```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <string>
#include <iomanip>
#include <fstream>
#include <sys/stat.h>
#include <xlsxwriter.h>

using namespace std;
using namespace chrono;

long long comparisons;
lxw_workbook  *workbook = nullptr;
lxw_worksheet *worksheet = nullptr;
lxw_format *header_format = nullptr;
lxw_format *row_format = nullptr;
lxw_format *separator_format = nullptr;

int current_row = 1;
int current_col = 0;
int block_counter = 0;
bool generate_excel = true;

vector<long double> selection_times, bubble_times, insertion_times, merge_times, quick_times;
vector<long long> selection_comparisons, bubble_comparisons, insertion_comparisons, merge_comparisons,
quick_comparisons;
vector<int> input_sizes;

void selectionSort(vector<int> arr) {
    comparisons = 0;
    for (size_t i = 0; i < arr.size() - 1; i++) {
        int min_idx = i;
        for (size_t j = i + 1; j < arr.size(); j++) {
            comparisons++;
            if (arr[j] < arr[min_idx]) min_idx = j;
        }
        swap(arr[i], arr[min_idx]);
    }
}

void bubbleSort(vector<int> arr) {
    comparisons = 0;
    for (size_t i = 0; i < arr.size() - 1; i++) {
        for (size_t j = 0; j < arr.size() - i - 1; j++) {
            comparisons++;
            if (arr[j] > arr[j + 1]) swap(arr[j], arr[j + 1]);
        }
    }
}

void insertionSort(vector<int> arr) {
    comparisons = 0;
    for (size_t i = 1; i < arr.size(); i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0) {
            comparisons++;
            if (arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            } else break;
        }
        arr[j + 1] = key;
    }
}

long long merge(vector<int>& arr, int l, int m, int r) {
    long long comps = 0;
    vector<int> L(arr.begin() + l, arr.begin() + m + 1);
    vector<int> R(arr.begin() + m + 1, arr.begin() + r + 1);
    int i = 0, j = 0, k = l;
    while (i < L.size() && j < R.size()) {
        comps++;
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < L.size()) arr[k++] = L[i++];
    while (j < R.size()) arr[k++] = R[j++];
    return comps;
}

long long mergeSortRec(vector<int>& arr, int l, int r) {
    long long comps = 0;
    if (l < r) {
        int m = l + (r - l) / 2;
        comps += mergeSortRec(arr, l, m);
        comps += mergeSortRec(arr, m + 1, r);
        comps += merge(arr, l, m, r);
    }
    return comps;
}

void mergeSort(vector<int> arr) {
    comparisons = mergeSortRec(arr, 0, arr.size() - 1);
}

long long quickSortRec(vector<int>& arr, int low, int high) {
    long long comps = 0;
    if (low < high) {
        int pivot = arr[high], i = low - 1;
        for (int j = low; j < high; j++) {
            comps++;
            if (arr[j] < pivot) {
                i++;
                swap(arr[i], arr[j]);
            }
        }
        swap(arr[i + 1], arr[high]);
        int pi = i + 1;
        comps += quickSortRec(arr, low, pi - 1);
        comps += quickSortRec(arr, pi + 1, high);
    }
    return comps;
}

void quickSort(vector<int> arr) {
    comparisons = quickSortRec(arr, 0, arr.size() - 1);
}

void writeExcelHeader(int row, int col_offset) {
    worksheet_write_string(worksheet, row, col_offset + 0, "n", header_format);
    worksheet_write_string(worksheet, row, col_offset + 1, "Algorithm", header_format);
    worksheet_write_string(worksheet, row, col_offset + 2, "Comparisons", header_format);
    worksheet_write_string(worksheet, row, col_offset + 3, "Time(ms)", header_format);
}

void writeExcelRow(int row, int col_offset, int n, const string& algo, long long comps, long double time_ms) {
    worksheet_write_number(worksheet, row, col_offset + 0, n, row_format);
    worksheet_write_string(worksheet, row, col_offset + 1, algo.c_str(), row_format);
    worksheet_write_number(worksheet, row, col_offset + 2, comps, row_format);
    worksheet_write_number(worksheet, row, col_offset + 3, time_ms, row_format);
}

void writeExcelSeparatorRow(int row, int col_offset) {
    for (int i = 0; i < 4; ++i)
        worksheet_write_blank(worksheet, row, col_offset + i, separator_format);
}

void runSort(void(*sortFunc)(vector<int>), vector<int> arr, const string& name, int n, int col_offset, vector<long long>&
comp_list, vector<long double>& time_list) {
    auto start = high_resolution_clock::now();
    sortFunc(arr);
    auto end = high_resolution_clock::now();
    long double duration_ns = duration_cast<nanoseconds>(end - start).count();
    long double duration_ms = duration_ns / 1'000'000.0;

    cout << left << setw(12) << name << setw(16) << comparisons << fixed << setprecision(4) << duration_ms << "
ms" << endl;

    if (generate_excel) {
        writeExcelRow(current_row, col_offset, n, name, comparisons, duration_ms);
        current_row++;
    }

    comp_list.push_back(comparisons);
    time_list.push_back(duration_ms);
}


void runSort(void(*sortFunc)(vector<int>), vector<int> arr, const string& name, int n, int col_offset, vector<long long>&
comp_list, vector<long long>& time_list) {
    auto start = high_resolution_clock::now();
    sortFunc(arr);
    auto end = high_resolution_clock::now();
    long long duration_ns = duration_cast<nanoseconds>(end - start).count();
    long long duration_ms = duration_ns / 1'000'000;

    cout << left << setw(12) << name << setw(16) << comparisons << duration_ms << " ms" << endl;

    if (generate_excel) {
        writeExcelRow(current_row, col_offset, n, name, comparisons, duration_ms);
        current_row++;
    }

    comp_list.push_back(comparisons);
    time_list.push_back(duration_ms);
}

void addSummaryAndChart(int start_row) {
    const char* algo_names[] = {"Selection", "Bubble", "Insertion", "MergeSort", "QuickSort"};
    worksheet_write_string(worksheet, start_row, 0, "n", header_format);
    for (int i = 0; i < 5; ++i)
        worksheet_write_string(worksheet, start_row, 1 + i, algo_names[i], header_format);

    for (size_t i = 0; i < input_sizes.size(); ++i) {
        worksheet_write_number(worksheet, start_row + 1 + i, 0, input_sizes[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 1, selection_comparisons[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 2, bubble_comparisons[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 3, insertion_comparisons[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 4, merge_comparisons[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 5, quick_comparisons[i], row_format);
    }

    lxw_chart *chart = workbook_add_chart(workbook, LXW_CHART_LINE);
    chart_axis_set_name(chart->x_axis, "Input Size (n)");
    chart_axis_set_name(chart->y_axis, "Element Comparisons");
    chart_title_set_name(chart, "Comparisons vs Input Size");

    int chart_start = start_row + 1;
    int chart_end = chart_start + input_sizes.size() - 1;

    for (int i = 0; i < 5; ++i) {
        lxw_chart_series *series = chart_add_series(chart, NULL, NULL);
        chart_series_set_name(series, algo_names[i]);
        chart_series_set_categories(series, "Sheet1", chart_start, 0, chart_end, 0);
        chart_series_set_values(series, "Sheet1", chart_start, 1 + i, chart_end, 1 + i);
    }

    worksheet_insert_chart(worksheet, start_row, 7, chart);
}

void addTimeChart(int start_row) {
    const char* algo_names[] = {"Selection", "Bubble", "Insertion", "MergeSort", "QuickSort"};

    worksheet_write_string(worksheet, start_row, 0, "n", header_format);
    for (int i = 0; i < 5; ++i)
```

```
        worksheet_write_string(worksheet, start_row, 1 + i, algo_names[i], header_format);

    for (size_t i = 0; i < input_sizes.size(); ++i) {
        worksheet_write_number(worksheet, start_row + 1 + i, 0, input_sizes[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 1, selection_times[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 2, bubble_times[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 3, insertion_times[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 4, merge_times[i], row_format);
        worksheet_write_number(worksheet, start_row + 1 + i, 5, quick_times[i], row_format);
    }

    lxw_chart *chart = workbook_add_chart(workbook, LXW_CHART_LINE);
    chart_axis_set_name(chart->x_axis, "Input Size (n)");
    chart_axis_set_name(chart->y_axis, "Runtime (ms)");
    chart_title_set_name(chart, "Runtime vs Input Size");

    int chart_start = start_row + 1;
    int chart_end = chart_start + input_sizes.size() - 1;

    for (int i = 0; i < 5; ++i) {
        lxw_chart_series *series = chart_add_series(chart, NULL, NULL);
        chart_series_set_name(series, algo_names[i]);
        chart_series_set_categories(series, "Sheet1", chart_start, 0, chart_end, 0);
        chart_series_set_values(series, "Sheet1", chart_start, 1 + i, chart_end, 1 + i);
    }

    worksheet_insert_chart(worksheet, start_row, 7, chart);
}

int main() {
    workbook = workbook_new("projectexcel.xlsx");
    worksheet = workbook_add_worksheet(workbook, NULL);

    header_format = workbook_add_format(workbook);
    format_set_bold(header_format);
    format_set_font_color(header_format, LXW_COLOR_WHITE);
    format_set_bg_color(header_format, 0x1F4E78);
```

```cpp
format_set_align(header_format, LXW_ALIGN_CENTER);
format_set_border(header_format, LXW_BORDER_THIN);
format_set_border_color(header_format, LXW_COLOR_WHITE);

row_format = workbook_add_format(workbook);
format_set_bg_color(row_format, 0xDDEEFF);
format_set_align(row_format, LXW_ALIGN_CENTER);
format_set_border(row_format, LXW_BORDER_THIN);
format_set_border_color(row_format, LXW_COLOR_WHITE);

separator_format = workbook_add_format(workbook);
format_set_bg_color(separator_format, LXW_COLOR_WHITE);

srand(time(0));
vector<int> sizes = {100, 500, 1000, 1500, 2000, 3000, 4000, 5000};

for (int n : sizes) {
    input_sizes.push_back(n);

    if (block_counter == 2) {
        writeExcelSeparatorRow(current_row, current_col);
        current_row = 1;
        current_col += 5;
        block_counter = 0;
    }

    long long theo_nlogn = static_cast<long long>(n * log2(n));
    cout << "\nInput Size: " << n << endl;
    cout << left << setw(12) << "Algorithm" << setw(16) << "Comparisons" << "Time" << endl;
    cout << "-------------------------------------------------------" << endl;
    cout << left << setw(12) << "nlogn" << setw(16) << theo_nlogn << "-" << endl;

    if (current_row == 1)
        writeExcelHeader(0, current_col);

    writeExcelRow(current_row, current_col, n, "nlogn", theo_nlogn, 0);
    current_row++;
```

```
        vector<int> base(n);
        generate(base.begin(), base.end(), []() { return rand() % 101; });

        runSort(selectionSort, base, "Selection", n, current_col, selection_comparisons, selection_times);
        runSort(bubbleSort, base, "Bubble", n, current_col, bubble_comparisons, bubble_times);
        runSort(insertionSort, base, "Insertion", n, current_col, insertion_comparisons, insertion_times);
        runSort(mergeSort, base, "MergeSort", n, current_col, merge_comparisons, merge_times);
        runSort(quickSort, base, "QuickSort", n, current_col, quick_comparisons, quick_times);

        block_counter++;

        if (n != sizes.back()) {
            writeExcelSeparatorRow(current_row, current_col);
            current_row++;
        }
    }

    cout << endl << endl;
    int summary_start_row = current_row + 3;
    addSummaryAndChart(summary_start_row);
    addTimeChart(summary_start_row + input_sizes.size() + 5);

    workbook_close(workbook);
    return 0;
}
```