Design & Analysis of Algorithms

LAB - 1

Submitted by:
Venkatesan M
22BAI1259

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Submitted to:
Dr. Indira B
53139

AIM

Problem for a problem using STL - with a stress on 'Containers' and 'Iterators'.

ALGORITHM

1. Read the number of mark details of a Quiz, N.
2. Read the data of each participant
3. Assume the first participant scored the minimum and maximum
4. From the second participant, for each participant ‘p’ repeat step 5 and 6
5. If mark of participant ‘p’ is less than current minimum mark participant the make ‘p’ as minimum mark participant
6. If mark of participant ‘p’ is greater than current maximum mark participant then make ‘p’ as maximum mark participant
7. Print details of time taken to execute, minimum mark participant and maximum mark participant

Code

```
#include <iostream>
#include <vector>
#include <ctime>
#include <limits>
#include <string>
#include <algorithm>

using namespace std;

struct Participant {
    string name;
    int score;
};

int main() {
    int N;
    cout << "Enter the number of participants: ";
    cin >> N;

    vector<Participant> participants(N);

    for (int i = 0; i < N; ++i) {
        cout << "Enter name and score for participant "
<< i + 1 << ": ";
        cin >> participants[i].name >>
participants[i].score;
    }
    // Timing the execution
    clock_t start = clock();

    auto minMaxParticipant = minmax_element(
        participants.begin(), participants.end(),
        [](const Participant &p1, const Participant &p2)
{
            return p1.score < p2.score;
        });

    Participant minParticipant =
*minMaxParticipant.first;
    Participant maxParticipant =
*minMaxParticipant.second;

    // Calculate execution time
    double duration = (clock() - start) /
(double)CLOCKS_PER_SEC;

    cout << "Time taken to execute: " << duration << "
seconds\n";
    cout << "Minimum mark participant: " <<
minParticipant.name << " with score " <<
minParticipant.score << "\n";
    cout << "Maximum mark participant: " <<
maxParticipant.name << " with score " <<
maxParticipant.score << "\n";

    return 0;
}
```

Sample Input and Output

Sample Input:

```
Enter the number of participants: 5
Enter name and score for participant 1: Alex 92
Enter name and score for participant 2: Tony 96
Enter name and score for participant 3: Mike 91
Enter name and score for participant 4: Ken 88
Enter name and score for participant 5: Marshell 75
```

Sample Output:

```
Time taken to execute: 2e-05 seconds
Minimum mark participant: Marshell with score 75
Maximum mark participant: Tony with score 96
```

Graph Between Inputs and Time

```
void measureExecutionTime(int N) {
  vector<Participant> participants(N);

  // Generate random data
  for (int i = 0; i < N; ++i) {
    participants[i].name = "Participant" + to_string(i +
1);
    participants[i].score = rand() % 101; // Random
score between 0 and 100
  }

  // Timing the execution
  clock_t start = clock();

  auto minMaxParticipant = minmax_element(
    participants.begin(), participants.end(),
    [](const Participant &p1, const Participant &p2) {
      return p1.score < p2.score;
    });

  Participant minParticipant =
*minMaxParticipant.first;
  Participant maxParticipant =
*minMaxParticipant.second
```

```
 // Calculate execution time
 double duration = (clock() - start) /
(double)CLOCKS_PER_SEC;

  cout << "Number of Participants: " << N << "\n";
  cout << "Time taken to execute: " << duration << "
seconds\n";
  cout << "Minimum mark participant: " <<
minParticipant.name << " with score " <<
minParticipant.score << "\n";
  cout << "Maximum mark participant: " <<
maxParticipant.name << " with score " <<
maxParticipant.score << "\n";
  cout << "-------------------------------------
--\n";
}

int main() {
  srand(time(0)); // Seed for random number
generation

  // List of participant counts to test
  vector<int> participantCounts = {10, 100, 1000, 10000,
100000};

  for (int count : participantCounts) {
    measureExecutionTime(count);
  }

  return 0;
}
```

Plotting the time taken

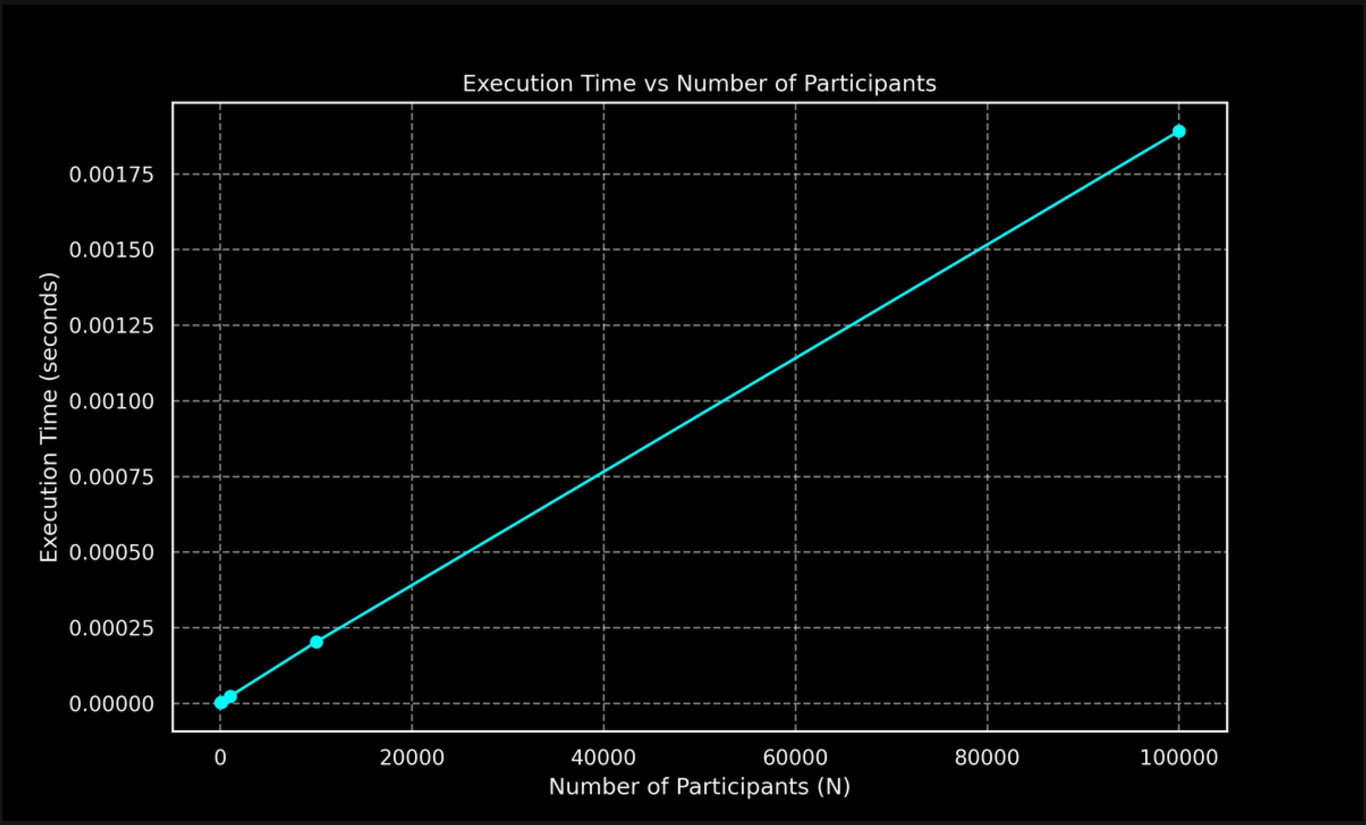
Output

```
Number of Participants: 10 Time taken to execute: 3e-
06 seconds Minimum mark participant: Participant9
with score 3 Maximum mark participant: Participant2
with score 99 ----------------------------------
------ Number of Participants: 100 Time taken to
execute: 4e-06 seconds Minimum mark participant:
Participant97 with score 1 Maximum mark
participant: Participant62 with score 100 -----------
------------------------------ Number of
Participants: 1000 Time taken to execute: 2.4e-05
seconds Minimum mark participant: Participant10
with score 0 Maximum mark participant:
Participant874 with score 100 ---------------------
-------------------- Number of Participants: 10000
Time taken to execute: 0.000204 seconds Minimum
mark participant: Participant8 with score 0 Maximum
mark participant: Participant9963 with score 100 ----
-------------------------------------- Number of
Participants: 100000 Time taken to execute: 0.001892
seconds Minimum mark participant: Participant175
with score 0 Maximum mark participant:
Participant99908 with score 100 -------------------
----------------------
```

Script to plot the Time Taken

```
import matplotlib.pyplot as plt

# Data
participants = [10, 100, 1000, 10000, 100000]
execution_times = [3e-06, 4e-06, 2.4e-05, 0.000204,
0.001892]

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(participants, execution_times, marker='o',
linestyle='-', color='b')

# Set the labels and title
plt.xlabel('Number of Participants (N)')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time vs Number of Participants')
plt.grid(True)

# Save the plot as a PNG file
plt.savefig('execution_time_vs_participants.png')
```

Plotting this data on a graph with the number of participants on the x-axis and execution time on the y-axis would show a linear relationship, indicating the efficiency of the algorithm for input size.



Complexity Analysis

The algorithm uses the minmax_element function from the STL, which performs a single pass over the range to find both the minimum and maximum elements.

- Time Complexity: O(N), where N is the number of participants. This is because minmax_element iterates through the entire vector once to determine the minimum and maximum scores.
- Space Complexity: O(1) for additional space, as no extra space is allocated apart from variables needed to store the participants' information and the results.

Result

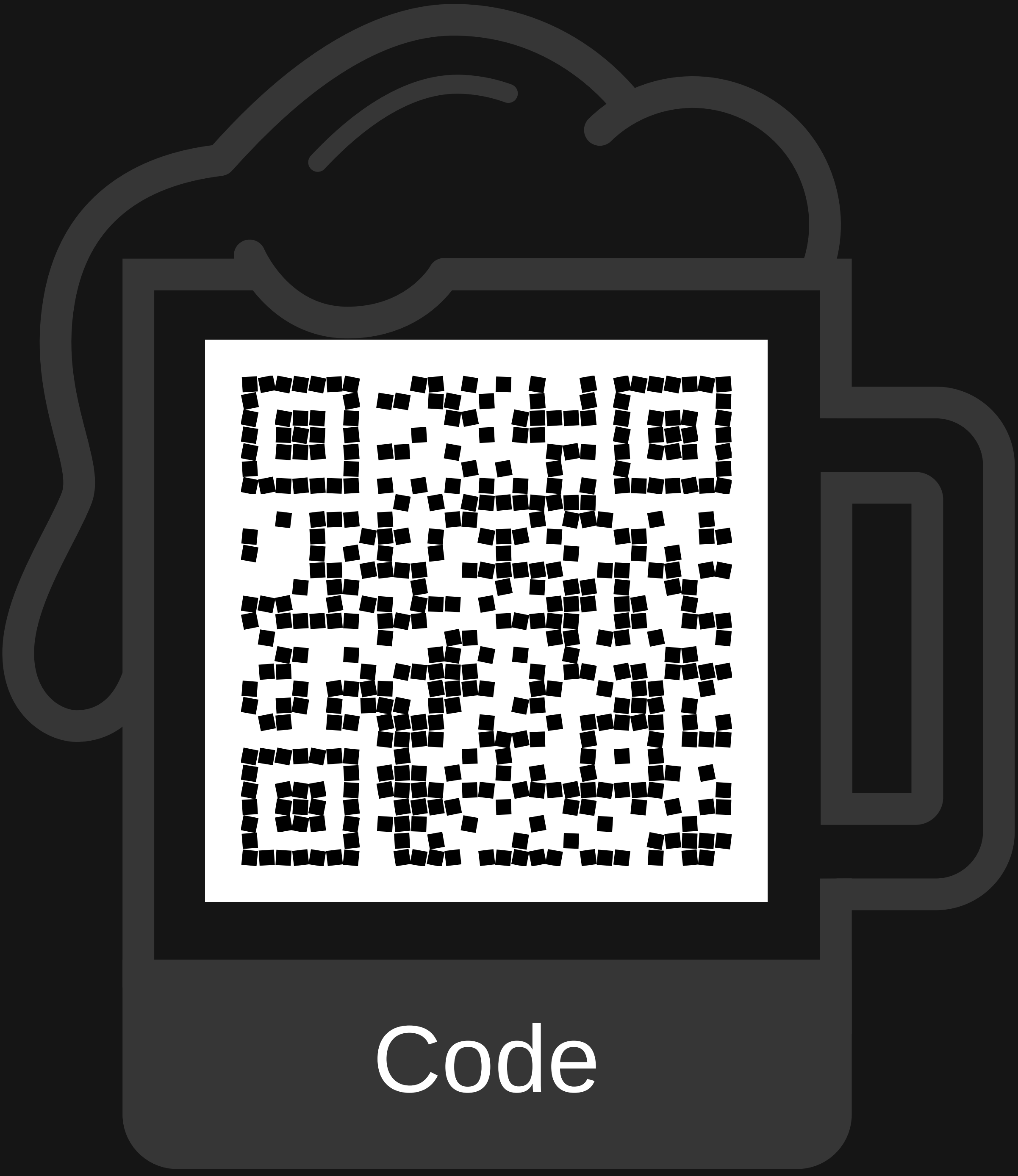
The program uses STL containers and iterators to determine the participants with the minimum and maximum scores.

The execution time is minimal due to the single-pass algorithm, and the results are accurate and easily understandable.

Using vector for storage and minmax_element for finding the minimum and maximum elements showcases the power and simplicity of STL in handling common data processing tasks.

The linear time complexity ensures that the program performs well even with large input sizes, making it suitable for practical applications.

Github Codebase



Thank you !

Venkatesan M

venkatesan.m2022@vitstudent.ac.in