第一部分　高精度计算

核心思想：将高精度数字转化为数组存储。

一、整型数字处理

```cpp
#include <iostream>
#include <cstdio>
using namespace std;
#define MAX 1000
int main()
{
    int n, temptop = 0;
    cin >> n;
    int temp[MAX];
     while (n / 10 != 0)
     {
         temp[temptop] = n % 10;
         temptop++;
         n /= 10;
     }
     temp[temptop] = n;
 }
```

二、使用字符串通过键盘输入并转化为数组

```cpp
gets(stra);
lena=strlen(stra);
memset(a, 0, sizeof(a));
for(i=0;i<lena;i++)
{
 a[i]=stra[i]-48;
}
```

三、实现两高精度数组加法运算

```cpp
lenc = 0;
x = 0;
while (lenc < lena || lenc < lenb)
{
 c[lenc] = a[lenc] + b[lenc] + x;
 x = c[lenc] / 10;
 c[lenc] %= 10;
 lenc++;
```

```
}
 c[lenc] = x;
 if (c[lenc] == 0)
  lenc--;
```

四、减法运算

在运算前要对被减数和减数大小进行比较

```
if((lena<lenb)||(lena==lenb&&strcmp(stra,strb)<0))
{
 strcpy(t,stra);
 strcpy(stra,strb);
 strcpy(strb,t);
 temp=lena;
 lena=lenb;
 lenb=temp;
 cout<<"-";
}
```

进行减法运算

```
for(lenc=0;lenc<lena;lenc++)
{
 if(a[lenc]<b[lenc])
 {
 a[lenc+1]=a[lenc+1]-1;
 a[lenc]=a[lenc]+10;
 }
 c[lenc]=a[lenc]-b[lenc];
}
 while(c[lenc]==0)    lenc--; //确保首位是0的情况下不输出
```

五、乘法运算

```
for (i = 0; i < lena; i++)
{
    for (j = 0; j < lenb; j++)
    {
        c[i + j] = c[i + j] + a[i] * b[j] + x;
        x = c[i + j] / 10;
        c[i + j] %= 10;
 }
 c[i + j] = x;
  x = 0;
}
```

```
lenc = i + j;
while (c[lenc] == 0)
 lenc--;
```

六、除法运算

```
x=0;
int pf=0;
for(i=0;i<lena;i++)
{
 x*=10;
 c[i]=(a[i]+x)/b;
 x=(a[i]+x)%b;
}
i=0;
while(c[i]==0&&i<lena)
{
 i++;
}
```

七、阶乘运算

```
#include <iostream>
#include <cstdio>
#include <cstring>
#define MAX 100000
using namespace std;
int main()
{
    int a[MAX], b[MAX], c[MAX];
    int n, i, j, k, x, m, num, lena, lenb, lenc;
     cin >> n;
     memset(a, 0, sizeof(a));
     memset(c, 0, sizeof(c));
     a[0] = 1;
     lena = 1;
     for (i = 2; i <= n; i++)
     {
         num = i;
         memset(b, 0, sizeof(b));
         lenb = 0;
         while (num / 10 != 0)
         {
```

```
            b[lenb] = num % 10;
            lenb++;
            num /= 10;
        }
        b[lenb++] = num;
        x = 0;
        for (j = 0; j < lena; j++)
        {
            for (k = 0; k < lenb; k++)
            {
                c[j + k] = c[j + k] + a[j] * b[k] + x;
                x = c[j + k] / 10;
                c[j + k] %= 10;
            }
            c[j + k] = x;
            x = 0;
        }
        lenc = j + k + 1;
        memset(a, 0, sizeof(a));
        for (m = 0; m < lenc; m++)
        {
            a[m] = c[m];
        }
        lena = lenc;
        memset(c, 0, sizeof(c));
    }
    while (a[lena] == 0)
        lena--;
    for (i = lena; i >= 0; i--)
    {
        cout << a[i];
    }
    cout << endl;
    return 0;
}
```

第二部分　数据排序

一、选择排序

```
//选择排序
```

```cpp
#include<iostream>
#include<cstdio>
#define MAX 100
using namespace std;
int main()
{
 int a[MAX],temp,k;
 cin>>a[0];
  for(int i=1;i<=a[0];i++)
  {
  cin>>a[i];
  }
  for(int i=1;i<=a[0];i++)
  {
  k=i;
  for(int j=i;j<=a[0];j++)
  {
  if(a[i]>=a[j])
  {
  k=j;
  }
  }
  if(k!=i)
  {
  temp = a[k];
  a[k] = a[i];
  a[i] = temp;
  }
  }
  for(int i=1;i<=a[0];i++)
  {
  cout<<a[i];
  cout<<" " ;
  }
  cout<<endl;
 }
```

二、冒泡排序

```cpp
//冒泡排序
#include<iostream>
#include<cstdio>
```

```
#define MAX 100
using namespace std;
int main()
{
 int a[MAX],temp;
 cin>>a[0];
  for(int i=1;i<=a[0];i++)
  {
  cin>>a[i];
  }
  for(int i=1;i<=a[0];i++)
  {
  for(int j=a[0];j>i;j--)
  {
  if(a[j]<a[j-1])
  {
  temp=a[j];
  a[j]=a[j-1];
  a[j-1]=temp;
  }
  }
  }
  for(int i=1;i<=a[0];i++)
  {
  cout<<a[i]<<" ";
  }
  cout<<endl;
  return 0;
 }
```

冒泡算法的改进算法

```
void bubblesort(int a[],int n)
{
 int b,exchanged=n-1,temp;
 while(exchanged!=0)
 {
 b=exchanged;exchanged=0;
 for(int i=0;i<b;i++)
 {
 if(a[i]>a[i+1]){
  temp=a[i];a[i]=a[i+1];a[i+1]=temp;
```

```
  exchanged=i;
  }
  }
  }
 }
```

冒泡算法改进的原因：

冒泡排序的运行时间只要浪费在循环语句上，循环的次数取决于带查找记录的个数，和查找值在数组的位置，每执行一次循环一次，所以造成了时间上的浪费。而改进后的冒泡排序，算法由两层嵌套循环，内层循环的执行次数取决于每一趟待排序区间的长度，也就是待排序记录的个数，外层循环的终止条件时在一趟排序过程中没有交换记录。是否有交换取决于相邻两个数的比较结果。

三、插入排序

```
void charu(int a[],int n)
{
 int temp,loc;
 for(int j=1;j<n;j++)
 {
 temp=a[j];
 loc=j;
 while(a[loc-1]>temp&&loc>0)
 {
  a[loc]=a[loc-1];
  loc--;
  }
  a[loc]=temp;
  }
 }
```

四、桶排序

```
void tong(int a[],int n)
{
 int tong[MAX],temp;
 memset(tong,0,sizeof(tong));
 for(int i=0;i<n;i++)
 {
 temp=a[i];
 tong[temp]++;
 }
  for(int j=0;j<MAX;j++)
  {
```

```
  while(tong[j]>0)
  {
  cout<<j<<" ";
  tong[j]--;
  }
  }
}
```

第三部分　分治法

设计思想：

1、划分：把规模为n的原问题划分为k个规模较小的子问题。

2、求解子问题：各子问题解法与原问题解法相同，可以用递归的方法求解各个子问题。

3、合并：各个子问题的解合并起来。

一、数字旋转方阵

```
#include<iostream>
#define MAX 1000
using namespace std;
int data[MAX][MAX];
void Full(int number,int begin,int size)
{
 int i,j,k;
 if(size==0)
 {
  return;
 }
 if(size==1)
 {
 data[begin][begin]=number;
 return;
 }
 i=begin;j=begin;
 for(k=0;k<size-1;k++)
 {
 data[i][j]=number;
 number++;
 i++;
 }
 for(k=0;k<size-1;k++)
 {
```

```
 data[i][j]=number;
 number++;
 j++;
 }
 for(k=0;k<size-1;k++)
 {
 data[i][j]=number;
 number++;
 i--;
 }
 for(k=0;k<size-1;k++)
 {
 data[i][j]=number;
 number++;
 j--;
 }
 Full(number,begin+1,size-2);
}
int main()
{
 int size;
 cin>>size;
 Full(1,0,size);
 for(int i=0;i<size;i++)
 {
 for(int j=0;j<size;j++)
 {
 cout<<data[i][j]<<"\t";
 }
 cout<<"\n";
 }
 return 0;
}
```

二、归并排序

```
void Merge(int r[],int r1[],int s,int m,int t)
{
 int i=s,j=m+1,k=s;
 while(i<=m&&j<=t)
 {
```

```
6  if(r[i]<=r[j])
7  {
8  r1[k++]=r[i++];
9  }
10  else
11  {
12  r1[k++]=r[j++];
13  }
14  }
15  while(i<=m)
16  {
17  r1[k++]=r[i++];
18  }
19  while(j<=t)
20  {
21  r1[k++]=r[j++];
22  }
23 }
24 void MergeSort(int r[],int s,int t)
25 {
26  int m,r1[1000];
27  if(s==t)
28  {
29  return;
30  }
31  else
32  {
33  m=(s+t)/2;
34  MergeSort(r,s,m);
35  MergeSort(r,m+1,t);
36  Merge(r,r1,s,m,t);
37  for(int i=s;i<=t;i++)
38  {
39  r[i]=r1[i];
40  }
41  }
42 }
```

三、快速排序

```
1 int Partition(int r[],int first,int end)
```

```
{
 int i=first,j=end;
 while(i<j)
 {
 while(i<j&&r[i]<=r[j]) j--;
 if(i<j)
 {
 int temp=r[i];
  r[i]=r[j];
  r[j]=temp;
  i++;
  }
  while(i<j&&r[i]<=r[j]) i++;
  if(i<j)
  {
  int temp=r[i];
  r[i]=r[j];
  r[j]=temp;
  j--;
  }
  }
  return i;
}
void QuickSort(int r[],int first,int end)
{
 int pivot;
 if(first<end)
 {
 pivot=Partition(r,first,end);
 QuickSort(r,first,pivot-1);
 QuickSort(r,pivot+1,end);
 }
}
```

四、最大子段和

```
int MaxSum(int a[],int left,int right)
{
 int sum=0,midsum=0,leftsum=0,rightsum=0;
 int center,s1,s2,lefts,rights;
 if(left==right)
```

```
{
sum=a[left];
}
else{
 center=(left+right)/2;
 leftsum=MaxSum(a,left,center);
 rightsum=MaxSum(a,center+1,right);
 s1=0;lefts=0;
 for(int i=center;i>=left;i--)
 {
 lefts+=a[i];
 if(lefts>s1){
 s1=lefts;
 }
 }
 s2=0;rights=0;
 for(int j=center+1;j<=right;j++)
 {
 rights+=a[i];
 if(rights>s2){
 s2=rights;
 }
 }
 midsum=s1+s2;
 if(midsum<leftsum)
 {
 sum=leftsum;
 }
 else{
 sum=midsum;
 }
 if(sum<rightsum)
 {
 sum=rightsum;
 }
 }
 return sum;
}
```

五、棋盘覆盖问题

```
void ChessBoard(int tr,int tc,int dr,int dc,int size)
{
 int s,t1;
 if(size==1) return;
 t1=++t;
 s=size/2;
 if(dr<tr+s&&dc<tc+s)
 ChessBoard(tr,tc,dr,dc,s);
 else{
  board[tr+s-1][tr+s-1]=t1;
  ChessBoard(tr,tc,tr+s-1,tc+s-1,s);
  }
  if(dr<tr+s&&dc>=tc+s)
  ChessBoard(tr,tc+s,dr,dc,s);
  else{
  board[tr+s-1][tc+s]=t1;
  ChessBoard(tr,tc+s,tr+s-1,tc+s,s);
  }
  if(dr>=tr+s&&dc<tc+s)
  ChessBoard(tr+s,tc,dr,dc,s);
  else{
  board[tr+s][tc+s-1]=t1;
  ChessBoard(tr+s,tc,,tr+s,tc+s-1,s);
  }
  if(dr>=tr+s&&dc>=tc+s)
  ChessBoard(tr+s,tc+s,dr,dc,s);
  else{
  board[tr+s][tc+s]=t1;
  ChessBoard(tr+s,tc+s,tr+s,tc+s,s);
  }
}
```

第四部分　减治法

设计思想

1、原问题的解只存在于其中一个较小规模的子问题中

2、原问题的解与其中一个较小规模的解之间存在某种对应关系

一、两个序列的中位数

```
int SearchMid(int A[],int B[],int n)
{
```

```
int s1=0,e1=n-1,s2=0,e2=n-1;
int mid1,mid2;
while((s1<e1)&&(s2<e2))
{
mid1=(s1+e1)/2;
mid2=(s2+e2)/2;
if(A[mid1]==B[mid2])
 {
 return A[mid1];
 }
 if(A[mid1]<B[mid2])
 {
 if((s1+e1)%2==0)
 {
 s1=mid1;
 }
 else{
 s1=mid1+1;
 }
 e2=mid2;
 }
 else{
 if((s2+e2)%2==0)
 {
 s2=mid2;
 }
 else{
 s2=mid2+1;
 }
 e1=mid1;
 }
 }
 if(A[s1]<B[s2])
 {
 return A[s1];
 }
 else{
 return B[s2];
 }
}
```

二、二叉查找树

```
struct BiNode
{
 int data;
 BiNode *lchild,*rchild;
};
BiNode *SearchBST(BiNode *root,int k)
{
 if(root==NULL) return NULL;
 else if(root->data==k) return root;
  else if(k<root->data)
  return SearchBST(root->lchild,k);
  else
  return SearchBST(root->rchild,k);
 }
BiNode *InsertBST(BiNode *root,int data)
{
  if(root==NULL)
  {
  root=new BiNode;
  root->data = data;
  root->lchild=root->rchild=NULL;
  return root;
  }
  if(data<=root->data)
  root->lchild=InsertBST(root->lchild,data);
  else
  root->rchild=InsertBST(root->rchild,data);
  return root;
 }
BiNode *createBST(int a[],int n)
{
  BiNode *T=NULL;
  for(int i=0;i<n;i++)
  T=InsertBST(T,a[i]);
  return T;
 }
```

三、假币问题

```
int Coin(int a[],int low,int high,int n)
```

```
{
 int i,num1,num2,num3;
 int add1=0,add2=0;
 if(n==1)
 {
 return low+1;
 }
 if(n%3==0)
  {
  num1=num2=n/3;
  }
  else{
  num1=num2=n/3+1;
  }
  num3=n-num1-num3;
  for(i=0;i<num1;i++)
  {
  add1=add1+a[low+i];
  }
  for(i=num1;i<num1+num2;i++)
  {
  add2=add2+a[low+i];
  }
  if(add1<add2)
  {
  return Coin(a,low,low+num1-1,num1);
  }
  else if(add1>add2)
  {
  return Coin(a,low+num1,low+num1+num2-1,num2);
  }
  else{
  return Coin(a,low+num1+num2,high,num3);
  }
}
```

第五部分　动态规划法

设计思想：

1、划分子问题：将原问题分解为若干个子问题，每个子问题对应一个决策阶段，并且子问题之间具有重叠关系。

2、确定动态规划函数（关键）：根据子问题之间重叠关系找到子问题满足的递推关系式（动态规划函数）

3、填写表格：设计表格，自底向上计算各个子问题的解并填表

一、数塔问题

```
int DataTower(int n)
{
 int maxAdd[n][n],path[n][n];
 int i,j;
 for(j=0;j<n;j++)
 {
 maxAdd[n-1][j]=d[n-1][j];
 }
 for(i=n-2;i>=0;i--)
  {
 for(j=0;j<=i;j++)
 {
 if(maxAdd[i+1][j]>maxAdd[i+1][j+1])
 {
 maxAdd[i][j]=d[i][j]+maxAdd[i+1][j];
 path[i][j]=j;
 }
 else{
 maxAdd[i][j]=d[i][j]+maxAdd[i+1][j+1];
 path[i][j]=j+1;
 }
 }
 }
 cout<<"路径为:"<<d[0][0];
 j=path[0][0];
 for(i=1;i<n;i++)
 {
 cout<<"-->";
 cout<<d[i][j];
 j=path[i][j];
 }
 cout<<endl;
 return maxAdd[0][0];
```

```
}
```

二、最短路径

```
int CreatGraph()
{
 int i,j,k;
 int weight;
 int vnum,arcnum;
 cout<<"please input the point number and the line number:";
 cin>>vnum>>arcnum;
 for(i=0;i<vnum;i++)
 {
  for(j=0;j<vnum;j++)
  {
  arc[i][j]=MAX;
  }
  }
  for(k=0;k<arcnum;k++)
  {
  cout<<"please input two point and the length:";
  cin>>i>>j>>weight;
  arc[i][j]=weight;
  }
  return vnum;
}
int BackPath(int n)
{
 int i,j,temp;
 int cost[n],path[n];
 for(i=1;i<n;i++)
 {
 cost[i]=MAX;
 path[i]=-1;
 }
 cost[0]=0;
 path[0]=-1;
 for(j=1;j<n;j++)
 {
 for(i=j-1;i>=0;i--)
 {
```

```
 if(arc[i][j]+cost[i]<cost[j])
 {
 cost[j]=arc[i][j]+cost[i];
 path[j]=i;
 }
 }
 }
 cout<<n-1;
 i=n-1;
 while(path[i]>=0)
 {
 cout<<"<-"<<path[i];
 i=path[i];
 }
 cout<<endl;
 return cost[n-1];
}
```

第六部分　贪心算法

设计思想：

将复杂问题分解为一系列较为简单的局部最优选择，每一步选择都是对当前解的一个拓展，直到获得问题完整解。

在解决问题策略上目光短浅，只根据已有信息做出选择。

一、埃及分数

```
int gongyueshu(int m,int n)
{
 int r=m%n;
 while(r!=0)
 {
 m=n;
 n=r;
 r=m%n;
 }
 return n;
}
void anjiFenshu(int a,int b)
{
 int e,r;
 cout<<a<<"/"<<b<<"=";
```

```
 do{
 e=b/a+1;
 cout<<"1/"<<e<<"+";
 a=a*e-b;
 b=b*e;
 r=gongyueshu(b,a);
 if(r>1)
 {
 a=a/r;b=b/r;
 }
 }while(a>1);
 cout<<"1/"<<b<<endl;
 return;
}
```

二、TSP问题

```
int TSP1(int w)
{
 int edgeCount=0,TSPLength=0;
 int min,u,v;
 int flag[n]={0};
 u=w;flag[w]=1;
 while(edgeCount<n-1)
 {
 min=100;
 for(int j=0;j<n;j++)
 {
 if((flag[j]==0)&&(arc[u][j]!=0)&&(arc[u][j]<min))
 {
 v=j;min=arc[u][j];
 }
 }
 TSPLength+=arc[u][v];
 flag[v]=1;edgeCount++;
 cout<<u<<"-->"<<v<<endl;
 u=v;
 }
 cout<<v<<"-->"<<w<<endl;
 return(TSPLength+arc[u][w]);
}
```