**Week 2**

姓名：李雅帆 学号：2213041

**E2-1**

**1.代码**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int BinarySearch(const vector<int>& A, int x) {

int left = 0, right = A.size() - 1;

while (left <= right) {

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

if (A[mid] == x) {

return mid;

}

else if (A[mid] < x) {

left = mid + 1;

}

else {

right = mid - 1;

}

}

return -1; // 下x未找到

}

int main() {

int n, x;

cout << "输入排序数组的元素个数: ";

cin >> n;

vector<int> A(n);

cout << "输入" << n << "个排序整数: ";

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

cin >> A[i];

}

cout << "输入要搜索的整数: ";

cin >> x;

int index = BinarySearch(A, x);

if (index != -1) {

cout << x << "在数组中的索引是: " << index << endl;

}

else {

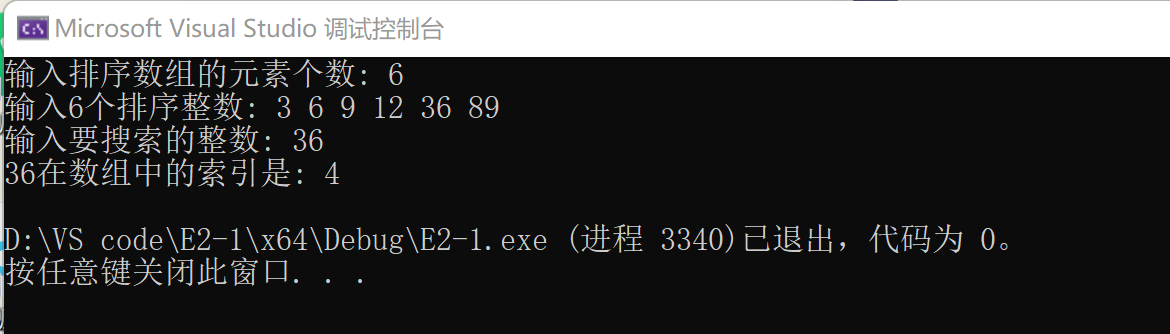
cout<< "在数组中未找到" << x <<endl;

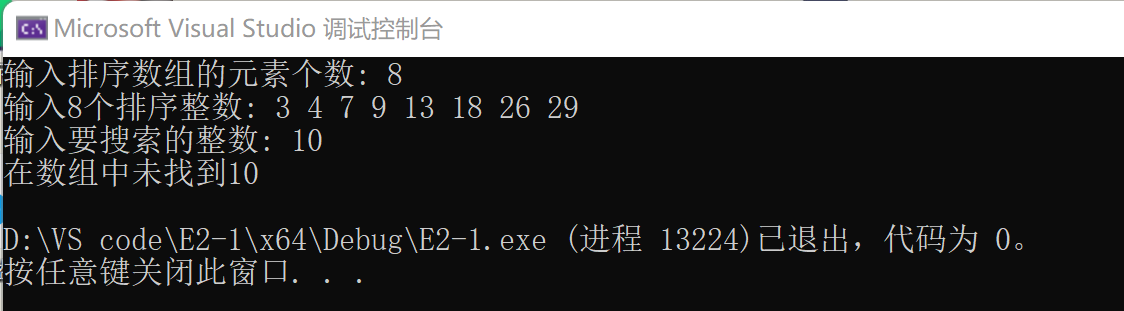
}

return 0;

}

**2.运行结果截图**





**E2-2**

**1.代码**

#include <iostream>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

ListNode(int x) : val(x), next(NULL) {}

};

// 尾插法创建链表

ListNode\* createListTail() {

ListNode\* head = new ListNode(0);

ListNode\* tail = head;

int num;

while (cin >> num && num != -1) {

ListNode\* newNode = new ListNode(num);

tail->next = newNode;

tail = newNode;

}

tail->next = NULL;

return head;

}

// 输出链表

void printList(ListNode\* head) {

ListNode\* p = head->next;

while (p) {

cout << p->val << " ";

p = p->next;

}

cout << endl;

}

// 合并两个链表

ListNode\* mergeLists(ListNode\* list1, ListNode\* list2) {

ListNode\* p1 = list1->next;

ListNode\* p2 = list2->next;

ListNode\* dummy = new ListNode(0);

ListNode\* tail = dummy;

while (p1 && p2) {

if (p1->val <= p2->val) {

tail->next = p1;

p1 = p1->next;

}

else {

tail->next = p2;

p2 = p2->next;

}

tail = tail->next;

}

// 处理剩余节点

if (p1) {

tail->next = p1;

}

else if (p2) {

tail->next = p2;

}

list1->next = dummy->next;

delete dummy;

return list1;

}

int main() {

cout << "请输入链表A的元素（输入-1表示结束）：" << endl;

ListNode\* listA = createListTail();

cout << "请输入链表B的元素（输入-1表示结束）：" << endl;

ListNode\* listB = createListTail();

cout << "链表A和链表B合并后的结果为：" << endl;

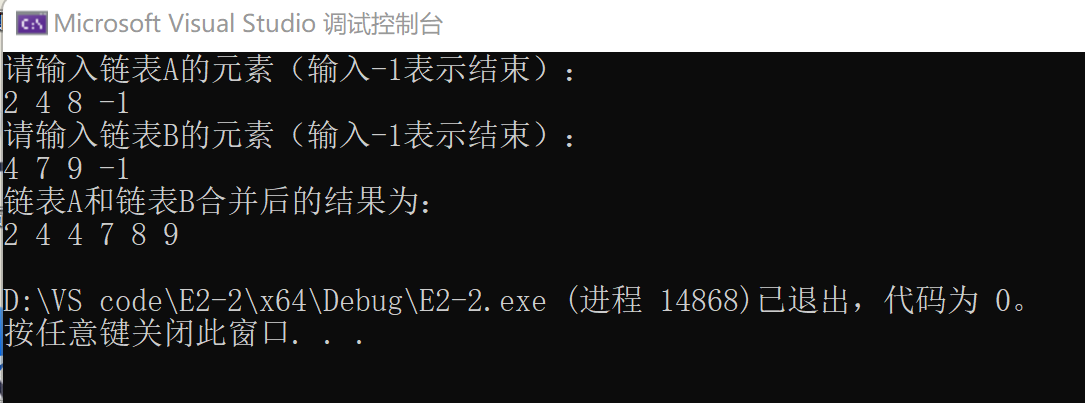
ListNode\* listC = mergeLists(listA, listB);

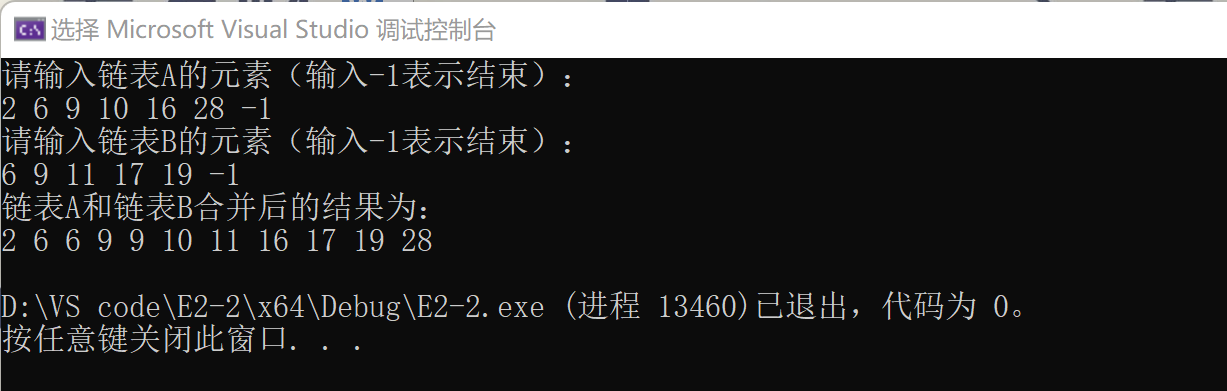
printList(listC);

return 0;

}

**2.运行结果截图**





**E2-3**

**E2-3.1** O(n2）

**1.代码**

#include <iostream>

#include <vector>

using namespace std;

pair<int, int> Sum(vector<int>& arr, int target) {

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

for (int j = i + 1; j < arr.size(); j++) {

if (arr[i] + arr[j] == target) {

return make\_pair(arr[i], arr[j]);

}

}

}

return make\_pair(-1, -1); // 没有找到和为T的两个整数，输出-1 -1

}

int main() {

vector<int> arr;

int n, target, element;

cout << "输入排序数组中元素的个数: ";

cin >> n;

cout << "输入排序后的数组元素: ";

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

cin >> element;

arr.push\_back(element);

}

cout << "T: ";

cin >> target;

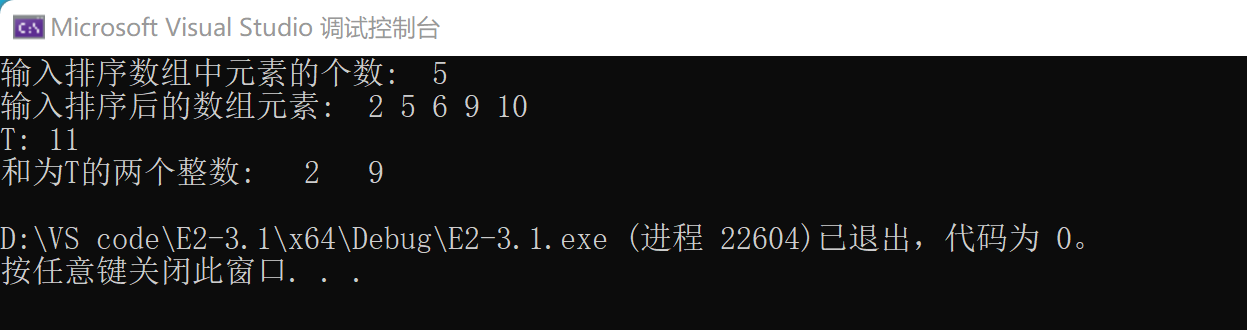
pair<int, int> result = Sum(arr, target);

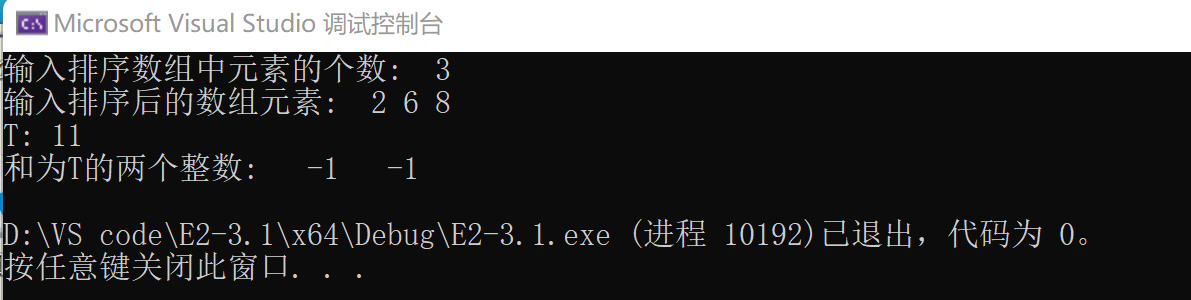
cout << "和为T的两个整数: " << result.first << " " << result.second << endl;

return 0;

}

**2.运行结果截图**





**E2-3.2** O(nlog(n)）

**1.代码**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

pair<int, int> Sum(vector<int>& arr, int target) {

sort(arr.begin(), arr.end());

int left = 0, right = arr.size() - 1;

while (left < right) {

int sum = arr[left] + arr[right];

if (sum == target) {

return make\_pair(arr[left], arr[right]);

}

else if (sum < target) {

left++;

}

else {

right--;

}

}

return make\_pair(-1, -1); // 没有找到和为T的两个整数，输出-1 -1

}

int main() {

vector<int> arr;

int n, target, element;

cout << "输入排序数组中元素的个数: ";

cin >> n;

cout << "输入排序后的数组元素: ";

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

cin >> element;

arr.push\_back(element);

}

cout << " T: ";

cin >> target;

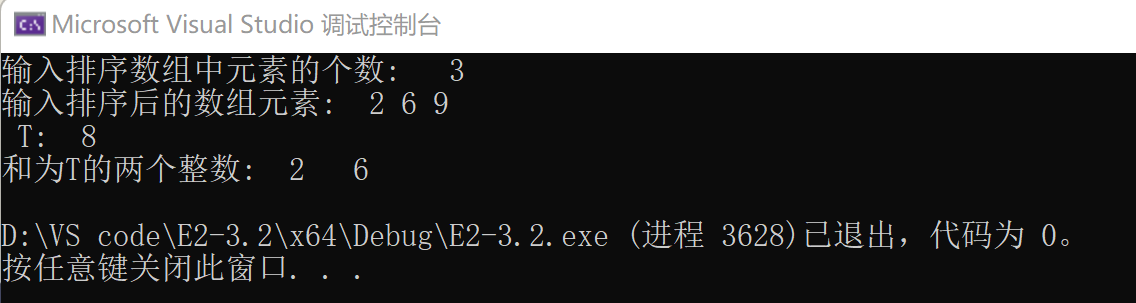
pair<int, int> result = Sum(arr, target);

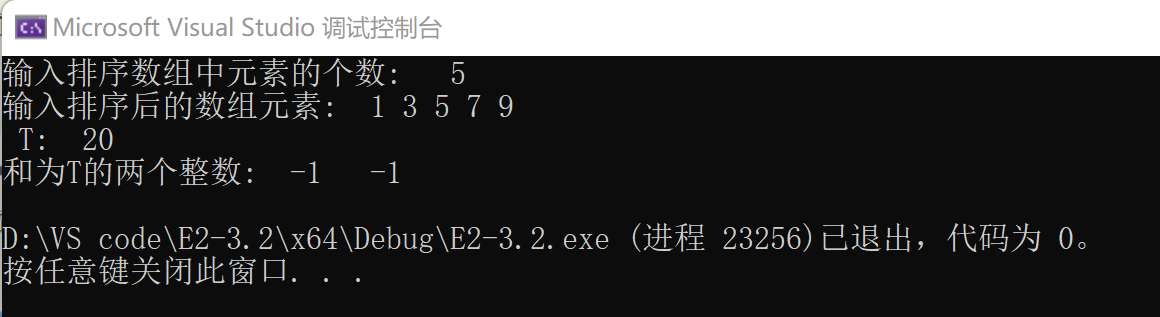
cout << "和为T的两个整数: " << result.first << " " << result.second << endl;

return 0;

}

**2.运行结果截图**





**E2-3.3** O(n）

**1.代码**

#include <iostream>

#include <vector>

#include <unordered\_set>

using namespace std;

pair<int, int> Sum(vector<int>& arr, int target) {

unordered\_set<int> complements;

for (int num : arr) {

int complement = target - num;

if (complements.count(complement)) {

return make\_pair(complement, num);

}

complements.insert(num);

}

return make\_pair(-1, -1); // 没有找到和为T的两个整数，输出-1 -1

}

int main() {

vector<int> arr;

int n, target, element;

cout << "输入排序数组中元素的个数: ";

cin >> n;

cout << "输入排序后的数组元素: ";

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

cin >> element;

arr.push\_back(element);

}

cout << "T: ";

cin >> target;

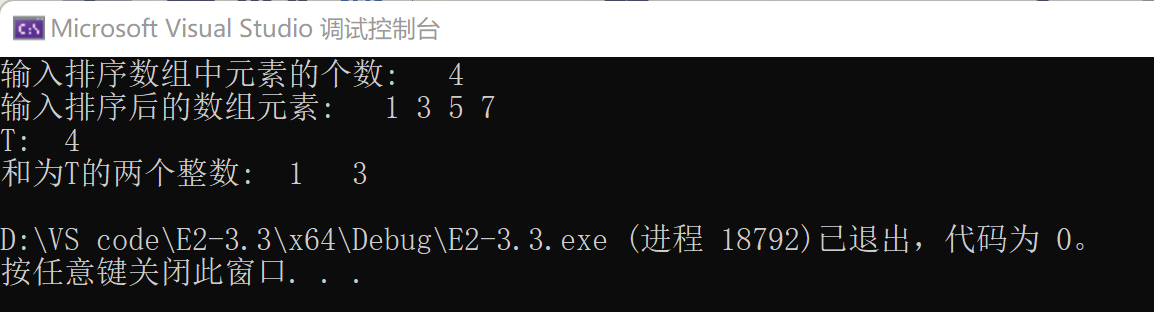
pair<int, int> result = Sum(arr, target);

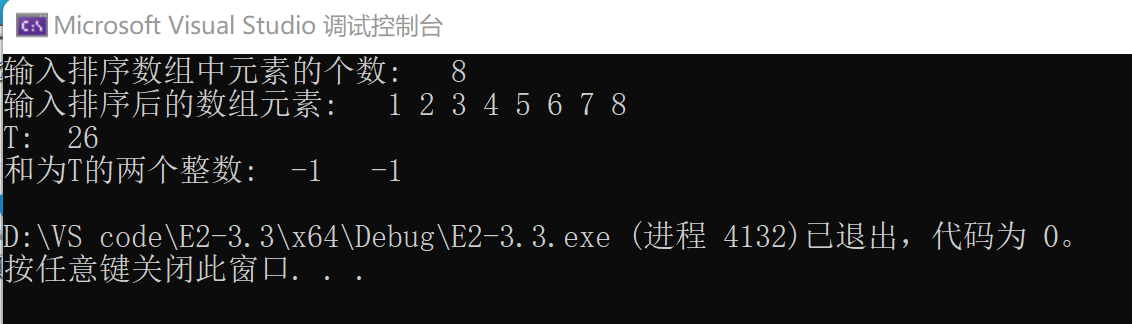
cout << "和为T的两个整数: " << result.first << " " << result.second << endl;

return 0;

}

**2.运行结果截图**



**E2-4**

**1.代码**

#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>

using namespace std;

struct Point {

int x;

int y;

};

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

return a.x < b.x;

}

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

return a.y < b.y;

}

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));

}

pair<Point, Point> Force(vector<Point>& points) {

int n = points.size();

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

pair<Point, Point> closestPair;

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

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

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

if (dist < minDist) {

minDist = dist;

closestPair = make\_pair(points[i], points[j]);

}

}

}

return closestPair;

}

pair<Point, Point> recursiveClosestPair(vector<Point>& points, int left, int right);

pair<Point, Point> ClosestPair(vector<Point>& points);

pair<Point, Point> recursiveClosestPair(vector<Point>& points, int left, int right) {

if (right - left <= 3) {

return Force(points);

}

int mid = (left + right) / 2;

auto leftPair = recursiveClosestPair(points, left, mid);

auto rightPair = recursiveClosestPair(points, mid + 1, right);

double leftDist = distance(leftPair.first, leftPair.second);

double rightDist = distance(rightPair.first, rightPair.second);

double minDist = min(leftDist, rightDist);

pair<Point, Point> closestPair;

if (leftDist < rightDist) {

closestPair = leftPair;

}

else {

closestPair = rightPair;

}

vector<Point> strip;

for (int i = left; i <= right; i++) {

if (abs(points[i].x - points[mid].x) < minDist) {

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

}

}

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

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

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

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

if (dist < minDist) {

minDist = dist;

closestPair = make\_pair(strip[i], strip[j]);

}

}

}

return closestPair;

}

pair<Point, Point> ClosestPair(vector<Point>& points) {

sort(points.begin(), points.end(), sortByX);

return recursiveClosestPair(points, 0, points.size() - 1);

}

int main() {

int n;

cout << "输入点对数: ";

cin >> n;

vector<Point> points(n);

cout << "输入点对(x y): " << endl;

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

cin >> points[i].x >> points[i].y;

}

pair<Point, Point> closestPair = ClosestPair(points);

cout << "最近的点对: (" << closestPair.first.x << ", " << closestPair.first.y << ")和("

<< closestPair.second.x << ", " << closestPair.second.y << ")" << endl;

return 0;

}

**2.运行结果截图**

