**Algorithm for Harbor Ship Management GUI**

**Done By** **-->** **Prem kumar R ( 23MIP10019 )**

**Introduction**

This document details an efficient algorithm for managing harbor operations, focusing on ship scheduling, container allocation, and slot visualization within a Python Tkinter GUI.

**Algorithm Steps**

**1. Initialization**

**Main Window Setup**: Configure the primary Tk window with appropriate title, dimensions, background, and an application icon.

**Global Variables**: Establish variables for ship data, container details, and slot availability using lists and dictionaries.

**Helper Functions**: Create modular functions for sorting, scheduling, and container-to-slot mapping, ensuring real-time GUI updates.

**2. Core Functionalities**

**A. Harbor Slot Registration**

**Input Interface**: Facilitate the collection of slot availability through labeled input fields accompanied by a registration button.

**Visualization**: Display a dynamic grid showcasing available slots to streamline slot management.

**B. Ship Arrival Management (Using Job Sequencing Algorithm)**

**Data Entry**: Enable the input of ship-specific details, such as name, container count, and priority, via structured form fields.

**Dynamic Display**: Append the collected ship data to a scrollable table, ensuring real-time updates and data accessibility.

**C. Unloading Optimization (Using Traveling Salesperson Algorithm)**

**Scheduling Algorithm**: Sequence ships based on deadlines and priority, optimizing unloading operations for efficiency.

**Visualization**: Depict the generated schedule within a dedicated section of the GUI.

**D. Container Management (Using Multistage Algorithm)**

**Input Interface**: Collect container details, including ship association, destination, and deadlines, through user-friendly input fields.

**Dynamic Table**: Maintain a scrollable table to reflect container data, updated dynamically as new entries are made.

**E. Slot Allocation (Using Multistage Algorithm)**

**Allocation Logic**: Strategically assign containers to slots while prioritizing proximity and meeting deadlines.

**Overflow Detection**: Alert users when slot capacity is exceeded to facilitate proper resource management.

**Visualization**: Update the slot grid dynamically, reflecting current occupancy.

**F. Container Distribution (Using Traveling Salesperson Algorithm)**

**Input Mechanism**: Support interactive user assignments of containers to ships.

**Optimization**: Minimize transport distances using advanced algorithms, with results presented visually for user clarity.

**3. GUI Design**

**Layout**

Organize the interface into logical panels for slot registration, ship management, and container operations.

Employ Canvas and Scrollbar widgets to ensure smooth navigation of extensive datasets.

**Dynamic Updates**

Automate the refresh of tabular data and slot grids upon new entries or updates, enhancing responsiveness.

**4. Core Algorithms**

**A. distribute\_containers\_among\_ships**

Implement a greedy strategy to minimize transport distances by sorting containers based on proximity and deadlines.

**B. job\_sequencing**

Sequence ships by evaluating deadlines and importance, ensuring optimal slot allocation.

**C. store\_containers\_in\_harbor**

Allocate containers to the nearest available slots using an optimized placement approach.

**D. display\_harbor\_status**

Iteratively update the GUI to provide a visual representation of slot occupancy.

**5. Error Management**

Enforce input validation to guarantee data accuracy and integrity.

Incorporate overflow detection mechanisms to effectively handle capacity constraints.

**6. Workflow Summary**

1. Register harbor slots and update the grid visualization.
2. Input ship details, displaying them in a dynamically updated scrollable table.
3. Record container information and reflect updates within the table.
4. Allocate containers to ships, visualizing assignments interactively.
5. Assign containers to slots and refresh the grid to reflect real-time occupancy.

**7. Exit Procedure**

Implement a structured closure process, ensuring all data is securely saved before exiting the application.

Time Complexity Report for the Harbor Ship Management GUI Application

* Adding Ship Details: O(m), where m is the number of ships.
* Job Sequencing: O(m log m) , where m is the number of ships.
* Container Management: O(n), where n is the number of containers.
* Storing Containers in Harbor: O(m k.log.k), where mmm is the number of ships and k is the number of containers per ship.
* Harbor Availability Visualization: O(s), where s is the number of Harbor slots.
* Distributing Containers Among Ships: O(n log n), where n is the number of containers.
* Scrollable Tables and Dynamic Entries: O(m + n), where m is the number of ships and n is the number of containers.
* Updating Containers Table: O(n), where n is the number of containers.

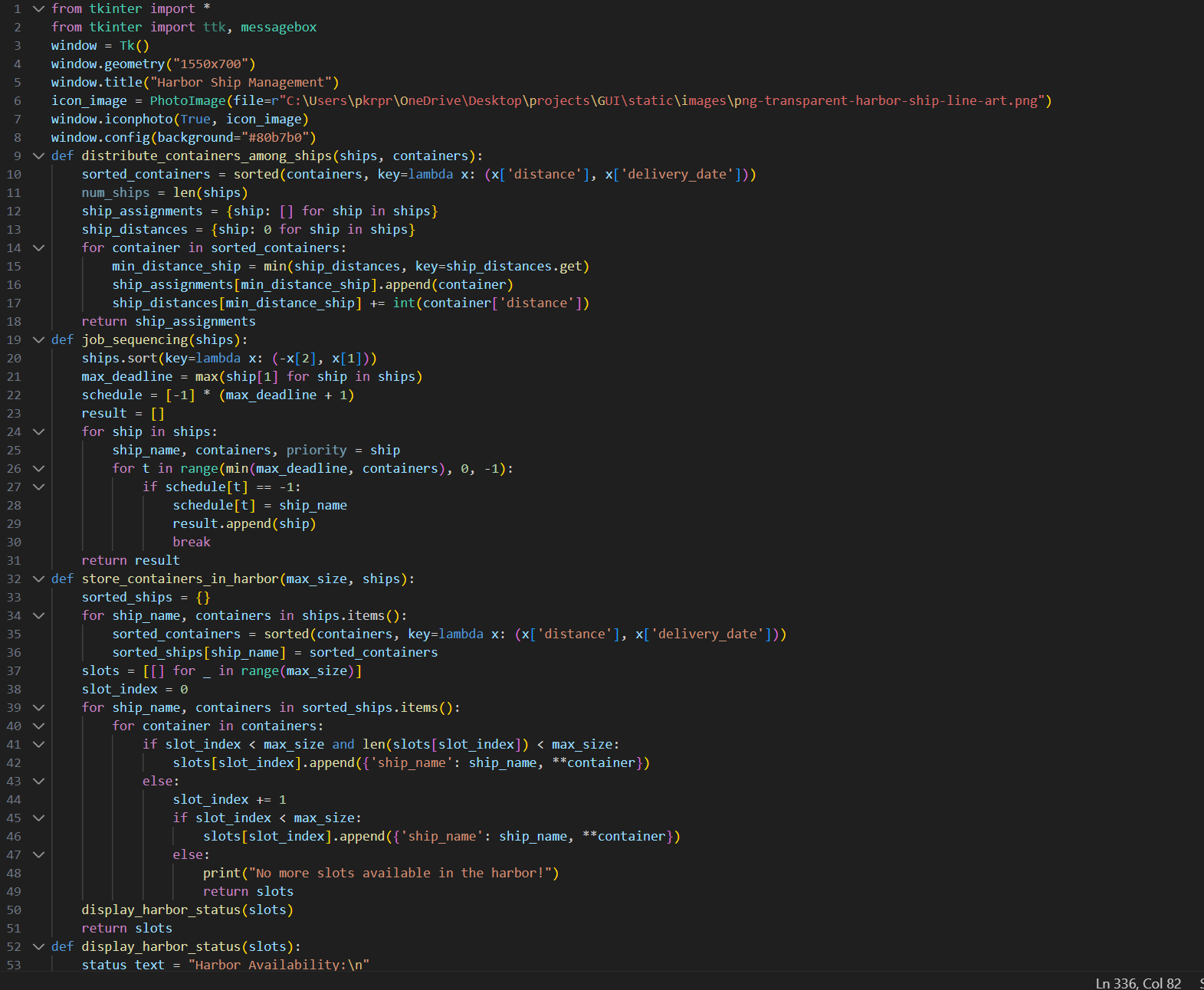
Thus, the **overall time complexity** of the system is:

**O(m log m + m ⋅ k log k + n log n)**

Where:

* m = Number of ships
* n = Number of containers
* k = Number of containers per ship
* s = Number of harbor slots (which is typically much smaller in comparison and can be ignored for the overall complexity)

Code :



A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated A screen shot of a computer code

Description automatically generated

A screen shot of a computer screen

Description automatically generated

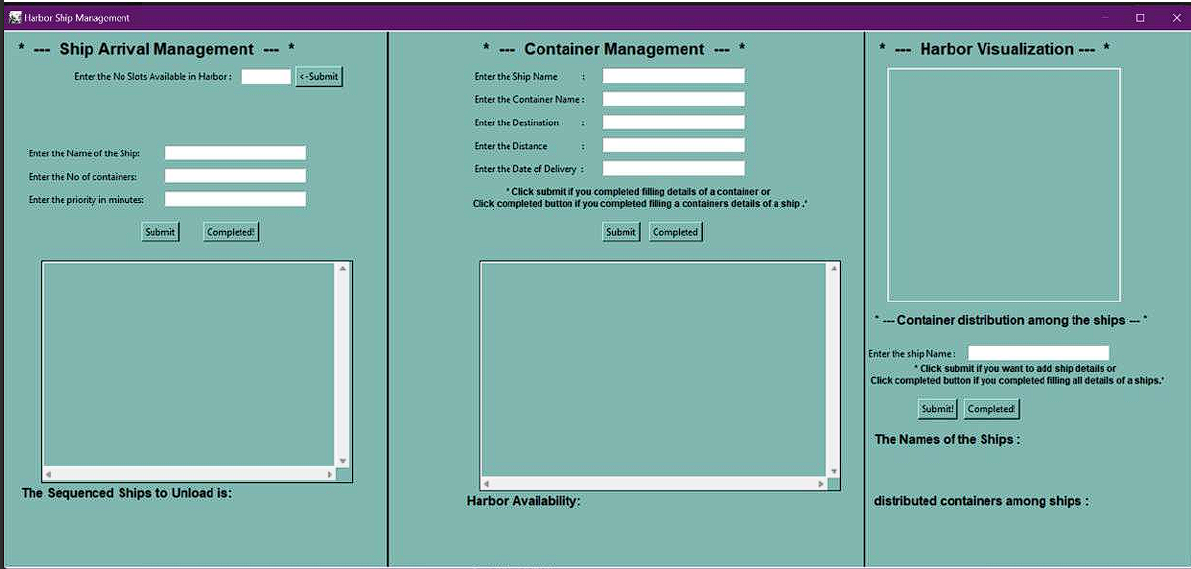
A computer screen shot of a building

Description automatically generated

A screen shot of a computer program

Description automatically generated

Output :



A screenshot of a computer

Description automatically generated