

Practical 1

Name- Tanishq Mankari

Section – A8_B2

Roll no – 31

Q1-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.

Code for task a and b

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

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

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

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

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

```
    float* temp = (float*)malloc(sizeof(float) * 10000);  
    float* pressure = (float*)malloc(sizeof(float) * 10000);
```

```
    int inp;    printf("Enter input size (up to  
    10000): ");    scanf("%d", &inp);
```

```
    if (inp <= 0 || inp > 10000) {
```

```

    printf("Invalid input size.\n");
return 1;
}

for (int i = 0; i < inp; i++) {
    temp[i] = generateRandomNumber(-20.0f, 50.0f);
pressure[i] = generateRandomNumber(950.0f, 1050.0f);
}

float min_temp = temp[0];
clock_t start_time = clock();
for (int i = 1; i < inp; i++) {
    if (temp[i] < min_temp) {
        min_temp = temp[i];
    }
}

clock_t end_time = clock();    printf("Minimum temperature (Linear Search): %.2f°C\n",
min_temp);    printf("Time taken: %.4f ms\n\n", ((double)(end_time - start_time) /
CLOCKS_PER_SEC) * 1000);

float max_pressure = pressure[0];
start_time = clock();    for (int i = 1; i
< inp; i++) {        if (pressure[i] >
max_pressure) {
max_pressure = pressure[i];
        }
    }

    end_time = clock();    printf("Maximum pressure (Linear Search): %.2f hPa\n", max_pressure);
printf("Time taken: %.4f ms\n\n", ((double)(end_time - start_time) / CLOCKS_PER_SEC) * 1000);
min_temp = temp[0];    start_time = clock();

```

```

    for (int i = 0; i < inp; i++) {
int is_min = 1;    for (int j =
0; j < inp; j++) {    if
(temp[i] > temp[j]) {
is_min = 0;        break;
    }
    }
    if (is_min) {
min_temp = temp[i];
        break;
    }
}

end_time = clock();  printf("Minimum temperature (Naive Search): %.2f°C\n", min_temp);
printf("Time taken: %.4f ms\n\n", ((double)(end_time - start_time) / CLOCKS_PER_SEC) * 1000);

```

```

    max_pressure = pressure[0];
start_time = clock();  for (int i = 0;
i < inp; i++) {    int is_max = 1;
for (int j = 0; j < inp; j++) {    if
(pressure[i] < pressure[j]) {
is_max = 0;        break;
    }
    }
    if (is_max) {
        max_pressure = pressure[i];
        break;
    }
}

end_time = clock();  printf("Maximum pressure (Naive Search): %.2f hPa\n", max_pressure);
printf("Time taken: %.4f ms\n\n", ((double)(end_time - start_time) / CLOCKS_PER_SEC) * 1000);

```

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

For input n=100

Output
Enter input size (up to 100): 100 Minimum temperature (Linear Search): -19.95°C Time taken: 0.0010 ms Maximum pressure (Linear Search): 1049.95 hPa Time taken: 0.0010 ms Minimum temperature (Naive Search): -19.95°C Time taken: 0.0030 ms Maximum pressure (Naive Search): 1049.95 hPa Time taken: 0.0010 ms

For input n=10000

Output
Enter input size (up to 10000): 10000 Minimum temperature (Linear Search): -20.00°C Time taken: 0.0120 ms Maximum pressure (Linear Search): 1049.99 hPa Time taken: 0.0230 ms Minimum temperature (Naive Search): -20.00°C Time taken: 0.0660 ms Maximum pressure (Naive Search): 1049.99 hPa Time taken: 0.1260 ms

Task 3 -1. Generate sorted data for temperature (range: 20 to 50)

2. Find the first Occurrence of temperature ≥ 30 .

- Apply Linear search
- Apply Binary Search

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

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

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

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

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

```
    float* temp = (float*)malloc(sizeof(float) * 10000);  
    int inp;
```

```
    printf("Enter input size (max 10000): ");  
    scanf("%d", &inp);
```

```
    // Accept input size up to 10000  
    if (inp <= 0 || inp > 10000) {  
        printf("Invalid size.\n");  
        free(temp);    return 1;  
    }
```

```
    // Generate random temps  
    for (int i = 0; i < inp; i++) {  
        temp[i] = generateRandomNumber(-20, 50);  
    }
```

```

// Bubble Sort (inefficient for large inputs, but simple)
for (int i = 0; i < inp - 1; 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;
        }
    }
}

```

```

// Linear Search: find first temperature >= 30
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("Linear Search index (temp >= 30): %d\n", index);
printf("Time taken: %.4f ms\n\n", ((double)(end_time - start_time) / CLOCKS_PER_SEC) * 1000);

```

```

// Binary Search: find first temperature >= 30
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("Binary Search index (temp >= 30): %d\n", index);    printf("Time  
taken: %.4f ms\n\n", ((double)(end_time - start_time) / CLOCKS_PER_SEC) * 1000);  
  
    free(temp);  
    return 0;  
}
```

Fir input n =100

Output

```
Enter input size (max 10000): 100  
Linear Search index (temp >= 30): 69  
Time taken: 0.0020 ms
```

```
Binary Search index (temp >= 30): 69  
Time taken: 0.0010 ms
```

```
=== Code Execution Successful ===
```

For input n=10000

Output

```
Enter input size (max 10000): 10000  
Linear Search index (temp >= 30): 7193  
Time taken: 0.0150 ms
```

```
Binary Search index (temp >= 30): 7193  
Time taken: 0.0020 ms
```

```
=== Code Execution Successful ===
```