Yash Wanjari
A2_B3_40

Ramdeobaba University (RBU), Nagpur
Department of Computer Science & Engineering and Emerging Technology
Session: 2025-26

Design and Analysis of Algorithms Lab **III Semester**

PRACTICAL NO. 1

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.

Expected Output / Report Format

Task	Loop Type	Time Complexity	Parameters	n = 10^2	n = 10^4	n = 10^6
Task-A	Linear	O(n)	Temperature	-18.858959	-19.996719	-19.999960
			Pressure	1049.999390	1049.999146	1049.999878
Task-B	Quadratic	O(n^2)	Temperature	-19.773798	-19.998169	-19.999971
			Pressure	1048.646729	1049.988159	1049.999878

n=100

```
Task A: Min_Temperature = -18.858959     Max_Pressure = 1049.999390     Time = 0.000003 sec
Task B: Min_Temperature = -19.773798     Max_Pressure = 1048.646729     Time = 0.000006 sec
```

n=10000

```
Task A: Min_Temperature = -19.996719     Max_Pressure = 1049.999146     Time = 0.000050 sec
Task B: Min_Temperature = -19.998169     Max_Pressure = 1049.988159     Time = 0.000480 sec
```

n=1000000

```
Task A: Min_Temperature = -19.999960     Max_Pressure = 1049.999878     Time = 0.005113 sec
Task B: Min_Temperature = -19.999971     Max_Pressure = 1049.999878     Time = 0.077554 sec
```

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

Task	Algorithm	Time Complexity	n = 10^2	n = 10^4	n = 10^6
Task-C	Linear Search	O(n)	30.056683	30.000320	30.000021
	Binary Search	O(log n)	30.772861	30.000179	30.000019

n=100

```
Linear Search: first Occurrence of temperature >= 30 at index 26 is 30.056683
Time taken: 0.000002 secs
Binary Search: first Occurrence of temperature >= 30 at index 37 is 30.772861
Time taken: 0.000001 secs
```

n=10000

```
Linear Search: first Occurrence of temperature >= 30 at index 3379 is 30.000320
Time taken: 0.000011 secs
Binary Search: first Occurrence of temperature >= 30 at index 3371 is 30.000179
Time taken: 0.000002 secs
```

n=1000000

```
Linear Search: first Occurrence of temperature >= 30 at index 333342 is 30.000021
Time taken: 0.000785 secs
Binary Search: first Occurrence of temperature >= 30 at index 333234 is 30.000019
Time taken: 0.000004 secs
```

Self-Study Assignment-1 [Fast Learners]:

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.

FEW HINTS:

- **Random data generation in a range max to min**
  float num = min + ((float) rand() / RAND_MAX) * (max - min); **[C Code]**
  Java: Can be achieved using the java.util.Random class or java.lang.Math.random().
- **Linear Search to min and max: write a function**
  - **Logic:** iterate through the list, comparing each element with the current minimum or max.
  - Set min/max to the first element of the list.
  - If the current element is less than min, update min to the current element.
  - After iterating through the entire list, min will hold the smallest element.
- **Pass the generated array of temp/ pressure**
- **Time computation**
  - Use the clock() function from the <time.h> header.
  - Sample code to compute time for a simple for loop **[C Code]** is given below. [in java use System.nanoTime()]

```
#include <stdio.h>
#include <time.h> // Required for clock_t, clock(), and CLOCKS_PER_SEC

int main() {
   clock_t start_time, end_time;
   double cpu_time_used;
   int i;
   long long sum = 0; // Use long long for sum to avoid overflow with large loops

   // Record the starting time
   start_time = clock();
```

```
    // The loop whose execution time is to be measured
    for (i = 0; i < 100000000; i++) {
        sum += i; // Perform some operation inside the loop
    }

    // Record the ending time
    end_time = clock();

    // Calculate the CPU time used
    cpu_time_used = ((double) (end_time - start_time)) / CLOCKS_PER_SEC;

    printf("Loop finished. Sum: %lld\n", sum);
    printf("Time taken by the loop: %f seconds\n", cpu_time_used);

    return 0;
}
```

Inferences

1) The linear search performed well for lower values of n and gave results quickly. However, as `n` increased the time started increasing linearly.
2) The pairwise approach took longer time to execute for larger datasets like `n = 10^6`. This confirms that algorithms with quadratic complexity are not scalable for real-time monitoring.
3) As we can see that in both linear search and pairwise search the execution time varies greatly. The choice of algorithm plays a vital role in designing real-time systems like sensor monitoring applications.
4) In Task-C binary search demonstrated a much faster execution time than linear search, especially for large values of `n`. This validates that it is more suitable for searching in sorted data.