codes\Practical 3\Practical3rd.cpp

```cpp
#include <iostream>
#include <vector>
#include <chrono>
#include <random>
#include <time.h>
#include <fstream>
#include <sys/stat.h>
#include <sstream>
using namespace std;

#define vi vector<int>

// Function to create "test" folder if it doesn't exist
void create_directory(const string &folder_name)
{
    struct stat info;
    if (stat(folder_name.c_str(), &info) != 0)
    {
        system(("mkdir " + folder_name).c_str()); // Create folder
    }
}
// Function to generate a unique file name
string get_unique_filename(const string &folder_name, const string &base_name)
{
    int counter = 0;
    string filename;

    do
    {
        stringstream ss;
        ss << folder_name << "/" << base_name << "_" << counter << ".txt";
        filename = ss.str();
        ifstream file_check(filename);
        if (!file_check.good())
        { // If file does not exist, return this name
            return filename;
        }
        counter++; // Increment counter if file exists
    } while (true);
}
void merge(vi &arr, int st, int mid, int end)
{
    vi temp;
    int siz, i = st, j = mid + 1;
    // siz = arr.size();
    while (i <= mid && j <= end)
    {
        if (arr[i] <= arr[j])
```

```
        {
            temp.push_back(arr[i]);
            i++;
        }
        else
        {
            temp.push_back(arr[j]);
            j++;
        }
    }
    while (i <= mid)
    {
        temp.push_back(arr[i]);
        i++;
    }
    while (j <= end)
    {
        temp.push_back(arr[j]);
        j++;
    }
    for (int id = 0; id < temp.size(); id++)
    {
        arr[st + id] = temp[id];
    }
}
void mergesort(vi &arr, int st, int end)

{
    if (st < end)
    {
        int mid = st + (end - st) / 2;
        mergesort(arr, st, mid);
        mergesort(arr, mid + 1, end);
        merge(arr, st, mid, end);
    }
}
int main()
{
    string folder_name = "test";
    create_directory(folder_name); // Ensure test folder exists

    int numb;
    cout << "Enter the number of tests to perform:";
    cin >> numb;
    cout << endl;
    for (int p = 0; p < numb; p++)
    {
        string filename = get_unique_filename(folder_name, "Test"); // Generate unique file name
        ofstream file(filename);
        // creating a file that loads the execution time in a FILE START
```

```
        if (!file)
        {
            cerr << "Error opening file!" << endl;
            return 1;
        }

        // creating a file that loads the execution time in a FILE end
        file << "["; // Start the array format
        // giving a array which has numbers of values to generate
        int n, min_val, max_val;
        int numbers[] = {10, 100, 500, 1000, 5000, 10000, 50000, 100000};

        // loopin arra on the code
        for (int n : numbers)
        {

            // Random number generation

            // cout << "Enter the number of random integers to generate: ";
            // cin >> n;
            // cout << "Enter the minimum and maximum range: ";
            // cin >> min_val >> max_val;         comment out this line if you want to give min_val and max_val

            int arr[n]; // Declare an array to store random numbers

            // Random number generator setup
            random_device rd;  // Seed generator
            mt19937 gen(rd()); // Mersenne Twister engine
            uniform_int_distribution<int> dist(1, n + 10000);
            //   you can change (1)->(min_val) and (n)->(max_val)

            cout << "Generated Random Integers" << endl;
            for (int i = 0; i < n; i++)
            {
                arr[i] = dist(gen); // Store in array
            }
            // random number generation end
            // Convert array to vector
            vi vec(arr, arr + n);

            // Sorting start
            cout << "Array before sorting:";
            for (int i = 0; i < vec.size(); i++)
            {
                cout << arr[i] << " ";
            }
            cout << endl;
            auto start = chrono::high_resolution_clock::now();
            mergesort(vec, 0, vec.size() - 1);
```

```
            auto end = chrono::high_resolution_clock::now();
            cout << "Array after sorting:";

            for (int i = 0; i < vec.size(); i++)
            {

                cout << vec[i] << " ";
            }
            cout << endl;
            auto start_time = chrono::duration_cast<chrono::microseconds>
(start.time_since_epoch()).count();
            auto end_time = chrono::duration_cast<chrono::microseconds>
(end.time_since_epoch()).count();
            auto duration_ns = chrono::duration_cast<chrono::nanoseconds>(end - start);
            if (n != numbers[sizeof(numbers) / sizeof(numbers[0]) - 1]) {
    file << duration_ns.count() << ",";  // ✅ No extra comma at the end
} else {
    file << duration_ns.count();  // ✅ No comma for the last element
}

            cout << "start_time was:" << start_time << " ns" << endl
                 << "end_time was:" << end_time << " ns" << endl;
            cout << "Execution Time:";
            cout << duration_ns.count() << " ns ";
            // _sleep(10000);
        }
        file << "]"; // end the array format
        file.close();
    }
    return 0;
}

```