SMART COMMAND MANAGER

PROJECT REPORT

Contents

[1.Problem statement overview 3](#_Toc202275084)

[2.thought process and approach 4](#_Toc202275085)

[3.data structures used 5](#_Toc202275086)

[4.time and space complexity analysis 6](#_Toc202275087)

[5.how to run application 7](#_Toc202275088)

[6.sample input output commands 8](#_Toc202275089)

[7.flow chart 9](#_Toc202275090)

# 1.Problem statement overview

The goal of this project is to design and implement a **Smart Inventory Tracker** tailored for a small retail store. The store requires a simple but efficient system to manage its product inventory digitally. The application should allow store staff to:

* **Add new products** to the inventory, along with their quantity and timestamp.
* **Remove products** that are no longer in stock or need to be updated.
* **Check the availability** of specific products instantly.
* **View the entire inventory** in a sorted and user-friendly format.

To ensure a smooth and responsive user experience, the system must be designed with **optimized data structures** that allow operations like adding, removing, and checking products to occur in **constant (O(1)) or near-constant time**. Additionally, it should support **sorted display** using efficient sorting mechanisms such as heaps.

The application aims to minimize manual errors, speed up inventory tasks, and provide clear, organized insights into stock levels—making it especially useful for small businesses with limited resources.

# 2.thought process and approach

In today’s world, businesses require systems that can efficiently manage large inventories with speed and accuracy. Our goal was to design a **Smart Inventory Tracker** that is fast, scalable, and user-friendly.

We focused on:

* **Speed**: Using efficient data structures for quick add, remove, and search operations.
* **Simplicity**: A clean interface so even non-technical users can use it easily.
* **Scalability**: Capable of handling large inventories without slowing down.
* **Reliability**: Handles invalid inputs and prevents crashes.

Python was chosen for its simplicity and flexibility.The result is a smart, responsive, and easy-to-use inventory system fit for modern business needs.

# 3.data structures used

**priority queue(heap)**

We use a **min-heap** to always pick the command with the **highest priority (smallest number)** first. It's perfect for auto-sorting tasks that must be executed in priority order.

**Stack (undo/redo)**

We store executed commands in a **stack** so we can easily undo the last action — just like going one step back in a browser. Redo uses another stack to bring it back.

**Dictionary lookup**

We use a **dictionary** to store command IDs and their text for **quick access** — it’s way faster than searching a list every time.

**Enumerate and sorting**

helps us show serial numbers (like 1, 2, 3...) alongside commands. Sorting keeps them ordered by priority, so the table looks clean and logical.

**ANSI color coding**

We add **colors and bold text** using special terminal codes — making output easier to read, like red for high priority or green for executed.

**Modular design with separate files**

Each feature (execution, undo, view, etc.) is in its **own Python file**, making the code clean, organized, and easier to maintain or upgrade.

**Input validation and error handling**

We check every user input — like only allowing numbers or handling out-of-range options — so the program doesn’t crash and feels smooth to use.

# 4.time and space complexity analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Dictionary (dict)** | **Queue - list based** | **Queue - deque based** |
| **Access by Key (e.g., dict[k])** | 🟢 **O(1)** average | 🔴 **O(n)** (search loop) | 🔴 **O(n)** (search loop) |
| **Insert (Add)** | 🟢 O(1) | 🟢 O(1) (append) | 🟢 O(1) (append) |
| **Delete by Key** | 🟢 O(1) | 🔴 O(n) (search + remove) | 🔴 O(n) (search + remove) |
| **Pop First Item** | ❌ N/A (unordered) | 🔴 O(n) (slow in list) | 🟢 O(1) (fast in deque) |
| **Search by Value** | 🔴 O(n) | 🔴 O(n) | 🔴 O(n) |
| **Space Complexity** | 🟢 O(n) | 🟢 O(n) | 🟢 O(n) |

**O(1) – Constant Time 🟢**

The operation takes the same amount of time no matter how much data there is.

**O(n) – Linear Time 🔴**

The time grows directly with the number of elements.

# 5.how to run application.

### **Requirements:**

* Python 3.x installed on your system.
* Required libraries:
  + ***pyfiglet***(for stylized title)
  + ***heapq*** (built-in, no install needed)
  + ***datetime*** (built-in, no installation needed)
  + ***tabulation(****for tables printing)*
  + ***emoji****(for printing various emojis on terminal)*

You can install required libraries by running:

*pip install pyfiglet*

*pip install tabulation*

*pip install emoji*

### **Steps to Run (Command-Line Version):**

1. Open **Command Prompt** (CMD).
2. Navigate to the project folder directory in your computer :

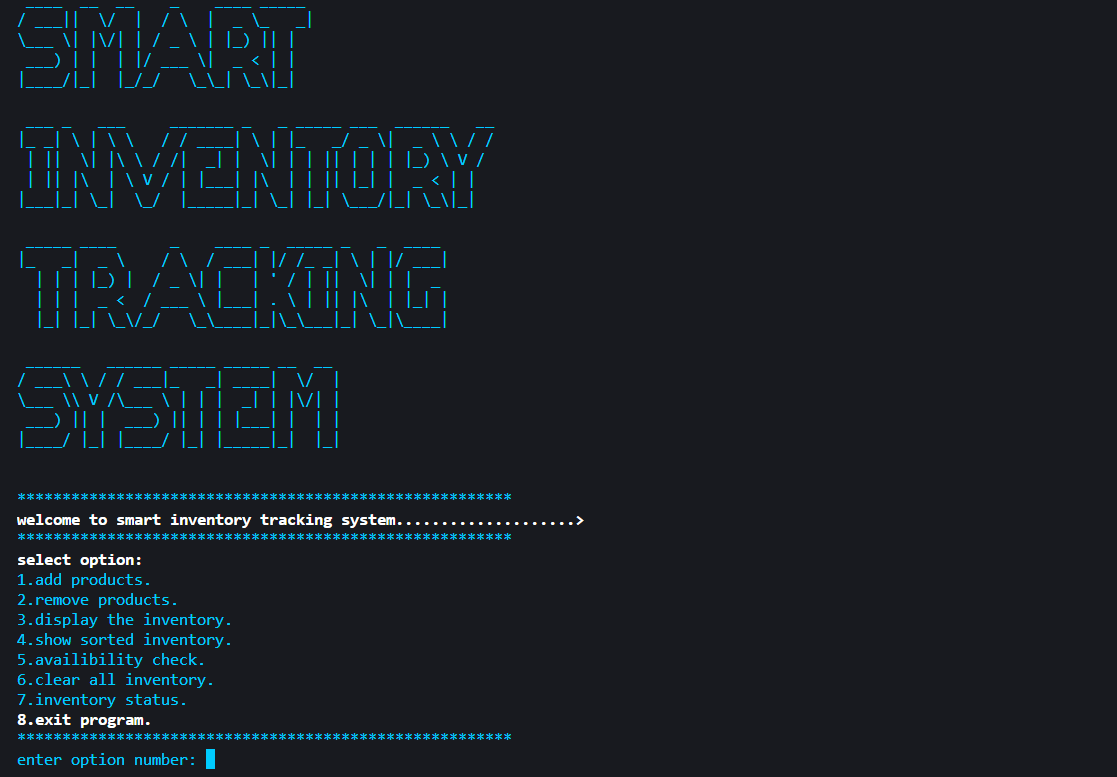
For example: *cd D: \DSA\projects\project2*

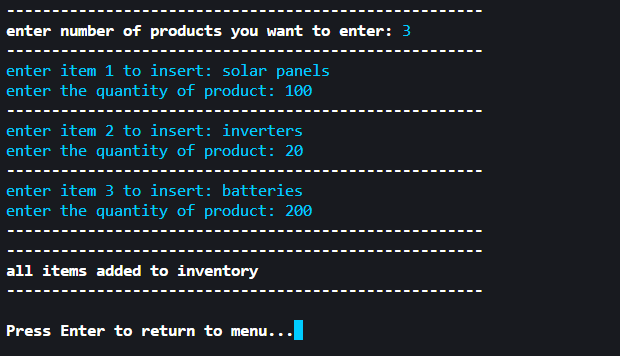
1. Run the Python script:

*Python main.py*

1. The program will launch in terminal.
2. Follow on-screen instructions to use the inventory system.

# 6.sample input output commands





A screen shot of a computer

AI-generated content may be incorrect.

# 7.flow chart

**START PROGRAM**

**DISPLAY TITLE**

**SMART COMMAND MANAGER**

**DISPLAY MENU**

**1–ADD COMMAND 2–EXECUTE**

**3–DISPLAY TABLE 4–DELETE**

**5-PENDING COMMANDS**

**6–EXECUTED COMMANDS**

**7-STATUS OF COMMANDS**

**8- EXIT PROGRAM**

**8–EXIT**

**ADD: ADD COMMAND TO TABLE**

**DELETE: DELETE COMMAND**

**UNDO AND REDO FUNCTIONS**

**STATUS: COMMANDS STATUS**

**CLEAR: EMPTY ARRAY**

**DISPLAY: DISPLAY TABLE**

**LOOP OR EXIT(8)**

**PERFORM ACTION BASED ON**

**INPUT (USING OOP CLASS)**