RBU, Nagpur
CSE III Sem
PRACTICAL NO. 1

Name:	**Kishan Saini**
Sec-Batch-Rollno:	**A4-B4-55**

Aim: Time and complexity analysis of loops for a sensor data monitoring system by generating random sensor readings such as temperature, and pressure. The goal is to analyze and compare the performance of different algorithms.

Data Generation: Simulate Sensor Data Generation

- Generate random sensor data such as:
  - Temperature (°C) — e.g., range: -20 to 50
  - Pressure (hPa) — e.g., range: 950 to 1050
- Store the data in a structured format (e.g., arrays or classes)

Objective: Apply different type of algorithms to study effective design technique

- Find
  - Minimum temperature
  - Maximum pressure
- Measure and analyze execution time for each parameter

- Analyze Time Complexity

Tasks

Task-A: Apply Linear Search Approach

- Implement a linear search algorithm, linear search algorithm (O(n)) is used to traverse the data and determine the min/max values for each sensor type.

Task-B: Naive Pairwise Comparison Approach

- For each element, compare it with every other element.

  For minVal:

  For i = 0 to n-1: check if arr[i] is less than every other arr[j].

  Mark as minimum if it satisfies all conditions.

- Repeat similarly for maxVal.

Task-C:

1. Generate sorted data for temperature (range: 20 to 50)
2. Find the first Occurrence of temperature >=30.
   - Apply Linear search
   - Apply Binary Search

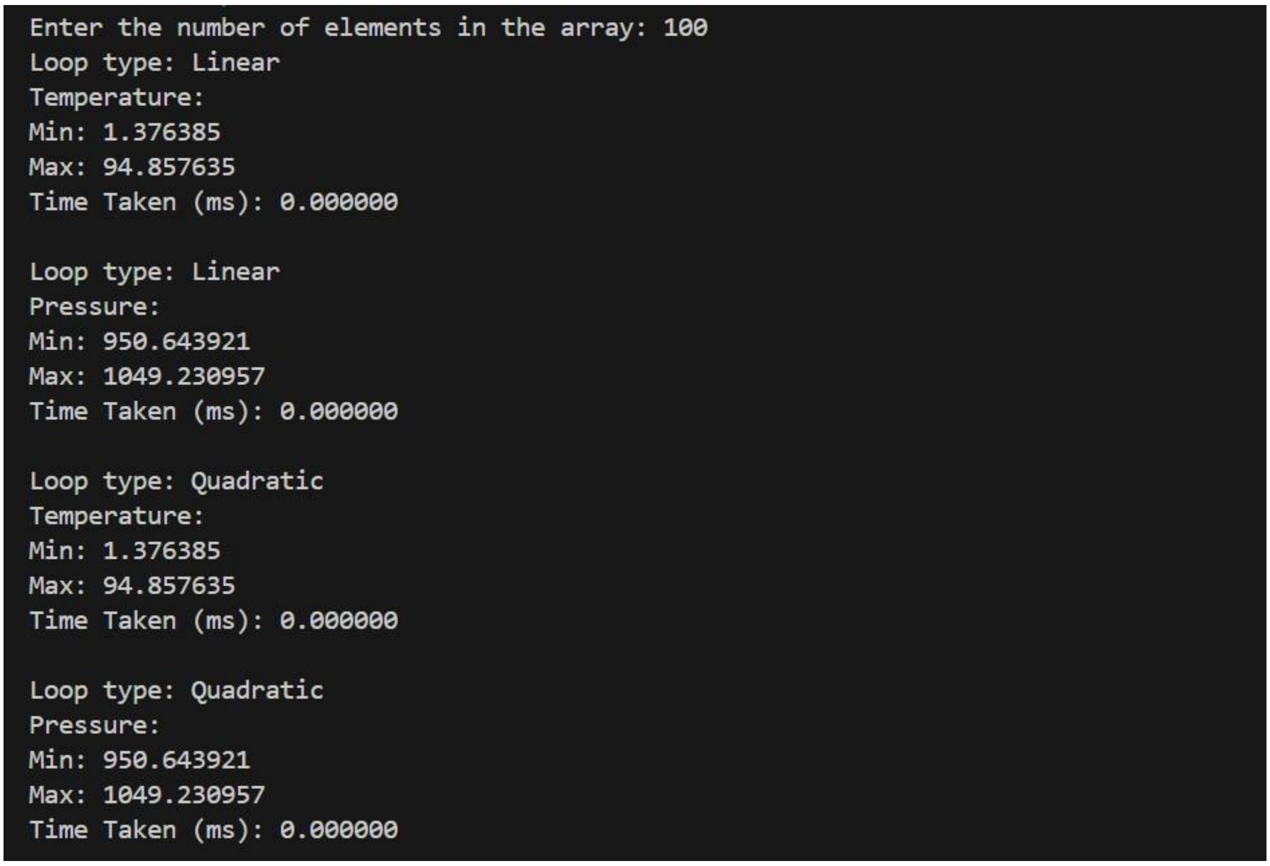
CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    srand(time(NULL));
    int n;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
```

```c
if (n <= 0) {
    printf("Invalid array size.\n");
    return 1;
}

float *temp = (float*)malloc(sizeof(float) * n);
float *press = (float*)malloc(sizeof(float) * n);

if (temp == NULL || press == NULL) {
    printf("\nMemory not allocated!\n");
    return 1;
}

for (int i = 0; i < n; i++) {
    temp[i] = (float)rand() / RAND_MAX * 100;
    press[i] = 950 + (float)rand() / RAND_MAX * 100;
}

clock_t start, end;

// Linear Search
float minTemp = temp[0], maxTemp = temp[0];
start = clock();
for (int i = 1; i < n; i++) {
    if (temp[i] < minTemp) minTemp = temp[i];
    if (temp[i] > maxTemp) maxTemp = temp[i];
}
end = clock();
printf("Loop type: Linear\nTemperature:\nMin: %f\nMax: %f\nTime Taken (ms): %f\n\n",
    minTemp, maxTemp, ((double)(end - start) / CLOCKS_PER_SEC) * 1000);



float minPress = press[0], maxPress = press[0];
start = clock();
```

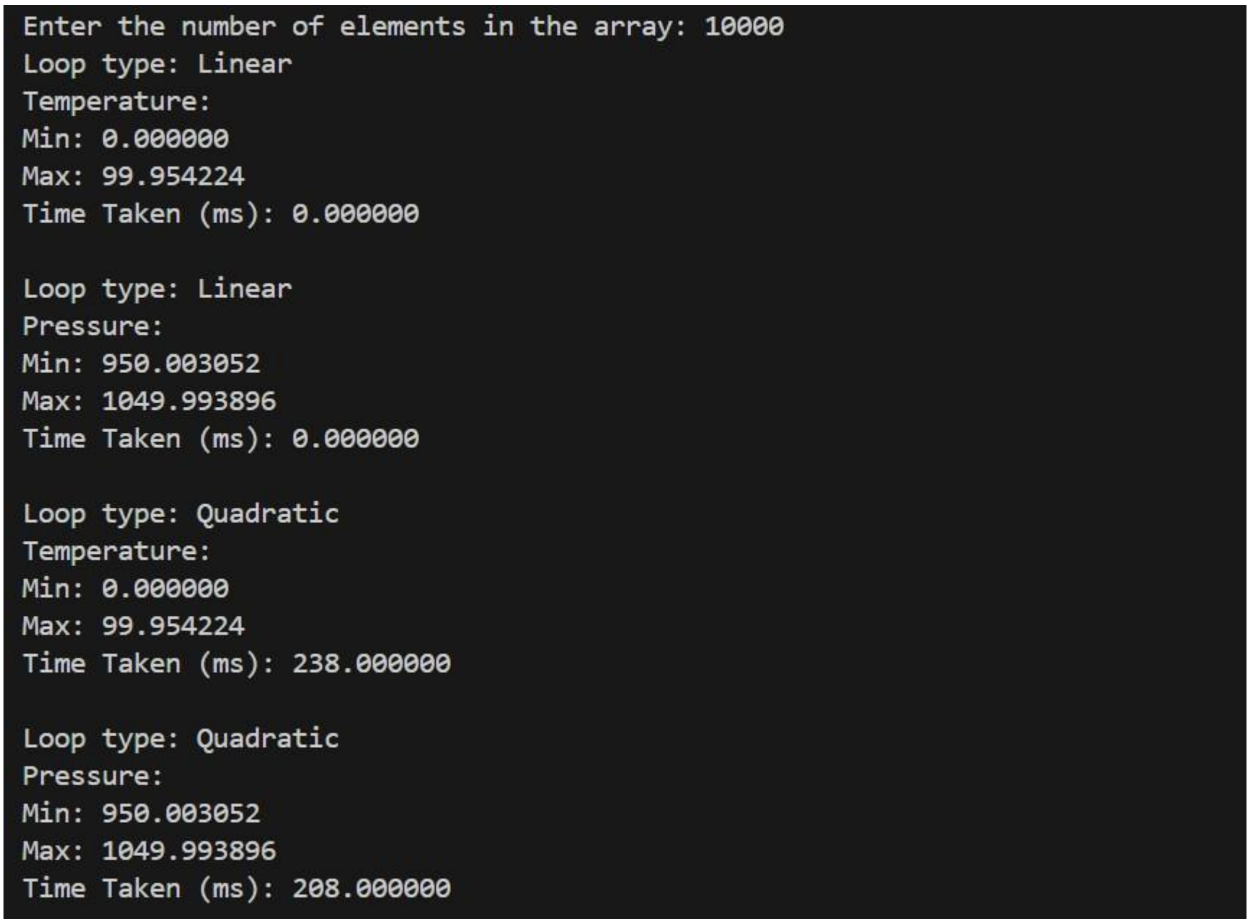
```c
    for (int i = 1; i < n; i++) {
        if (press[i] < minPress) minPress = press[i];
        if (press[i] > maxPress) maxPress = press[i];
    }
    end = clock();
    printf("Loop type: Linear\nPressure:\nMin: %f\nMax: %f\nTime Taken (ms): %f\n\n",
           minPress, maxPress, ((double)(end - start) / CLOCKS_PER_SEC) * 1000);

    // Quadratic Search
    minTemp = temp[0];
    maxTemp = temp[0];
    start = clock();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (temp[j] < minTemp) minTemp = temp[j];
            if (temp[j] > maxTemp) maxTemp = temp[j];
        }
    }
    end = clock();
    printf("Loop type: Quadratic\nTemperature:\nMin: %f\nMax: %f\nTime Taken (ms): %f\n\n",
           minTemp, maxTemp, ((double)(end - start) / CLOCKS_PER_SEC) * 1000);

    minPress = press[0];
    maxPress = press[0];
    start = clock();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (press[j] < minPress) minPress = press[j];
            if (press[j] > maxPress) maxPress = press[j];
        }
    }
    end = clock();
    printf("Loop type: Quadratic\nPressure:\nMin: %f\nMax: %f\nTime Taken (ms): %f\n\n",
        minPress, maxPress, ((double)(end - start) / CLOCKS_PER_SEC) * 1000);
    free(temp);
    free(press);

    return 0;
}
```

OUTPUT 1] n= 10^2:

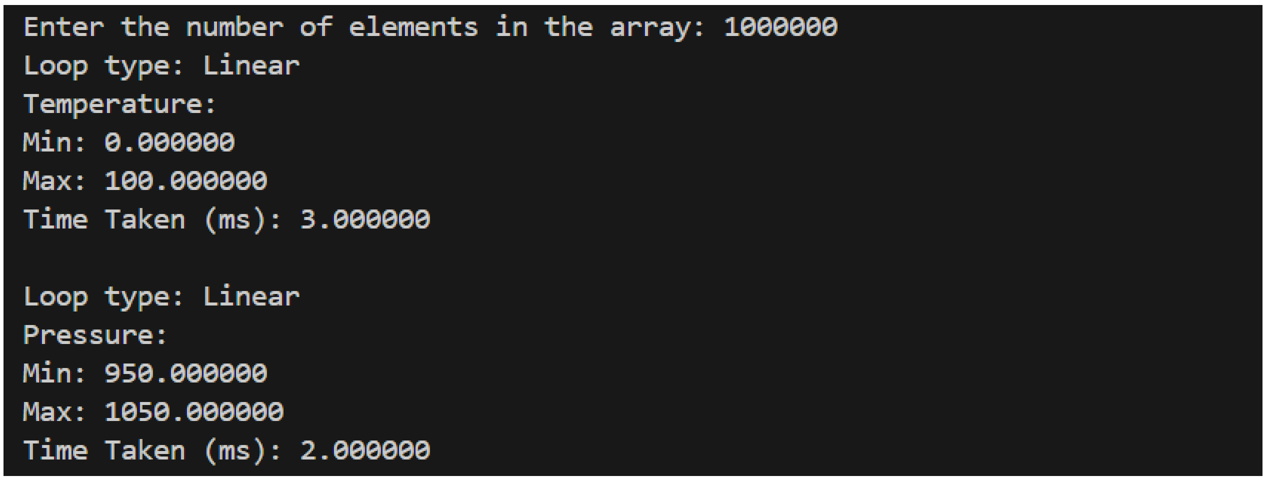


```
Enter the number of elements in the array: 100
Loop type: Linear
Temperature:
Min: 1.376385
Max: 94.857635
Time Taken (ms): 0.000000

Loop type: Linear
Pressure:
Min: 950.643921
Max: 1049.230957
Time Taken (ms): 0.000000

Loop type: Quadratic
Temperature:
Min: 1.376385
Max: 94.857635
Time Taken (ms): 0.000000

Loop type: Quadratic
Pressure:
Min: 950.643921
Max: 1049.230957
Time Taken (ms): 0.000000
```

OUTPUT 2] n=10^4:


```
Enter the number of elements in the array: 10000
Loop type: Linear
Temperature:
Min: 0.000000
Max: 99.954224
Time Taken (ms): 0.000000

Loop type: Linear
Pressure:
Min: 950.003052
Max: 1049.993896
Time Taken (ms): 0.000000

Loop type: Quadratic
Temperature:
Min: 0.000000
Max: 99.954224
Time Taken (ms): 238.000000

Loop type: Quadratic
Pressure:
Min: 950.003052
Max: 1049.993896
Time Taken (ms): 208.000000
```

OUTPUT 3] n = 10^6 :



```
Enter the number of elements in the array: 1000000
Loop type: Linear
Temperature:
Min: 0.000000
Max: 100.000000
Time Taken (ms): 3.000000

Loop type: Linear
Pressure:
Min: 950.000000
Max: 1050.000000
Time Taken (ms): 2.000000
```

Expected Output / Report Format

Task	**Loop Type**	**Time Complexity**	**Parameters**	**n = 10^2**	**n = 10^4**	**n = 10^6**
Task-A	Linear		Temperature	Min: 1.376385 Max: 94.857635	Min: 0.000000 Max: 99.954224	Min: 0.000000 Max: 100.000000
			Pressure	Min: 950.643921 Max: 1049.230957	Min: 950.003052 Max: 1049.993896	Min: 950.000000 Max: 1050.000000

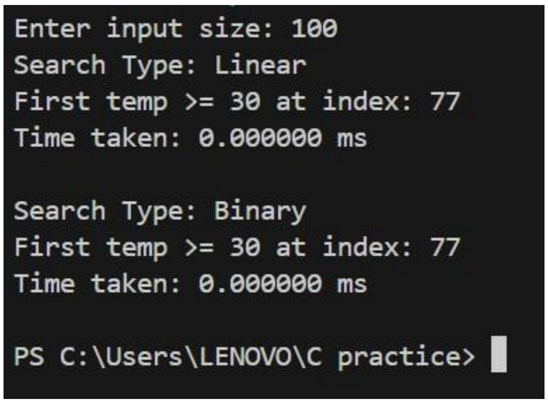
Task-B	Quadratic		Temperature	Min: 1.376385 Max: 94.857635	Min: 0.000000 Max: 99.954224	-
			Pressure	Min: 950.643921 Max: 1049.230957	Min: 950.003052 Max: 1049.993896	-

Task-C

CODE:

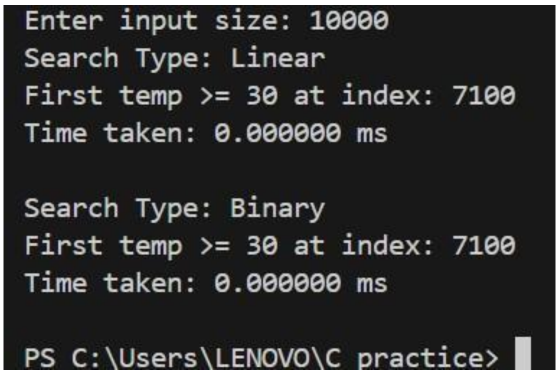
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

float generateRandomNumber(float min, float max) {
    return min + ((float)rand() / RAND_MAX) * (max - min);
}

int main() {
    srand(time(NULL));
    int inp;
    printf("Enter input size: ");
    scanf("%d", &inp);

    float* temp = (float*)malloc(sizeof(float) * inp);
    float* pressure = (float*)malloc(sizeof(float) * inp);
    if (temp == NULL || pressure == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    for (int i = 0; i < inp; i++) {
        temp[i] = generateRandomNumber(-20, 50);
        pressure[i] = generateRandomNumber(950, 1050);
    }

    // Bubble sort temperatures
    for (int i = 0; i < inp; i++) {
        for (int j = 0; j < inp - i - 1; j++) {
            if (temp[j] > temp[j + 1]) {
                float t = temp[j];
                temp[j] = temp[j + 1];
                temp[j + 1] = t;
            }
        }
    }

    int index = -1;
    clock_t start_time = clock();
    for (int i = 0; i < inp; i++) {
        if (temp[i] >= 30) {
            index = i;
            break;
        }
    }
    clock_t end_time = clock();
    printf("Search Type: Linear\nFirst temp >= 30 at index: %d\nTime taken: %f
ms\n\n",
           index, ((double)(end_time - start_time) / CLOCKS_PER_SEC) * 1000);
    int left = 0, right = inp - 1;
    index = -1;
    start_time = clock();
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (temp[mid] >= 30) {
            index = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    end_time = clock();
    printf("Search Type: Binary\nFirst temp >= 30 at index: %d\nTime taken: %f
ms\n\n",
           index, ((double)(end_time - start_time) / CLOCKS_PER_SEC) * 1000);

    free(temp);
    free(pressure);
    return 0;
}
```

OUTPUT 1]n=10^2



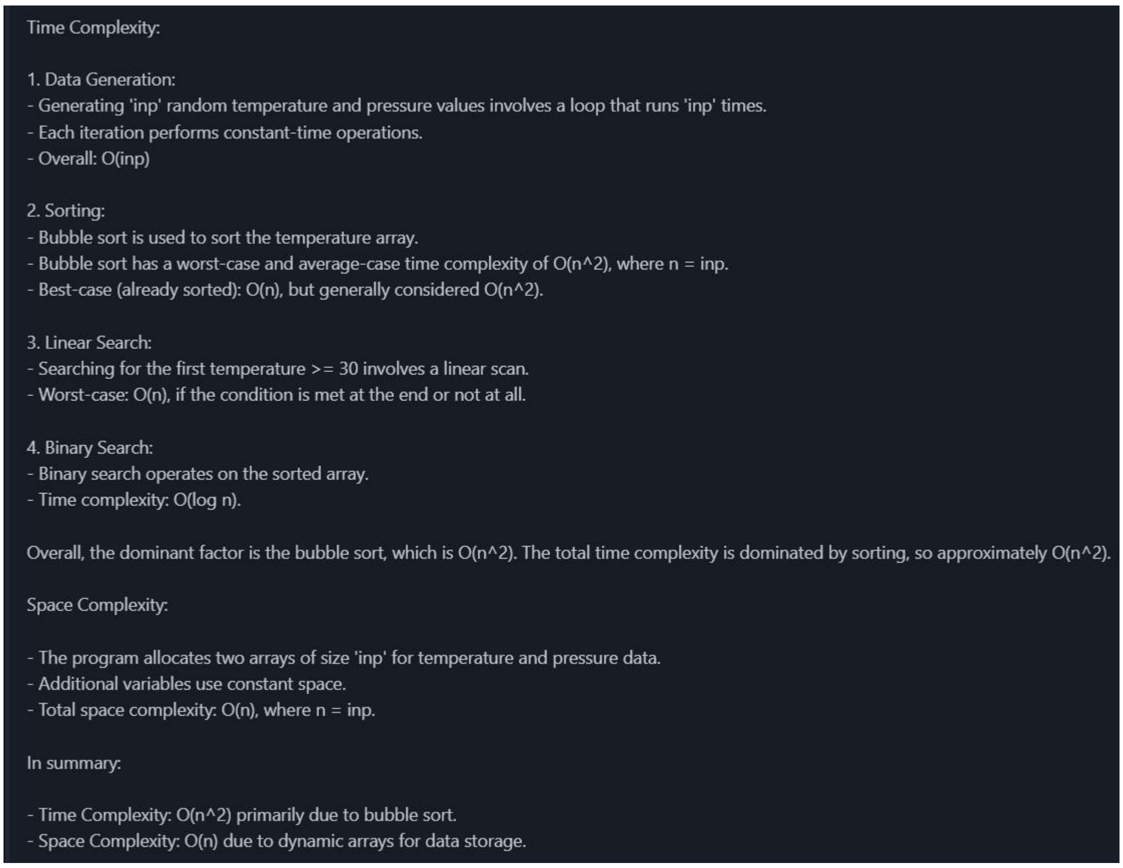
```
Enter input size: 100
Search Type: Linear
First temp >= 30 at index: 77
Time taken: 0.000000 ms

Search Type: Binary
First temp >= 30 at index: 77
Time taken: 0.000000 ms

PS C:\Users\LENOVO\C practice> 
```

OUTPUT 2] n = 10^4



```
Enter input size: 10000
Search Type: Linear
First temp >= 30 at index: 7100
Time taken: 0.000000 ms

Search Type: Binary
First temp >= 30 at index: 7100
Time taken: 0.000000 ms

PS C:\Users\LENOVO\C practice>
```

Task	**Algorithm**	**Time Complexity**	**n = 10^2**	**n = 10^4**	**n = 10^6**
Task-C	Linear Search	O(n)	Search Type: Linear First temp >= 30 at index: 77 Time taken:	First temp >= 30 at index: 7100 Time taken: 0.000000 ms	-
			0.000000 ms		
	Binary Search	O(log n)	First temp >= 30 at index: 77 Time taken: 0.000000 ms	First temp >= 30 at index: 7100 Time taken: 0.000000 ms	-



```
Time Complexity:

1. Data Generation:
- Generating 'inp' random temperature and pressure values involves a loop that runs 'inp' times.
- Each iteration performs constant-time operations.
- Overall: O(inp)

2. Sorting:
- Bubble sort is used to sort the temperature array.
- Bubble sort has a worst-case and average-case time complexity of O(n^2), where n = inp.
- Best-case (already sorted): O(n), but generally considered O(n^2).

3. Linear Search:
- Searching for the first temperature >= 30 involves a linear scan.
- Worst-case: O(n), if the condition is met at the end or not at all.

4. Binary Search:
- Binary search operates on the sorted array.
- Time complexity: O(log n).

Overall, the dominant factor is the bubble sort, which is O(n^2). The total time complexity is dominated by sorting, so approximately O(n^2).

Space Complexity:

- The program allocates two arrays of size 'inp' for temperature and pressure data.
- Additional variables use constant space.
- Total space complexity: O(n), where n = inp.

In summary:

- Time Complexity: O(n^2) primarily due to bubble sort.
- Space Complexity: O(n) due to dynamic arrays for data storage.
```

Self-Study and Assignment-1:

1. Consider two algorithms for finding square root of an integer, Babylonian method (Newton-Raphson method, check wiki) and Binary search. Which one is faster and why? Implement both and show comparison of implementation time.

   Solution :

   Analyzing loop complexity in a sensor data monitoring system involves evaluating how the execution time of algorithms scales with the amount of sensor data. For loops that process sensor readings, the time complexity is often determined by the number of iterations and the operations within each iteration. Common loop types include simple loops (O(n)), nested loops (O(n^2) or worse), and logarithmic loops (O(log n)). Comparing algorithm performance involves analyzing their time and space complexity

for different data sizes and complexities of operations within the loops. 1. Simple Loop (O(n))
• Scenario:

Iterating through a list of sensor readings to calculate an average temperature.
• Analysis:

If you have n sensor readings, a simple for or while loop that processes each reading once will have a time complexity of O(n). The space complexity would be O(1) if only a few variables are used, or O(n) if you need to store all the sensor readings. 2. Nested Loops (O(n^2) or worse)
• Scenario:

Comparing each sensor reading with every other reading to identify potential anomalies.
• Analysis:

A nested loop structure (one loop inside another) can lead to quadratic time complexity. For example, if you have n sensor readings and an inner loop iterates through them for each outer loop iteration, the complexity becomes O(n^2). Further nested loops will increase the complexity (O(n^3), O(n^4), etc.). 3. Logarithmic Loop (O(log n))
• Scenario:

Using a binary search algorithm to locate a specific sensor reading in a sorted list.
• Analysis:

If the sensor data can be organized in a way that allows for efficient searching (e.g., sorted), a binary search approach can achieve logarithmic time complexity, O(log n).
4. Analyzing Different Algorithms •
Example:

Consider two algorithms: one using a simple loop for calculating the average temperature and another using a more complex algorithm that involves sorting and then averaging.
• Comparison:

The simple loop algorithm will likely be faster for smaller datasets, but for larger datasets, the overhead of sorting in the complex algorithm might make it slower than the simple loop.
5. Time and Space Complexity

- Time Complexity: Refers to the amount of time an algorithm takes to run as the input size grows, usually expressed using Big O notation (e.g., O(n), O(n^2), O(log n)).

- <b>Space Complexity</b>: Refers to the amount of memory an algorithm uses as the input size grows.

6. Factors Affecting Performance

• Data Size:

As the number of sensor readings increases, the impact of time complexity becomes more significant.

• Algorithm Choice:

The specific algorithm used for processing the data can drastically affect performance.

• Hardware Limitations:

The processing power and memory of the system monitoring the sensors can also influence the practical performance of the algorithms. 7. Practical Considerations

• Real-time Requirements:

In real-time monitoring systems, the time it takes to process sensor data is critical. Algorithms with lower time complexity are generally preferred.

• Resource Constraints:

On embedded systems with limited resources, space complexity is also a crucial factor.

• Data Filtering and Preprocessing:

Techniques like data filtering and outlier detection can reduce the amount of data that needs to be processed, potentially improving performance.

• Sensor Fusion:

In some cases, combining data from multiple sensors (sensor fusion) can provide more accurate and reliable information.

By analyzing the time and space complexity of different algorithms and considering the practical constraints of the monitoring system, developers can choose the most efficient and effective approach for processing sensor data.