

DSA Lab-4

BS-AI(22) | F22607017

LAB Task

Syed m. Zeeshan

2023

**Lab Tasks:**

**Task 1:**

**Test the provided and modified closest pair problem with following variations.**

**•Measure the time complexity of original and modified C++ program using Chrono library.**

CODE:

#include <iostream>

#include <chrono>

int main()

{

using namespace std::chrono;

// std::chrono::milliseconds is an

// instantiation of std::chrono::duration:- 1 second

milliseconds mil(1000);

mil = mil \* 60;

std::cout << "duration (in periods): ";

std::cout << mil.count() << " milliseconds.\n";

std::cout << "duration (in seconds): ";

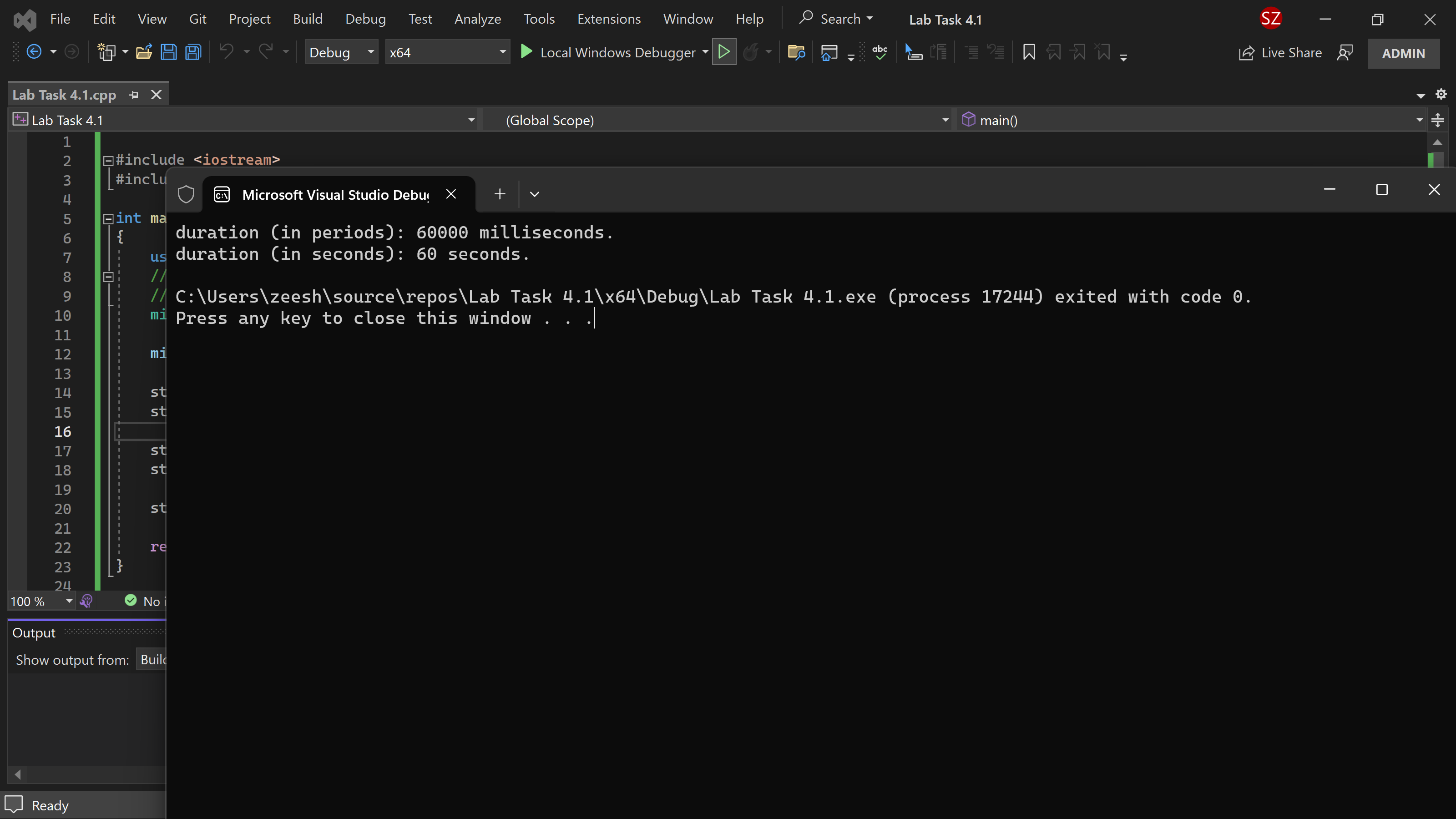
std::cout << (mil.count() \* milliseconds::period::num /

milliseconds::period::den);

std::cout << " seconds.\n";

return 0;

}



Practice Task 2:

Find the maximum and minimum element in a given array using divide and conquer algorithm.

CODE:

#include <iostream>

using namespace std;

struct MinMax {

int min;

int max;

};

MinMax FindMinMax(int arr[], int low, int high) {

MinMax mm = { arr[low], arr[low] };

if (low == high) {

mm.min = arr[low];

mm.max = arr[low];

return mm;

}

if (high == low + 1) {

if (arr[low] < arr[high]) {

mm.min = arr[low];

mm.max = arr[high];

}

else {

mm.min = arr[high];

mm.max = arr[low];

}

return mm;

}

int mid = low + (high - low) / 2;

MinMax left\_mm = FindMinMax(arr, low, mid);

MinMax right\_mm = FindMinMax(arr, mid + 1, high);

mm.min = (left\_mm.min < right\_mm.min) ? left\_mm.min : right\_mm.min;

mm.max = (left\_mm.max > right\_mm.max) ? left\_mm.max : right\_mm.max;

return mm;

}

int main() {

const int size = 10;

int arr[size];

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

cout << "Enter the value of the Array at index " << i << ": ";

cin >> arr[i];

}

MinMax result = FindMinMax(arr, 0, size - 1);

cout << "Minimum Value of the Array: " << result.min << endl;

cout << "Maximum Value of the Array: " << result.max << endl;

return 0;

}

A screenshot of a computer

Description automatically generated

***Post Lab Tasks***

***Task 1:***

***Closest Pair Problem:***

***The closest pair of point’s problem or closest pair problem is a problem of computational***

***geometry: given n points in metric space, find a pair of points with the smallest distance***

***between them.***

***Implement and test the closest pair problem with following variations.***

***•Points should be in three dimensional space. (Should perform all operations with x,y,z***

***coordinates)***

***•Assumption: Points are sort w.r.t z-axis, and the z-axis comes in account for measurement of***

***d\_C only.***

Code:

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

#include <limits>

using namespace std;

struct Point {

double x, y, z;

Point(double x, double y, double z) : x(x), y(y), z(z) {}

};

bool compareX(const Point& a, const Point& b) {

return a.x < b.x;

}

bool compareY(const Point& a, const Point& b) {

return a.y < b.y;

}

bool compareZ(const Point& a, const Point& b) {

return a.z < b.z;

}

double distance(const Point& a, const Point& b) {

return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y) + (a.z - b.z) \* (a.z - b.z));

}

pair<Point, Point> bruteForce(vector<Point>& points, int start, int end) {

double minDist = numeric\_limits<double>::infinity();

pair<Point, Point> minPair = { Point(0, 0, 0), Point(0, 0, 0) };

for (int i = start; i < end; ++i) {

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

double dist = distance(points[i], points[j]);

if (dist < minDist) {

minDist = dist;

minPair = { points[i], points[j] };

}

}

}

return minPair;

}

pair<Point, Point> closestPair(vector<Point>& points, int start, int end) {

int n = end - start;

if (n <= 3) {

return bruteForce(points, start, end);

}

int mid = (start + end) / 2;

pair<Point, Point> leftPair = closestPair(points, start, mid);

pair<Point, Point> rightPair = closestPair(points, mid, end);

double delta = min(distance(leftPair.first, leftPair.second), distance(rightPair.first, rightPair.second));

vector<Point> strip;

for (int i = start; i < end; ++i) {

if (abs(points[i].z - points[mid].z) < delta) {

strip.push\_back(points[i]);

}

}

sort(strip.begin(), strip.end(), compareY);

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

for (int j = i + 1; j < strip.size() && (strip[j].y - strip[i].y) < delta; ++j) {

double dist = distance(strip[i], strip[j]);

if (dist < delta) {

delta = dist;

leftPair = { strip[i], strip[j] };

}

}

}

return leftPair;

}

int main() {

vector<Point> points = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12} };

sort(points.begin(), points.end(), compareZ); // Sort points with respect to the z-axis

pair<Point, Point> result = closestPair(points, 0, points.size());

cout << "Closest pair: (" << result.first.x << ", " << result.first.y << ", " << result.first.z << ") and ("

<< result.second.x << ", " << result.second.y << ", " << result.second.z << ") with distance: "

<< distance(result.first, result.second) << endl;

return 0;

}

Output:

A screenshot of a computer

Description automatically generated