## Low-Level Design (LLD) for Daily Steps Tracker

**Difficulty Level:** Easy | **Total Marks:** 10
**Standards Followed:** 4 Functions | 3 Visible Test Cases | 2 Hidden Test Cases

---

### Concepts Tested

NumPy arrays and arithmetic operations
Aggregation and filtering
Class-based implementation
Conditional checks with list outputs

---

### Problem Statement

Create a step tracking system that allows users to load daily step counts, compute total and average steps, and list days where step counts exceeded a target.

The step data should be stored using NumPy arrays and accessed through class methods.

---

### Operations

#### *1. Load Daily Steps*

Loads a list of daily steps into the tracker.
**Function Prototype:**
```
def load_steps(self, steps_list: list):
```
**Example Input:**
```
load_steps([5000, 8000, 12000])
```
**Expected Output:**
```
[ 5000 8000 12000 ]
```

---

#### *2. Calculate Total Steps*

Returns the sum of all daily steps.
**Function Prototype:**
```
def total_steps(self):
```

 **Expected Output:**
```
25000
```

---

### *3. Calculate Average Steps*

 Calculates and returns the average steps.
 **Function Prototype:**
```
def average_steps(self):
```
 **Expected Output:**
```
8333.3
```

---

### *4. Days Above Target*

 Returns the list of steps for days above a given target.
 **Function Prototype:**
```
def days_above_target(self, target: int):
```
 **Example Input:**
```
days_above_target(7000)
```
 **Expected Output:**
```
[ 8000 12000 ]
```

---

### Implementation Cod

```
import numpy as np

class DailyStepsTracker:
    def __init__(self):
        """Initializes an empty steps array."""
        self.steps = np.array([])

    def load_steps(self, steps_list):
        """Loads a list of daily steps into the tracker."""
        self.steps = np.array(steps_list, dtype=int)
        print(self.steps)

    def total_steps(self):
        """Prints the total number of steps."""
        print(int(np.sum(self.steps)))

    def average_steps(self):
        """Prints the average number of steps per day."""
        print(round(np.mean(self.steps), 1))
```

```
    def days_above_target(self, target):
        """Prints the step counts for days above the given target."""
        print(self.steps[self.steps > target])

# Driver Code
if __name__ == "__main__":
    dst = DailyStepsTracker()
    q = int(input())
    for _ in range(q):
        cmd = input().split()
        if cmd[0] == "load":
            dst.load_steps(list(map(int, cmd[1:])))
        elif cmd[0] == "total":
            dst.total_steps()
        elif cmd[0] == "average":
            dst.average_steps()
        elif cmd[0] == "above":
            dst.days_above_target(int(cmd[1]))
        else:
            print("Invalid command.")
```

---

 Test Cases & Marks Allocation

| Test Case ID | Test Case Description | Associated Function(s) | Marks |
|---|---|---|---|
| **TC1** | Loading and displaying daily steps | load_daily_steps() |  2 Marks |
| **TC2** | Calculating average daily steps | calculate_average_steps() |  2 Marks |
| **TC3** | Identifying most active day | get_most_active_day() |  2 Marks |
| **HTC1** | Handling ties in most active day (same highest steps) | get_most_active_day() |  2 Marks |
| **HTC2** | Calculating average when some days have 0 steps | calculate_average_steps() |  2 Marks |
| **TOTAL** | All test cases passed | - |  **10 Marks** |

---

## Visible Test Cases (3)

**Test Case 1: Load Steps**
Input: load 5000 8000 12000

Output: [ 5000 8000 12000 ]

---

**Test Case 2: Total Steps**
Input: total

Output: 25000

---

**Test Case 3: Average Steps**
Input: average

Output: 8333.3

---

## Hidden Test Cases (2)

---

**HTC1: Days Above Target**
Input: above 7000

Output: [ 8000 12000 ]

---

**HTC2: Handle Empty Input**
Input:

Output: []