重庆师范大学

实验报告

实验课程名称	算法设计与分析
实验序号	4
实验内容	数组逆序对计数与排序
班级	23 计科 6 班
姓名	周子依
学号	2023051603162

2024 年 3 月 20 日

实验目的与要求

1. 学习目标

1) 掌握逆序对计数的核心概念，明晰逆序对在数组中的定义与判定准则，能够准确识别给定数组中的逆序对。
2) 熟练运用合并排序算法思想实现逆序对计数功能，深入理解排序过程中元素移动与逆序对数量统计之间的内在联系，完成对给定数组的逆序对计数操作。

13	**8**	**10**	**6**	**15**	**18**	**12**	**20**	**9**	**14**	**17**	**19**

实验内容

1. 算法原理与设计思路

（1）学习伪代码

逆序对计数通过两个函数 CountInver 和 MergeCount 来实现：

CountInver 函数

功能：用于计算数组中逆序对的数量。

流程：

首先判断 left 是否大于等于 right，如果是，说明子数组只有一个元素或为空，直接返回数组 A[left...right]。

递归计算左子数组的逆序对数量 S1，即 S1 = CountInver(A, left, mid) 。

递归计算右子数组的逆序对数量 S2，即 S2 = CountInver(A, mid + 1, right) 。

调用 MergeCount 函数合并左右子数组，并计算跨越子数组的逆序对数量 S3，即 S3 = MergeCount(A, left, mid, right)。

总逆序对数量 S = S1 + S2 + S3 ，最后返回总逆序对数量 S 以及合并后的数组 A[left...right] 。

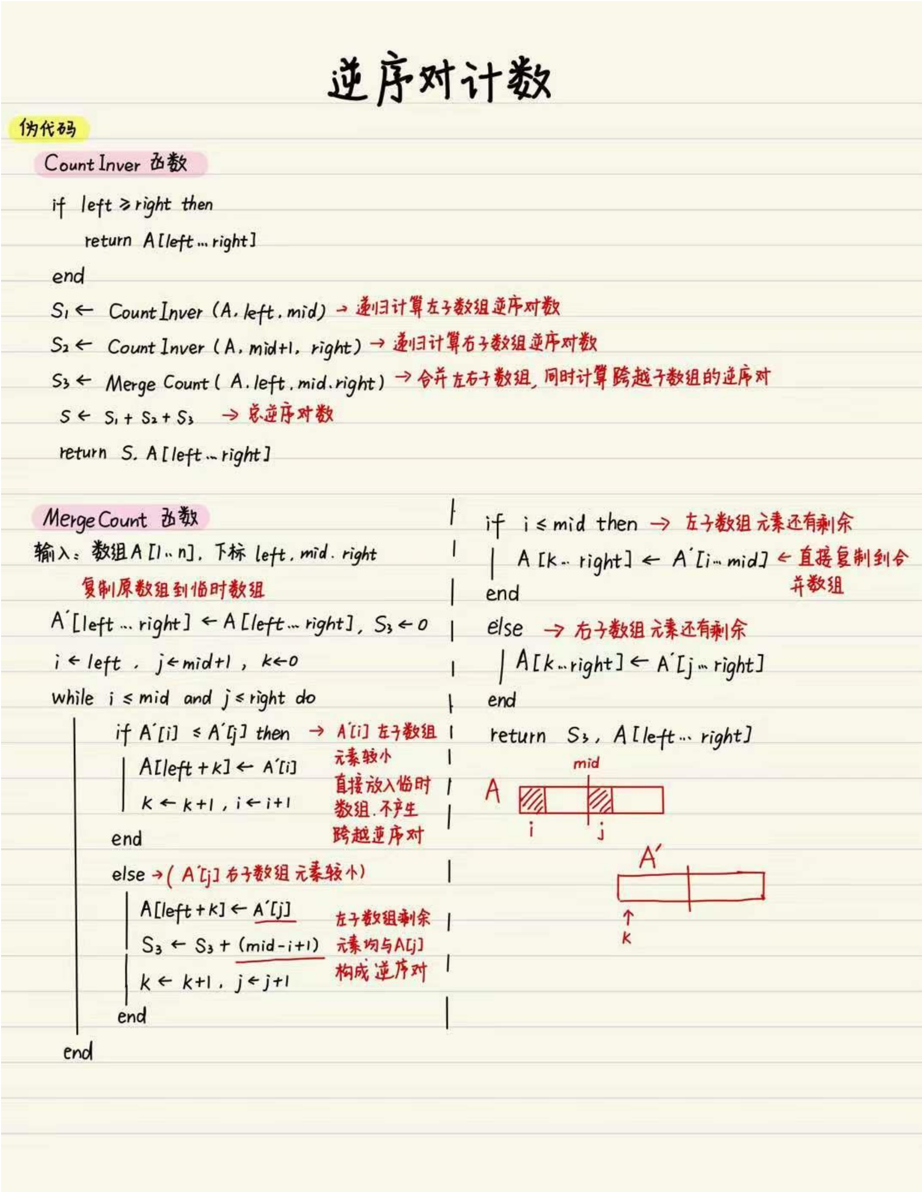
MergeCount 函数

功能：输入数组 A[1..n] 以及下标 left、mid、right ，负责合并左右子数组并计算跨越子数组的逆序对数量。

流程：

先将原数组 A[left...right] 复制到临时数组 A'[left...right]，并初始化跨越子数组的逆序对数量 S3 = 0 ，设置指针 i = left ，j = mid + 1 ，k = 0 。进入循环，当 i <= mid 且 j <= right 时：如果 A'[i] <= A'[j] ，说明左子数组元素较小，将 A'[i] 直接放入临时数组 A[left + k]，k 自增 1，i 自增 1 ，此过程不产生逆序对。 否则，即 A'[j] 较小，将 A'[j] 放入临时数组 A[left + k] ，此时左子数组剩余元素 A'[i...mid] 都与 A'[j]构成逆序对，所以 S3 = S3 + (mid - i + 1)，k 自增 1 ，j 自

增 1。最后返回跨越子数组的逆序对数量 S3 以及合并后的数组 A[left...right]。
图一为伪代码笔记。



逆序对计数

伪代码

CountInver 函数

```
if left ≥ right then
    return A[left ... right]
end
S1 ← CountInver (A, left, mid) → 递归计算左子数组逆序对数
S2 ← CountInver (A, mid+1, right) → 递归计算右子数组逆序对数
S3 ← MergeCount (A, left, mid, right) → 合并左右子数组，同时计算跨越子数组的逆序对
S ← S1 + S2 + S3 → 总逆序对数
return S, A[left ... right]
```

MergeCount 函数

输入：数组 A[1..n]，下标 left, mid, right

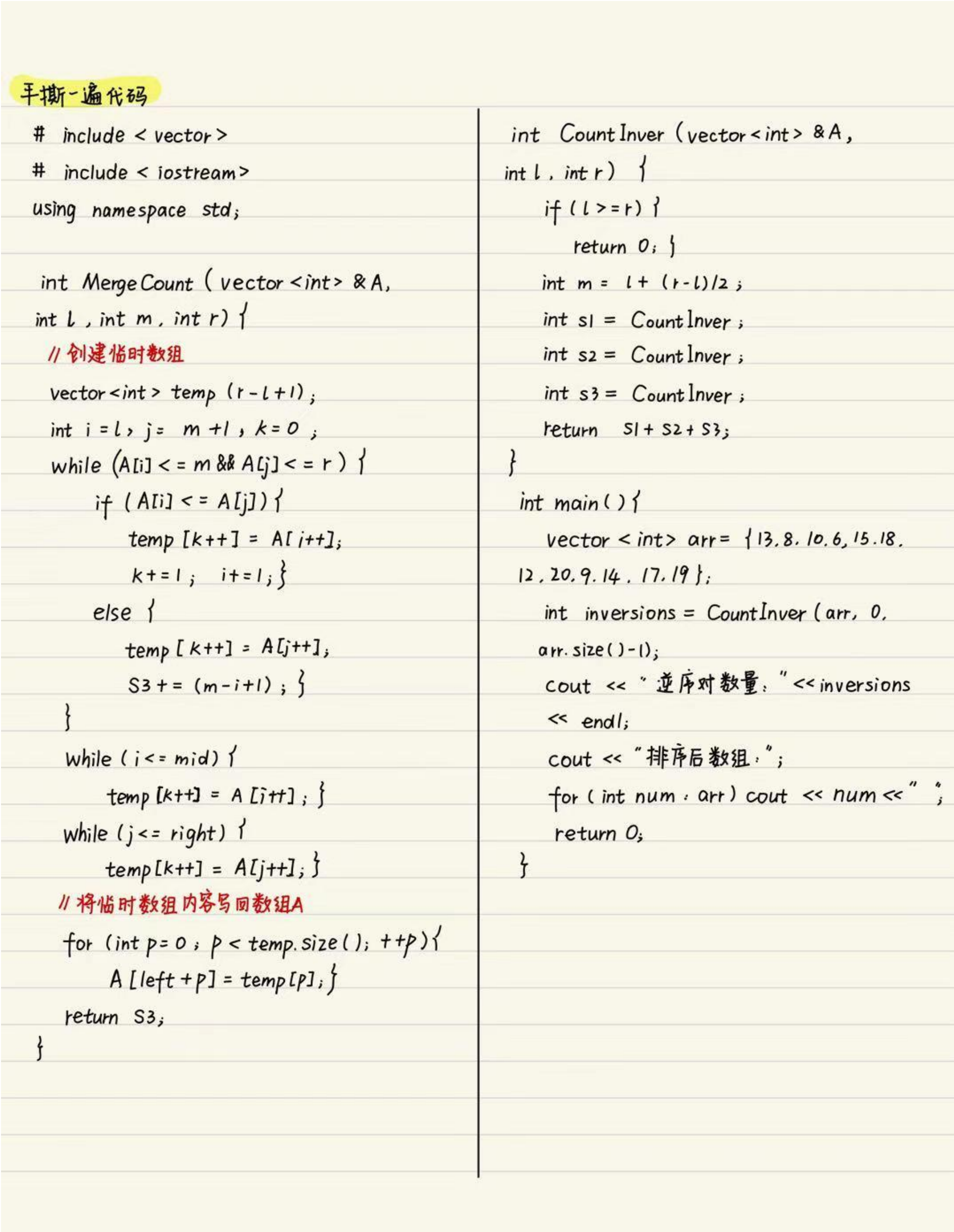
```
复制原数组到临时数组
A'[left ... right] ← A[left ... right], S3 ← 0
i ← left , j ← mid+1 , k ← 0
while i ≤ mid and j ≤ right do
    if A'[i] ≤ A'[j] then → A'[i] 左子数组元素较小直接放入临时数组，不产生跨越逆序对
        A[left+k] ← A'[i]
        k ← k+1 , i ← i+1
    end
    else → (A'[j] 右子数组元素较小)
        A[left+k] ← A'[j]
        S3 ← S3 + (mid-i+1)    左子数组剩余元素均与A[j]构成逆序对
        k ← k+1 , j ← j+1
    end
end
if i ≤ mid then → 左子数组元素还有剩余
    A[k .. right] ← A'[i .. mid] ← 直接复制到合并数组
end
else → 右子数组元素还有剩余
    A[k .. right] ← A'[j .. right]
end
return S3 , A[left ... right]
```


图 1 伪代码笔记

（2）实现代码

注意创建数组时使用 vector。

图 2 为手撕一遍代码。



手撕一遍代码

```
# include < vector >
# include < iostream>
using namespace std;

int MergeCount ( vector <int> &A,
int l , int m , int r) {
 // 创建临时数组
 vector<int > temp (r - l +1) ;
 int i = l, j = m +1 , k = 0 ;
 while (A[i] < = m && A[j] < = r ) {
     if ( A[i] < = A[j]) {
         temp [k++] = A[ i++];
         k+=1 ;  i+=1 ; }
     else {
         temp [ k++] = A[j++];
         S3 += (m-i+1) ; }
 }
 while ( i<= mid) {
     temp [k++] = A [i++] ; }
 while (j<= right) {
     temp[k++] = A[j++]; }
 // 将临时数组内容写回数组A
 for (int p= 0 ; p < temp.size(); ++p){
     A [left +p] = temp[p]; }
 return S3;
}
```

```
int CountInver (vector <int> &A,
int l , int r) {
   if ( l >= r) {
     return 0; }
   int m = l + (r-l)/2 ;
   int s1 = CountInver ;
   int s2 = CountInver ;
   int s3 = CountInver ;
   return s1 + s2 + s3;
}
int main ( ){
   vector < int> arr = {13, 8, 10, 6, 15, 18,
12, 20, 9, 14, 17, 19};
   int inversions = CountInver (arr, 0,
  arr.size()-1);
   cout << "逆序对数量: " << inversions
   << endl;
   cout << "排序后数组: ";
   for ( int num : arr) cout << num << " ";
   return 0;
}
```

图 2 手撕代码笔记

（3）复杂度分析

时间复杂度：算法基于归并排序的分治思想，将数组不断拆分为子数组。每次合并子数组时，需遍历元素完成逆序对计数与数组合并，时间复杂度为(O(n))。设总数据规模为 n，递归拆分的深度为(log n)，满足递推公式:

$$T(n) = O(1) \quad n < 1$$

$$T(n) = 2T(n/2) + O(n) \quad n >= 1$$

通过主定理推导可得整体时间复杂度为 (O(n log n))，相比插入排序的(O(n^2))，更适合大规模数据的逆序对计数。

空间复杂度：合并过程中使用临时数组 temp 存储当前处理区间的元素，空间大小与输入数组规模相关，为(O(n))。该空间用于辅助合并操作，确保原数

组有序更新，虽占用额外空间，但换取了更优的时间复杂度。

实验代码

附在最后。

实验结果

总逆序对数为 20。

stdout

```
逆序对数量: 20
排序后的数组: 6 8 9 10 12 13 14 15 17 18 19 20
```

总结与体会

1. 实验总结

通过本次实验，深入掌握了逆序对的核心概念，能够精准判定数组中的逆序对。基于插入排序思想实现逆序对计数时，深刻理解了元素移动与逆序对统计的关联：插入排序中元素移动次数直接对应逆序对数量，例如对小规模数组计数时，每一次元素移位都隐含着逆序对的统计。同时，结合伪代码分析归并排序实现逆序对计数的流程，明晰了分治思想下“子数组逆序对统计”与“跨子数组逆序对统计”的逻辑，从算法执行流程中体会到不同规模数据下算法性能的差异，如插入排序在小规模数据中实现直观，但时间复杂度限制其处理大规模数据的效率。

2. 踩坑与解决

（1） 逻辑理解偏差：

初期对伪代码中 MergeCount 函数的跨子数组逆序对统计逻辑理解模糊，尤其在计算 S3 = S3 + (mid - i + 1) 时，未清晰关联左子数组剩余元素与当前右子数组元素的逆序关系。通过对照实例推演，逐步明确每一步操作对应的逆序对生成逻辑，最终正确实现计数功能。

（2） 代码实现细节：

在复现算法代码时，因临时数组复制边界处理不当，导致合并数组结果错误。通过细化代码调试，严格检查数组下标范围，确保原数组与临时数组复制操作的准确性，解决了结果偏差问题。

3. 学习收获

深化了对算法分治思想的应用理解，归并排序实现逆序对计数的过程，展现

了分治策略在复杂问题拆解（子问题求解、合并）中的高效性。

认识到逆序对计数在算法分析中的意义，其不仅是一个计数问题，更能反映数组元素的有序程度，为后续学习排序算法性能分析（如排序算法的最优、最坏情况与逆序对关联）奠定了实践基础。

通过伪代码分析与代码实现的结合，提升了算法逻辑转化为代码的能力，对“算法设计—伪代码描述—代码实现”的完整流程有了更系统的认知。

附完整 C++代码

```
#include <vector>
#include <iostream>
using namespace std;

// 合并数组并计算跨越区间的逆序对
// 输入：数组 A，左边界 left，中间位置 mid，右边界 right
// 输出：跨越左右子数组的逆序对数量
int MergeCount(vector<int>& A, int left, int mid, int right) {
    // 创建临时数组，存储当前处理区间的元素，用于合并操作
    vector<int> temp(right - left + 1);
    int i = left;            // 左子数组（[left, mid]）的遍历指针
    int j = mid + 1;         // 右子数组（[mid+1, right]）的遍历指针
    int k = 0;               // 临时数组 temp 的写入指针
    int s3 = 0;       // 存储跨越左右子数组的逆序对数量

    // 合并左右子数组，直到其中一个子数组遍历完毕
    while (i <= mid && j <= right) {
        if (A[i] <= A[j]) {
            // 左子数组当前元素较小，直接放入临时数组，不产生跨越逆序对
            temp[k++] = A[i++];
        } else {
            // 右子数组当前元素较小，此时左子数组剩余元素（mid - i + 1 个）
            // 都与 A[j]构成逆序对（因为左子数组剩余元素都比 A[j]大）
            temp[k++] = A[j++];
            s3 += (mid - i + 1); // 累加跨越逆序对数量
        }
    }

    // 处理左子数组剩余元素（若有）
    while (i <= mid) {
        temp[k++] = A[i++];
    }
    // 处理右子数组剩余元素（若有）
    while (j <= right) {
        temp[k++] = A[j++];
```

```
    }

    // 将临时数组的内容写回原数组 A，保证 A[left..right]有序
    for (int p = 0; p < temp.size(); ++p) {
        A[left + p] = temp[p];
    }
    return s3; // 返回跨越逆序对数量
}

// 递归计算数组区间[left, right]的逆序对总数
// 输入：数组 A，左边界 left，右边界 right
// 输出：区间[left, right]的逆序对总数
int CountInver(vector<int>& A, int left, int right) {
    // 递归终止条件：区间只有一个元素或没有元素，逆序对为 0
    if (left >= right) {
        return 0;
    }
    int mid = left + (right - left) / 2; // 计算中间位置，避免 (left+right)/2 溢出

    // 递归计算左子数组[left, mid]的逆序对
    int s1 = CountInver(A, left, mid);
    // 递归计算右子数组[mid+1, right]的逆序对
    int s2 = CountInver(A, mid + 1, right);
    // 合并左右子数组并计算跨越逆序对
    int s3 = MergeCount(A, left, mid, right);

    // 总逆序对 = 左子数组逆序对 + 右子数组逆序对 + 跨越逆序对
    return s1 + s2 + s3;
}

int main() {
    // 测试用例，使用图片中的数组
    vector<int> arr = {13, 8, 10, 6, 15, 18, 12, 20, 9, 14, 17, 19};
    // 计算逆序对数量，初始调用时处理整个数组区间[0, arr.size()-1]
    int  inversions = CountInver(arr, 0, arr.size() - 1);

    // 输出逆序对数量
    cout << "逆序对数量: " << inversions << endl;
    // 输出排序后的数组（算法执行过程中会对数组排序）
    cout << "排序后的数组: ";
    for (int num : arr) {
        cout << num << " ";
    }
```

```
	return 0;
}
```