中山大学计算机学院本科生实验报告

(2025 学年第 1 学期)

课程名称：数据结构与算法　　实验任课教师：张子臻

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	242325157	姓名	梁玮麟
电话	18620062182	Email	3371676041@qq.com
开始日期	2025.10.15	结束日期	2025.10.19

第一题

1、实验题目

z6-明明的随机数

题目描述　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。

输入描述　输入包含多个测试数据。每个测试数据有 2 行，第 1 行为 1 个正整数，表示所生成的随机数的个数 N，第 2 行有 N 个用空格隔开的正整数，为所产生的随机数。

输出描述　对每个测试数据输出 2 行。第 1 行为 1 个正整数 M，表示不相同的随机数的个数。第 2 行为 M 个用空格隔开的正整数，为从小到大排好序的不相同的随机数。

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
```

2、实验目的

- 掌握“去重 + 排序”的两种常见实现：基于有序集合与基于计数桶。
- 体会多组数据的读写与输出格式控制（先数量、后有序序列）。

3、算法设计

设计思路如下：

- **整体思路：**输入一组整数，完成去重并按升序输出。
- **方案 A：有序集合** `set<int>`——逐个插入，容器自动去重且保持有序；最终输出 `size()` 和遍历元素。

  - 复杂度：插入均摊 O(log K)，整体 O(N log K)；空间 O(K)（K 为不重复元素个数）。
- **方案 B：计数桶/位图**——题目给定取值区间 [1,1000]，使用长度为 1001 的标记数组；读到 x 就置 bucket[x]=1，最后顺序扫描输出。
  - 复杂度：O(N+U)（U 为范围 1000），常数小；空间 O(U)。

细节注意：

- 我首先想到的是直接使用 set，但是使用桶排序的效率更高。
- 多组输入需在每组开始时清空容器或重置桶。
- 输出格式严格：先数量、换行，再空格分隔的升序序列（末尾空格不敏感）。

具体实现：

```
//z5-方程求解
//version1
#include <iostream>
#include <set>
using namespace std;
int main() {
    int N;
    while (cin >> N) {
        set<int> s;
        for (int i = 0; i < N; i++) {
            int m;
            cin >> m;
            s.insert(m);
        }
        cout << s.size()<<endl;
        for (auto val : s) { cout << val << " "; }
    }
    return 0;
}
```

```
//z5-方程求解
//verison2
#include<iostream>
using namespace std;
int main(){
    int n;
    while(cin>>n){
        int bucket[1001]={0};
        int sum=0;
        for(int i=0;i<n;i++){
            int m;
            cin>>m;
            if(bucket[m]!=1){
                sum++;
                bucket[m]=1;
            }
        }
        cout<<sum<<endl;
        for(int i=1;i<=1000;i++){
            if(bucket[i]==1) cout<<i<<" ";
        }
    }
```

```
    return 0;
}
```

4、程序运行与测试

运行结果：

数据点一

- 标准输入：

```
70
302 457 215 315 437 447 231 189 301 407 89 306 407 515 40 321 120 40 321 617 627 637 679 67 430 230 8 3
```

- 实际输出：

```
39
8 15 20 40 67 89 120 125 189 215 230 231 240 241 301 302 303 306 315 321 327 402 407 408 410 430 437 44
```

- 期望输出：

```
39
8 15 20 40 67 89 120 125 189 215 230 231 240 241 301 302 303 306 315 321 327 402 407 408 410 430 437 44
```

数据点二

- 标准输入：

```
90
7 2 315 437 447 231 189 301 407 8 407 84 341 408 525 41 322 220 40 321 302 457 215 315 437 447 231 189 3
```

- 实际输出：

```
48
2 5 7 8 15 20 40 41 67 84 89 120 185 189 201 215 220 231 240 241 301 302 315 321 322 341 407 408 409 41
```

- 期望输出：

```
48
2 5 7 8 15 20 40 41 67 84 89 120 185 189 201 215 220 231 240 241 301 302 315 321 322 341 407 408 409 41
```

5、实验总结与心得

- 范围小且已知时，桶法最简单高效；范围未知或很大时，`set` 更稳妥也更通用。
- 易错点在于多组输入的状态重置与输出格式（尤其是数量与序列的分隔换行）。

第二题

1、实验题目

z6-Mergesort for List

题目描述 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.

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

提示 1->3->2->4->NULL 其中 1 是 head

2、实验目的

- 理解归并排序在**单链表**上的落地实现：只用指针完成“分 + 合”。
- 掌握哑结点（dummy head）与子段**断开/回接**的边界处理。

3、算法设计

设计思路如下：

- **总体结构：** `mergesort(head, len)` → 递归 `sort(head, left, right)` → 每次将 `[left, right]` 拆为 `[left, mid]` 与 `[mid+1, right]`，分别排序后调用 `merge()` 归并。
- `merge()` **关键步骤（就地归并）：**
  1. 建哑结点 `top` 令其指向 `head`，便于头结点变化时统一处理。
  2. 线性定位：`begin`（`left` 的前驱）、`lptr`（左段头）、`mptr`（mid）、`rptr`（右段头 = `mptr->next`）、`rightNode`（`right`）、`end`（`right` 的下一个）。
  3. **先截断**：`mptr->next=nullptr`、`rightNode->next=nullptr`，把左右两段变成独立小链表，避免环或误连接。
  4. **双指针归并**：谁小接谁到 `begin->next` 并推进对应指针；某一侧耗尽后把另一侧整段接上。
  5. **回接区间外**：把新段尾接回 `end`；更新 `head = top->next` 并释放哑结点。
- **复杂度：**每层线性归并，整体 `O(nlog n)`；除递归栈外为原地指针操作，额外空间 `O(1)`（不含栈）。

细节注意：

- 头节点 `head` 会在排序中改变，每次排序完之后需要重新将 `head` 节点设置成 `top->next`。
- 将左右两段用 `nullptr` 截断并非必要，但是必须要将**最后一个节点的** `next` 指向 `end`。
- 注意千万不要出现 `nullptr->next` 的情况!!!

具体实现：

```
//z6-Merge sort for list
#include<iostream>
#include"mergeSort.h"
using namespace std;
void sort(linkedlist *&head,int left, int right );
void merge(linkedlist*& head,int left, int mid, int right);
void mergesort(linkedlist *&head, int len){
    sort(head, 1,len);
}
void sort(linkedlist *&head,int left, int right ){
    if(right<=left) return ;
    else {
        int mid = (left+right)/2;
        sort(head,left,mid);
        sort(head,mid+1,right);
        merge(head,left,mid,right);
    }
```

```
}
void merge(linkedlist*& head,int left, int mid, int right){
    //
    linkedlist* top=new linkedlist;
    top->next=head;
    linkedlist* t=top;//用来找到对应节点
    linkedlist* end=nullptr;;//用来保存尾节点
    linkedlist* begin=top;//用来找到 Left 的上一个节点
    linkedlist* lptr=nullptr,*mptr=nullptr,*rptr=nullptr;
    for(int i=0;i<=right;i++){
        if(i==left-1) begin=t;
        if(i==left) lptr=t;
        if(i==mid){
            mptr=t;
            rptr=mptr->next;
            t=rptr;
            mptr->next=nullptr;//提前截断，避免出现环的情况
            continue;
        }
        if(i==right){
            end=t->next;
            t->next=nullptr;//同理
            continue;
        }
        t=t->next;
    }
    while(lptr&&rptr){
        if(lptr->data<=rptr->data){
            begin->next=lptr;
            begin=begin->next;
            lptr=lptr->next;
        }else{
            begin->next=rptr;
            begin=begin->next;
            rptr=rptr->next;
        }
    }
    while(lptr){
        begin->next=lptr;
        begin=begin->next;
        lptr=lptr->next;
    }
    while(rptr){
        begin->next=rptr;
        begin=begin->next;
        rptr=rptr->next;
    }
    begin->next=end;
    head=top->next;
    delete top;
}
```

4、程序运行与测试

运行结果：

- 标准输入：

```
500 7146 744 3638 6476 5655 3026 7007 4160 6294 4837 6054 7632 4152 9225 4375 1976 3179 3418 7657 4088
```

- 实际输出：

```
32 39 65 71 86 92 99 100 103 112 135 247 300 302 407 475 497 513 531 546 549 568 575 620 633 636 638 694
```

- 期望输出：

```
32 39 65 71 86 92 99 100 103 112 135 247 300 302 407 475 497 513 531 546 549 568 575 620 633 636 638 694
```

5、实验总结与心得

- 链表版归并的难点是**边界与顺序**：定位 → 截断 → 归并 → 回接，任何一步遗漏都可能成环或丢链。
- 哑结点显著简化了头部变化，无需特殊处理头节点。
- 严格区分 `left/mid/right` 与其前驱/后继能有效避免错误处理将链表变成环状。

第三题

1、实验题目

z6-Inversion Number

题目描述 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.

输入描述 There are multiple cases. Each case contains an integer `n (n<=100,000)` followed by `A(1) , ..., A(n)`.

输出描述 For each case, output the inversion number.

输入样例

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

输出样例

```
4
```

提示 The answer may exceed `2^31`.

2、实验目的

- 掌握“归并排序 + 计数”在线性对数时间内统计逆序对的方法。
- 理解辅助数组 `temp` 在归并时防止覆盖未处理数据的作用。

3、算法设计

设计思路如下：

- **分治思路：**

- `divide(left,right)`：若区间长度 $<= 1$ 返回 0；否则递归求解 `left..mid` 与 `mid+1..right` 的逆序对数，再在 `merge(left,mid,right)` 中统计**跨区间**的逆序对数。
- `merge` 归并时：当 `num[i] > num[j]`（右侧更小），说明左侧从 `i..mid` 的所有元素都大于 `num[j]`，一次性累加 `mid - i + 1`；无论哪边更小，都先写入 `temp`，最后再整体拷回 `num[left..right]`，保持两段有序不变式。
- **复杂度：**与归并排序相同，时间 `O(nlog n)`，空间 `O(n)`；计数使用 `long long` 以防溢出。

细节注意：

- 在 `divide()` 中，左右两段数据都已经存在 `num` 中，而且**已经排序**好。而 `temp` 仅仅作为一个缓存，记录正确的排序。在函数最后需要将 `num` 的对应部分**重新设置成**和 `temp` 一样的顺序。
- 传入的是 `temp` 的**引用**，避免过分开辟内存空间。

具体实现：

```
//z6-Inversion Number
#include<iostream>
#include<vector>
using namespace std;
typedef long long int ll;
ll merge(vector<ll > &num, ll left, ll mid, ll right,vector<ll>& temp){//
    //左右两部分已经排序好，先计算逆序对，在排序。
    ll sum=0;
    ll l=left, r=mid+1,p=left;
    while(l<=mid&&r<=right){
        if(num[r]<num[l]){
            sum+=mid+1-l;
            temp[p++]=num[r++];
        }
        else{
            temp[p++]=num[l++];
        }
    }
    while(l<=mid){
        temp[p++]=num[l++];
    }
    while(r<=right){
        temp[p++]=num[r++];
    }
    for(ll j=left;j<=right;j++ ){
        num[j]=temp[j];
    }
    return sum;
}
ll divide(vector<ll>& num, ll left, ll right, vector<ll >& temp){
    if(right>left){
        ll mid =(left+right)/2;
        ll sum=0;
        sum+=divide(num, left,mid,temp);
        sum+=divide(num, mid+1,right,temp);//算两部分的逆序对
        sum+=merge(num,left,mid,right,temp);//算跨越两部分的逆序对
        return sum;
    }
    else return 0;
```

```
}
ll inversion_number(vector<ll>& num){
    vector<ll> temp(num.size());
    return divide(num,0,num.size()-1,temp);

}
int main(){
    ll n;
    while(cin>>n){
        vector<ll> num(n);
        for(ll i=0;i<n;i++){
            cin>>num[i];
        }
        cout<<inversion_number(num)<<endl;
    }
    return 0;
}
```

4、程序运行与测试

运行结果：

数据点 1

- 标准输入：

```
50678
98207687 243481617 741938601 834390500 505305848 80055817 395643562 822153543 897241074 340915642 52465
```

- 实际输出：

```
644522972
```

- 期望输出：

```
644522972
```

数据点 2

- 标准输入：

```
20701
352125598 758780116 578280270 550441682 561673814 710436570 27713218 224459541 56725008 779955019 72520
```

- 实际输出：

```
107839546
```

- 期望输出：

```
107839546
```

5、实验总结与心得

- 关键体会是“**在归并时顺手计数**”与“**用 temp 隔离读写**”。
- 与 O(n^2) 暴力相比，归并计数在规模上万时仍能稳定通过；实现时注意边界与类型（long long）。