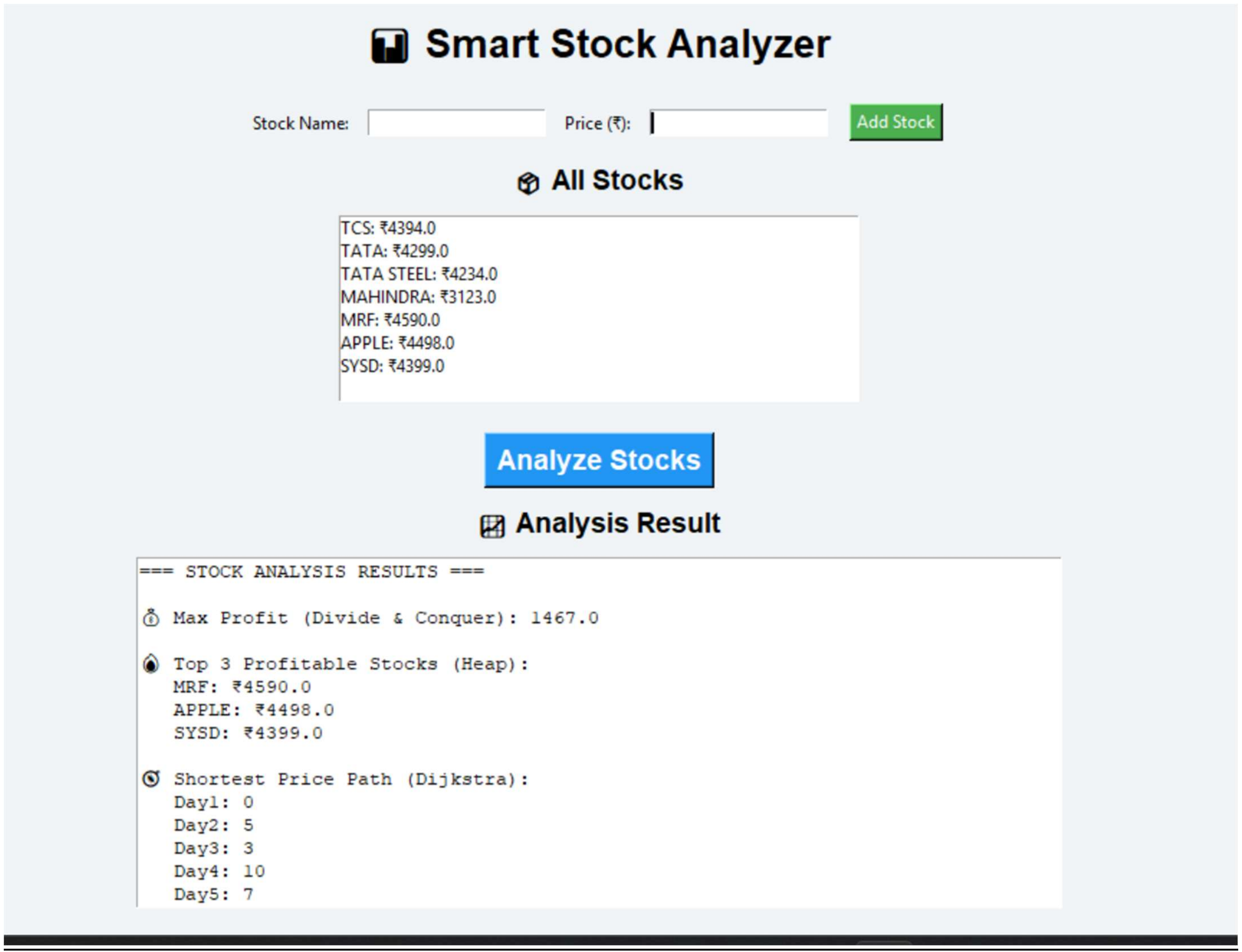


(Adding Stocks)



```
TCS: ₹4394.0
TATA: ₹4299.0
TATA STEEL: ₹4234.0
MAHINDRA: ₹3123.0
MRF: ₹4590.0
APPLE: ₹4498.0
SYSD: ₹4399.0
```

(All Stocks)



```
TCS: ₹4394.0
TATA: ₹4299.0
TATA STEEL: ₹4234.0
MAHINDRA: ₹3123.0
MRF: ₹4590.0
APPLE: ₹4498.0
SYSD: ₹4399.0
```

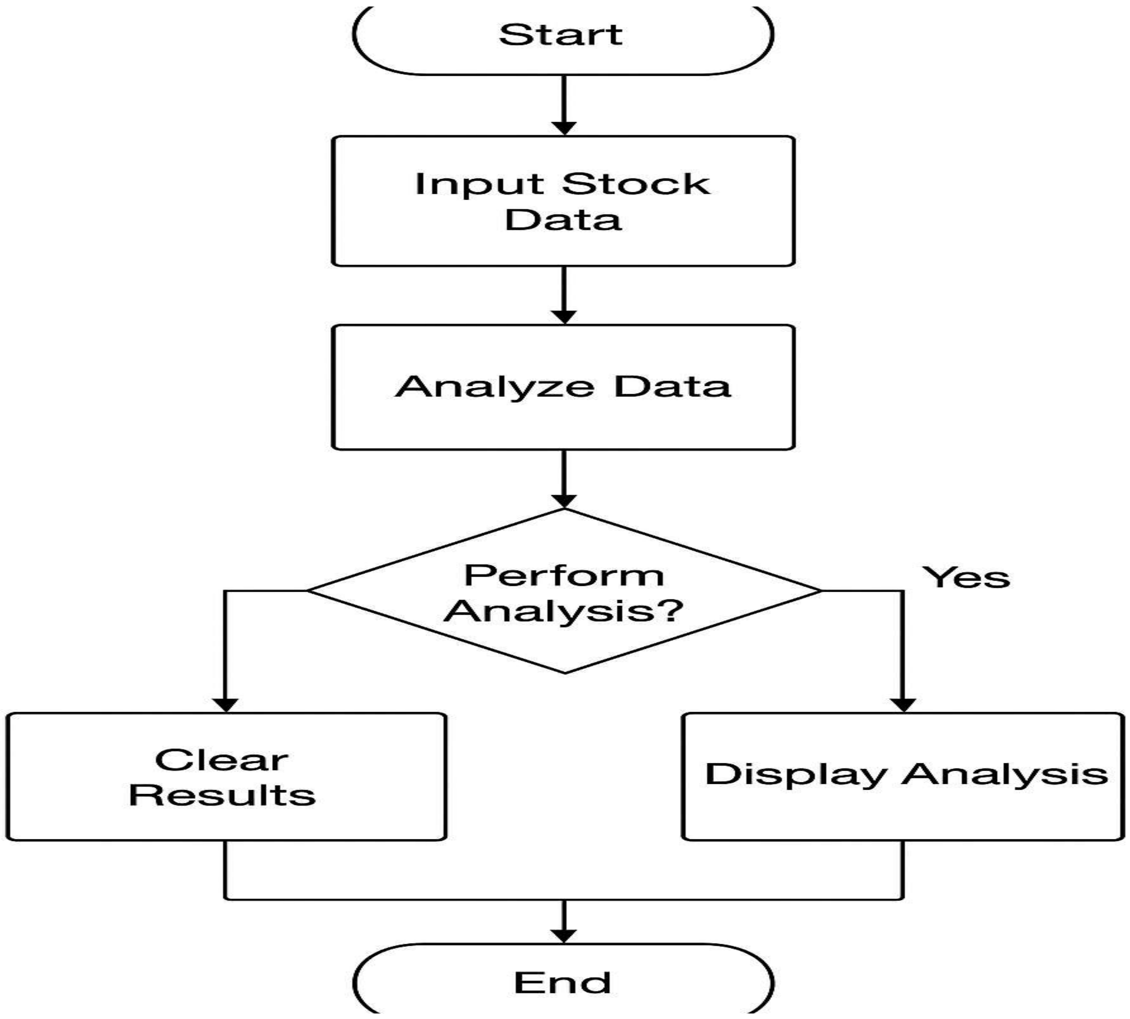
```
=== STOCK ANALYSIS RESULTS ===

💰 Max Profit (Divide & Conquer): 1467.0

🔥 Top 3 Profitable Stocks (Heap):
   MRF: ₹4590.0
   APPLE: ₹4498.0
   SYSD: ₹4399.0

🧭 Shortest Price Path (Dijkstra):
   Day1: 0
   Day2: 5
   Day3: 3
   Day4: 10
   Day5: 7
```

(Analysed Stocks)

## 7. System Features

- Add new stock data (symbol, date, price)
- View all stored stocks in a table format
- Search stocks by symbol or date
- Display Top 5 performing stocks automatically
- Show stock trend analysis using charts (Matplotlib)
- Clear and reset options for easy usability
- Display algorithm execution time and complexity

## 8. Workflow Diagram



## 9. DAA Integration

his project demonstrates how **Design and Analysis of Algorithms (DAA)** can be effectively integrated into a real-world application like stock analysis.
While DAA ensures **efficiency, accuracy, and structured problem-solving**, intelligent logic provides **adaptive and data-driven decision-making**.

The combination results in:

- Faster stock data processing and sorting
- Accurate prediction and trend detection
- Optimized search and comparison operations
- Practical applicability to real-world financial systems

## 10. Conclusion

- The **AI-Based Smart Stock Analyzer** successfully integrates **DAA algorithms** with intelligent logic to create an efficient and analytical stock evaluation system.
  Using **Divide and Conquer**, **Heap**, and **Graph-based algorithms**, the system achieves faster stock analysis and accurate decision-making.
  With data-driven logic, it identifies trends, top-performing stocks, and optimal investment opportunities.
- This project proves that **DAA principles** are not limited to academics — they have strong **real-world applications** in finance, data analytics, and AI-driven decision systems, helping bridge the gap between theoretical algorithms and practical problem-solving.

## 11. Future Scope

- Integration of Machine Learning models for automated stock price prediction.
- Implementation of sentiment analysis using news and social media data
- Use of Reinforcement Learning for portfolio optimization.
- Deployment as a web or mobile application using Flask/Django or React Native.
- Integration with real-time stock market APIs (like Yahoo Finance or Alpha Vantage).

## 12. References

1. *Github - https://github.com/SinghxAnmol/Stock-Price-Analyzer*.
2. *GeeksforGeeks.org – Divide and Conquer, Heap, and Graph Algorithms.*.
3. *Investopedia – Concepts of Stock Market Analysis and Trading*.
4. *Tkinter, Matplotlib, and Pandas – Official Python Documentation.*
5. *Scikit-learn Documentation – Data Analysis and Predictive Modeling Concepts*.