1. 欧几里得算法：

```
Euclid(m,n)
while n≠0 do
    r=m mod n;
    m=n;
    n=r;
return m;
```

2. 连续整数检测算法：

```
Function(m,n)
t=min(m,n);
while t≥0 do
    if m mod t == 0 and n mod t == 0
        return t;
    --t;
```

3. 素数筛选法：

```
Function(n)
for p=2 to n do
    A[p]=p;
for p=2 to sqrt(n) do
    if A[p]≠0
        j=p*p;
    while j<n do
        A[j]=0;
        j=j+p;
i=0;
for p=2 to n do
    if A[p]≠0
        L[i]=A[p];
        ++i;
Return L;
```

4. 矩阵乘积：

```
Function(A[0..n-1,0..n-1],B[0..n-1,0..n-1])
for i=0 to n-1 do
    for j=0 to n-1 do
        C[i][j]=0;
        for k=0 to n-1 do
            C[i,j]=C[i,j]+A[i,k]*B[k,j];
return C;
```

5. 汉诺塔(A->B:A->C,C->B)：

```
void hanoi(int n,int A,int B,int C)
{
    if(n>0)
    {
        hanoi(n-1,A,C,B);
        move(A,B);
        hanoi(n-1,C,A,B);
    }
}
```

6. 选择排序：

(选择当前序列最小值与当前序列首值交换)

```
SelectionSort(A[0..n-1])
for i=0 to n-1 do
    min=i;
    for j=i+1 to n-1 do
        if A[j]<A[min]
            min=j;
    swap(A[i],A[min]);
```

7. 冒泡排序：

```
BubbleSort(A[0..n-1])
for i=0 to n-2 do
    for j=i to n-2-i do
        if A[j+1]<A[j]
            swap(A[j+1],A[j])
```

8. 字符串蛮力匹配：

(逐字符匹配，失败则从下一字符重新开始)

```
Function(T[0..n-1],P[0..m-1])
for i=0 to n-m do
    j=0;
    while j<m and P[j]=T[i+j] do
        ++j;
        if j==m return i;
return 0;
```

9. 蛮力平面距离最近两点：

```
Function(p)
d=∞
for i=0 to n-1 do
    for j=i+1 to n do
        d=min(d,sqrt((xi-xj)^2+(yi-yj)^2));
return d;
```

10. 深度优先搜索遍历：

```
DFS(G)
count=0;
for each vertex v in V do
    if v is marked with 0
        dfs(v)
dfs(v)//递归访问和 v 相连接未访问顶点
++count;mark v with count;
for each vertex w in V adjacent to v do
```

```
        if w is marked with 0
            dfs(w);
```

11. 广度优先搜索遍历：

```
BFS(G)
count=0;
for each vertex v in V do
    if v is marked with 0
        bfs(v)
bfs(v)
    ++count;mark v with count and
    init a queue with v;
while the queue is not empty do
    for each vertex w in V adjacent
    to the front vertex do
    if w is marked with 0
        ++count;mark w with count;
        add w to the queue;
    remove the front vertex from the queue;
```

12. 插入排序：

(往前找小于的值，插在其后)

```
InsertionSort(A[0..n-1])
for i=0 to n-1 do
    v=A[i];
    j=i=1;
    while j≥0 and A[j]>v do
        A[j+1]=A[j];
        --j;
    A[j+1]=v;
```

13. 拓扑排序：

执行一次 DFS 遍历；

并记住顶点变成死端(即退出遍历栈)的顺序

将该次序反序即得拓扑排序一个解

14. 生成排列算法：

```
将第一个排列初始化为带左方向标志 12..n;
while 存在一个可移动元素 do
    求最大移动元素 k;
    把 k 和它箭头指向元素互换;
    调转所有大于 k 的元素的方向;
    将新排列添加到列表中;
```

15. 反射格雷码：

```
BRG(n)
if n=1 表 L 包含位串 0 和 1;
else 调用 BRG(n-1)生成长度 n-1 位串列表 L1;
    把表 L1 倒序后复制给表 L2;
    把 0 加到表 L1 中的每个位串前面;
    把 1 加到表 L2 中的每个位串后面;
    把表 L2 添加到表 L1 后面得到表 L;
Return L;
```

16. 折半查找：

```
BinarySearch(A[0..n-1],k)
l=0,r=n-1;
while l≤r do
    m=(l+r)/2;
    if k==A[m] return m;
    else if k<A[m] r=m-1;
    else l=m+1;
return -1;
```

17. 俄式乘法：

n 为偶数：n*m=n/2*2m;

n 为奇数：n*m=(n-1)/2*2m+m;

n	m	
50	65	
25	130	130
12	260	
6	520	
3	1040	1040
1	2080	2080
		3250

18. 约瑟夫斯问题：

对 n 向左做一次循环移位；

19. 三重查找：

```
l = 0;r = n - 1;
while l<=r
    lmid = l+(r - l) / 3;//一二段分割点
    rmid = r-(r - l) / 3;//二三段分割点
    if k == A[lmid]
        return lmid;
    else if k == A[rmid]
        return rmid;
    else if k < A[lmid]   //K 在第一段
        r = lmid - 1;
    else if k < A[rmid]   //K 在第二段
        l = lmid + 1, r = rmid - 1;
    else             //K 在第三段
        l = rmid + 1;
return -1;
```

20. Lomuto 划分：

```
LomutoPartition(A[l..r])
```

```
p=A[l];s=l;
for i=l+1 to r do
      if A[i]<p
            ++s;swap(A[s],A[i]);
swap(A[l],A[s]);
return s;
```

21. 快速选择：

```
Quickselect(A[l..r],k)
s= LomutoPartition(A[l..r]);
if s=l+k-1 return A[s];
else if s<l+k-1 Quickselect(A[l..s-1],k)
else Quickselect(A[s+1..r],l+k-1-s)
```

22. 快速排序：

```
Quicksort(A[l..r])
if l<r
      s= LomutoPartition(A[l..r]);
      Quicksort(A[l..s-1]);
      Quicksort(A[s+1..r]);
```

23. 合并排序：

```
Mergesort(A[0..n-1])
if n>1
      copy A[0..n/2-1] to B[0..n/2-1];
      copy A[n/2..n-1] to C[0..n/2-1];
      Mergesort(B[0..n/2-1]);
      Mergesort(C[0..n/2-1]);
      Merge(B,C,A);
```

24. Merge(B[0..p-1],C[0..q-1],A[0..p+q-1])

```
i=0;j=0;k=0;
while i<p and j<q do
      if B[i]≤C[j]
            A[k]=B[i];++i;
      else A[k]=C[j];++j;
      ++k;
if i==p
      copy C[j..q-1] to A[k..p+q-1];
else copy B[i..p-1] to A[k..p+q-1];
```

25. 二叉树遍历：

```
peroder(BTNode *p)
if p!=null
      printf(p->data);
      perorder(p->lchild);
      perorder(p->rchild);
```

26. 高斯消去法：

```
Function(A[1..n,1..n],b[1..n])
for i=1 to n do A[i,n+1]=b[i];
for i=1 to n-1 do
      for j=i+1 to n do
            for k=n+1 downto i do
                  A[j,k]=A[j,k]-
A[I,k]*A[j,i]/A[i,i];
```

27. 构造堆：

```
Function(H[1..n])
for i=n/2 downto 1 do
      k=i;v=H[k];heap=false;
      while not heap and 2*k≤n do
            j=2*k;
            if j<n
                  if H[j]<H[j+1]
                        ++j;
            if v≥H[j] heap=true;
            else H[k]=H[j];k=j;
H[k]=v;
```

28. 霍纳法则：

```
Horner(P[0..n],x)
p=P[n];
for i=n-1 downto 0 do
      p=x*p+P[i];
return p;
```

29. 从左至右二进制幂：

```
LRBE(a,b(n))
Product=a;
for i=I-1 downto 0 do
      product=product*product;
      if bi=1 product=product*a;
return product;
```

30. 从右至左二进制幂：

```
RLBE(a,b(n))
term=a;
if b0=1 product=a;
else product=1;
for i=1 to I do
      term=term*term;
      if bi=1 product=product*term;
return product;
```

31. 比较计数排序：

```
Function(A[0..n-1])
for i=0 to n-1 do Count[i]=0;
for i=0 to n-2 do
```

```
    for j=i+1 to n-1 do
        if A[i]<A[j]
            ++Count[j];
        else ++Count[i];
for i=0 to n-1 do
    S[Count[i]]=A[i];
return S;
```

32. 分布计数排序：

```
Function(A[0..n-1],l,u)
for j=0 to u-l do D[j]=0
for i=0 to n-1 do ++D[A[i]-l];
for j=1 to u-l do D[j]=D[j-1]+D[j];
for i=n-1 downto 0 do
    j=A[i]-l;
    S[D[j]-1]=A[i];
    --D[j];
return S;
```

33. 填充移动表：

```
ShiftTable(P[0..m-1])
for i=0 to size-1 do Table[i]=m;
for j=0 to m-2 do Table[P[i]]=m-1-j;
return Table;
```

34. Horspool 字符串匹配算法：

```
HorspoolMatching(P[0..m-1],T[0..n-1])
ShiftTable(P[0..m-1]);//生成移动表
i=m-1;
while i≤n-1 do
    k=0;
    while k≤m-1 and P[m-1-k]=T[i-k] do
        ++k;
    if k==m return i-m+1;
    else i=i+Table[T[i]];
return -1;
```

35. 币值最大化问题：

```
CoinRow(C[1..n])
F[0]=0;F[1]=C[1];
for i=2 to n do
    F[i]=max(C[i]+F[i-2].F[i-1]);
return F[n];
```

36. 找零问题：

```
ChangeMaking(D[1..m],n)
F[0]=0;
for i=1 to n do
    temp=∞;j=1;
    while j≤m and i≥D[j] do
        temp=min(F[i-D[j]],tmp);
        ++j;
    F[i]=temp+1;
return F[n];
```

37. 硬币收集问题：

```
RobotCoinCollection(C[1..n,1..m])
F[1,1]=C[1,1];
for j=2 to m do
    F[1,j]=F[1,j-1]+C[1,j];
for i=2 to n do
    F[i,1]=F[i-1,1]+C[i,1];
    for j=2 to m do
        F[i,j]=max(F[i-1,j],F[i,j-1])+C[i,j];
return F[n,m];
```

38. 背包记忆化：

```
Function(i,j)
if F[i,j]<0
    if j<Weights[i]
        value=Function(i-1,j);
    else
        value=max(Function(i-1,j),
        Values[i]+ Function(i-1,j-
    Weights[i]));
    F[i,j]=value;
return F[i,j];
```

39. 最优二叉查找树：

```
OptimalBST(P[1..n])
for i=1 to n do
    C[i,i-1]=0;
    C[i,i]=P[i];
    R[i,i]=i;
C[n+1,n]=0;
for d=1 to n-1 do
    for i=1 to n-d do
        j=i+d;
        minval=∞
        for k=i to j do
            if C[i,k-1]+C[k+1,j]<minval;
                minval=C[i,k-1]+C[k+1,j];
                kmin=k;
        R[i,j]=kmin;
        sum=P[i];
        for s=i+1 to j do
```

sum=sum+P[s];

C[i,j]=minval+sum;

return C[1,n],R;

40.Warshall 算法：

Warshall(A[1..n,1..n])

$R^{(0)}$=A;

for k=1 to n do

for i=1 to n do

for j=1 to n do

$R^{(k)}$[i,j]=R^{k-1}[i,j] or

$R^{(k-1)}$[i,k] and $R^{(k-1)}$[k,j];

return $R^{(n)}$

41.Floyd 算法：

Floyd[W[1..n,1..n]

D=W;

for k=1 to n do

for i=1 to n do

for j=1 to n do

D[i,j]=min{D[i,j],D[i,k]+D[k,j]};

return D;

42.最小生成树 Prim 算法：

Prim(G)

V_T={v_0};E_T=空;

for i=1 to |V|-1 do

在所有的边(v,u)中，求权重最小的边

e*=(v*,u*);

使得 v 在 V_T 中，而 u 在 V-V_T 中;

V_T=V_T∪{u^*};E_T=E_T∪{e^*};

return E_T;

43.最小生成树 Kruskal 算法：

Kruskal(G)

按照边权重非递减顺序对集合 E 排序

E_T=空;ecounter=0;//初始化边顶点集合及其规模

k=0;//初始化已处理边数量

while ecounter<|V|-1 do

++k;

if E_T∪{e_{ik}}无回路

E_T=E_T∪{e_{ik}};++ecounter;

return E_T;

44.最短路径 Dijkstra 算法:

Dijkstra(G,s)

Init(Q);//顶点优先队列初始化为空

for V 中每一个顶点

d_v=∞;p_v=null;

Insert(Q,v,d_v);//初始化优先队列顶点优先级

d_s=0; Decrease(Q,s,d_s);//将 s 的优先级更新为 d_s

V_T=空;

for i=0 to |V|-1 do

u^*=DeleteMin(Q);//删除优先级最小的元素

V_T=V_T∪{u^*};

for V- V_T 中每一个和 u^*相邻的顶点 u do

if d_{u*}+w(u^*,u)<d_u

d_u= d_{u*}+ w(u^*,u);p_u=u^*;

Decrease(Q,u,d_u);

45.平分法求方程 x^3+x-1=0 的根：

do

mid=(a+b)/2;

t3=f(mid);

t1=f(a);t2=f(b);

if t1*t3>0 a = mid;

else b=mid;

while fabs(t3)>1e-2

return t3;

46.试位法求方程 x^3+x-1=0 的根：

do

x=(a*f(b)-b*f(a))/(f(b)-f(a));

y=f(x);

if y*fa > 0 a = x;fa = y;

else b = x;fb = y;

while fabs(y)>eps;

return x;

47.牛顿法求方程 x^3+x-1=0 的根：

f1 为原方程，f2 为其导数

do

x=x0-f1(x0)/f2(x0);

x0=x;

while fabs(f1(x))>eps;

return x;