数据结构期末复习

期末 70%，实验 20%，平时 10%

下周二机房答疑，周四下午期末前答疑

考试题型

填空 15-20 分，选择 10-20 分，简答 35 分，写程序 10 分，补齐算法 10 分，写算法 10 分

第一章 基本概念

必考算法的时间复杂度，常用的时间复杂度有 $1 < \log n < n < 2n < n \log n < n^2 < n^3 < n!$

例题

x=2;
while(x<n/2)
    x=2*x;
A.O($\log_2 n$) B.O(n) C.O($n\log_2 n$) D.O(n^2)
	x 以指数逼近 n/2，时间复杂度是 log n

课堂小测验

1、下面程序的时间复杂度为 ________。
```
void fun( int n) { int i=1, k=100; while(i<=n) {k++;  i+=2;} }
```
(a) O(log2n)
(b) O(n)
(c) O(nlog2n)
(d) O(n^2)

2、算法分析的目的是 ________。
(a) 找出数据结构的合理性
(b) 分析算法的易读性和文档性
(c) 分析算法的效率以求改进
(d) 研究算法中输入和输出的关系

3、算法的时间复杂度与 ________ 有关。
（a）计算机硬件性能
（b）编译程序质量
（c）问题规模
（d）程序设计语言

答案 B C C

第二章 线性表

基本概念名词

单链表：每个结点只有自己的数据，和指向后继元素的 next 指针域

双向链表：每个结点除数据外有两个指针域，一个指向后继元素，另一指向前趋元素

循环链表： 尾结点的 next 指向头结点

双向循环链表：头结点的 prior 指向尾结点，尾结点的 next 指向头结点

静态链表：不设指针，借助一维数组来描述线性链表

例题

考上面几种链表的结构、适用场合和特点，链表指针操作，按算法判断功能的填空

下面的函数是对不带头结点的单链表进行就地逆置的算法，该算法用L返回逆置后的链表的头指针，试在空缺处填入适当的语句。
void reverse(linklist &L){
p=null; q=L;
while(q!=NULL)
{ ① L=L->next ;
q->next=p; p=q;
② q=L ; }
③ L=p ;
}
	链表题用草稿纸铅笔画一下箭头，注意：修改指针的顺序不能丢结点
	还需要注意括号位置，此处每个循环是
	L=L->next; q->next=p; p=q; q=L;

```
/* 链表原地逆置 */
List LocalReverseList(List &L)
{
    Pointer tmp, older, newer;
    newer = L->Next;
    older = newer->Next;
    while(older->Next)
    {
        tmp = older->Next;
        older->Next = newer;
        newer = older;
        older = tmp;
    }
    L->Next->Next = nullptr;
    L->Next = older;
    L->Next->Next = newer;
    return L;
}
```

课堂小测验

1、含有 n 个元素的顺序表中插入一个元素时移动元素的平均次数是________。
(a) n/2
(b) (n+1)/2
(c) (n-1)/2
(d) O(n)

2、对于双链表，在两个结点之间插入一个新结点是，需要修改 ________ 个指针域。
(a) 1
(b) 2
(c) 3
(d) 4

3、在一个双链表中，在*p 结点之后插入结点*q 的操作是 ________。
(a) q->next = p->next; p->next->prior=q; p->next=q; q->prior=p;
(b) q->prior = p; p-> next=q; p -> next -> prior =q; q ->next = p -> next;
(c) p-> next=q; q->prior = p; q ->next = p -> next; p -> next -> prior =q;
(d) p -> next -> prior =q; q->prior = p; p-> next=q; q ->next = p -> next;

4、非空的循环单链表 L 的尾结点（由 p 所指向）满足 ________。
(a) p == NULL
(b) p -> next == L
(c) p-> next == NULL
(d) p==L

5、某线性表最常用的操作是在尾元素之后插入一个元素和删除尾元素，则采用 ________ 存储方式最节省运算时间。
(a) 单链表
(b) 双链表
(c) 循环单链表
(d) 循环双链表

6、下面算法实现的功能是，在顺序表 L 的第 i 个元素之前插入新的元素 e，划线处要填写的语句是_______ :

```
Status ListInsert_Sq(SqList &L, int i, ElemType e) {
 if (i < 1 || i > L.length+1) return ERROR;
 q = &(L.elem[i-1]);
 for (p = &(L.elem[L.length-1]); p >= q;  --p)
     ( ______________)
*q = e;
++L.length;
return OK;
     } // ListInsert_Sq
```

(a) *p = *(p-1);
(b) *(p+1) = *p;
(c) *(p+1) = *q;
(d) *p = *(q-1);

7、下面算法的功能是_______ 。

```
int 算法1(SqList L, ElemType e) {
     i = 1; p = L.elem;
     while (i <= L.length && (*p++! = e)
        ++i;
     if (i <= L.length)
        return i;
     else
        return 0;
}
```

(a) 在顺序表中查找第一个与元素 e 相等的元素，查找成功返回元素 e 的值，查找不成功返回 0
(b) 在顺序表中查找所有与元素 e 相等的元素，查找成功返回元素 e 的值，查找不成功返回 0
(c) 在顺序表中查找第一个与元素 e 相等的元素，查找成功返回位序，查找不成功返回 0
(d) 在顺序表中查找所有与元素 e 相等的元素，查找成功返回位序，查找不成功返回 0

8、线性表的静态链表存储结构与顺序存储结构相比，优点是 _______。
(a) 所有的操作算法实现简单
(b) 便于随机存取
(c) 便于利用零散的存储器空间
(d) 便于插入和删除

答案：A D A B D B C D
课上做错第 5 题，【删除操作需要循环链表，才能以 O(1)比较方便的访问到尾节点的前一个和后一个元素】

第三章 栈和队列

基本概念名词

栈：只允许在表尾进行插入（进栈）或删除（出栈）操作的线性表；

允许进行插入和删除的一端称为栈顶（表尾）；不允许操作的另一端称为栈底（表头）；

最后进栈的元素最先离开（LIFO 特性）；

表中没有元素时，称为空栈；

顺序存储栈：用一维数组依次存放从栈底到栈顶的元素，数组的上界 maxsize 表示栈的最大容量

另设栈顶指针 Top，指示目前栈顶元素在数组中的位置

顺序栈判满：S.top - S.base >= S.size

顺序栈判空：S.top == S.base

链式栈：用链表存放从栈底到栈顶的元素，规定只能在栈顶进行操作

队列：只允许在表的一端进行插入，在另一端删除元素的线性表；

允许插入的一端称为队尾(rear)；

允许删除的一端称为队头(front)；

循环队列判满：(Q.rear+1) % MAXSIZE = Q.front

循环队列判空：Q.front = Q.rear

```
/* 汉诺塔问题 */
void hanoi(int n, char x, char y, char z)
{
     if(n==1)
        move(x,1,z);
     else{
        hanoi(n-1,x,z,y);

    move(x,n,z);
    hanoi(n-1,y,x,z);
    }
}
```

课堂小测验 1

1、判定一个顺序栈 st 为（元素个数最多为 MaxSize）为栈满的条件为 ________。
(a) st.top==-1
(b) st.top! =-1
(c) st.top==MaxSize-1
(d) st.top==st.base

2、判定一个顺序栈 st 为（元素个数最多为 MaxSize）为栈空的条件为 ________。
(a) st.top==-1
(b) st.top==MaxSize
(c) st.top!=MaxSize
(d) st.top!=MaxSize
(e) st.top-st.base= MaxSize

3、若一个栈用数组 data[1..n]存储，初始栈顶指针 top 为 n+1，则以下元素 x 进入栈的正确操作是________。
(a) top++; data[top]=x;
(b) top--; data[top]=x;
(c) data[top]=x;top++;
(d) data[top]=x;top--;

4、从一个不带头结点的栈顶指针为 lst 的栈链中删除一个结点时，用 x 保存被删结点的值，则执行 ________。
(a) x=lst->data; lst= lst->next;
(b) x=lst; lst = lst->next ;
(c) x=lst->data
(d) lst=lst->next; x=lst->data;

5、链栈与顺序栈相比有一个明显的优点，即 ________。
(a) 插入操作更方便
(b) 总是不会出现栈空的情况
(c) 通常不会出现栈满的情况
(d) 删除操作更加方便

答案：C A B A C
【第三题应该选 B，这个初始栈顶指针 top 为 n+1 说明数组是倒着用的，n+1 是初始】

课堂小测验 2

1、表达式 3* 2^(4+2*2-6*3)-5 求值过程中当扫描到 6 时，对象栈和算符栈为（　　），其中^为乘幂 。
（a）3,2,4,1,1 ；　(*^(+*-
（b）3,2,8 ；　(*^-
（c）3,2,4,2,2；(*^(-
（d）3,2,8；(*^(-
2、设 a=6,b=4,c=2,d=3,e=2,则后缀表达式 abc-/de*+的值是__________。
（a）7
（b）8
（c）9
（d）都不对

答案 D C

1、下列关于栈的叙述中错误的是__________。
Ⅰ．采用非递归方式重写递归程序时必须使用栈
Ⅱ．函数调用时，系统要用栈保存必要的信息
Ⅲ．只要确定了入栈次序，即可确定出栈次序
Ⅳ．栈是一种受限的线性表，允许在其两端进行操作
（a）仅 I
（b）仅 I 、 II 、III
（c）仅 I 、III、 IV
（d）仅 II 、III、 IV

2、假设用一个不带头结点的单链表表示队列，队头和队尾指针分别为 front 和 rear，则判断队空的条件是 _______。
（a）front == NULL
（b）front == rear
（c）front!==NULL
（d）rear!==NULL

答案：C A

第四章 串

重点 KMP 算法，next 函数和 next 函数改进的 nextval 函数

主要考点是给字符串计算 next，但一般不要求能手写求 next 的代码

KMP 算法 next 函数的手动求法

求 next 函数

首先 next[1]填 0，next[2]填 1，从 next[3]开始，此后每次计算 next[j+1]时，k 取前面 next[j]的值（是前面一位的 next 值）

s[j]表示字符串的第 j 位，从 1 开始数

退出条件：k=0 或 s[j] = s[k]，得 next[j+1] = k+1

其他情况：不断向前回溯 k = next[k]，直到满足上面的条件，最后也得 next[j+1] = 回溯后的 k+1

求 nextval 函数（已有 next 函数结果后）

先把 next 函数的所有结果填到 nextval，然后从 nextval[2]开始，此后每次计算 nextval[j]时，k 取 next[j]（就是自身）

若 s[j] != s[k]：nextval[j]就等于 next[j]

若 s[j] == s[k]：不断向前回溯 k = nextVal[k]，nextVal[j] = k，直到 s[j] != s[k]为止

求 next 和 nextval 函数的代码

```
void get_next(string s, vector<int> &next)
//求 next 的函数
{
    next[1]=0;
    next[2]=1;
    int k,j;
    j=2;
    while(1)
    {
        if(j >= next.size()-1)  //到达最后的字符就退出循环
            break;

        //以下代码设法求 j+1 位的 next 值
        k = next[j];  //先让 k 取 j 位的 next 值
```

```
        while(k!=0)   //当 k 不是第 0 位，s[j] 也不等于 s[k] 时
        {
            if(s[j] == s[k])
                break;
            else
                k = next[k]; //回溯
        }
        next[j+1]=k+1;  // j+1 位的 next 值 等于 k+1
        j++;  //再求下一位的 next
    }
}

void get_nextVal(string p, vector<int> next, vector<int> &nextVal)
//在已有 next 的基础上，求修正值
{
    nextVal = next; //赋初值
    int j, k;
    for(j=2; j<nextVal.size(); j++)
    {
        k = nextVal[j];      //先让 k 取 j 位的 next 值
        //1: 当 j 位和 k 位字符不相等时，nextVal[j]就取 next[j]的值，不操作

        while(p[j] == p[k])  //2: 当 j 位和 k 位字符相等时
        {
            k = nextVal[k];  //回溯直到 j 位和 k 位字符不再相等
            nextVal[j] = k;  //使 j 位的 nextVal = k
        }
    }
}

int main(void)
{
     string p="$abaabcac";
     cout << "模式串为: "  << endl;
     for(int i=1; i <= p.size()-1; i++)
         cout << p[i] << "  ";
     cout << endl;

    // 计算 next 函数
    vector<int> next;
    next.resize(p.size());
    get_next(p,next);
    // 计算 next 函数修正值
    vector<int> nextVal;
    get_nextVal(p, next, nextVal);

  for(int i=1;i<=nextVal.size()-1;i++)
     cout << nextVal[i] << "  ";
  cout << endl;
  return 0;
}
```

例题

例 1："abcabaa"

字符串 s	a	b	c	a	b	a	a
j	1	2	3	4	5	6	7
k			1->0	1->0	1	2	3->1
next[j]	0	1	0+1=1	0+1=1	1+1=2	2+1=3	1+1=2
k		0	1	1->0	2->1	3	3
nextval[j]	0	1	1	1->0	2->1	3	2

例 2："abcaabbabcab"

字符串 s	a	b	c	a	a	b	b	a	b	c	a	b
j	1	2	3	4	5	6	7	8	9	10	11	12
k			1->0	1->0	1	2->1	2	3->1->0	1	2	3	4
next[j]	0	1	1	1	2	2	3	1	2	3	4	5
k		1	1	1->0	2	2->1	3	1->0	2->1	3->1	4->0	5
nextval[j]	0	1	1	1->0	2	2->1	3	1->0	2->1	3->1	4->0	5

第五章 数组和广义表

基本概念名词

考数组怎么换算到一维矩阵的下标变换

稀疏矩阵：假设 m 行 n 列的矩阵含 t 个非零元素，则称**$\delta=t/(m*n)$**为稀疏因子；

通常认为 **$\delta<=0.05$** 的矩阵为稀疏矩阵

广义表

是线性表的推广，也称为列表，是 n 个单元素或子表所组成的有限序列

记作 **LS=(α_1, α_2,…, α_n)**

广义表运算：

注意算表尾是把这个表里的 head 去掉，括号保留

任何一个非空广义表 LS = (α1, α2, ⋯, αn) 均可分解为

表头 Head(LS) = α1 和
表尾 Tail(LS) = (α2, ⋯, αn) 两部分。

表头可能是原子，也可能是列表，表尾必定是列表

例如: D = (E, F) = ((a, (b, c)), F)

Head(D) = E　　Tail(D) = (F)

Head(E) = a　　Tail(E) = ((b, c))

Head(((b, c))) = (b, c)　Tail(((b, c))) = ( )

Head((b, c)) = b　Tail((b, c)) = (c)

Head((c)) = c　Tail((c)) = ( )

*注意：*列表 () 和 (()) 不同。

注意去线代复习一下矩阵的性质（对称矩阵，上三角矩阵，对角线矩阵）

例题

一个N阶对称矩阵，矩阵元为A_{ij}，将其下三角部分以行序为主序存放在一维数组M[0, n(n+1)/2-1]中，设矩阵最左上角矩阵元为A_{00}，则M[29]对应的矩阵元为___________。	第 6 行（从 0 开始计数）的最后一个元素 27 小于 29，因此 29 在(7,1)的位置；又因矩阵为对称矩阵，对应矩阵元是 A71 和 A17；
广义表A((x，(a ,b))， (x，(a，b)，y))，则运算Head(Head(Tail(A)))为_____________。 A．x B．(a, b) C．(x, (a, b)) D． A	Tail(A) = ((x, (a, b), y)) 注意两层括号 Head(Tail(A)) = (x, (a, b), y) Head(Tail(A))) = x

课堂小测验 1

1、数组 a[1.. 6][1.. 5] （无 0 行 0 列）以列序优先顺序存储，第一个元素 a[1][1]的地址为 1000，每个元素占 2 个存储单元，则 a[3][4]的地址是____。
(a) 1026
(b) 1038
(c) 1040
(d) 1046

2、设有一个 5 行 4 列的矩阵 A，采用行序优先存储方式，A[0][0]为第一个元素，其存储地址为 1000，A[2][2]的地址为 1040，则 A[3][0]的地址为__________。
(a) 1024
(b) 1048
(c) 1060
(d) 1096

3、设 10*10 的对称矩阵下三角保存 SA[1.. 55]中，其中 A[1][1]保存在 SA[1]中，A[5][3]保存在 SA[k]中，这里 k 等于__________。
(a) 12
(b) 13
(c) 14
(d) 15

4、表头和表尾均为空义表的广义表是___________。
(a) ( )
(b) (( ))
(c) ((( )))
(d) (( ),( ))

5、广义表（a,(b,c),d）的表长是______。
（a） 3
（b） 4
（c） 5
（d） 6

6、广义表((a,()),(b,(c)),(()))的深度是______。
（a） 3
（b） 4
（c） 5
（d） 6

7、广义表((a,b),(c),(d))的表尾是______。
（a） d
（b） (d)
（c） (a,b)
（d） ((c),(d))

答案：C B B B A A D

课堂小测验 2（暂无答案）

1、表头和表尾均为空义表的广义表是___________。
(a) ( )
(b) (( ))
(c) ((( )))
(d) (( ),( ))

2、广义表（a,(b,c),d）的表长是_______。
(a) 3
(b) 4
(c) 5
(d) 6

3、广义表((a,()),(b,(c)),(()))的深度是_______。
(a) 3
(b) 4
(c) 5
(d) 6

4、广义表((a,b),(c),(d))的表尾是_______。
(a) d
(b) (d)
(c) (a,b)
(d) ((c),(d))

自己做的：A A A D （经赵路路提醒，第一个改成 B，表尾是除了第一个元素之外的元素组成的表，表头是第一个元素）

第六章 树和二叉树（重点）

基本概念名词

二叉树性质：(节点/叶子/高度的数字关系需要掌握)

（1）在二叉树的第 i 层上至多有 2^{i-1} 个结点。($i\geq1$)

（2）深度为 k 的二叉树上至多含 2k-1 个结点（$k\geq1$）。

（3）任何一棵二叉树，若它含有 n0 个叶子结点、n2 个度为 2 的结点，则必存在关系式：

$$n_0 = n_2+1$$

满二叉树：指的是深度为 k 且含有 2k-1 个结点的二叉树。

完全二叉树

树中所含的 n 个结点，和满二叉树中编号为 1 至 n 的结点一一对应。

具有 n 个结点的完全二叉树的深度为 **$\lfloor \log_2 n \rfloor + 1$**

前序/中序/后序遍历中的“前、中、后”指的都是什么时间访问到根节点，递归遍历可直接写；

层序遍历顾名思义按层遍历，访问完上一层才访问下一层；

线索二叉树：在二叉链表的结点中增加两个标志域 LTag 和 RTag，分别指向遍历该树过程中的前驱结点和后继结点；填充这两个域的过程称为二叉树的线索化；

最优二叉树（哈夫曼树）：带权路径长度取最小值的树；

哈夫曼算法同离散数学里那个，这里不列了，可以看第六章作业整理的版本

需要掌握怎样用两种不同的遍历顺序列表确定树形；

例题

一棵完全二叉树的第六层上有 9 个结点，则结点总数是_____个。	先数到第 6 层再数到第 9 个，40
深度为 5 的完全二叉树拥有的最少结点数为_____	16
证明叶子结点数为 n 的哈夫曼树内部结点数为 n-1：	哈夫曼树没有度为 1 的结点，由于 n0=n2+1，所以 n2 = n0-1

非递归的前序/中序/后序/层序遍历代码

```
void PreOrderTraverse(BinTree b)     // 非递归的前序遍历（借助栈）
{
    InitStack(S);  //初始化创建栈
    BinTree p=b;  //p 为工作指针
    while(p||!isEmpty(s))
    {
        while(p)  //到最左下的孩子
        {
            printf(" %c ",p->date);  //先序先遍历结点
            Push(S,p);  //入栈
            p=p->lchild;
        }
        if(!isEmpty(s))  //在栈不为空的情况下，左孩子为空，弹出该结点，遍历右孩子
        {
            p=Pop(s);
            p=p->rchild;
        }
    }
}

void InOrderTraverse(BinTree b)      // 非递归的中序遍历（借助栈）
{
    InitStack(S);  //初始化创建栈
    BinTree p=b;  //p 为工作指针
    while(p||!isEmpty(s))
    {
        while(p)
        {
            Push(S,p);  //中序现将结点进栈保存
            p=p->lchild;
        }  //遍历到左下角尽头再出栈访问
        if(!isEmpty(s))  //在栈不为空的情况下，左孩子为空，弹出该结点，遍历右孩子
```

```
        {
            p=Pop(s);
            printf(" %c ",p->data);
            p=p->rchild;  //遍历右孩子
        }
    }
}

void PostOrderTraverse(BinTree b)       // 非递归的后序遍历（借助栈）
{
    InitStack(S);  //初始化创建栈
    BinTree p=b, r=NULL;  //p 为工作指针,辅助指针 r
    while(p||!isEmpty(s))
    {
        if(p)  //从根节点到最左下角的左子树都入栈
        {
            Push(S,p);  //中序现将结点进栈保存
            p=p->lchild;
        }
        else
        {
            GetTop(S,p);  //取栈顶，注意! 不是出栈!
            if(p->rchild && p->rchild!=r)
             //1.右子树还没有访问并且右子树不空，第一次栈顶
            {
                p=p->rchild;  //进入右子树
            }
            else  //2.右子树已经访问或为空，接下来出栈访问结点，第二次栈顶
            {
                p=Pop(s);
                printf(" %c ",p->data);
                r=p;  //指向访问过的右子树结点
                p=NULL;  //使 p 为空继续访问栈顶
            }
        }
    }
}

void LevelOrder(BiTree b)     // 非递归的层序遍历（借助队列）
{
    InitQueue(Q);  //初始化建立队列
    BinTree p;
    EnQueue(Q,b);  //根节点入队
    while(!isEmpty(Q))  //队列不空循环
    {
        DeQueue(Q,p);  //队头元素出队
        printf(" %c ",p->data);
          //左右孩子入队
        if(p->lchild!=NULL)
        {
            EnQueue(Q,p->lchild);
        }
```

```
        if(p->rchild!=NULL)
        {
            EnQueue(Q,p->rchild);
        }
    }
}
```

课堂小测验 1

1．一棵二叉树上有 1001 个结点，其中叶子结点的个数是（　　）
（a）250
（b）500
（c）254
（d）505
（e）以上答案都不对

2．已知一棵二叉树的前序遍历结果为 ABCDEF，中序遍历结果为 CBAEDF，则后序遍历的结果是（　　）
（a）CBEFDA
（b）FEDCBA
（c）CBEDFA
（d）不定

3．在二叉树中的先序序列、中序序列和后序序列中，所有叶子结点的先后顺序（　　）
（a）都不相同
（b）完全相同
（c）先序和中序相同，而与后序不同
（d）中序和后序相同，而与先序不同

4、树中所有结点的度等于所有结点数加（　　）
（a） 0
（b） 1
（c） -1
（d） 2

5、在一棵二叉树的二叉链表中，空指针域数（n+1）等于非空指针域数（n-1）加（　　）
（a） 2
（b） 1
（c） 0
（d） -1

7．二叉树的先序遍历序列和后序遍历序列正好相反，则该二叉树一定满足的条件是（　　）。
（a）任一结点无左孩子
（b）任一结点无右孩子
（c）空或只有一个结点
（d）高度等于其结点数

8．设某棵二叉树的中序遍历序列为 ABCD，先序遍历序列为 CABD，则后序遍历该二叉树得到序列为（　　）。
（a）BCDA
（b）BADC

(c) CDAB
(d) CBDA

答案：E A B C A D B

课堂小测验 2

1．判断线索二叉链表中*p 结点有右孩子结点的条件是（　　）。
(a) p!=NULL
(b) p->rchild!=NULL
(c) p->rtag==0
(d) p->rtag==1

2．基于中序线索化链表，其头结点指针为 head，对应的二叉树为空的判断条件是（　　）。
(a) head->ltag==0
(b) head->ltag==1
(c) head->lchild==head && head->rchild==head
(d) head==NULL

3、设森林 F 有 3 棵树，分别有 9、8 和 7 个结点，则 F 此排列次序转换成二叉树后根结点的右子树上结点的个数是（　　）。
(a) 7
(b) 15
(c) 16
(d) 17

4、给定一棵树的二叉链表存储结构，把这棵树转换为二叉树后，这棵二叉树的形态是（　　）。
(a) 唯一的
(b) 有多种
(c) 有多种，但根结点都没有左孩子
(d) 有多种，但根结点都没有右孩子

5．哈夫曼树中叶子结点数为 n，则内部结点数为（　　）。
A. n　　B. 2n-1　　C. n-1　　D. n-2

答案：C C B A C

第七章 图

基本概念名词

网：带权的图

假设图中有 n 个顶点，e 条边，则含有 **e=n(n-1)/2** 条边的无向图称作完全图；

含有 **e=n(n-1)** 条弧的有向图称作有向完全图；

若边或弧的个数 **e<nlogn**，则称作稀疏图，否则称作稠密图；

连通分量：若无向图为非连通图，则图中各个极大连通子图称作此图的连通分量；

强连通分量：若有向图不连通，则各个强连通子图称作它的强连通分量

生成树：假设一个连通图有 n 个顶点和 e 条边，其中 n-1 条边和 n 个顶点构成一个极小连通子图，该极小连通子图称为生成树；

图的储存结构主要分邻接矩阵（易判定是否有边/弧相连，易求各个点的度）和邻接表（节省储存空间，易于找到邻接点）两种，还有十字链表、三元数组等；

图的遍历算法 DFS 和 BFS

DFS（深度优先搜索）类似先序遍历，BFS（广度优先搜索）类似层序遍历；

<table>
<tr><th>DFS 的伪代码</th><th>BFS 的伪代码</th></tr>
<tr><td><pre>DFS(v):// v 可以是图中的一个顶点，也可
以是抽象的概念，如 dp 状态等
    visited[v] = true
    for each edge(u, v)
        if (!visited[v]) then
            DFS(u)</pre></td><td><pre>bfs(s):
    q = new queue()
    q.push(s)
    visited[s] = true
    while (!q.empty())
        u = q.pop()
        for each edge(u, v)
            if (!visited[v]) then
                q.push(v)
                visited[v] = true</pre></td></tr>
</table>

求最小生成树的 Prim 和 Kruskal 算法

Kruskal 是**加边**，贪心算法每次加入一个权最小的边；

Prim 是**加点**，每次加入一个和当前点距离最小的点；

<table>
<tr><th>Kruskal 的伪代码：</th><th>Prim 的伪代码：</th></tr>
<tr><td><pre>Kruskal:
sort edges into increasing order by
weight w
for each (u,v,w) in sorted edges:
    if u and v are not connected in
    the union-find set:
        connect u and v in the
        union-find set
        result += {(u,v,w)}
 return result</pre></td><td><pre>Prim:
choose an arbitrary node in V to be
the root
dis(root) = 0
for each node v in (V-{root})
      dis(v) = ∞
  rest = V
  while !rest.empty():
      cur = the node with the
      minimum dis in rest
      result = result+dis(cur)
      rest = rest-{cur}
      for each node v in adj(cur):
          dis(v) = min(dis(v),g(cur,v))
  return result</pre></td></tr>
</table>

```
void Kruskal(vector<DEdge> edges, int n)    // Kruskal 算法
{
    vector<int> uset;
    vector<DEdge> MST;
    int Cost = 0, e1, e2;
    MakeSet(uset, n);
    for (int i = 0; i < edges.size(); i++) //按权值从小到大的顺序取边
    {
        e1 = FindSet(uset, edges[i].V1);
        e2 = FindSet(uset, edges[i].V2);
        if (e1 != e2)
        //若当前边连接的两个结点在不同集合中，选取该边并合并这两个集合
        {
            MST.push_back(edges[i]);
            Cost += edges[i].Weight;
            uset[e1] = e2; //合并当前边连接的两个顶点所在集合
        }
    }
    cout << "Results of the Kruskal algorithm: " << endl;
    cout << "Cost = " << Cost << endl;
    cout << "Edges:" << endl;
    for (int j = 0; j < MST.size(); j++)
        cout << "<" << MST[j].V1 << ", " << MST[j].V2 << ">, Weight = " <<
MST[j].Weight << endl;
    cout << endl;
```

```
}

void Prim(vector<list<Node>> &Adj, int n, int m)    // Prim 算法
{
    priority_queue<DEdge, vector<DEdge>, greater<>> Q;  // 重载>操作符，实现顶
端元素最小的优先队列

    int Cost = 0;                    // 用于统计最小生成树的权值之和
    vector<DEdge> MST;               // 用于保存最小生成树的边
    vector<int> uset(n,0);  // 用数组来实现并查集
    DEdge E(0, 0, 0);
    for (int i = 0; i < n; i++)    //并查集初始化
        uset[i] = i;

    // 根据 Prim 算法，开始时选取结点 0，并将结点 0 与剩余部分的边保存在优先队列中
    // 循环中每次选取优先队列中权值最小的边，并更新优先队列
    for (auto it = Adj[0].begin(); it != Adj[0].end(); it++)
        Q.push(DEdge(0, it->v, it->w));

    while (!Q.empty())
    {
        E = Q.top();
        Q.pop();
        if (FindSet(uset, E.V2) != 0)
        {
            Cost += E.Weight;
            MST.push_back(E);
            uset[E.V2] = E.V1;
            for (auto it = Adj[E.V2].begin(); it != Adj[E.V2].end(); it++)
                if (FindSet(uset, it->v) != 0)
                    Q.push(DEdge(E.V2, it->v, it->w));
        }
    }
    cout << "Results of the Prim algorithm:" << endl;
    cout << "Cost = " << Cost << endl;
    cout << "Edges:" << endl;
    for (int j = 0; j < MST.size(); j++)
        cout << "<" << MST[j].V1 << ", " << MST[j].V2 << ">, Weight = " <<
MST[j].Weight << endl;
    cout << endl;
}
```

求最短路径（Dijkstra 算法）

步骤	第一步	第二步	第三步	第四步	第五步	第六步
b	15（a,b）	15（a,b）	15（a,b）	15（a,b）	15（a,b）	15（a,b）
c	2（a,c）					
d	12（a,d）	12（a,d）	2+4+5（a,c,f,d）	2+4+5（a,c,f,d）		
e		2+8（a,c,e）	2+8（a,c,e）			
f		2+4（a,c,f）				
g			2+4+10（a,c,f,g）	2+4+10（a,c,f,g）	2+4+5+3（a,c,f,d,g）	
vj	c	f	e	d	g	b
S	{a,c}	{a,c,f}	{a,c,f,e}	{a,c,f,e,d}	{a,c,f,e,d,g}	{a,c,f,e,d,g,b}

拓扑排序

建立一个序列，能保证在序列中顶点 Vi 在顶点 Vj 之前 ↔ 有向图 G 中从顶点 Vi 到顶点 Vj 有一条路径，则这一顶点线性序列(Vi1, Vi2 ,…,Vin)称作一个拓扑序列，建立过程称为拓扑排序；

AOE-网(Activity On Edge)

顶点表示事件，弧表示活动，权值表示活动持续时间的带权有向无环图

关键路径：路径长度最长的路径； 关键路径上的活动为关键活动，(该弧上的权值增加将使有向图上的最长路径的长度增加。)

例题

<table>
<tr><td>
已知如图所示的有向图，请给出该图的

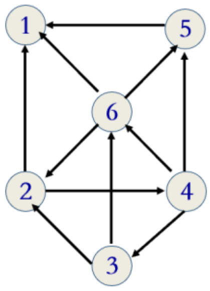
(1) 每个顶点的入/出度；

(2) 邻接矩阵；

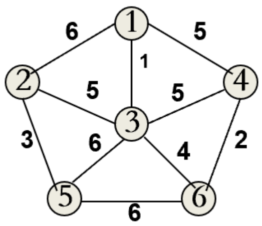
(3) 邻接表；

(4) 逆邻接表；

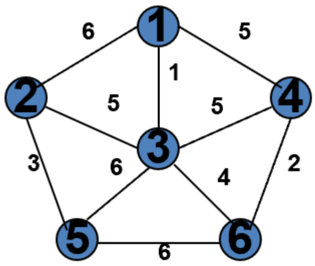
(5) 强连通分量；



</td><td></td></tr>
<tr><td>
Prim 算法构造最小生成树：



</td><td></td></tr>
<tr><td>
Kruskal 算法构造最小生成树：



</td><td></td></tr>
</table>

课堂小测验 1

1．图中，所有顶点的度数之和等于图的边数的（　　）倍。
(a) 1/2
(b) 1
(c) 2
(d) 4

2．已知图的邻接矩阵，则从顶点 0 出发按照深度优先遍历的结点序列是（　　）。

```
0  1  1  1  1  0  1
1  0  0  1  0  0  1
1  0  0  0  1  0  0
1  1  0  0  1  1  0
1  0  1  1  0  1  0
0  0  0  1  1  0  1
1  1  0  0  0  1  0
```

(a) 0 2 4 3 1 5 6
(b) 0 1 3 6 5 4 2
(c) 0 1 3 4 2 5 6
(d) 0 3 6 1 5 4 3

3．已知图的邻接矩阵同上题，则从顶点 0 出发按照广度优先遍历的结点序列是（　　）。
(a) 0 2 4 3 6 5 1
(b) 0 1 2 3 4 5 6
(c) 0 4 2 3 1 5 6
(d) 0 1 3 4 2 5 6

4．对某个无向图的邻接矩阵来说，下面哪个说法是正确的（　　　　）。
(a) 第 i 行上的非零元素个数和第 i 列的非零元素个数一定相等
(b) 矩阵中的非零元素个数等于图中的边数
(c) 第 i 行上，第 i 列上非零元素总数等于顶点 Vi 的度数
(d) 矩阵中非全零行的行数等于图中的顶点数

5．具有 10 个顶点的无向图至少有多少条边才能保证连通（　　　　）
(a) 9
(b) 10
(c) 11
(d) 12

6．一个有 n 个顶点和 n 条边的无向图一定是（　　　　）
(a) 连通的
(b) 不连通的
(c) 无环的
(d) 有环的

7．在图采用邻接表进行存储时，求最小生成树的 prim 算法的时间复杂度是（　　　）
(a) O(n)
(b) O(n+e)
(c) O(n^2)
(d) O(n^3)

8．设有无向图 G=(V，E)和 G’=(V’，E’)，如果 G’是 G 的生成树，则下面说法不正确的是（ ）
（a）G’是 G 的子图
（b）G’是 G 的连通分量（又名极大连通子图）
（c）G’是 G 的极小连通子图且 V’=V
（d）G’是 G 的无环子图

答案：C C B A A D B B

课堂小测验 2

1．可借助于__________判别有向图中是否存在回路。
（a）迪杰斯特拉算法
（b）FLOYD 算法
（c）拓扑排序算法
（d）PRIM 算法

2．下列关于工程计划的 AOE 网的叙述中，不正确的是__________。
（a）关键活动不按期完成，会影响整个工程的完成时间
（b）任何一个关键活动的提前完成，整个工程的完成时间都会提前
（c）所有关键活动都提前完成，会提前整个工程的完成时间
（d）某个关键活动提前完成，可能会提前整个工程的完成时间

3．关键路径是 AOE 网中（ ）。
（a）从源点到汇点的最短路径
（b）从源点到汇点的最长路径
（c）从源点到汇点边数最多的路径
（d）从源点到汇点边数最少的路径

4．使用弗洛伊德算法，求任意 2 个顶点的最短路径，该算法的时间复杂度为_________。
（a）O(n log n)
（b）O(log n * log n)
（c）O(n * n)
（d）O(n * n * n)

答案：C B B D

第九章 查找

基本概念名词

静态查找表：只做查询不做修改

动态查找表：除了查询还有插入和删除功能

顺序有序索引（分块查找）表：

若把所有 n 个数据元素分成 m 块，第一块中任一元素的关键字都小于第二块中任一元素的关键字，……，第 m-1 块中任一元素的关键字都小于第 m 块中的任一元素的关键字，但每一块中元素的关键字不一定有序；主要优点是可以先根据索引表缩小查找范围；

动态查找树：查找成功时返回关键值，查找不成功时则插入给定值，需要掌握这几种树节点/叶子/高度的数字关系；

二叉排序树（BST 树）

1. 是一棵空树；
2. 或者是具有如下特性的二叉树：

（1）若左子树不空，则左子树上所有结点的值均小于根结点的值；

（2）若右子树不空，则右子树上所有结点的值均大于根结点的值；

（3）左、右子树也都分别是二叉排序树。

查找过程：从根结点出发，沿着左分支或右分支逐层向下，直到遇空树或目标关键字；

平衡二叉树（AVL 树）

1. 是一棵空树
2. 或者是具有下列性质的二叉树：

（1）它的左子树或右子树都是平衡二叉树；

（2）左子树和右子树的深度之差的绝对值不超过 1；

（3）查找时，和给定值进行比较的次数 = log(n)

平衡旋转技术

遇到题就在草稿纸上画一下；

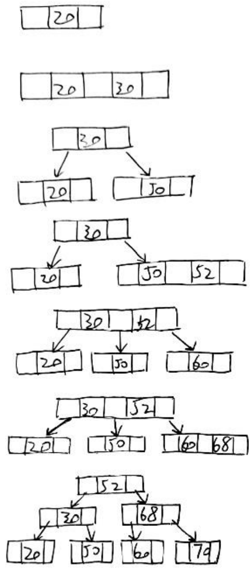
```
AVLNode *LL_Rotation(AVLNode *root) //left-left rotation
{
    AVLNode *temp = root->left;
    root->left = temp->right;
    temp->right = root;
    return temp;
}

AVLNode *RR_Rotation(AVLNode *root) //right-right rotation
{
    AVLNode *temp = root->right;
    root->right = temp->left;
    temp->left = root;
    return temp;
}

AVLNode *LR_Rotation(AVLNode *root)  //left-right rotation
{
    AVLNode *temp = root->left;
    root->left = LL_Rotation(temp);
    return RR_Rotation(root);
}

AVLNode *RL_Rotation(AVLNode *root)  //right-left rotation
{
    AVLNode *temp = root->right;
    root->right = RR_Rotation(temp);
    return LL_Rotation(root);
}
```

B-树

一棵 m 阶的 B-树，或为空树，或为满足下列特性的 m 叉树：

（1）树中每个结点至多有 m 棵子树；

（2）若根结点不是叶子结点，则至少有两棵子树；(至少含 1 个关键字)



（3）除根之外的所有非终端结点至少有⌈m/2⌉棵子树。(至少含⌈m/2⌉-1 个关键字)

（4）所有的非终端结点中包含下列信息数据：

$(n, A_0, K_1, A_1, K_2, A_2, \ldots, K_n, A_n)$

n 个关键字 K_i（$1 \le i \le n$）（$n < m$）

n 个指向记录的指针 D_i（$1 \le i \le n$）

n+1 个指向子树的指针 A_i（$0 \le i \le n$）

（5）所有的叶子结点都出现在同一层次上，并且不带信息（可以看作是外部结点或查找失败的结点，实际上这些结点不存在，指向这些结点的指针为空）

哈希表

关键字和位置一一相对，映射关系称为哈希函数；

六种常用的哈希函数构造方法

（1）直接定址法：H(key) = key 或 H(key) = a × key + b；

（2）数字分析法：假设每个关键字都是由 s 位数字组成（u1, u2, …, us)，以从中提取分布均匀的若干位、或它们的组合作为地址；

（3）平方取中法：以关键字的平方值的中间几位作为存储地址；

（4）折叠法：将关键字分割成若干部分，然后以它们的叠加和为哈希地址；一般用移位叠加(将分割后的每一部分的最低位对齐相加)，和间界叠加(从一端向另一端沿分割符来回折叠，对齐相加)；

（5）除余数法：H(key) = key % p

（6）随机数法：H(key) = Random(key)

四种常见的处理冲突的方法

（1）开放定址法：为产生冲突的地址 H(key) 求得一个地址序列：$H_0, H_1, H_2, \ldots, H_s$（$1\leq s\leq m-1$）

其中 $H_0 = H(key)$，$H_i = (H(key) + d_i)\% m$，$i=1, 2, \ldots, s$，d_i 为增量序列，m 为哈希表表长

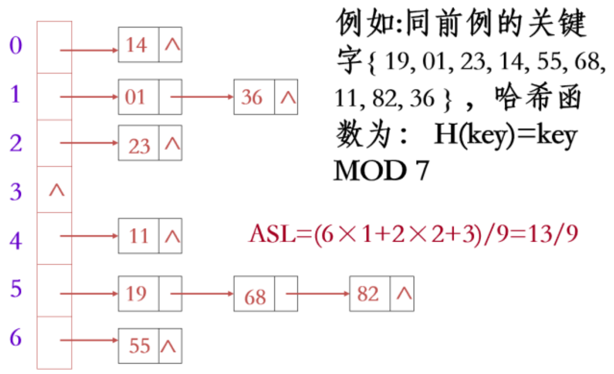
增量序列取法（增量序列应该具有完备性，可以取到哈希表所有位置）：

i. 线性探测再散列 $d_i = c\times i = 1,2,3,\ldots m-1$ ，最简单的情况 c=1

ii. 平方探测再散列（二次探测再散列）$d_i = 1^2, -1^2, 2^2, -2^2, \ldots, +k^2, -k^2$

iii. 随机探测再散列（双散列函数探测）d_i 是一组伪随机数列或 $d_i=i\times H_2(key)$

（2）链地址法：



（3）再哈希法（再哈希法）：当产生地址冲突时，计算另一个哈希函数地址

（4）建立公共溢出区：设向量 HashTable[0..m-1]为基本表存放记录；另设向量 OverTable[0..v]为溢出表，只要发生冲突，不管哈希地址是什么都填入溢出表。

计算平均查找长度（ASL）

$$ASL = \frac{\sum（查找到每个关键字所需的比较次数）}{关键字个数}$$

例题

<table>
<tr><td>一棵 5 阶的 B_树中某个结点的关键字个数最多为_________个。</td><td>4</td></tr>
<tr><td>已知一组关键字序列为：(4、5、7、2、1、3、6)，

请画出其构成的二叉排序树和平衡二叉树</td><td></td></tr>
<tr><td>已知一组关键字序列为（12，51，8，22，26，80，11，16，54，41)，其散列地址空间为[0,...,12]，若 Hash 函数定义为：H(key) = key MOD 13，采用线性探查法处理冲突，请给出它们对应的散列表
<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td></tr>
<tr><td>51</td><td>26</td><td>80</td><td>16</td><td>54</td><td>41</td><td></td><td></td><td>8</td><td>22</td><td></td><td>11</td><td>12</td></tr>
</table>
</td><td></td></tr>
</table>

课堂小测验 1

1．在关键字序列（10,20,30,40,50）中采用折半查找 20，依次与（　　）关键字进行了比较。
(a) 20
(b) 30,20
(c) 30,10,20
(d) 40,20

2．对于长度为 11 的有序表，按折半查找，在等概率情况下查找成功时，其平均查找长度是（）
(a) 1
(b) 2
(c) 3
(d) 4

3．在下列查找算法中，（　　）算法要求关键字序列是有序的。
(a) 顺序查找
(b) 折半查找
(c) 分块查找
(d) 二叉树查找

4．假设查找表长为 n，对于分块查找，如过采用顺序查找确定待查值可能所在的块，那么每块的关键字个数为（　　）时，分块查找的平均查找长度可以达到最佳。
(a) 根号 n 减 1
(b) 根号 n
(c) 根号 n 加 1
(d) ln(n)

答案 C C B B

课堂小测验 2

1、一棵 3 阶 B_树中含有 2047 个关键字，包含叶子结点层，该树的最大深度为（　　）。
（a）11
（b）12
（c）13
（d）14

2、设哈希标的地址范围是 0-17，哈希函数是 H(k)=K MOD 16，K 为关键字，用线性探测再散列法处理冲突，若输入关键字序列为：(10,24,32,17,31,30,46,47,40,63,49)。
构造哈希表，回答以下问题：
（1）查找关键字 63，需比较几次？
（2）查找关键字 60，需比较几次？
（3）设查找每个关键字的概率相等，求查找成功时的平均查找长度。

答案：B　6、1、1.55

第十章 排序

常见排序算法

需要理解排序方法的稳定性，特点、应用，性能；排序过程；时间复杂度分析

（有交换操作的一般不稳定）

简单选择排序不稳定!!!

排序方法	时间复杂度			空间复杂度	稳定性
	最好情况	最坏情况	平均情况		
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
折半插入排序	$O(n\log_2 n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
希尔排序			$O(n^{1.3})$	$O(1)$	不稳定
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	**不稳定**
快速排序	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	$O(\log_2 n)$	不稳定
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序	$O(d(n+rd))$	$O(d(n+rd))$	$O(d(n+rd))$	$O(n+rd)$	稳定

插入排序

```
void insertion_sort(int* a, int n) {    //插入排序
    for (int i = 2; i <= n; ++i) {
        int key = a[i];
        int j = i - 1;
        while (j > 0 && a[j] > key) {
            a[j + 1] = a[j];
            --j;
        }
        a[j + 1] = key;
    }
}
```

冒泡排序

```
void bubble_sort(int *a, int n) {    // 改进的冒泡排序
    bool flag = true;
    while (flag) {
        flag = false;
        for (int i = 0; i < n; ++i) {
            if (a[i] > a[i + 1]) {
                flag = true;
                int t = a[i];
                a[i] = a[i + 1];
                a[i + 1] = t;
            }
        }
    }
}
```

快速排序（不稳定）

```
void quickSort(vector<T> &list, int l, int r) { // 快速排序，已整合子集划分过程
    T temp;
    int pos = 0;
    int low = l;
    int high = r;
    if(l < r) {
        temp = list[l];
        while(low < high) {
            while((low < high) && (list[high] >= temp)) {
                high--;
            }
            if(low < high) {
                list[low] = list[high];
                low++;
            }
            while((low < high) && (list[low] < temp)) {
                low++;
            }
            if(low < high) {
                list[high] = list[low];
                high--;
            }
        }
        list[low] = temp;
        pos=low;
        quickSort(list, l, pos-1);
        quickSort(list, pos+1, r);
    }
}
```

简单选择排序（不稳定）

```
void selection_sort(int* a, int n) {    // 简单选择排序
    for (int i = 1; i < n; ++i) {
        int ith = i;
        for (int j = i + 1; j <= n; ++j) {
            if (a[j] < a[ith]) {
                ith = j;
            }
         }
    swap(a[i], a[ith]);
    }
}
```

堆排序（不稳定）

```
void sift_down(int arr[], int start, int end) {
  // 建立父结点指标和子结点指标
  int parent = start;
  int child = parent * 2 + 1;
  while (child <= end) {  // 子结点指标在范围内才做比较
      if (child + 1 <= end && arr[child] < arr[child + 1])
    // 先比较两个子结点大小，child 选择最大的
          child++;
      if (arr[parent] >= arr[child])
      // 如果父结点比子结点大，代表调整完毕，直接跳出函数
          return;
      else {
      // 否则交换父子内容，子结点再和孙结点比较
          swap(arr[parent], arr[child]);
          parent = child;
          child = parent * 2 + 1;
    }
  }
}

void heap_sort(int arr[], int len) {
  // 从最后一个节点的父节点开始 sift down 以完成堆化(heapify)
  for (int i = (len - 1 - 1) / 2; i >= 0; i--)
      sift_down(arr, i, len - 1);
  // 先将第一个元素和已经排好的元素前一位做交换，再重新调整（刚调整的元素之前的元
素），直到排序完毕
  for (int i = len - 1; i > 0; i--) {
      swap(arr[0], arr[i]);
      sift_down(arr, 0, i - 1);
  }
}
```

希尔排序（不稳定）

```
void shell_sort(T array[], int length) {
    int h = 1;
    while (h < length / 3) {
        h = 3 * h + 1;
    }
    while (h >= 1) {
        for (int i = h; i < length; i++) {
            for (int j = i; j >= h && array[j] < array[j - h]; j -= h) {
                swap(array[j], array[j - h]);
            }
        }
        h = h / 3;
```

```
    }
}
```

归并排序

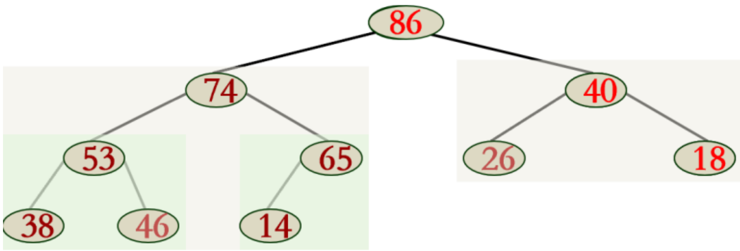
```
void merge(int ll, int rr) {
    // 用来把 a[ll.. rr - 1] 这一区间的数排序。 t 数组是临时存放有序的版本用的。
    if (rr - ll <= 1)
       return;
    int mid = ll + (rr - ll >> 1);
    merge(ll, mid);
    merge(mid, rr);
    int p = ll, q = mid, s = ll;
    while (s < rr) {
        if (p >= mid || (q < rr && a[p] > a[q])) {
            t[s++] = a[q++];
        }
       else
            t[s++] = a[p++];
    }
    for (int i = ll; i < rr; ++i)
        a[i] = t[i];
}
//关键点在于一次性创建数组，避免在每次递归调用时创建，以避免对象的无谓构造和析构。
```

基数排序

原理：将待排序的元素拆分为 k 个关键字（比较两个元素时，先比较第一关键字，如果相同再比较第二关键字……），然后先对第 k 关键字进行稳定排序，再对第 k-1 关键字进行稳定排序，再对第 k-2 关键字进行稳定排序……最后对第一关键字进行稳定排序，完成对整个待排序序列的稳定排序；需要借助一种**稳定算法**完成内层对关键字的排序；

例题

已知一组关键字序列为：(46, 74, 18, 53, 14, 26, 40, 38, 86, 65)，若采用堆排序方法进行排序，请构造初始堆 要求：最后输出结果按关键字非递减有序排列	

	
有 n 个互不重复的正整数型关键字进行堆排序，在构造的初始堆中，叶子结点数有多少？结点总数有多少？分支结点数有多少？	答案：叶子结点数为 $n-\lfloor \frac{n}{2} \rfloor$；结点总数为n；分支结点数为 $\lfloor \frac{n}{2} \rfloor$。

课堂小测验 1（课上没做过，没答案）

1．在第一趟排序之后，不能确保将数据表中某一个元素放在其最终位置上的排序算法是（　）。
（a）冒泡排序
（b）快速排序
（c）选择排序
（d）归并排序

2．对关键字序列（30，26，18，16，5，66），进行 2 遍（　）排序后得到序列（5，16，18，26，30，66）。
（a）插入
（b）冒泡
（c）选择
（d）归并

3．假设两个有序表长度分别为 n 和 m，将其归并成一个有序表最多需要（　）次关键字之间的比较。
（a）n+m-2
（b）n+m-1
（c）n+m
（d）n+m+1

4.对关键字序列（149,138,165,197,176,113,127），采用基数排序的第一趟之后所得结果为（　）。
（a）（113，127，138，149，165，176，197）
（b）(113，165，176，127，197，138，149)
（c）(113，165，176，197，127，138，149)
（d）(149，138，165，197，176，113，127)

5．对于下列排序，（　）的时间效率与关键字初始序列有直接关系。
（a）直接插入排序
（b）冒泡排序（没改进的）
（c）归并排序
（d）基数排序

自己做的：B D B C A(?)

课堂小测验 2

1．在下列排序算法中，（　）排序算法可能出现如下情况：在最后一趟排序之前，所有元素均不在其最终的位置上。
（a）插入
（b）冒泡
（c）快速
（d）堆

2．下列四种排序中，（　）的辅助空间复杂度是最高的。
（a）直接插入排序
（b）快速排序
（c）简单选择排序
（d）堆排序

3．在下列排序算法中，在待排序序列为有序的情况下，（　）的时间复杂度是 O(n*n)，其中 n 为待排序序列的数据元素个数。
（a）直接插入排序
（b）冒泡排序
（c）快速排序
（d）堆排序

4．假设待排序的表长为 n，那么创建堆需要时间复杂度为（　）。
（a）O(1)
（b）O(log n)
（c）O(n)
（d）O(n log n)

5．对于关键字序列(49,38,65,97,76,13,27,49)，完成创建的大根堆是（　）。
（a）(97,76,65,49,49,38,27,13)
（b）(97,76,65,49,49,13,27,38)
（c）(13,27,38,49,49,65,76,97)
（d）(97,65,76,49,49,13,27,38)

答案 A B C C B