Code

```
import random

class ClimateRecords:
    def __init__(self, num_years, num_cities):
        self.num_years = num_years
        self.num_cities = num_cities
        self.records = [[None for _ in range(num_cities)] for _ in
range(num_years)]

    def auto_fill(self):
        for y in range(self.num_years):
            for c in range(self.num_cities):
                self.records[y][c] = round(random.uniform(15, 40), 1)
        print("Data auto-filled with random values.")

    def add_entry(self, yr, ct, temperature):
        if 0 <= yr < self.num_years and 0 <= ct < self.num_cities:
            self.records[yr][ct] = temperature
            print("Entry added successfully.")
        else:
            print("Invalid index for year/city.")

    def remove_entry(self, yr, ct):
        if 0 <= yr < self.num_years and 0 <= ct < self.num_cities:
            self.records[yr][ct] = None
            print("Entry removed successfully.")
        else:
            print("Invalid index for year/city.")

    def fetch(self, yr, ct):
        if 0 <= yr < self.num_years and 0 <= ct < self.num_cities:
            return self.records[yr][ct] if self.records[yr][ct] is not None
else "No Data"
        return "Invalid index"

    def row_view(self):
        print("\nRow-Major Display:")
        for r in self.records:
            print([v if v is not None else "NA" for v in r])
    def column_view(self):
        print("\nColumn-Major Display:")
        for c in range(self.num_cities):
            column = []
            for y in range(self.num_years):
                column.append(self.records[y][c] if self.records[y][c] is not
None else "NA")
            print(column)

    def sparse_check(self):
        print("\nSparse Data Handling:")
        for y in range(self.num_years):
            row = []
            for c in range(self.num_cities):
                row.append(self.records[y][c] if self.records[y][c] is not
None else "NA")
            print(row)

    def complexity_info(self):
        print("\nOperation Complexity:")
        print("Add -> O(1)")
        print("Remove -> O(1)")
        print("Fetch -> O(1)")
        print(f"Space -> O(years * cities) = O({self.num_years} *
{self.num_cities})")


if __name__ == "__main__":
    num_years = int(input("Enter total years: "))
    num_cities = int(input("Enter total cities: "))
    climate = ClimateRecords(num_years, num_cities)

    while True:
        print("\n--- Climate Records Manager ---")
        print("1. Auto Fill")
        print("2. Add Entry")
        print("3. Remove Entry")
        print("4. Fetch Entry")
        print("5. Row View")
        print("6. Column View")
        print("7. Sparse Check")
        print("8. Complexity Info")
print("9. Exit")

option = input("Choose an option: ")

if option == "1":
   climate.auto_fill()
elif option == "2":
   y = int(input("Enter year index: "))
   c = int(input("Enter city index: "))
   t = float(input("Enter temperature: "))
   climate.add_entry(y, c, t)
elif option == "3":
   y = int(input("Enter year index: "))
   c = int(input("Enter city index: "))
   climate.remove_entry(y, c)
elif option == "4":
   y = int(input("Enter year index: "))
   c = int(input("Enter city index: "))
   print("Fetched:", climate.fetch(y, c))
elif option == "5":
   climate.row_view()
elif option == "6":
   climate.column_view()
elif option == "7":
   climate.sparse_check()
elif option == "8":
   climate.complexity_info()
elif option == "9":
   print("Exiting... Have a nice day!")
   break
else:
   print("Invalid choice. Try again.")
```

Summary

Program Overview

This program is designed to manage temperature records for multiple years and cities using a two-dimensional list (matrix). It provides a menu-based interface that lets the user insert, remove, fetch, and display data in different formats. It also deals with missing entries and shows the computational complexity of the operations.

What the Program Does

1. Creates a matrix of size years × cities.
2. Stores and updates temperature values in this matrix.
3. Manages empty/missing entries (sparse data).
4. Prints the data row-wise as well as column-wise.
5. Explains the time and space complexity of operations.

Functions Used

1. __init__(self, years, cities)
   Initializes the matrix with None in each cell.
2. populate_auto(self)
   Randomly assigns temperatures between **15°C and 40°C** to all cells.
3. insert(self, year, city, temp)
   Places a temperature value at the given row (year) and column (city).
4. delete(self, year, city)
   Clears an entry by setting it back to None.
5. retrieve(self, year, city)
   Fetches the stored temperature at the given indices.
   - Returns **"No Data"** if missing.
   - Returns **"Invalid index"** if out of range.
6. show_row_major(self)
   Displays all values row by row. Missing entries are printed as "NA".

7. show_column_major(self)
   Displays values column by column. Missing entries appear as "NA".
8. handle_sparse_data(self)
   Prints the dataset with "NA" placeholders for missing information.
9. analyze_complexity(self)
   - Insert → O(1)
   - Delete → O(1)
   - Retrieve → O(1)
   - Space → O(years × cities)