Mann Dave

A8-B4-56

Practical-1

Practical 1: 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 Task B:

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

float generateRandomNumber(float min, float max) {
    float offset = 0;
    int flag = 0;
    float temp;

    if (min < 0 || max < 0) {
        offset = (min < max) ? min : max;
        min -= offset;
        max -= offset;
        flag = 1;
```

```
    } else {
        temp = min;
        max = max - min;
        min = 0;
    }

    float number = rand() % (int)max;
    number = number + (float)rand() / (float)RAND_MAX;
    if (flag) {
        number += offset;
    } else {
        number += temp;
    }

    return number;
}

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);

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

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

float min = temp[0];
clock_t start_time = clock();
for (int j = 0; j < inp; j++) {
   if (temp[j] < min) {
      min = temp[j];
   }
}
clock_t end_time = clock();
```

```
   printf("Minimum temperature is: %f\nTime Taken to
execute is: %f ms\n\n",
          min, ((double)(end_time - start_time) /
CLOCKS_PER_SEC) * 1000);


   float max = pressure[0];
   start_time = clock();
   for (int j = 0; j < inp; j++) {
      if (pressure[j] > max) {
         max = pressure[j];
      }
   }
   end_time = clock();
   printf("Maximum pressure is: %f\nTime Taken to execute
is: %f ms\n\n",
          max, ((double)(end_time - start_time) /
CLOCKS_PER_SEC) * 1000);


   min = temp[0];
   start_time = clock();
   for (int j = 0; j < inp; j++) {
      for (int i = 0; i < inp; i++) {
         if (temp[j] < temp[i]) {
```

```
            min = temp[j];
        }
    }
}
end_time = clock();
printf("Minimum temperature is: %f\nTime Taken to execute is: %f ms\n\n",
        min, ((double)(end_time - start_time) / CLOCKS_PER_SEC) * 1000);

max = pressure[0];
start_time = clock();
for (int j = 0; j < inp; j++) {
    for (int i = 0; i < inp; i++) {
        if (pressure[j] < pressure[i]) {
            max = pressure[j];
        }
    }
}
end_time = clock();
printf("Maximum pressure is: %f\nTime Taken to execute is: %f ms\n\n",
```

```
        max, ((double)(end_time - start_time) /
CLOCKS_PER_SEC) * 1000);


    free(temp);
    free(pressure);


    return 0;
}
```

For Input Size 100

```
Enter input size: 100
Minimum temperature is: -17.404230
Time Taken to execute is: 0.002000 ms

Maximum pressure is: 1049.669922
Time Taken to execute is: 0.001000 ms

Minimum temperature is: 34.455132
Time Taken to execute is: 0.047000 ms

Maximum pressure is: 950.512878
Time Taken to execute is: 0.061000 ms
```

For Input Size 10000

```
Enter input size: 10000
Minimum temperature is: -19.984932
Time Taken to execute is: 0.028000 ms

Maximum pressure is: 1049.996094
Time Taken to execute is: 0.021000 ms

Minimum temperature is: 23.096539
Time Taken to execute is: 526.193000 ms

Maximum pressure is: 981.315002
Time Taken to execute is: 522.054000 ms
```

Code For Task-3

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


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

```
    return number;
}

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

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

    if (inp <= 0) {
        printf("Invalid input size.\n");
        return 1;
    }

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

    for (int i = 0; i < inp; i++) {
```

```
    temp[i] = generateRandomNumber(-20.0f, 50.0f);
}

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;
        }
    }
}

clock_t start_time = clock();
int index = -1;
for (int i = 0; i < inp; i++) {
    if (temp[i] >= 30.0f) {
        index = i;
        break;
    }
}
clock_t end_time = clock();
```

```
   printf("Element found at index for Linear Search:
%d\nTime Taken to execute: %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] >= 30.0f) {
         index = mid;
         right = mid - 1;
      } else {
         left = mid + 1;
      }
   }
   end_time = clock();
   printf("Element found at index for Binary Search:
%d\nTime Taken to execute: %f ms\n\n",
          index, ((double)(end_time - start_time) /
CLOCKS_PER_SEC) * 1000);
```

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

For Input Size 100

```
Enter input size: 100
Element found at index for Linear Search: 73
Time Taken to execute: 0.002000 ms

Element found at index for Binary Search: 73
Time Taken to execute: 0.001000 ms
```

For Input Size 10000

```
Enter input size: 10000
Element found at index for Linear Search: 7173
Time Taken to execute: 0.009000 ms

Element found at index for Binary Search: 7173
Time Taken to execute: 0.001000 ms
```