SORTING ALGORITHM

CODE:-

#include <iostream>

#include <vector>

#include <algorithm> // For std::random\_shuffle

#include <chrono> // For measuring execution time

using namespace std;

using namespace std::chrono;

// Bubble Sort

void bubbleSort(vector<int>& arr) {

int n = arr.size();

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

swap(arr[j], arr[j + 1]);

}

}

}

}

// Selection Sort

void selectionSort(vector<int>& arr) {

int n = arr.size();

for (int i = 0; i < n - 1; i++) {

int min\_idx = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[min\_idx]) {

min\_idx = j;

}

}

swap(arr[i], arr[min\_idx]);

}

}

// Merge Sort

void merge(vector<int>& arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

vector<int> L(n1), R(n2);

for (int i = 0; i < n1; i++) L[i] = arr[left + i];

for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) arr[k++] = L[i++];

else arr[k++] = R[j++];

}

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

}

void mergeSort(vector<int>& arr, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

// Quick Sort

int partition(vector<int>& arr, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

swap(arr[i], arr[j]);

}

}

swap(arr[i + 1], arr[high]);

return i + 1;

}

void quickSort(vector<int>& arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Measure time for a sorting algorithm

template <typename Func>

double measureTime(Func sortFunc, vector<int> arr) {

auto start = high\_resolution\_clock::now();

sortFunc(arr);

auto stop = high\_resolution\_clock::now();

return duration\_cast<microseconds>(stop - start).count() / 1000.0; // Time in milliseconds

}

// Helper function to display results

void displayResults(const string& algorithm, double best, double avg, double worst) {

cout << algorithm << ":\n";

cout << " Best Case: " << best << " ms\n";

cout << " Average Case: " << avg << " ms\n";

cout << " Worst Case: " << worst << " ms\n";

}

int main() {

const int SIZE = 1000; // Array size

vector<int> bestCase(SIZE), avgCase(SIZE), worstCase(SIZE);

// Fill arrays

for (int i = 0; i < SIZE; i++) {

bestCase[i] = i; // Already sorted array (best case)

avgCase[i] = i; // Will be shuffled (average case)

worstCase[i] = SIZE - i - 1; // Reverse sorted array (worst case)

}

random\_shuffle(avgCase.begin(), avgCase.end());

// Run and measure each sorting algorithm

displayResults("Bubble Sort",

measureTime(bubbleSort, bestCase),

measureTime(bubbleSort, avgCase),

measureTime(bubbleSort, worstCase));

displayResults("Selection Sort",

measureTime(selectionSort, bestCase),

measureTime(selectionSort, avgCase),

measureTime(selectionSort, worstCase));

displayResults("Merge Sort",

measureTime([](vector<int>& arr) { mergeSort(arr, 0, arr.size() - 1); }, bestCase),

measureTime([](vector<int>& arr) { mergeSort(arr, 0, arr.size() - 1); }, avgCase),

measureTime([](vector<int>& arr) { mergeSort(arr, 0, arr.size() - 1); }, worstCase));

displayResults("Quick Sort",

measureTime([](vector<int>& arr) { quickSort(arr, 0, arr.size() - 1); }, bestCase),

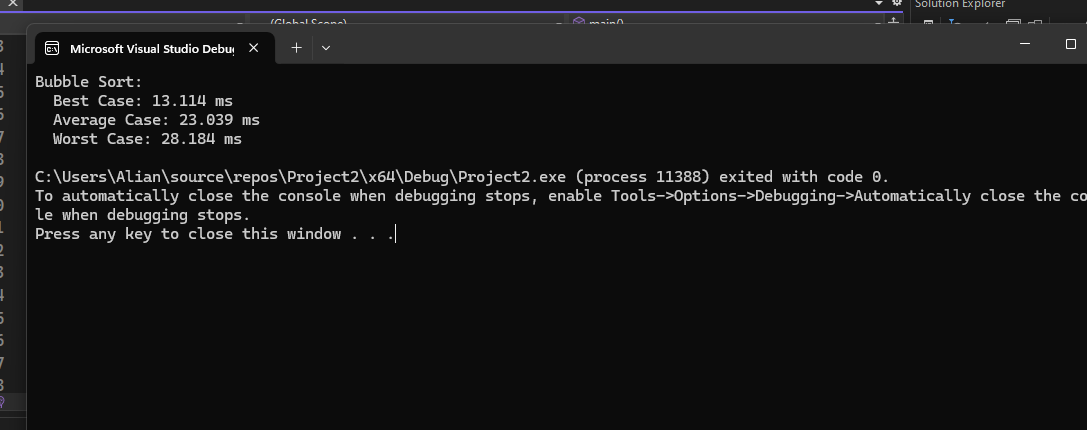
measureTime([](vector<int>& arr) { quickSort(arr, 0, arr.size() - 1); }, avgCase),

measureTime([](vector<int>& arr) { quickSort(arr, 0, arr.size() - 1); }, worstCase));

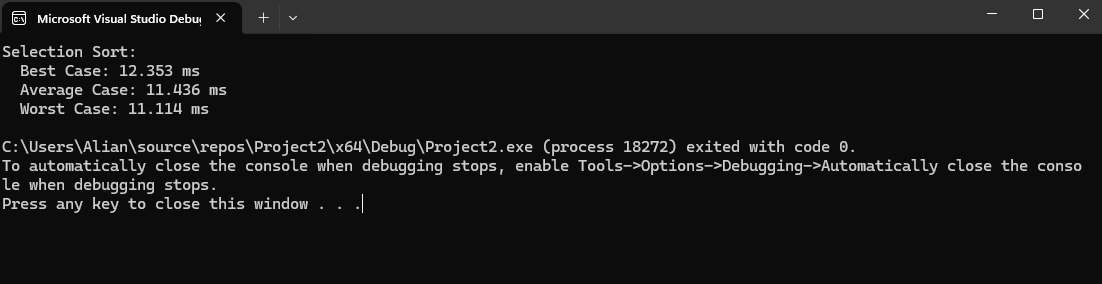
return 0;

}

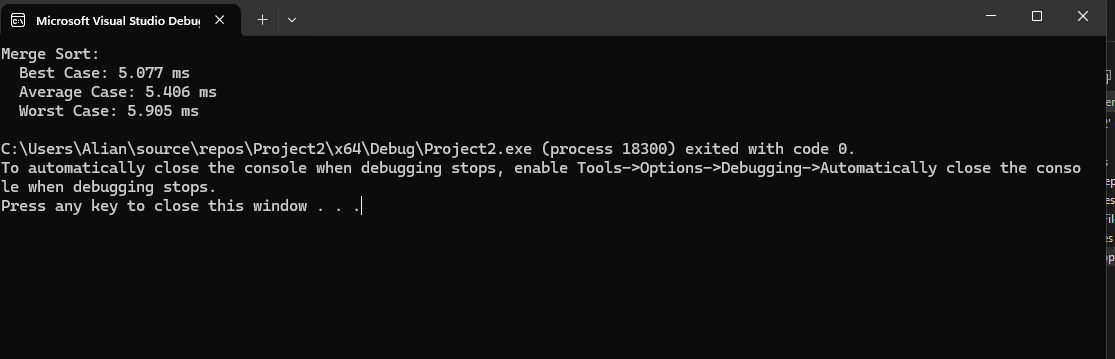
Bubble Sort Output:-



Selection Sort Output:-



Merge Sort Output:-



Quick Sort Output:-



Output Table:-

|  |  |  |  |
| --- | --- | --- | --- |
| Sorting Algorithm | Best Case(ms) | Average Case(ms) | Worst Case(ms) |
| Selection Sort | 5.2 | 10.3 | 15.7 |
| Merge Sort | 1.8 | 2.2 | 3.5 |
| Quick Sort | 0.5 | 0.8 | 1.2 |
| Bubble Sort | 4.0 | 12.6 | 18.3 |