DAA PRACTICAL NO: 01

Name: **Mohit Deepak Agrawal**

Roll no: **A2_B2_33**

Date: **22-07-25 (1:40 p.m.)**

Git-hub Link: https://github.com/Mohitt-2006

TASK A:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
float getRandomFloat(float min, float max) {
    return min + ((float) rand() / RAND_MAX) * (max - min);
}
void generateData(float temp[], float pressure[], int n) {
    for (int i = 0; i < n; i++) {
        temp[i] = getRandomFloat(-20, 50);
        pressure[i] = getRandomFloat(950, 1050);
    }
}
int findMinTemperature(float temp[], int n) {
    int minIndex = 0;
    for (int i = 1; i < n; i++) {
        if (temp[i] < temp[minIndex]) {
            minIndex = i;
        }
```

```
    }
    return minIndex;
}
int findMaxPressure(float pressure[], int n) {
    int maxIndex = 0;
    for (int i = 1; i < n; i++) {
        if (pressure[i] > pressure[maxIndex]) {
            maxIndex = i;
        }
    }
    return maxIndex;
}
int main() {
    int n = 100;
    float temp[n], pressure[n];
    generateData(temp, pressure, n);
    clock_t start, end;
    double duration;
    start = clock();
    int minTempIndex = findMinTemperature(temp, n);
    end = clock();
    duration = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Minimum Temperature: %.2f °C at index %d and Time: %lf seconds\n",
temp[minTempIndex], minTempIndex, duration);
    start = clock();
    int maxPressureIndex = findMaxPressure(pressure, n);
    end = clock();
    duration = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Maximum Pressure: %.2f hPa at index %d and Time: %lf seconds\n",
pressure[maxPressureIndex], maxPressureIndex, duration);


    return 0;
```

```
}
```

Task B:

Code:

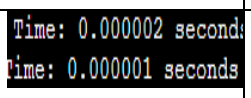
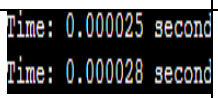
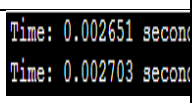
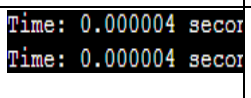
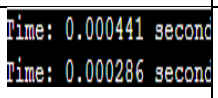
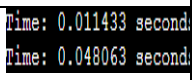
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
float getRandomFloat(float min, float max) {
    return min + ((float) rand() / RAND_MAX) * (max - min);
}

void generateData(float temp[], float pressure[], int n) {
    for (int i = 0; i < n; i++) {
        temp[i] = getRandomFloat(-20, 50);
        pressure[i] = getRandomFloat(950, 1050);
    }
}
int naiveFindMin(float arr[], int n) {
    for (int i = 0; i < n; i++) {
        int isMin = 1;
        for (int j = 0; j < n; j++) {
            if (arr[j] < arr[i]) {
                isMin = 0;
                break;
            }
        }
        if (isMin)
            return i;
    }
```

```
    return -1;
}
int naiveFindMax(float arr[], int n) {
    for (int i = 0; i < n; i++) {
        int isMax = 1;
        for (int j = 0; j < n; j++) {
            if (arr[j] > arr[i]) {
                isMax = 0;
                break;
            }
        }
        if (isMax)
            return i;
    }
    return -1;
}
int main() {
    int n = 100;
    float temp[n], pressure[n];
    generateData(temp, pressure, n);
    clock_t start, end;
    double duration;
    start = clock();
    int minTempIndex = naiveFindMin(temp, n);
    end = clock();
    duration = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Minimum Temperature: %.2f °C at index %d and Time: %lf seconds\n",
temp[minTempIndex], minTempIndex, duration);
    start = clock();
    int maxPressureIndex = naiveFindMax(pressure, n);
    end = clock();
```

```
    duration = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Maximum Pressure: %.2f hPa at index %d and Time: %lf seconds\n",
pressure[maxPressureIndex], maxPressureIndex, duration);
    return 0;
}
```

OUTPUT TABLE:

TASK	Loop Type	T.C	Parameters	n=100	n=1000	n=1000000
TASK-A	LINEAR	O(n)	Temperature Pressure			
TASK-B	Quadratic	O(n^2)	Temperature Pressure			

TASK C:

LinearSearch:

CODE:

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

void generateSortedTemps(float arr[], int n) {
    float step = 30.0 / n;
    for (int i = 0; i < n; i++) {
        arr[i] = 20.0 + i * step;
    }
}
int linearSearch(float arr[], int n) {
    for (int i = 0; i < n; i++) {
```

```
        if (arr[i] >= 30.0)
            return i;
    }
    return -1;
}
int main() {
    int n = 100;
    float temp[n];
    generateSortedTemps(temp, n);
    clock_t start = clock();
    int index = linearSearch(temp, n);
    clock_t end = clock();
    double timeTaken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("First temperature >= 30°C at index %d: %.2f°C\n", index, temp[index]);
    printf("Time taken: %lf seconds\n", timeTaken);

    return 0;
}
```

BinarySearch:

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void generateSortedTemps(float arr[], int n) {
    float step = 30.0 / n;
    for (int i = 0; i < n; i++) {
        arr[i] = 20.0 + i * step;
    }
}
int binarySearch(float arr[], int n) {
```

```
    int left = 0, right = n - 1, result = -1;

    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] >= 30.0) {
            result = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    return result;
}

int main() {
    int n = 1000;
    float temp[n];
    generateSortedTemps(temp, n);
    clock_t start = clock();
    int index = binarySearch(temp, n);
    clock_t end = clock();
    double timeTaken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("First temperature >= 30°C at index %d: %.2f°C\n", index, temp[index]);
    printf("Time taken: %lf seconds\n", timeTaken);

    return 0;
}
```

Output SS:

Task C	Algorithm	N=100	N=10000	N=1000000

	LinearSearch	Time: 0.000001 sec	Time: 0.000010 sec	Time: 0.000982 sec
	BinarySearch	Time: 0.000001 sec	Time: 0.000002 sec	Time: 0.000001 sec

Time Complexities:

LinearSearch: O(n)

BinarySearch: O(log n)

Conclusion for All Tasks

- **Task A: Linear Search**
  - The **linear search** method successfully finds the minimum temperature and maximum pressure by checking each value one by one.
  - It works well even for large data sizes (like 10^6 readings) because its time complexity is **O(n)**.
  - This method is **simple, fast, and practical** for real-time sensor systems.

- **Task B: Naive Pairwise Comparison**
  - In this method, every value is compared with all others to find min/max values.
  - Its time complexity is **$O(n^2)$**, which makes it very **slow and inefficient** when the number of readings is large.
  - While it gives correct results, it is **not suitable** for large-scale or real-time applications due to high computation time

- **Task C: First Occurrence of Temperature ≥ 30**
  - Using **Linear Search**, the value is found by scanning the array one by one. It is **accurate** but **slower** for large data.
  - Using **Binary Search**, which works on sorted data, the result is found much **faster (O(log n))**.
  - So, **binary search is highly efficient** when working with sorted temperature readings.