中山大学计算机学院本科生实验报告

（2025学年第1学期）

年级	2024级	专业（方向）	计算机科学与技术（人工智能与大数据）
学号	24325155	姓名	梁桂铭
电话	15817681625	Email	lianggm8@mail2.sysu.edu.cn
开始日期	2025年10月20日	完成日期	2025年10月20日

第一题

实验题目

明明想在学校中请一些同学一起做一项问卷调查，为了实验的客观性，他先用计算机生成了N个1到1000之间的随机整数（N≤100），对于其中重复的数字，只保留一个，把其余相同的数去掉，不同的数对应着不同的学生的学号。然后再把这些数从小到大排序，按照排好的顺序去找同学做调查。请你协助明明完成“去重”与“排序”的工作。

实验目的

1. 掌握使用**哈希表**进行数据去重的方法
2. 理解并实现简单的**桶排序**算法

算法设计

哈希去重法

1. 使用一个大小为1001的数组 hash 作为哈希表，将其中所有数据初始化为0
2. 对于每个输入的数字 tmp ，执行 hash[tmp]++ 进行计数
3. 遍历哈希表，当 hash[i] > 0 时，说明该数字存在，将其加入 nums 中

时间复杂度分析

- 输入阶段：$O(N)$
- 去重阶段：$O(1001) \approx O(1)$
- 总体时间复杂度：$O(N)$

空间复杂度

- 哈希表空间：$O(1001)$
- 结果向量：最坏情况$O(N)$
- 总体空间复杂度：$O(N)$

程序运行与测试

```
#include  <iostream>
#include  <vector>
using  namespace  std;
vector<int> nums;
int  main () {
    int N, tmp, count = 0;
    int  hash[1001] = {0};
    cin >> N;
    for (int i = 0; i < N; i++) {
        cin >> tmp;
        hash[tmp]++;
    }
    for (int i = 0; i <= 1000; i++) {
        if(hash[i] > 0) {
        count++;
        nums.push_back(i);
        }
    }
    cout << count << endl;
```

```
    for (int i = 0; i < count; i++) {
        cout << nums[i] << ' ';
    }
    return  0;
}
```

运行结果

输入样例：

```
10
20 40 32 67 40 20 89 300 400 15
3
3 2 1
```

输出样例：

```
8
15 20 32 40 67 89 300 400
3
1 2 3
```

实验总结

1. 通过本次实验，掌握了使用**哈希表**进行高效去重的方法
2. 理解了**桶排序**在特定场景下的优势
3. 实验过程中验证了算法设计的正确性和高效性，达到了预期目标

第二题

实验题目

Let A(1), …, A(n) be a sequence of n numbers. If i<j and A(i)>A(j), then the pair (i,j) is called an inversion pair.

The inversion number of a sequence is one common measure of its sortedness. Given the sequence A, calculate its inversion number.

实验目的

1. 理解**逆序数**的概念及其计算方法
2. 掌握**分治算法**在逆序数计算中的应用
3. 实现高效的**归并排序**算法来计算逆序数

算法设计

算法思路

1. **分治策略**：采用归并排序的思想，将序列不断**二分**
2. **合并计数**：在合并两个有序子序列时统计逆序数
    - 当右子序列元素**小于**左子序列元素时，产生逆序
    - 逆序数增加量为**左子序列剩余元素个数**
3. **递归处理**：对**左右子序列**分别递归计算逆序数

数学表达

设序列为A，其逆序数$Inv(A)$可表示为：

$$Inv(A) = Inv(A_{left}) + Inv(A_{right}) + Inv(A_{left}, A_{right})$$

其中$Inv(A_{left}, A_{right})$表示左右子序列间的逆序对数

时间复杂度分析

- 分治过程：$O(\log n)$
- 合并过程：$O(n)$
- 总体时间复杂度：$O(n \log n)$

空间复杂度分析

- 临时数组：$O(n)$
- 递归栈：$O(\log n)$
- 总体空间复杂度：$O(n)$

程序运行与测试

程序代码

```
#include  <iostream>
#include  <vector>
using  namespace  std;
long  long  int inverse_num = 0;
vector<int> nums;
void  inverse_numbers (int  left, int  right) {
    if (right - left <= 1) return;    // 递归的终止条件
    int mid = (left + right) / 2;
    vector<int> result;
    inverse_numbers(left, mid);
    inverse_numbers(mid, right);
    int x = left, y = mid;
    while (x < mid || y < right) {
        if (y >= right || (x < mid && nums[x] <= nums[y])) {
            result.push_back(nums[x++]);
        }
        else {
            result.push_back(nums[y++]);
            inverse_num += mid - x;    // 注意增量
        }
    }
    for (int i = left; i < right; i++) {
        nums[i] = result[i - left];
    }
}
int  main () {
    int N = 0;
    while (cin >> N) {
        inverse_num = 0;
        int tmp;
        for (int i = 0; i < N; i++) {
        cin >> tmp;
        nums.push_back(tmp);
        }
```

```
            inverse_numbers(0, N);
            cout << inverse_num << endl;
            nums.clear();
        }
        return  0;
}
```

运行结果

输入样例：

```
5
3 1 4 5 2
```

输出样例：

```
4
```

实验总结

1. 通过本次实验，深入理解了逆序数的概念和计算方法
2. 掌握了利用**归并排序**的分治思想高效计算逆序数的技巧
3. 认识到分治算法在解决复杂问题时的优势：
    - 将$O(n^2)$的暴力解法优化为$O(n \log n)$
    - 适用于大规模数据计算
4. 需要注意逆序数可能超过32位整数范围，应使用64位整数存储结果

第三题

实验题目

Please use mergesort to sort a linked list of data.

The linked list is defined as follows.

```
struct linkedlist{
 int data;
 linkedlist *next;
 };
```

Please implement the following function.

```
//sort the list by mergesort and return the head of it
void  mergesort(linkedlist *&head, int len){
 //add your code here
}
```

实验目的

1. 掌握链表数据结构的特点和操作方法
2. 理解归并排序算法在链表上的实现原理
3. 实现链表的**分割和合并**操作
4. 培养对递归算法的理解和应用能力

算法设计

算法思路

1. **分割阶段**：
    - 找到链表中点，将链表分为左右两部分
    - 使用快慢指针法或直接遍历找到中点
    - 找到中点之后需要将该中点与后续链表断开
2. **递归排序**：
    - 对左右两部分分别递归调用归并排序
3. **合并阶段**：
    - 创建**哑节点**简化合并操作
    - 比较左右链表节点值，按升序连接

关键步骤

1. 中点定位：遍历前$n/2-1$个节点找到中点，而不是$n/2$
2. 链表分割：将**中点节点**的next指针置为NULL
3. 递归排序：分别对左右子链表排序
4. 有序合并：使用双指针法合并两个有序链表

时间复杂度分析

- 分割操作：$O(n)$
- 递归深度：$O(\log n)$
- 合并操作：$O(n)$
- 总体时间复杂度：$O(n \log n)$

空间复杂度分析

- 递归栈：$O(\log n)$
- 临时节点：$O(1)$
- 总体空间复杂度：$O(\log n)$

程序运行与测试

程序代码

```
#include  "mergeSort.h"
#include  <iostream>
using  namespace  std;
//sort the list by mergesort and return the head of it
linkedlist*  combine(linkedlist*  &l, linkedlist*  &r) {
    linkedlist* dummy = new linkedlist{0, NULL};
    linkedlist* cur = dummy;
    while (l != NULL || r != NULL) {
        if (l == NULL || (r && l -> data > r -> data)) {
            cur -> next = r;
            r = r -> next;
        }
        else {
            cur -> next = l;
            l = l -> next;
        }
```

```
        cur = cur -> next;
    }
    return  dummy -> next;
}
void  mergesort(linkedlist  *&head, int  len){
    if (len <= 1) return;
    linkedlist* mid = head;
    for (int i = 0; i < len/2 - 1; i++) {   // 需要注意是len/2 - 1，而不是len,
        mid = mid -> next;
    }
    linkedlist* right = mid -> next;
    mid -> next = NULL;
    mergesort(head, len/2);
    mergesort(right, len - len/2);        // 注意不能直接用len/2，要考虑int的向
    head = combine(head, right);
}
```

运行示例

输入链表：

```
1 -> 3 -> 2 -> 4 -> NULL
```

输出样例：

```
1 -> 2 -> 3 -> 4 -> NULL
```

实验总结

1. 通过本次实验，掌握了链表归并排序的实现方法
2. 理解了递归分治思想在链表排序中的应用
3. 实现了链表的分割和合并操作，加深了对指针操作的理解
4. 认识到链表排序与数组排序的主要区别：
    - 链表不支持随机访问，需要特殊方法找中点
    - 合并操作需要仔细处理指针连接

附录、提交文件清单

第一题

```
#include  <iostream>
#include  <vector>
using  namespace  std;
vector<int> nums;
int  main () {
    int N, tmp, count = 0;
    int  hash[1001] = {0};
    cin >> N;
    for (int i = 0; i < N; i++) {
        cin >> tmp;
        hash[tmp]++;
    }
    for (int i = 0; i <= 1000; i++) {
        if(hash[i] > 0) {
        count++;
        nums.push_back(i);
        }
    }
    cout << count << endl;
    for (int i = 0; i < count; i++) {
        cout << nums[i] << ' ';
    }
    return  0;
}
```

第二题

```
#include  <iostream>
#include  <vector>
using  namespace  std;
long  long  int inverse_num = 0;
vector<int> nums;
void  inverse_numbers (int  left, int  right) {
    if (right - left <= 1) return;
    int mid = (left + right) / 2;
    vector<int> result;
    inverse_numbers(left, mid);
```

```
    inverse_numbers(mid, right);
    int x = left, y = mid;
    while (x < mid || y < right) {
        if (y >= right || (x < mid && nums[x] <= nums[y])) {
            result.push_back(nums[x++]);
        }
        else {
            result.push_back(nums[y++]);
            inverse_num += mid - x;
        }
    }
    for (int i = left; i < right; i++) {
        nums[i] = result[i - left];
    }
}
int  main () {
    int N = 0;
    while (cin >> N) {
        inverse_num = 0;
        int tmp;
        for (int i = 0; i < N; i++) {
        cin >> tmp;
        nums.push_back(tmp);
        }
        inverse_numbers(0, N);
        cout << inverse_num << endl;
        nums.clear();
    }
    return  0;
}
```

第三题

```
#include  "mergeSort.h"
#include  <iostream>
using  namespace  std;
//sort the list by mergesort and return the head of it
linkedlist*  combine(linkedlist*  &l, linkedlist*  &r) {
    linkedlist* dummy = new linkedlist{0, NULL};
    linkedlist* cur = dummy;
    while (l != NULL || r != NULL) {
```

```
            if (l == NULL || (r && l -> data > r -> data)) {
                cur -> next = r;
                r = r -> next;
            }
            else {
                cur -> next = l;
                l = l -> next;
            }
            cur = cur -> next;
        }
        return  dummy -> next;
    }
    void  mergesort(linkedlist  *&head, int  len){
        if (len <= 1) return;
        linkedlist* mid = head;
        for (int i = 0; i < len/2 - 1; i++) {   // 需要注意是len/2 - 1，而不是len,
            mid = mid -> next;
        }
        linkedlist* right = mid -> next;
        mid -> next = NULL;
        mergesort(head, len/2);
        mergesort(right, len - len/2);        // 注意不能直接用len/2，要考虑int的向
        head = combine(head, right);
    }
```