A1-B3-42

Amber Shukla

DAA LAB

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	0 (Too Small)	0.016	0.011
			Pressure	0 (Too Small)	0.016	0.011
Task-B	Quadratic	O(N^2)	Temperature	0 (Too Small)	0.551	3.755
			Pressure	0 (Too Small)	0.551	3.755

10^2 Trials

```
Time Taken By Generation: 0
Minimum Temperature: -20
Maximum Pressure: 1050
Time Taken By Linear Method: 0
Minimum Temperature: -20
Maximum Pressure: 1050
Time Taken By Quadratic Method: 0
```

10^4 Trials

```
Time Taken By Generation: 0.029
Minimum Temperature: -20
Maximum Pressure: 1050
Time Taken By Linear Method: 0.016
Minimum Temperature: -20
Maximum Pressure: 1050
Time Taken By Quadratic Method: 0.551
```

10^6Trials

```
Time Taken By Generation: 0.028
Minimum Temperature: -20
Maximum Pressure: 1050
Time Taken By Linear Method: 0.011
Minimum Temperature: -20
Maximum Pressure: 1050
Time Taken By Quadratic Method: 3.755
```

CODE (PART 1/B):

```cpp
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

vector<ll> resultTemp;
vector<ll> resultPressure;

clock_t start_time, end_time;

ll times = 10000;
int minTemp = INT_MAX, maxPressure = -1;

void generateTemp(ll times)
{
    for (ll i = 0; i < times; i++)
        resultTemp.push_back((rand() % (71)) - 20);
}

void generatePressure(ll times)
{
    for (ll i = 0; i < times; i++)
        resultPressure.push_back((rand() % (101)) + 950);
}

void linearSearch()
{
    for (auto i : resultTemp)
        if (i < minTemp)
            minTemp = i;

    for (auto i : resultPressure)
        if (i > maxPressure)
            maxPressure = i;

    cout << "Minimum Temperature: " << minTemp << endl;
    cout << "Maximum Pressure: " << maxPressure << endl;
}

void quadraticSearch()
{
    for (auto i : resultTemp)
    {
        auto test = i;
        bool found = true;

        for (auto j : resultTemp)
        {
            if (j < i)
                found = false;
        }
```

```
        if (found)
        {
            minTemp = i;
            cout << "Minimum Temperature: " << minTemp << endl;
            break;
        }
    }

    for (auto i : resultPressure)
    {
        auto test = i;
        bool found = true;

        for (auto j : resultPressure)
        {
            if (j > i)
                found = false;
        }

        if (found)
        {
            maxPressure = i;
            cout << "Maximum Pressure: " << maxPressure << endl;
            break;
        }
    }
}

int main()
{
    srand(time(0));
    start_time = clock();
    generateTemp(times);
    generatePressure(times);
    end_time = clock();
    cout << "Time Taken By Generation: " << ((double)(end_time - start_time)) /
CLOCKS_PER_SEC << endl;

    // Linear Method
    start_time = clock();
    linearSearch();
    end_time = clock();
    cout << "Time Taken By Linear Method: " << ((double)(end_time - start_time)) /
CLOCKS_PER_SEC << endl;

    // Quadratic Method
    start_time = clock();
    quadraticSearch();
    end_time = clock();
    cout << "Time Taken By Quadratic Method: " << ((double)(end_time - start_time)) /
CLOCKS_PER_SEC << endl;
}
```

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)	0.002	0.003	0.004
	Binary Search	O(Log(n))	0.001	0.002	0.001

10^2 Trials

```
Time Taken By Sorting: 0 Seconds


First Occourence: 30 At Index: 28
Time Taken By Linear Method: 0.002 Seconds

Found Occourence: 36 At Index: 49
Time Taken By Binary Method: 0.001 Seconds
```

10^4 Trials

```
Time Taken By Sorting: 0.002 Seconds


First Occourence: 30 At Index: 3200
Time Taken By Linear Method: 0.003 Seconds

Found Occourence: 35 At Index: 4999
Time Taken By Binary Method: 0.002 Seconds
```

10^6Trial

```
Time Taken By Sorting: 0.187 Seconds


First Occourence: 30 At Index: 322517
Time Taken By Linear Method: 0.004 Seconds

Found Occourence: 35 At Index: 499999
Time Taken By Binary Method: 0.001 Seconds
```

CODE (PART 1/B):

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

vector<ll> resultTemp;

clock_t start_time, end_time;

ll times = 1000000;

void generateTemp(ll times)
{
    for (ll i = 0; i < times; i++)
        resultTemp.push_back(rand() % (50 - 20 + 1) + 20);
}

void linearSearch()
{
    for (ll i = 0; i < resultTemp.size(); i++)
        if (resultTemp[i] >= 30)
        {
            cout << "\n\nFirst Occourence: " << resultTemp[i] << " At Index: " << i << endl;
            break;
        }
}

void binarySearch()
{
    ll start = 0, end = resultTemp.size() - 1, mid;
    bool found = false;

    while (start <= end && !found)
    {

        mid = (end + start) / 2;

        if (resultTemp[mid] >= 30)
        {
            cout << "\nFound Occourence: " << resultTemp[mid] << " At Index: " << mid <<
endl;
            found = true;
        }
        else if (resultTemp[mid] < 30)
        {
            start = mid + 1;
        }
        else
        {
            end = mid - 1;
        }
    }
}
```

```
int main()
{

    srand(time(0));

    generateTemp(times);

    // Sorting
    start_time = clock();
    sort(resultTemp.begin(), resultTemp.end());
    end_time = clock();
    cout << "\nTime Taken By Sorting: " << ((double)(end_time - start_time)) / CLOCKS_PER_SEC
<< " Seconds" << endl;

    // Linear Method
    start_time = clock();
    linearSearch();
    end_time = clock();
    cout << "Time Taken By Linear Method: " << ((double)(end_time - start_time)) /
CLOCKS_PER_SEC << " Seconds" << endl;

    // Binary Method
    start_time = clock();
    binarySearch();
    end_time = clock();
    cout << "Time Taken By Binary Method: " << ((double)(end_time - start_time)) /
CLOCKS_PER_SEC << " Seconds\n\n"
         << endl;
}
```

Inference:

a. Linear search (O(n)) is simple and works well for unsorted data but is slower for large datasets.

b. Naive pairwise comparison ($O(n^2)$) is much less efficient and not suitable for big data.

c. Sorting (O(n log n)) enables fast binary search but adds initial overhead.

d. Binary search (O(log n)) is very efficient on sorted data for repeated or fast lookups.