

**CENTRAL SOUTH UNIVERSITY**

**数据结构课程实验报告**

|  |  |
| --- | --- |
| 学生姓名 | 陈康坤 |
| 专业班级 | 计通2215班 |
| 学 号 | 8208221512 |
| 学 院 | 计算机学院 |
| 完成日期 | 2023年5月 |

目录

目录

[1、 实验目的 3](#_Toc14286)

[(1)线性表练习 3](#_Toc16380)

[(2)内部排序练习 3](#_Toc24532)

[2、 算法的思路及算法 4](#_Toc9625)

[(1)线性表练习 4](#_Toc14610)

[1、实现顺序表的初始化、在指定位置插入和删除元素 4](#_Toc4670)

[（2）实现单链表的初始化、在指定位置插入和删除元素 5](#_Toc17952)

[（3） 给定的2个有序线性链表的合并（合并到1个新的链表中） 7](#_Toc15802)

[(4) 实现双向链表的初始化、在指定位置插入和删除元素 10](#_Toc26845)

[(2)内部排序练习 11](#_Toc3725)

[1、实现直接插入排序、2-路插入排序和希尔排序； 11](#_Toc20836)

[2、实现起泡排序、双向起泡排序和快速排序； 14](#_Toc12699)

[3、实现简单选择排序和堆排序； 15](#_Toc2415)

[4、实现归并排序。 17](#_Toc119)

[3、 程序流程图 18](#_Toc10244)

[4、 实验结果及分析 28](#_Toc13026)

[注：程序放在附录中 31](#_Toc22372)

[顺序表 31](#_Toc32124)

[链表 34](#_Toc26511)

[双向链表 37](#_Toc3762)

[合并链表到新的链表 40](#_Toc14819)

[合并链表到第一个链表 44](#_Toc9330)

[排序 48](#_Toc30356)

[排序2 51](#_Toc2093)

# 实验目的

## (1)线性表练习

1、实现顺序表的初始化、在指定位置插入和删除元素；

2、实现单链表的初始化、在指定位置插入和删除元素；

3、给定的2个有序线性链表的合并（合并到1个新的链表中以及合并到其中的1个链表中两种方式）；

4、实现双向链表的初始化、在指定位置插入和删除元素。

## (2)内部排序练习

1、实现直接插入排序、2-路插入排序和希尔排序；

2、实现起泡排序、双向起泡排序和快速排序；

3、实现简单选择排序和堆排序；

4、实现归并排序。

# 算法的思路及算法

## (1)线性表练习

1、实现顺序表的初始化、在指定位置插入和删除元素；

定义：顺序表是用一段物理地址连续的存储单元依次存储数据元素的线性结构，一般情况下采用数组存储。在数组上完成数据的增删查改。

顺序表的初始化

如果创建失败，则返回溢出

typedef struct{

     ElemType \* elem;

     int length;

     int listsize;        //目前线性表的长度

}SqList;

  void InistSqList(SqList &l)//   \*l 则用l->elem,函数则用&q；&l 则用l->elem ,函数用q。

  //如果 L 是一个结构实例的指针，要用 -> 访问结构里的变量，而不能用点。

  //如果L 是一个结构的实例而非指针，只能用点而不能用 -> 。

  {

     l.elem=(ElemType \*)malloc(LIST\_INIT\_SIZE\*sizeof(ElemType));

  if(!l.elem) exit(OVERFLOW);

  l.length=0;

  l.listsize=LIST\_INIT\_SIZE;

  //return OK;

  }

在指定位置插入元素

如果插入的数字不符合规范，直接返回错误。如果数据超出原来给定的顺序表的大小，则需要给顺序表扩容，如扩容失败，则返回溢出。找到插入的位置，将i后面的的每个数据向后位移一位，空出来的位子插入元素e。

Status Insertlist(SqList &l,int i,ElemType e)//在线性表第i个位置插入元素e

{ ElemType \*q,\*newbase,\*p;

  if(i<1 || i>l.length+1) return ERROR;

  if(l.length>=l.listsize)

  {

    newbase=(ElemType \*)malloc((l.listsize+LIST\_INIT\_SIZE)\*sizeof(ElemType));

     if(!newbase) exit(OVERFLOW);

     l.elem=newbase;

     l.listsize+=LISTINCREMENT;

  }

  q=&(l.elem[i-1]);

  for(p=&(l.elem[l.length-1]);p>=q;--p)\*(p+1)=\*p;

  \*q=e;

  ++l.length;

  return OK;

}

顺序表删除元素

首先判断顺序表的长度是否符合规范，若不可删除操作，则返回ERROR，并且输出不可操作。删除中间的元素之后，需要把后面的元素往前平移，顺序表的长度要减少一位

Status deleteList(SqList &l, int i, ElemType e){

    if(l.length==0){   //判断目前的顺序表是否为空，若为空则不可进行删除操作

        printf("线性表的长度为0，不可进行删除操作");

        return ERROR;

    }

    if(i==0||i>l.length){

        printf("您输入的删除位置有误，不可进行删除操作");

        return ERROR;

    }

    e = l.elem[i-1];  //将要删除位置的元素存储起来

    if(i!=l.length){  //如果要删除的位置不在最后，那么要进行顺序表的前移操作

        for(int n=i;n<l.length;n++){

            l.elem[n-1] = l.elem[n];

        }

        l.length--;

        return e;

    }

    l.elem[i-1] = 0;  //如果要删除的位置在最后，那么直接将最后一个元素置0

    return e;

}

（2）实现单链表的初始化、在指定位置插入和删除元素；

单链表的初始化

单链表中，除头尾结点外，各结点的关系都是只有一个前驱和一个后继

typedef struct lnode{

  ElemType data;

  struct lnode \*next;

}lnode ,\*linklist;

bool InitList(linklist &l)

{

     l=(lnode\*)malloc(sizeof(lnode));

    if(l==NULL)  {

      return false;

    }

    l->next=NULL;

    return true;

}

void inita(linklist &l)

{

     int len;

     printf("请输入想要的数据长度：\n");

     scanf("%d",&len);

     lnode \*p=l;

     for(int i=1;i<=len;i++)

     {

       lnode \*s=(lnode\*)malloc(sizeof(lnode));

        scanf("%d",&s->data);

        s->next=NULL;

        p->next=s;

        p=s;//顺序读入

        /\*s->next=p->next;

        p->=s;

        逆序读入\*/

     }

}

在指定位置插入元素

Status listinsert(linklist l,int i,ElemType e)

{

   lnode \*p;

   p=l;

   int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

   if(!p||j>i)  return false;

   lnode \*s=(lnode\*)malloc(sizeof(lnode));

   s->data=e;

    s->next=p->next;

   p->next=s;

  return true;

}

删除指定位置的元素

找到需要删除的节点s之后，将下一位节点连接到当前的节点的前一位，并释放内存

Status deleteList(linklist l,int i)

{

  lnode \*p,\*s;

  p=l;

  int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

  s=p->next;

   p->next=s->next;

   free(s);

return true;

}

### 给定的2个有序线性链表的合并（合并到1个新的链表中）

获取指定位子的元素

Status Getelem(linklist l,int i,ElemType &e)

{

  lnode \*p;

  int j;

  p=l->next ;j=1;

  while(p&&j<i)

  {

    p=p->next;

    j++;

  }

  if(!p||j>i)  return false;

  e=p->data;

  return true;

}

获取链表的长度

Status listlength(linklist l)

{

  lnode \*p;

  p=l;

  int j=0;

  while(p!=NULL)

  {

    j++;

    p=p->next;

  }

  return j-1;

}

合并两个链表的到一起，如果有其中一个链表已经被完全排完，则直接把剩下的一个链表连接到后面，对于出现相同的数字，则只需要一个放入到新的链表

void mergelist (linklist l1,linklist l2,linklist &l3)

{

 InitList(l3);

 int i=1;

 int j=1;

 int k=0;

  int len1,len2;

  int e1,e2,e;

  len1=listlength(l1);

  len2=listlength(l2);

  Getelem(l1,i,e1);

  Getelem(l2,j,e2);

  while(i<=len1&&j<=len2)

  {

    if(e1<e2){

      listinsert(l3,++k,e1);

      Getelem(l1,++i,e1);

    }

    else if(e1>e2){

      listinsert(l3,++k,e2);

      Getelem(l2,++j,e2);

    }

    else{

      listinsert(l3,++k,e1);

      Getelem(l1,++i,e1);

      Getelem(l2,++j,e2);

    }

    while(i<=len1)

    {

      Getelem(l1,i++,e1);

      listinsert(l3,++k,e1);

    }

while(j<=len2)

{

  Getelem(l2,j++,e2);

  listinsert(l3,++k,e2);

}

      }

}

(合并到第一个链表)

void mergelist (linklist l1,linklist l2)

{

  lnode \*m,\*n;

  m=l1,n=l2;

  int len1,len2;

  int e1,e2,e;

  len1=listlength(l1);

  len2=listlength(l2);

//printf("%d %d\n",len1,len2);

  for(int i=1;i<=len2;i++)

  {

    Getelem(l2,i,e2);

    bool q=1;

    for(int j=1;j<=len1;j++)

    {

      Getelem(l1,j,e1);

      //printf("%d %d\n",e1,e2);

      if(e1==e2) {q=0;

      break;

      }

    }

if(q){

  listinsert(l1,++len1,(int)e2);

}

  }

}

### 实现双向链表的初始化、在指定位置插入和删除元素

双向链表结构

头节点 <-> 首元节点 <-> 节点2 <-> 节点3 <-> 节点4 <-> … <-> 节点n <-> NULL

双向链表的初始化

void inita(DuLinkList &l)

 {

    int len;

    printf("请输入想要的链表长度：");

    scanf("%d",&len);

    DulNode \*p=l;

    for(int i=1;i<=len;i++)

    {

      DulNode \*s=(DulNode\*)malloc(sizeof(DulNode));

     p->next=s;

     scanf("%d",&s->data);

     s->prior=p;

     s->next=NULL;

     p=s;

    }

 }

在指定位置插入

找到需要插入的位子，新建一个节点，将前一个节点的后节点连接到新的节点，将后一个节点的前节点连接到新节点，将新节点的前节点连接到p；

Status listinsert(DuLinkList l,int i,ElemType e)

{

   DulNode \*p;

   p=l;

   int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

   if(!p||j>i)  return false;

   DulNode \*s=(DulNode\*)malloc(sizeof(DulNode));

   s->data=e;

    s->next=p->next;

   p->next=s;

   s->prior=p;

   s->next->prior=s;

  return true;

}

删除元素

找到需要删除的位子，将前一个结点的后节点连接到后一个节点，后一个节点的前结点链接到前一个结点。释放s的空间

Status deleteList(DuLinkList l,int i)

{

  DulNode \*p,\*s;

  p=l;

  int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

  s=p->next;

s->next->prior=p;

   p->next=s->next;

   free(s);

return true;

}

## (2)内部排序练习

### 1、实现直接插入排序、2-路插入排序和希尔排序；

直接插入排序

在每一趟排序中，将后面部分的第一个元素插入到前面的有序部分，这样做后就形成了新的前面部分（有序）和后面部分（未排序）。再进行下一趟排序，又把后面部分的第一个元素插入到前面部分。重复这一过程，直到所有的元素都排好序。

void insertSort(int a[], int n)

{

  int temp, j, k;

  for (int i = 0; i < n; i++)

  {

    temp = a[i];

    for (j = i - 1; j >= 0; j--)

    {

      if (a[j] < temp)

        break;

    }

    if (j != i - 1)

    {

      for (k = i - 1; k > j; k--)

      {

        a[k + 1] = a[k];

      }

      a[k + 1] = temp;

    }

  }

}

1. 路插入排序

另外设置一个同存储记录的数组大小相同的数组 d，将无序表中第一个记录添加进 d[0] 的位置上，然后从无序表中第二个记录开始，同 d[0] 作比较：如果该值比 d[0] 大，则添加到其右侧；反之添加到其左侧。

void twoLineInsertSort(int a[],int d[],int n)

{

   int first=0,final=0;

   int k=0;

   d[0]=a[0];

   for(int i=1;i<n;i++)

   {

    if(a[i]<d[first])

    {

      first=(first-1+n)%n;

      d[first]=a[i];

    }

    else if(a[i]>d[final])

    {

      final=(final+1+n)%n;

      d[final]=a[i];

    }

    else {

      k = (final + 1 + n) % n;

            while (d[((k - 1) + n) % n] > a[i]) {

                d[(k + n) % n] =d[(k - 1 + n) % n];

                k = (k - 1 + n) % n;

            }

            d[(k + n) % n] = a[i];

            final = (final + 1 + n) % n;

    }

   }

   for(int j=0;j<n;j++)

   {

    a[j]=d[(j+first)%n];

   }

}

希尔排序

先将整个待排元素序列分割成若干个子序列（由相隔某个“增量”的元素组成的）分别进行直接插入排序，然后依次缩减增量再进行排序，待整个序列中的元素基本有序（增量足够小）时，再对全体元素进行一次直接插入排序。

void shell(int a[],int n)

{

  int temp;

  int j,i,s;

for( j=n/2;j>=1;j=j/2)

{

  for(i=j;i<n;i++)

  {

   temp=a[i];

   for( s=i-j;s>=0&&temp<a[s];s=s-j){

        a[s+j]=a[s];

      }

      a[s+j]=temp;

  }

}

}

### 2、实现起泡排序、双向起泡排序和快速排序；

气泡排序

第一次循环，让最小(最大)的值浮到第一位，

第二次循环，让剩余的数字中最小(最大)浮到第二位

依次类推。

void BubbleSort(int a[],int n)

{int temp;

  for(int i=1;i<n;i++)

  {

    for(int j=0;j<n-i;j++)

    {

      if(a[j+1]<a[j])

      {

         Swap(a[j+1],a[j]);

      }

    }

  }

}

双向起泡排序

左右两侧都进行比较，大的数向右移动，小的数向左移动

void twoBubbleSort(int a[],int n)

{

   int left, right, shift, i;

    left = 0;

    right = n - 1;

    shift = 1;

    while(left < right) {

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

            if(a[i] > a[i+1]) {

                Swap(a[i], a[i+1]);

                shift = i;

            }

        }

        right = shift;

        for(i = right-1; i >= left; i--) {

            if(a[i] > a[i+1]) {

                Swap(a[i], a[i+1]);

                shift = i + 1;

            }

        }

        left = shift;

    }

}

快速排序

快速排序主要有三个参数，left 为区间的开始地址，right 为区间的结束地址，Key 为当前的开始的值。

从待排序的记录序列中选取一个记录（通常第一个）作为基准元素（称为key）key=arr[left]，然后设置两个变量，left指向数列的最左部，right 指向数据的最右部。

void Quick\_Sort(int a[], int begin, int end)

{

    if(begin > end)

        return;

    int temp = a[begin];

    int i = begin;

    int j = end;

    while(i != j){

        while(a[j] >= temp && j > i)

            j--;

        while(a[i] <= temp && j > i)

            i++;

        if(j > i){

            Swap(a[i],a[j]);

        }

    }

    a[begin] = a[i];

    a[i] = temp;

    Quick\_Sort(a, begin, i-1);

    Quick\_Sort(a, i+1, end);

}

### 3、实现简单选择排序和堆排序；

简单选择排序

对于具有 n 个记录的无序表遍历 n-1 次，第 i 次从无序表中第 i 个记录开始，找出后序关键字中最小的记录，然后放置在第 i 的位置上

void selectSort(int a[], int n)

{

  int i, j;

  int min;

  int temp;

  for (i = 0; i < n - 1; ++i)

    min = i;

    for (j = i + 1; j < n; ++j)

    {

      if (a[j] < a[min])

      {

        min = j;

      }

    }

    Swap(a[min],a[i]);

  }

堆排序

1.根据拿到的数组构建大顶堆

2.从堆顶取走元素，放到其应该存在的位置中去。从堆底拿到堆中最后一个元素，放到堆顶，此时这个堆很可能不再合法也就是说不再是一个堆；

3.维护这个堆，通过自己写的方法调整堆中节点结构，让它重新变成一个堆；

4.重复2,3过程，直到堆被取空，此时数组也被完全排列好；

void HeapAdjust(int a[],int s,int m)//一次筛选的过程

{

    int rc,j;

    rc=a[s];

    for(j=2\*s;j<=m;j=j\*2)//通过循环沿较大的孩子结点向下筛选

    {

        if(j<m&&a[j]<a[j+1]) j++;//j为较大的记录的下标

        if(rc>a[j]) break;

        a[s]=a[j];s=j;

    }

    a[s]=rc;//插入

}

void HeapSort(int a[],int n)

{

    int temp,i,j;

    for(i=n/2;i>0;i--)//通过循环初始化顶堆

    {

        HeapAdjust(a,i,n);

    }

    for(i=n;i>0;i--)

    {

        temp=a[1];

        a[1]=a[i];

        a[i]=temp;//将堆顶记录与未排序的最后一个记录交换

        HeapAdjust(a,1,i-1);//重新调整为顶堆

    }

}

### 4、实现归并排序。

**归并排序对序列的元素进行逐层折半分组，然后从最小分组开始比较排序，合并成一个大的分组，逐层进行，最终所有的元素都是有序的**

void Merge(int A[], int left, int mid, int right)

{

  int \*B = new int[right - left + 1];

  int i = left;

  int j = mid + 1;

  int k = 0;                                                //k指向辅助数组B[]中待放置元素的位置

  while (i <= mid && j <= right)

  {                                                          //从小到大排序，将A[i]和A[j]中的较小元素放入B[]中

    if (A[i] <= A[j])

      B[k++] = A[i++];

else

  B[k++] = A[j++];

  }

  while (i <= mid)                                           //对序列A[left:mid]剩余的部分依次进行处理，与图中的（5）对应

    B[k++] = A[i++];

  while (j <= right)

      B[k++] = A[j++];

  for (i = left, k = 0; i <= right; i++)                        A[i] = B[k++];

  delete[] B;

}

void MergeSort(int A[], int left, int right)                   //归并排序

{

  if (left < right)                                          //当数组内的元素数大于1时进行二分操作，只有一个元素的时候，不作任何处理直接结束

  {

    int mid;

    mid = (left + right) / 2;                              //计算中间位置

    MergeSort(A, left, mid);                               //对数组A[left:mid]中的元素进行归并排序

    MergeSort(A, mid + 1, right);                          //对数组A[mid+1:right]中的元素进行归并排序

    Merge(A, left, mid, right);                            //进行合并操作

  }

}

# 程序流程图

1、实现顺序表的初始化、在指定位置插入和删除元素；

2、实现单链表的初始化、在指定位置插入和删除元素；

3、给定的2个有序线性链表的合并（合并到1个新的链表中以及合并到其中的1个链表中两种方式）；

4、实现双向链表的初始化、在指定位置插入和删除元素。

(2)内部排序练习

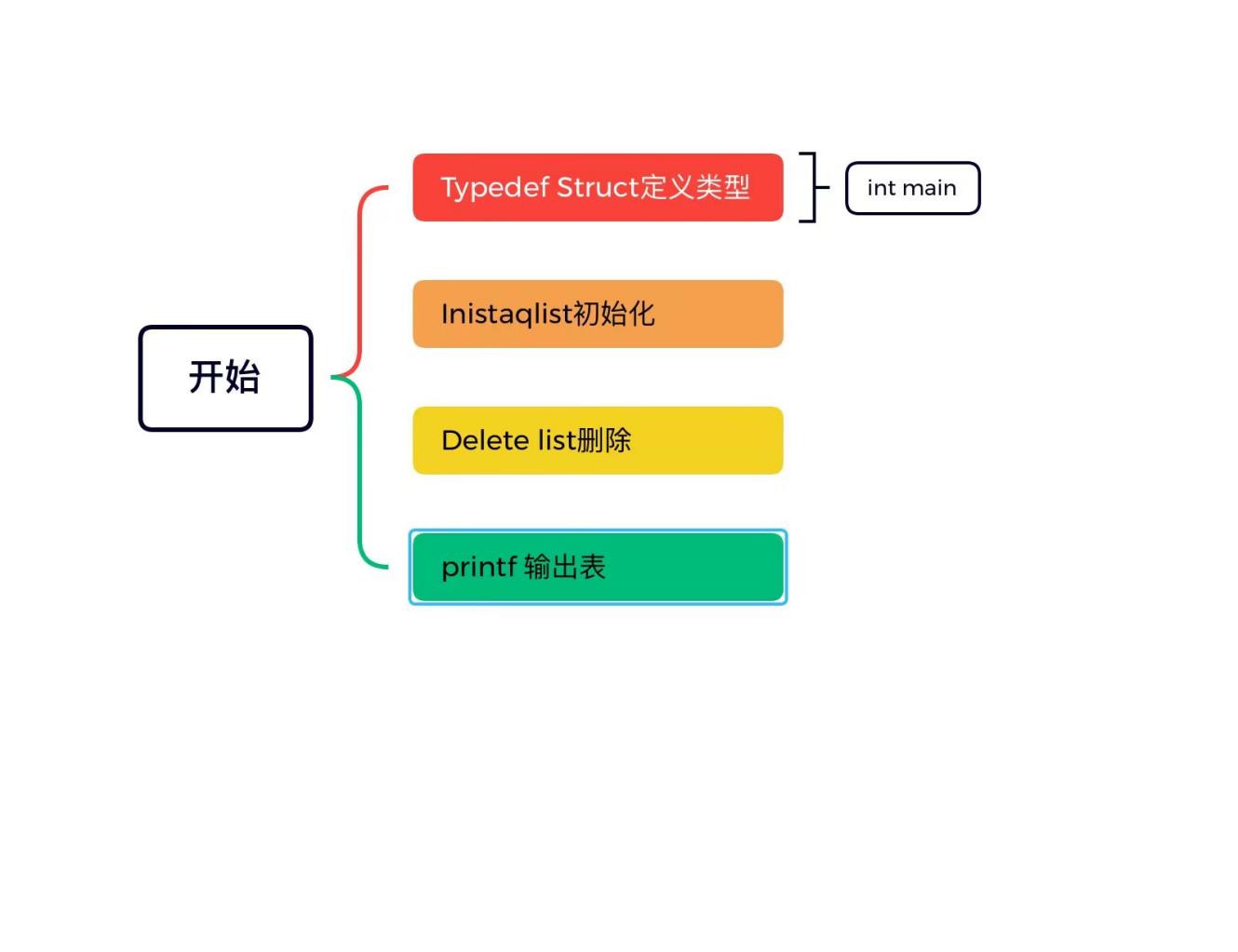
1、实现直接插入排序、2-路插入排序和希尔排序；

2、实现起泡排序、双向起泡排序和快速排序；

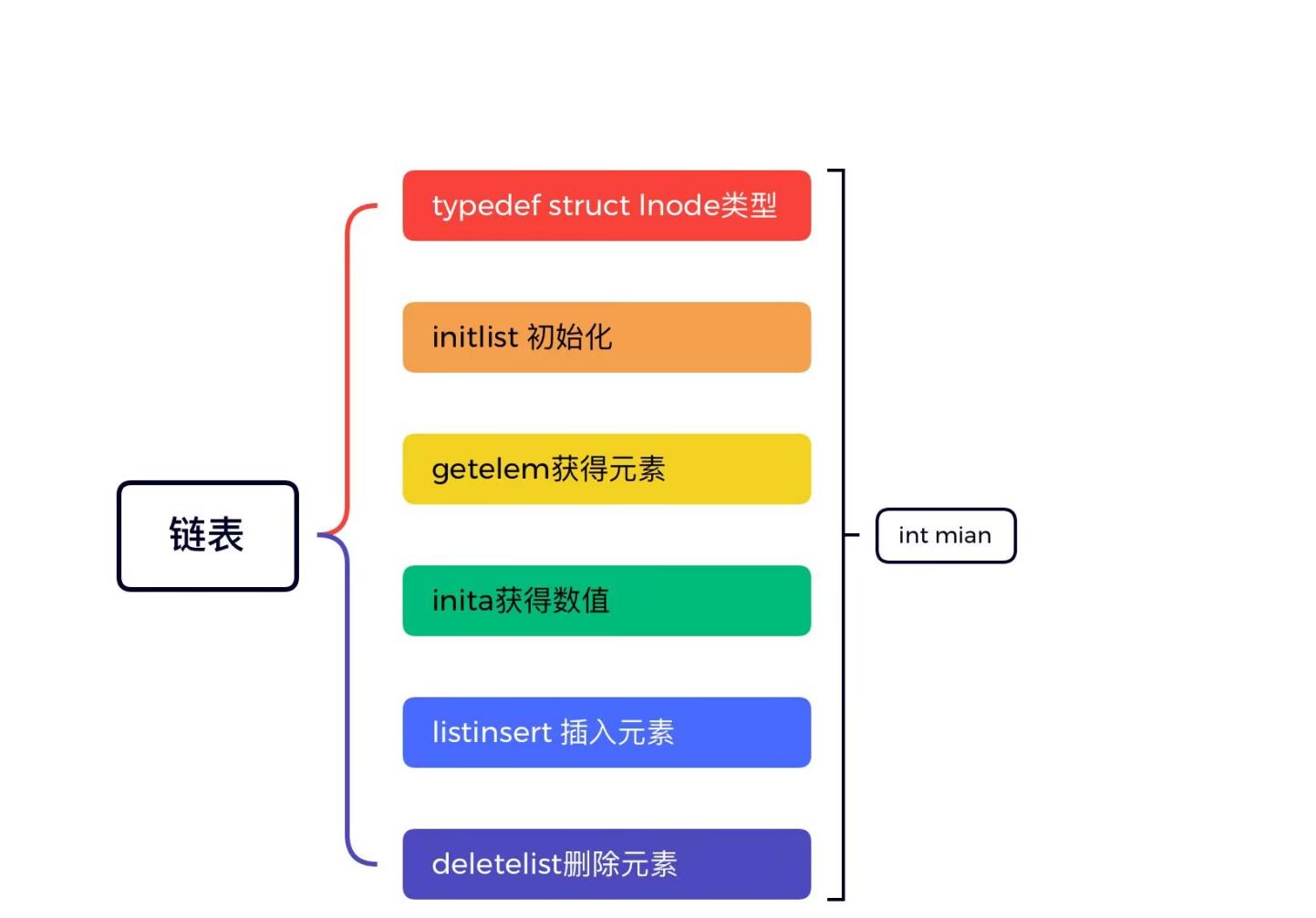
3、实现简单选择排序和堆排序；

4、实现归并排序。

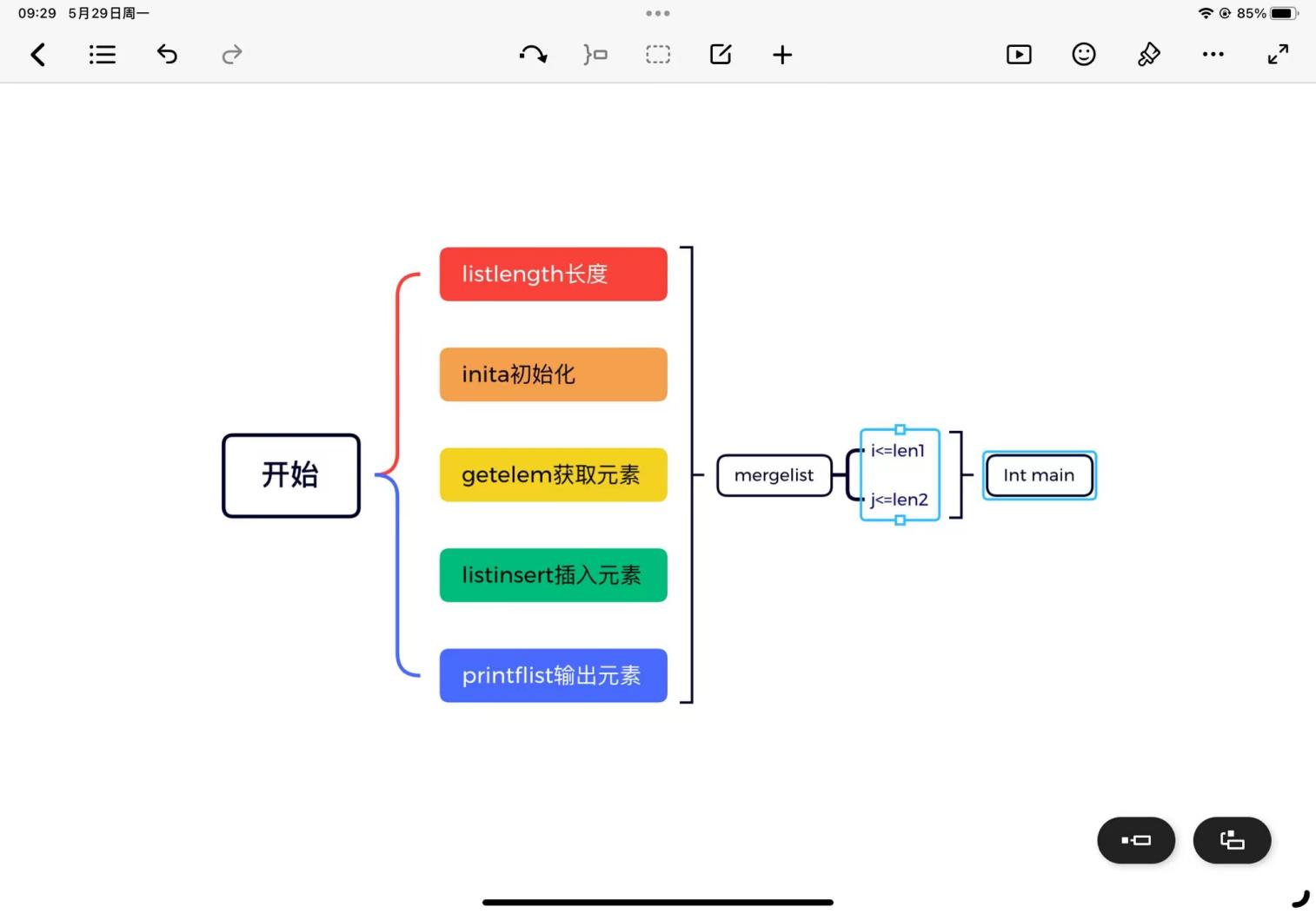
顺序表



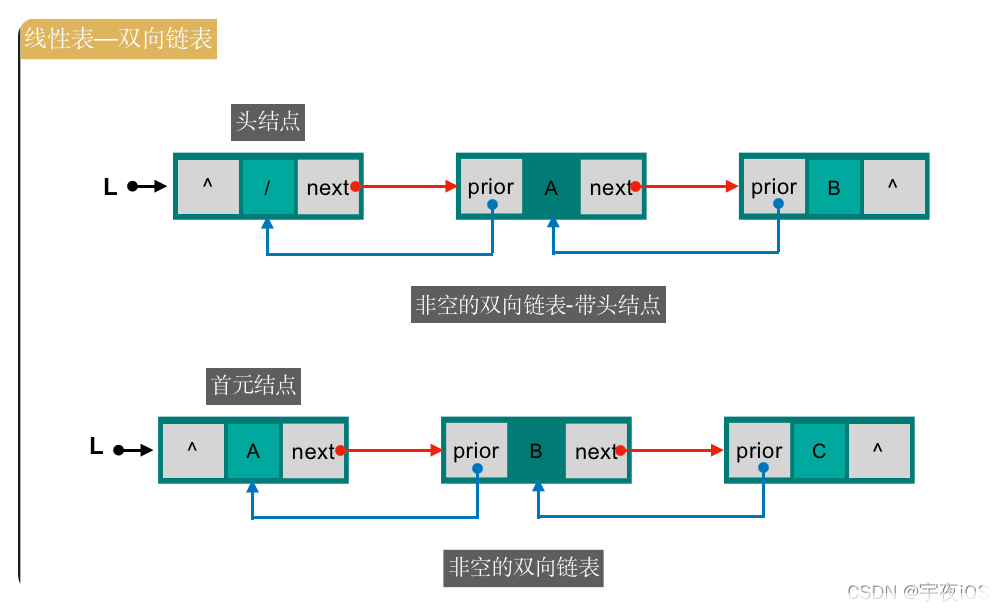
链表



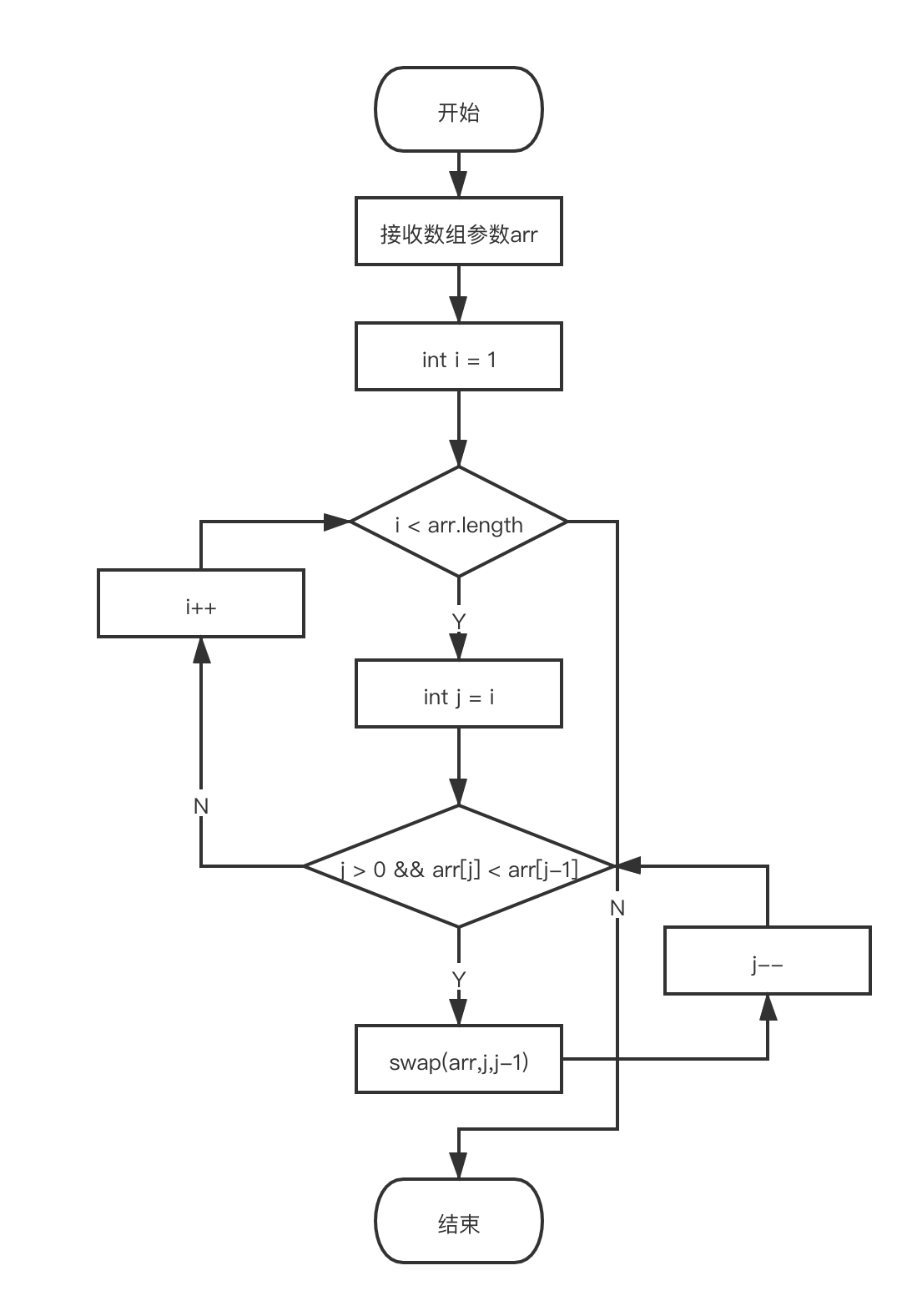
合并链表



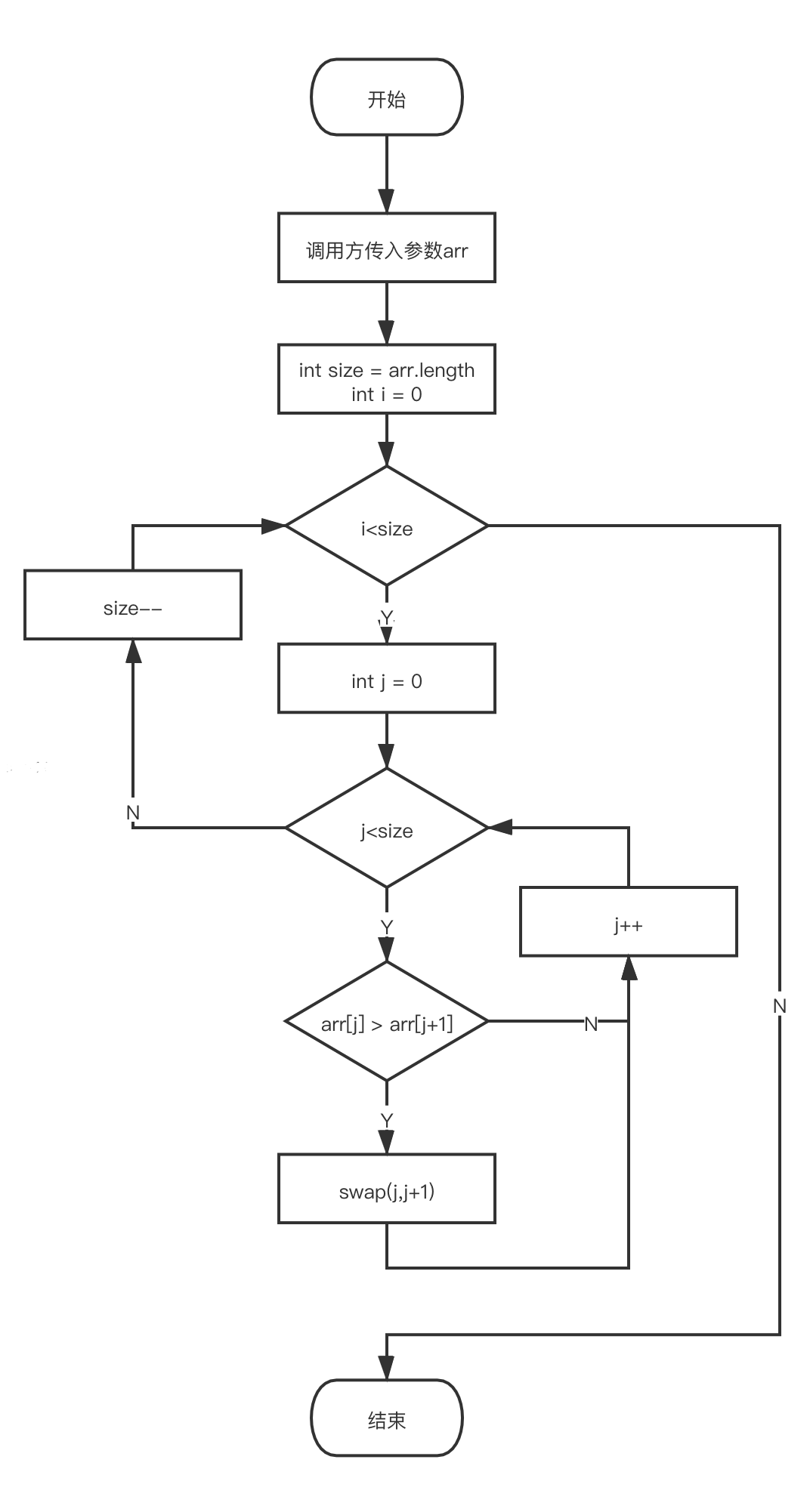
双向链表



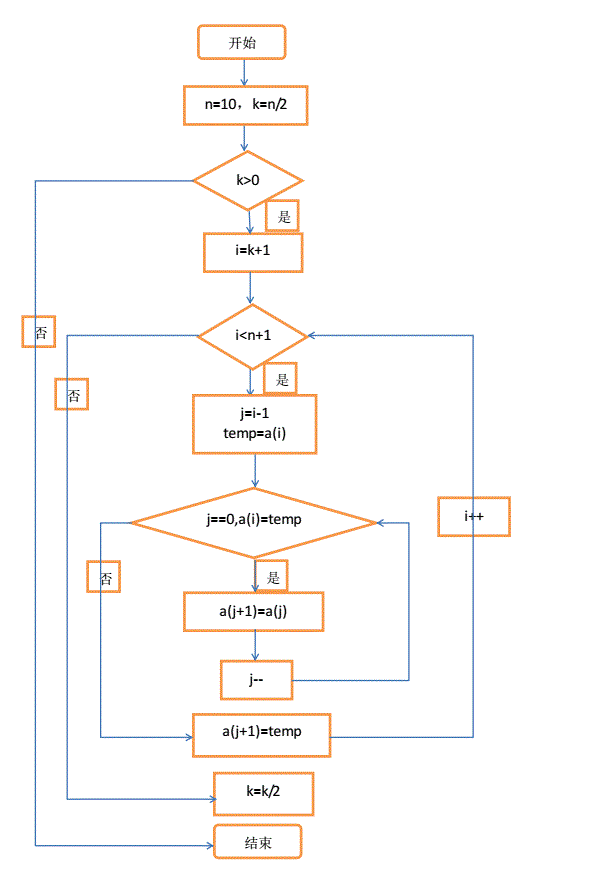
直接插入排序



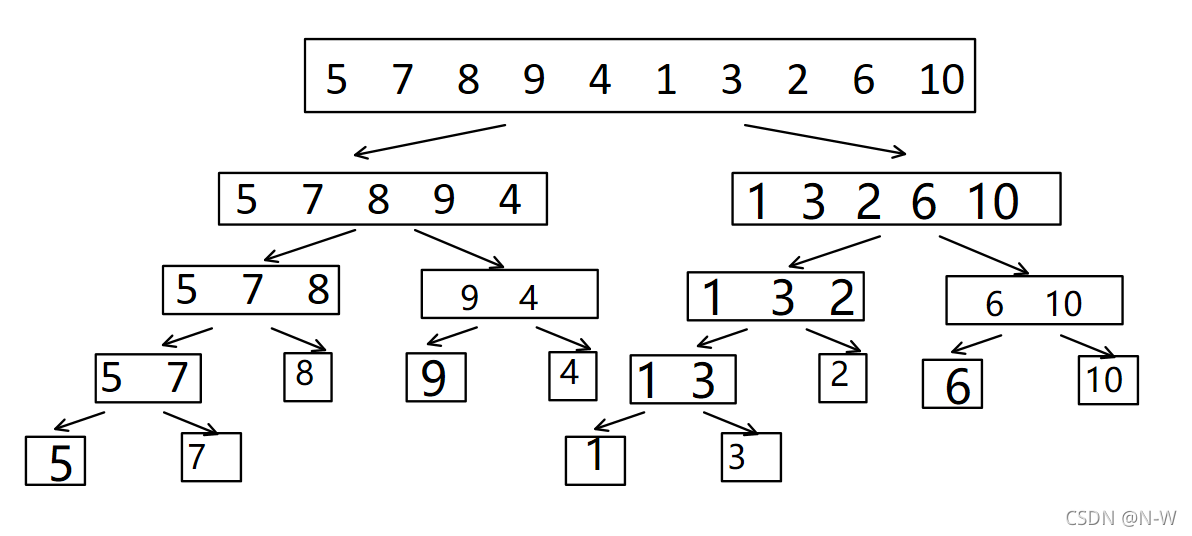
冒泡排序



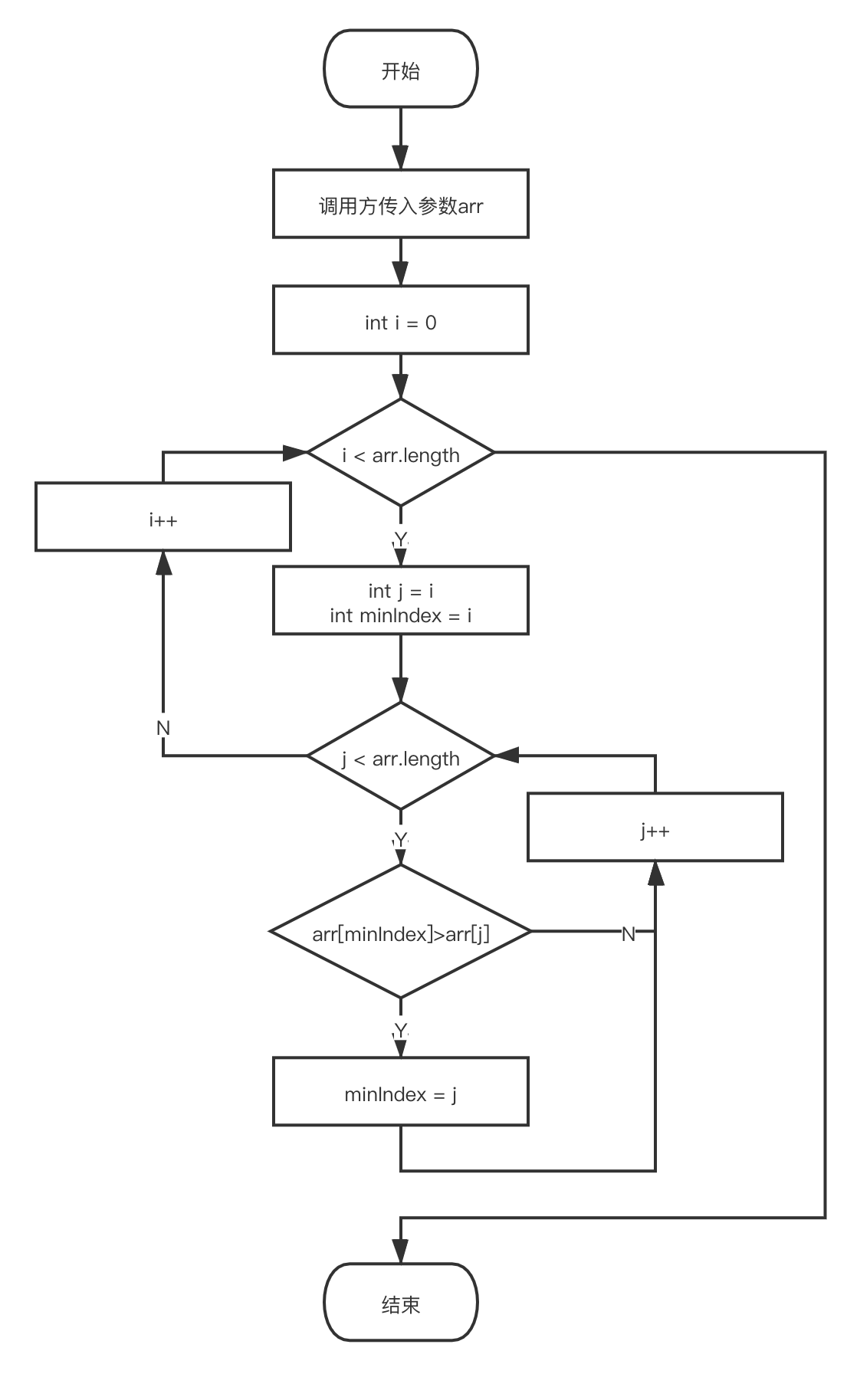
希尔排序



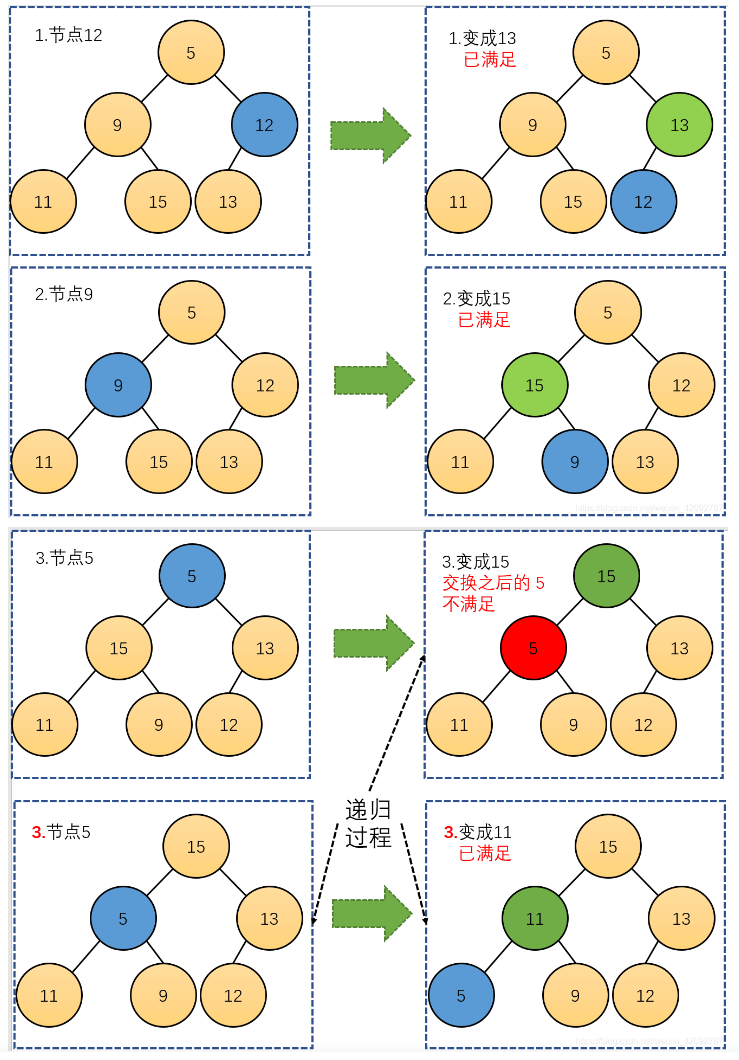
归并排序



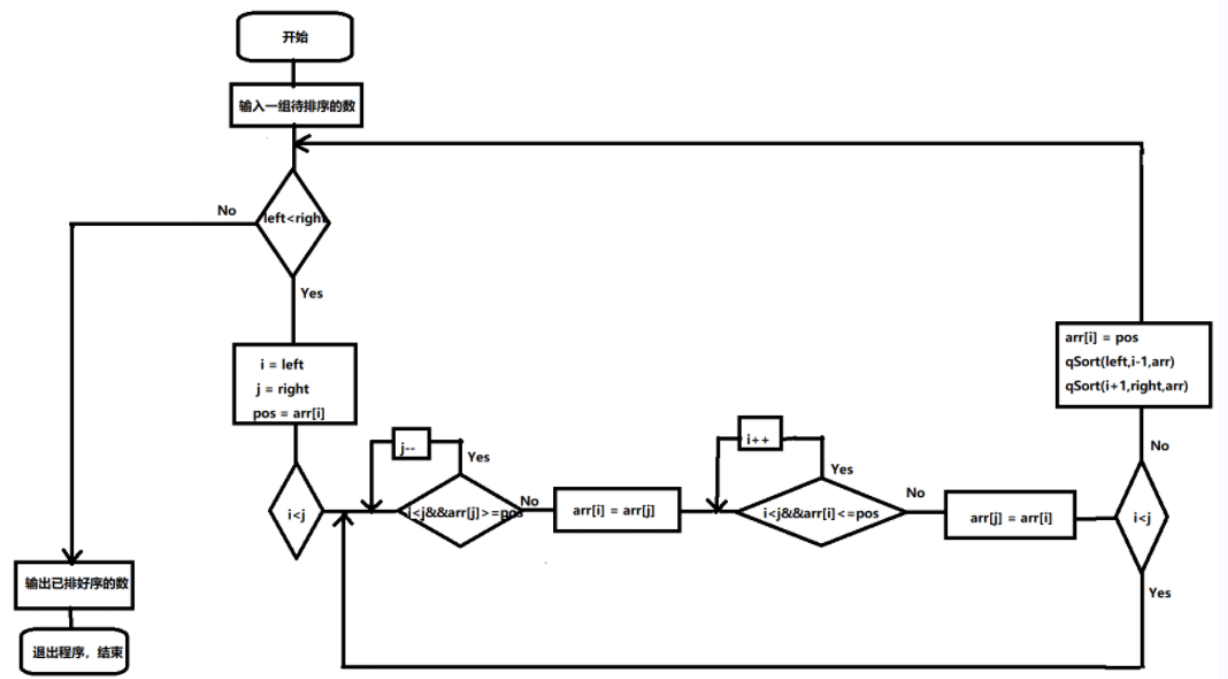
选择排序



堆排序

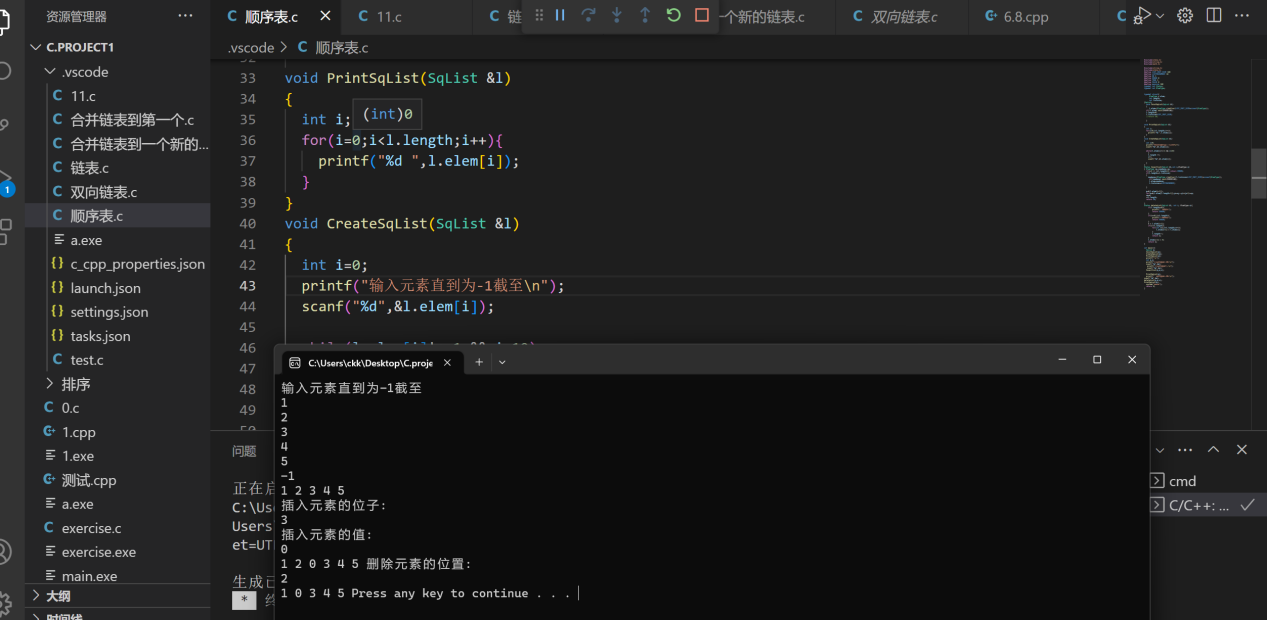


快速排序

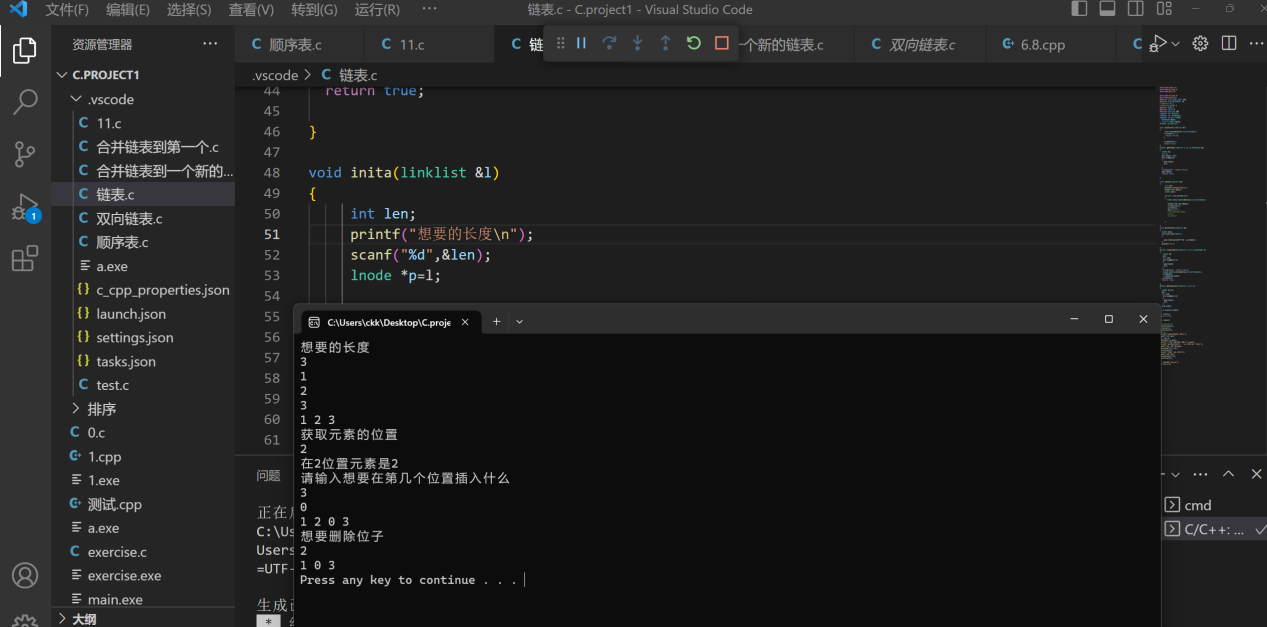


# 实验结果及分析

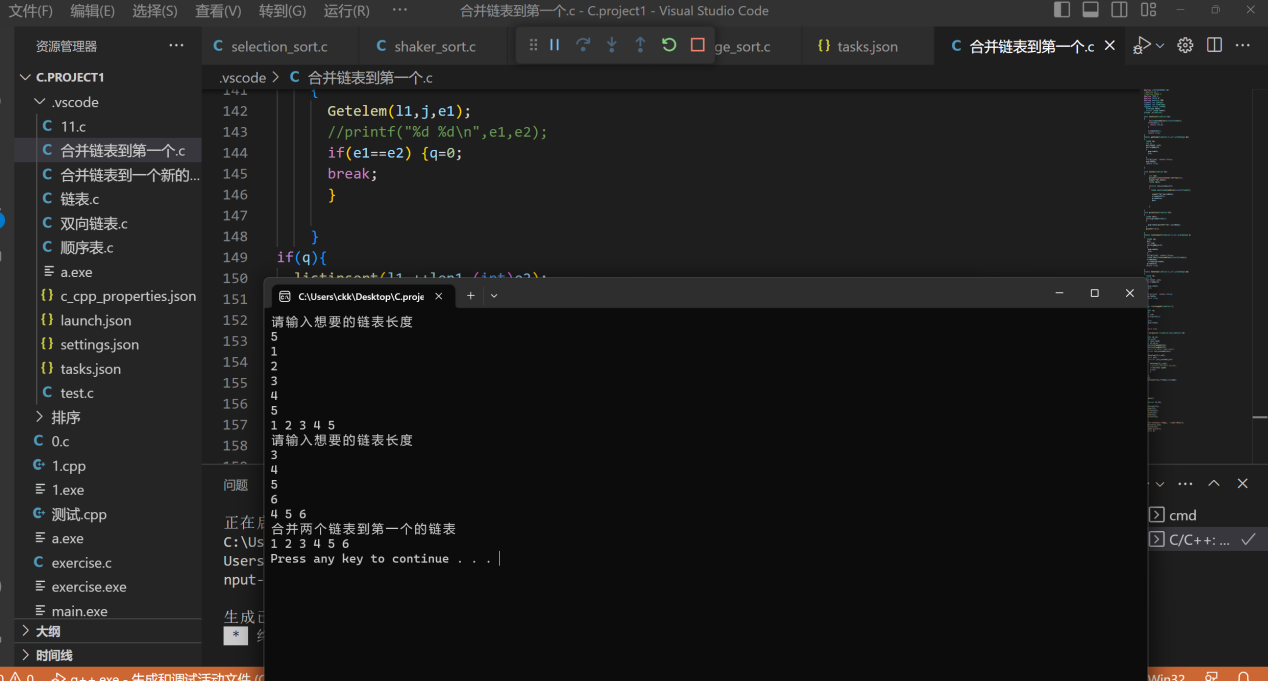
顺序表



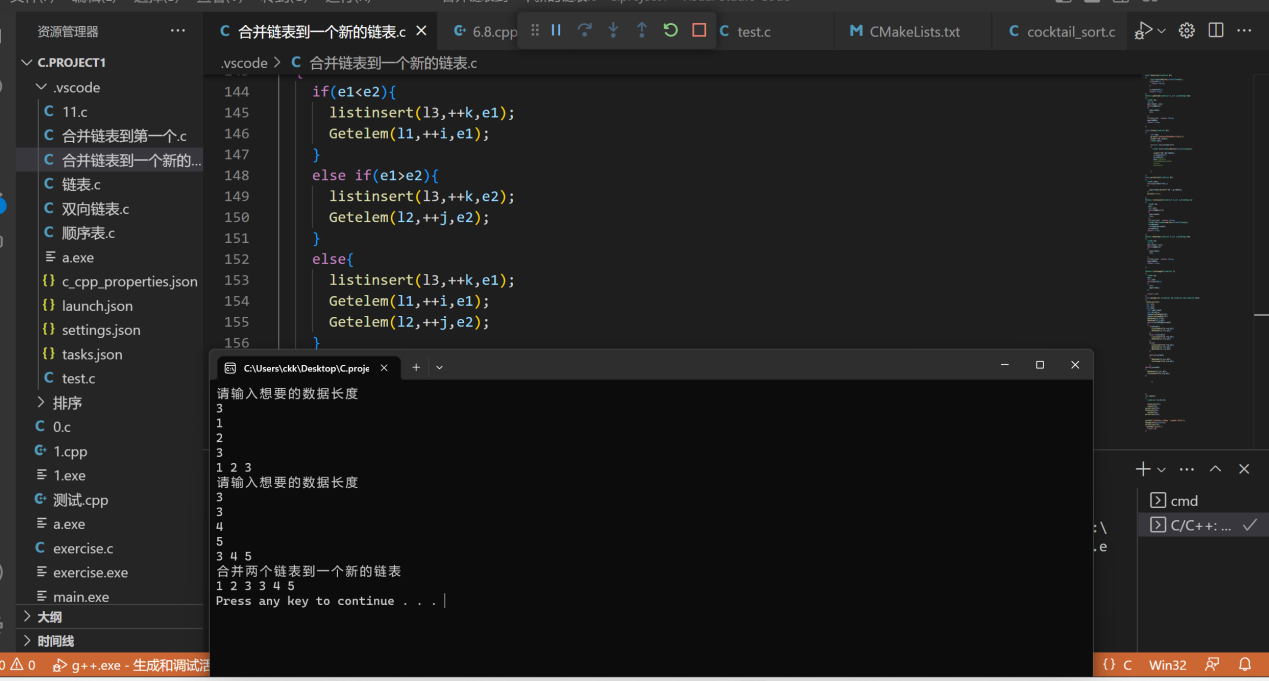
链表



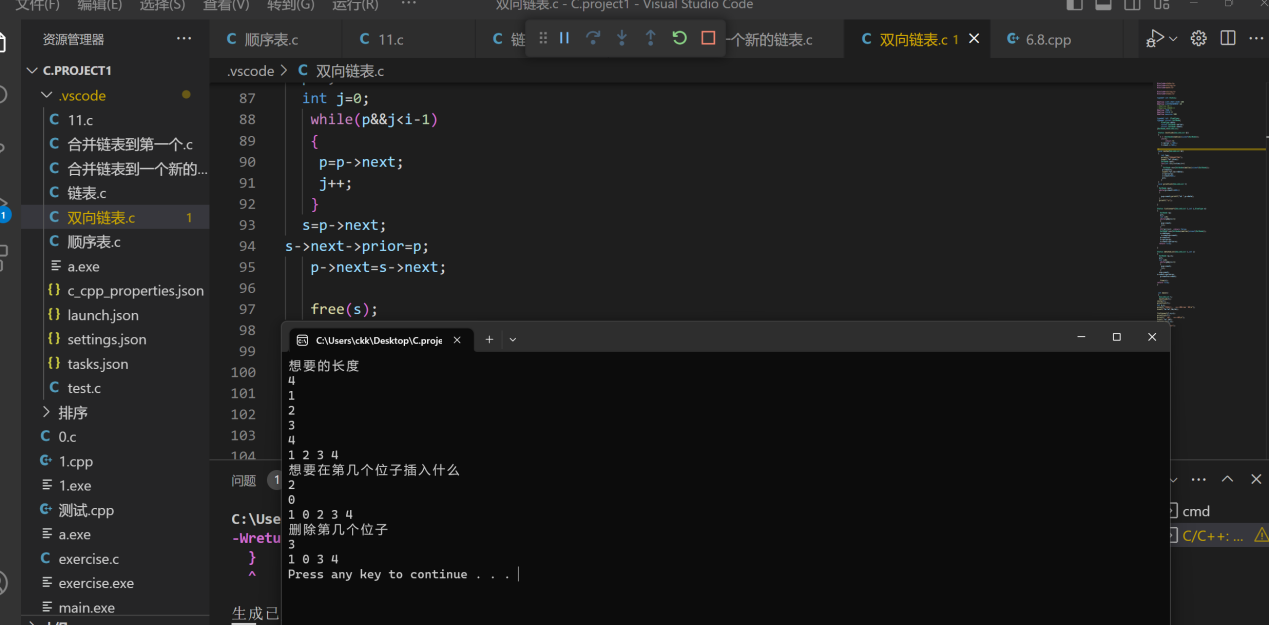
合并链表到第一个



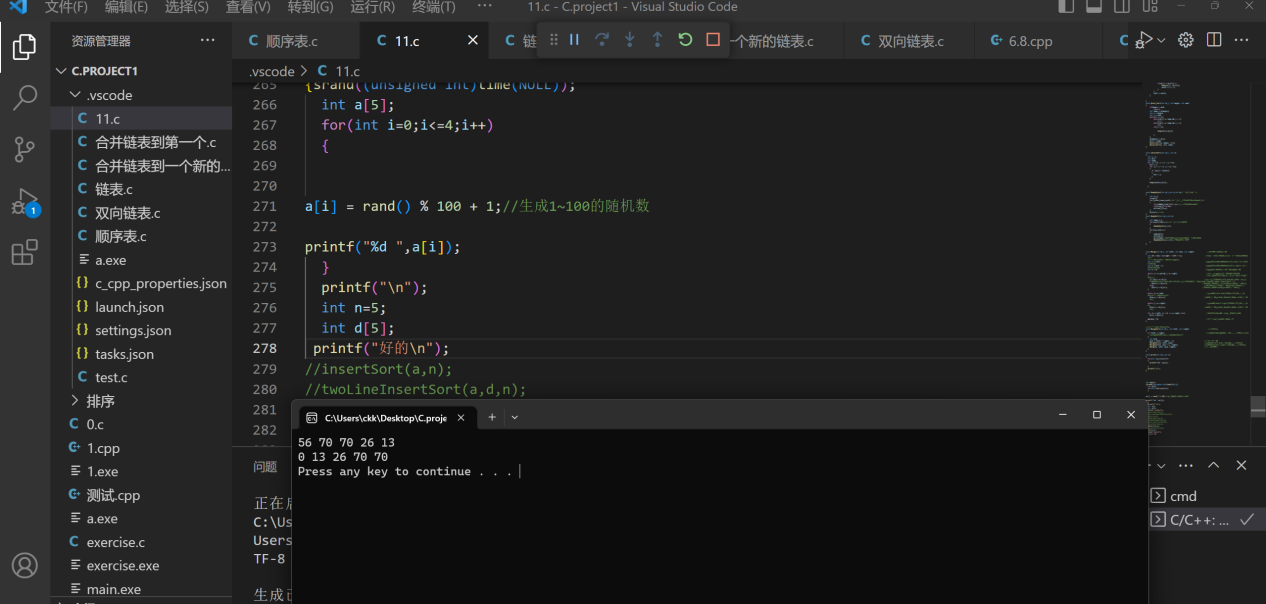
合并链表到一个新的链表



双向链表



排序结果都为由小到大



# 注：程序放在附录中

## 顺序表

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<string.h>

#include<stdio.h>

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

#define maxsize 100

typedef int Status;

typedef int ElemType;

typedef struct{

     ElemType \* elem;

     int length;

     int listsize;

}SqList;

  void InistSqList(SqList &l)

  {

     l.elem=(ElemType \*)malloc(LIST\_INIT\_SIZE\*sizeof(ElemType));

  if(!l.elem) exit(OVERFLOW);

  l.length=0;

  l.listsize=LIST\_INIT\_SIZE;

  //return OK;

  }

void PrintSqList(SqList &l)

{

  int i;

  for(i=0;i<l.length;i++){

    printf("%d ",l.elem[i]);

  }

}

void CreateSqList(SqList &l)

{

  int i=0;

  printf("输入元素直到为-1截至\n");

  scanf("%d",&l.elem[i]);

  while(l.elem[i]!=-1 && i<19)

    {

    l.length ++;

    i++;

    scanf("%d",&l.elem[i]);

  }

}

Status Insertlist(SqList &l,int i,ElemType e)

{ ElemType \*q,\*newbase,\*p;

  if(i<1 || i>l.length+1) return ERROR;

  if(l.length>=l.listsize)

  {

    newbase=(ElemType \*)malloc((l.listsize+LIST\_INIT\_SIZE)\*sizeof(ElemType));

     if(!newbase) exit(OVERFLOW);

     l.elem=newbase;

     l.listsize+=LISTINCREMENT;

  }

  q=&(l.elem[i-1]);

  for(p=&(l.elem[l.length-1]);p>=q;--p)\*(p+1)=\*p;

  \*q=e;

  ++l.length;

  return OK;

}

Status deleteList(SqList &l, int i, ElemType e){

    if(l.length==0){

        printf("删除失败");

        return ERROR;

    }

    if(i==0||i>l.length){

        printf("位置错误");

        return ERROR;

    }

    e = l.elem[i-1];

    if(i!=l.length){

        for(int n=i;n<l.length;n++){

            l.elem[n-1] = l.elem[n];

        }

        l.length--;

        return e;

    }

    l.elem[i-1] = 0;

    return e;

}

int main(){

  SqList q;

  InistSqList(q);

  CreateSqList(q);

  PrintSqList(q);

  printf("\n");

  int m,n;

  printf("插入元素的位子:\n");

  scanf("%d",&m);

   printf("插入元素的值:\n");

   scanf("%d",&n);

  Insertlist(q,m,n);

  PrintSqList(q);

  printf("删除元素的位置:\n");

scanf("%d",&m);

deleteList(q,m,n);

PrintSqList(q);

  system("pause");

  return 0;

}

## 链表

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<string.h>

#include<stdio.h>

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

//#define OK 1

//#define ERROR 0

#define TRUE 1

#define FALSE 0

#define maxsize 100

typedef int Status;

typedef int ElemType;

typedef struct lnode{

  ElemType data;

  struct lnode \*next;

}lnode ,\*linklist;

bool InitList(linklist &l)

{

     l=(lnode\*)malloc(sizeof(lnode));

    if(l==NULL)  {

      return false;

    }

    l->next=NULL;

    return true;

}

Status getelem(linklist l,int i,ElemType &e)

{

  lnode \*p;

  int j;

  p=l->next ;j=1;

  while(p&&j<i)

  {

    p=p->next;

    j++;

  }

  if(!p||j>i)  return false;

  e=p->data;

  return true;

}

void inita(linklist &l)

{

     int len;

     printf("想要的长度\n");

     scanf("%d",&len);

     lnode \*p=l;

     for(int i=1;i<=len;i++)

     {

       lnode \*s=(lnode\*)malloc(sizeof(lnode));

        scanf("%d",&s->data);

        s->next=NULL;

        p->next=s;

        p=s;

     }

}

void printlist(linklist &l)

{

  lnode \*p=l;

  while(p->next!=NULL)

  {

    p=p->next;printf("%d ",p->data);

  }

  printf("\n");

}

Status listinsert(linklist l,int i,ElemType e)

{

   lnode \*p;

   p=l;

   int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

   if(!p||j>i)  return false;

   lnode \*s=(lnode\*)malloc(sizeof(lnode));

   s->data=e;

    s->next=p->next;

   p->next=s;

  return true;

}

Status deleteList(linklist l,int i)

{

  lnode \*p,\*s;

  p=l;

  int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

  s=p->next;

   p->next=s->next;

   free(s);

return true;

}

int main()

{

  linklist l;

  InitList(l);

  inita(l);

printlist(l);

int i;

printf("获取元素的位置\n");

scanf("%d",&i);

int ans,e;

getelem(l,i,ans);

printf("在%d位置元素是%d\n",i,ans);

printf("请输入想要在第几个位置插入什么\n");

scanf("%d  %d",&i,&e);

listinsert(l,i,e);

printlist(l);

printf("想要删除位子\n");

scanf("%d",&i);

deleteList(l,i);

printlist(l);

   system("pause");

  return 0;

}

## 双向链表

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<string.h>

#include<stdio.h>

typedef int Status;

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

//#define OK 1

//#define ERROR 0

#define TRUE 1

#define FALSE 0

#define maxsize 100

typedef int  ElemType;

typedef struct DulNode{

    ElemType data;

    struct DulNode \*prior;

    struct DulNode \*next;

}DulNode,\*DuLinkList;

 Status Initlink(DuLinkList &l)

 {

   l = (DulNode\*)malloc(sizeof(DulNode));

    if(!l)

        return 0;

    l->prior = NULL;

    l->next = NULL;

 }

 void inita(DuLinkList &l)

 {

    int len;

    printf("想要的长度");

    scanf("%d",&len);

    DulNode \*p=l;

    for(int i=1;i<=len;i++)

    {

      DulNode \*s=(DulNode\*)malloc(sizeof(DulNode));

     p->next=s;

     scanf("%d",&s->data);

     s->prior=p;

     s->next=NULL;

     p=s;

    }

 }

 void printlist(DuLinkList l)

{

  DulNode \*p=l;

  while(p->next!=NULL)

  {

    p=p->next;printf("%d ",p->data);

  }

  printf("\n");

}

Status listinsert(DuLinkList l,int i,ElemType e)

{

   DulNode \*p;

   p=l;

   int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

   if(!p||j>i)  return false;

   DulNode \*s=(DulNode\*)malloc(sizeof(DulNode));

   s->data=e;

    s->next=p->next;

   p->next=s;

   s->prior=p;

   s->next->prior=s;

  return true;

}

Status deleteList(DuLinkList l,int i)

{

  DulNode \*p,\*s;

  p=l;

  int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

  s=p->next;

s->next->prior=p;

   p->next=s->next;

   free(s);

return true;

}

 int main()

 {

  DuLinkList l;

  Initlink(l);

inita(l);

printlist(l);

int m,n;

printf("想要在第几个位子插入什么\n");

scanf("%d %d",&m,&n);

listinsert(l,m,n);

printlist(l);

printf("删除第几个位子\n");

scanf("%d",&m);

deleteList(l,m);

printlist(l);

  system("pause");

  return 0;

 }

## 合并链表到新的链表

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<string.h>

#include<stdio.h>

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

//#define OK 1

//#define ERROR 0

#define TRUE 1

#define FALSE 0

#define maxsize 100

typedef int Status;

typedef int ElemType;

typedef struct lnode{

  ElemType data;

  struct lnode \*next;

}lnode ,\*linklist;

bool InitList(linklist &l)

{

     l=(lnode\*)malloc(sizeof(lnode));

    if(l==NULL)  {

      return false;

    }

    l->next=NULL;

    return true;

}

Status getelem(linklist l,int i,ElemType &e)

{

  lnode \*p;

  int j;

  p=l->next ;j=1;

  while(p&&j<i)

  {

    p=p->next;

    j++;

  }

  if(!p||j>i)  return false;

  e=p->data;

  return true;

}

void inita(linklist &l)

{

     int len;

     printf("请输入想要的数据长度\n");

     scanf("%d",&len);

     lnode \*p=l;

     for(int i=1;i<=len;i++)

     {

       lnode \*s=(lnode\*)malloc(sizeof(lnode));

        scanf("%d",&s->data);

        s->next=NULL;

        p->next=s;

        p=s;

     }

}

void printlist(linklist &l)

{

  lnode \*p=l;

  while(p->next!=NULL)

  {

    p=p->next;printf("%d ",p->data);

  }

  printf("\n");

}

Status listinsert(linklist l,int i,ElemType e)

{

   lnode \*p;

   p=l;

   int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

   if(!p||j>i)  return false;

   lnode \*s=(lnode\*)malloc(sizeof(lnode));

   s->data=e;

    s->next=p->next;

   p->next=s;

  return true;

}

Status Getelem(linklist l,int i,ElemType &e)

{

  lnode \*p;

  int j;

  p=l->next ;j=1;

  while(p&&j<i)

  {

    p=p->next;

    j++;

  }

  if(!p||j>i)  return false;

  e=p->data;

  return true;

}

Status listlength(linklist l)

{

  lnode \*p;

  p=l;

  int j=0;

  while(p!=NULL)

  {

    j++;

    p=p->next;

  }

  return j-1;

}

void mergelist (linklist l1,linklist l2,linklist &l3)

{

 InitList(l3);

 int i=1;

 int j=1;

 int k=0;

  int len1,len2;

  int e1,e2;

  len1=listlength(l1);

  len2=listlength(l2);

  Getelem(l1,i,e1);

  Getelem(l2,j,e2);

  while(i<=len1&&j<=len2)

  {

    if(e1<e2){

      listinsert(l3,++k,e1);

      Getelem(l1,++i,e1);

    }

    else if(e1>e2){

      listinsert(l3,++k,e2);

      Getelem(l2,++j,e2);

    }

    else{

      listinsert(l3,++k,e1);

      Getelem(l1,++i,e1);

      Getelem(l2,++j,e2);

    }

}

    while(i<=len1)

    {

      Getelem(l1,i++,e1);

      listinsert(l3,++k,e1);

    }

while(j<=len2)

{

  Getelem(l2,j++,e2);

  listinsert(l3,++k,e2);

}

}

int main()

{

  linklist l1,l2,l3;

  InitList(l1);

  inita(l1);

printlist(l1);

InitList(l2);

  inita(l2);

printlist(l2);

printf("合并两个链表到一个新的链表\n");

mergelist(l1,l2,l3);

printlist(l3);

 system("pause");

  return 0;

}

## 合并链表到第一个链表

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<string.h>

#include<stdio.h>

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

//#define OK 1

//#define ERROR 0

#define TRUE 1

#define FALSE 0

#define maxsize 100

typedef int Status;

typedef int ElemType;

typedef struct lnode{

  ElemType data;

  struct lnode \*next;

}lnode ,\*linklist;

bool InitList(linklist &l)

{

     l=(lnode\*)malloc(sizeof(lnode));

    if(l==NULL)  {

      return false;

    }

    l->next=NULL;

    return true;

}

Status getelem(linklist l,int i,ElemType &e)

{

  lnode \*p;

  int j;

  p=l->next ;j=1;

  while(p&&j<i)

  {

    p=p->next;

    j++;

  }

  if(!p||j>i)  return false;

  e=p->data;

  return true;

}

void inita(linklist &l)

{

     int len;

     printf("请输入想要的链表长度\n");

     scanf("%d",&len);

     lnode \*p=l;

     for(int i=1;i<=len;i++)

     {

       lnode \*s=(lnode\*)malloc(sizeof(lnode));

        scanf("%d",&s->data);

        s->next=NULL;

        p->next=s;

        p=s;

     }

}

void printlist(linklist &l)

{

  lnode \*p=l;

  while(p->next!=NULL)

  {

    p=p->next;printf("%d ",p->data);

  }

  printf("\n");

}

Status listinsert(linklist l,int i,ElemType e)

{

   lnode \*p;

   p=l;

   int j=0;

   while(p&&j<i-1)

   {

    p=p->next;

    j++;

   }

   if(!p||j>i)  return false;

   lnode \*s=(lnode\*)malloc(sizeof(lnode));

   s->data=e;

    s->next=p->next;

   p->next=s;

  return true;

}

Status Getelem(linklist l,int i,ElemType &e)

{

  lnode \*p;

  int j;

  p=l->next ;j=1;

  while(p&&j<i)

  {

    p=p->next;

    j++;

  }

  if(!p||j>i)  return false;

  e=p->data;

  return true;

}

Status listlength(linklist l)

{

  lnode \*p;

  p=l;

  int j=0;

  while(p!=NULL)

  {

    j++;

    p=p->next;

  }

  return j-1;

}

void mergelist (linklist l1,linklist l2)

{

  lnode \*m,\*n;

  m=l1,n=l2;

  int len1,len2;

  int e1,e2,e;

  len1=listlength(l1);

  len2=listlength(l2);

//printf("%d %d\n",len1,len2);

  for(int i=1;i<=len2;i++)

  {

    Getelem(l2,i,e2);

    bool q=1;

    for(int j=1;j<=len1;j++)

    {

      Getelem(l1,j,e1);

      //printf("%d %d\n",e1,e2);

      if(e1==e2) {q=0;

      break;

      }

    }

if(q){

  listinsert(l1,++len1,(int)e2);

}

  }

}

int main()

{

  linklist l1,l2;

  InitList(l1);

  inita(l1);

printlist(l1);

InitList(l2);

  inita(l2);

printlist(l2);

printf("合并两个链表到第一个的链表\n");

mergelist(l1,l2);

printlist(l1);

 system("pause");

  return 0;

}

## 排序

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void HeapAdjust(int a[],int s,int m)//一次筛选的过程

{

    int rc,j;

    rc=a[s];

    for(j=2\*s;j<=m;j=j\*2)//通过循环沿较大的孩子结点向下筛选

    {

        if(j<m&&a[j]<a[j+1]) j++;//j为较大的记录的下标

        if(rc>a[j]) break;

        a[s]=a[j];s=j;

    }

    a[s]=rc;//插入

}

void HeapSort(int a[],int n)

{

    int temp,i,j;

    for(i=n/2;i>0;i--)//通过循环初始化顶堆

    {

        HeapAdjust(a,i,n);

    }

    for(i=n;i>0;i--)

    {

        temp=a[1];

        a[1]=a[i];

        a[i]=temp;//将堆顶记录与未排序的最后一个记录交换

        HeapAdjust(a,1,i-1);//重新调整为顶堆

    }

}

void Merge(int A[], int left, int mid, int right)

{

  int \*B = new int[right - left + 1];

  int i = left;                                             //指向待排序子序列数组A[left:mid]中当前待比较的元素

  int j = mid + 1;                                          //指向待排序子序列数组A[mid+1:right]中当前待比较的元素

  int k = 0;                                                //k指向辅助数组B[]中待放置元素的位置

  while (i <= mid && j <= right)

  {

    if (A[i] <= A[j])

      B[k++] = A[i++];

    else                                                   //否则将后半部分辅助标记对应的值存入B[]中

      B[k++] = A[j++];                                   //存入辅助数组且与之对应的辅助标记后移

  }

  while (i <= mid)

    B[k++] = A[i++];

  while (j <= right)

    B[k++] = A[j++];

  for (i = left, k = 0; i <= right; i++)                     //将合并后的序列复制到原来的A[]序列

    A[i] = B[k++];

  delete[] B;                                                //释放动态创建的辅助数组空间

}

void MergeSort(int A[], int left, int right)

{

  if (left < right)

  {

    int mid;

    mid = (left + right) / 2;                              //计算中间位置

    MergeSort(A, left, mid);                               //对数组A[left:mid]中的元素进行归并排序

    MergeSort(A, mid + 1, right);                          //对数组A[mid+1:right]中的元素进行归并排序

    Merge(A, left, mid, right);                            //进行合并操作

  }

}

void print(int a[],int n)

{

  for(int i=1;i<=n;i++)

  {

    printf("%d ",a[i]);

  }

  printf("\n");

}

int main()

{srand((unsigned int)time(NULL));

  int a[6];

  for(int i=1;i<=5;i++)

  {

a[i] = rand() % 100 + 1;//生成1~100的随机数

printf("%d ",a[i]);

  }

  printf("\n");

  int n=5;

  int d[5];

//HeapSort(a,n);

MergeSort(a,1,n);

print(a,n);

system("pause");

  return 0;

}

## 排序2

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void insertSort(int a[], int n)

{

  int temp, j, k;

  for (int i = 0; i < n; i++)

  {

    temp = a[i];

    for (j = i - 1; j >= 0; j--)

    {

      if (a[j] < temp)

        break;

    }

    if (j != i - 1)

    {

      for (k = i - 1; k > j; k--)

      {

        a[k + 1] = a[k];

      }

      a[k + 1] = temp;

    }

  }

}

void twoLineInsertSort(int a[],int d[],int n)

{

   int first=0,final=0;

   int k=0;

   d[0]=a[0];

   for(int i=1;i<n;i++)

   {

    if(a[i]<d[first])

    {

      first=(first-1+n)%n;

      d[first]=a[i];

    }

    else if(a[i]>d[final])

    {

      final=(final+1+n)%n;

      d[final]=a[i];

    }

    else {

      k = (final + 1 + n) % n;

            while (d[((k - 1) + n) % n] > a[i]) {

                d[(k + n) % n] =d[(k - 1 + n) % n];

                k = (k - 1 + n) % n;

            }

            d[(k + n) % n] = a[i];

            final = (final + 1 + n) % n;

    }

   }

   for(int j=0;j<n;j++)

   {

    a[j]=d[(j+first)%n];

   }

}

void Swap(int &x, int &y) {

    int temp;

    temp = x;

    x = y;

    y = temp;

}

void shell(int a[],int n)

{

  int temp;

  int j,i,s;

for( j=n/2;j>=1;j=j/2)

{

  for(i=j;i<n;i++)

  {

   temp=a[i];

   for( s=i-j;s>=0&&temp<a[s];s=s-j){

        a[s+j]=a[s];

      }

      a[s+j]=temp;

  }

}

}

void BubbleSort(int a[],int n)

{int temp;

  for(int i=1;i<n;i++)

  {

    for(int j=0;j<n-i;j++)

    {

      if(a[j+1]<a[j])

      {

         Swap(a[j+1],a[j]);

      }

    }

  }

}

void twoBubbleSort(int a[],int n)

{

   int left, right, shift, i;

    left = 0;

    right = n - 1;

    shift = 1;

    while(left < right) {

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

            if(a[i] > a[i+1]) {

                Swap(a[i], a[i+1]);

                shift = i;

            }

        }

        right = shift;

        for(i = right-1; i >= left; i--) {

            if(a[i] > a[i+1]) {

                Swap(a[i], a[i+1]);

                shift = i + 1;

            }

        }

        left = shift;

    }

}

void Quick\_Sort(int a[], int begin, int end)

{

    if(begin > end)

        return;

    int temp = a[begin];

    int i = begin;

    int j = end;

    while(i != j){

        while(a[j] >= temp && j > i)

            j--;

        while(a[i] <= temp && j > i)

            i++;

        if(j > i){

            Swap(a[i],a[j]);

        }

    }

    a[begin] = a[i];

    a[i] = temp;

    Quick\_Sort(a, begin, i-1);

    Quick\_Sort(a, i+1, end);

}

void selectSort(int a[], int n)

{

  int i, j;

  int min;

  int temp;

  for (i = 0; i < n -1; ++i)

  {

    min = i;

    for (j = i + 1; j < n; ++j)

    {

      if (a[j] < a[min])

      {

        min = j;

      }

    }

    Swap(a[min],a[i]);

  }

  }

void print(int a[],int n)

{

  for(int i=0;i<n;i++)

  {

    printf("%d ",a[i]);

  }

  printf("\n");

}

int main()

{srand((unsigned int)time(NULL));

  int a[5];

  for(int i=0;i<=4;i++)

  {

a[i] = rand() % 100 + 1;//生成1~100的随机数

printf("%d ",a[i]);

  }

  printf("\n");

  int n=5;

  int d[5];

//insertSort(a,n);

//twoLineInsertSort(a,d,n);

//shell(a,n);

//BubbleSort(a,n);

//twoBubbleSort(a,n);

//Quick\_Sort(a,0,n-1);

selectSort(a,n);

print(a,n);

system("pause");

  return 0;

}