## 一、最接近中位数的k个数

#### 1.理解问题

给定由n个互不相同的数组成的集合S，及一个正整数k<=n，试设计一个O（n）时间算法找出S中最接近S的中位数的k个数。

输入：集合大小n，正整数k（k<=n），集合中的每个元素

示例：9 3 56 32 70 65 36 6 21 41 51

输出：41 36 51

#### **2.**算法设计

因为使用可以在线性时间内可以找到无序数组中第K小的元素。

1. 使用该算法找到中位数mid
2. 设置数组abs 放置原数组的每个元素和mid之差的绝对值
3. 使用该算法求abs中第k小的数 num（abs中的元素不能重复）
4. 输出abs中元素<=num的元素就是答案

#### 算法代码与分析

#include<iostream>

#include<cstdio>

#include<random>

using namespace std;

int n,k,len;

int partition(int a[],int p,int r);

void Swap(int &a,int &b);

int RandomPartition(int a[],int p,int r);

int Random(int x, int y);

int RandomSelect(int a[],int p,int r,int k);

//将数组首元素a[p]作为基准数将数组分割

int partition(int a[],int p,int r){

    int i=p;//首元素

    int j=r+1;//尾元素的下一个元素

    int x=a[p];//首元素是基准数

    while(1)

    {

        //i从基准数右边的元素开始找，直到找到第一个>=基准数的元素

        while (a[++i]<x && i<r);

        while (a[--j]>x);//j从尾元素开始找，直到找到第一个<=基准数的元素

        //如果i>=j说明基准数的位置已经找到，为j

        if (i>=j)

        {

            break;

        }

        Swap(a[i], a[j]);

    }

    a[p]=a[j];//将基准数归位

    a[j]=x;

    return j;//返回基准数位置

}

void Swap(int &a,int &b)

{

    int tmp;

    tmp=a;

    a=b;

    b=tmp;

}

int RandomPartition(int a[],int p,int r)

{

    int i = Random(p,r);

    Swap(a[i], a[p]);

    return partition(a, p, r);

}

int Random(int x, int y)

{

    return x+rand()%(y-x);

}

int RandomSelect(int a[],int p,int r,int k)

{

    //数组被分割只剩下一个元素就是第k小的元素

    if (p==r)

    {

        return a[p];

    }

    int i = RandomPartition(a,p,r);//随机数，分成小于等于的数组，和大于的数组

    len = i-p+1;//较小数组的长度

    if (k<=len)

    {

        return RandomSelect(a,p,i,k);

    }else return RandomSelect(a,i+1,r,k-len);

}

int main(){

    cout<<"length n:";

    cin>>n;

    cout<<"array a: ";

    int \*a=new int[n];

    for(int i=0;i<n;i++)

    {

        cin>>a[i];

    }

    cout<<"k:";

    cin>>k;

    int mid=RandomSelect(a,0,n-1,(n+1)/2); //整个数组的中位数

    int \*d=new int[n];

    for (int i = 0; i < n; i++)  //数组中每个数与中位数之差的绝对值

    {

        d[i]=abs(a[i]-mid);

    }

    //绝对值数组第K小的数

    cout<<endl<<"answer: ";

    int res = RandomSelect(d,0,n-1,k);

    for (int i = 0; i < n;++i)

    {

        if (abs(a[i]-mid)<res)

        {

            cout<<a[i]<<' ';

            k--;

        }

    }

    for (int i = 0; i < n; ++i)

    {

        //与中位数之差的绝对值等于第k小的绝对值的数都符合要求(这里要求绝对值不重复)

        if (abs(a[i] - mid) == res)

        {

            if (!k){    break;  }

            cout<<a[i]<<' ';

            k--;

        }

    }

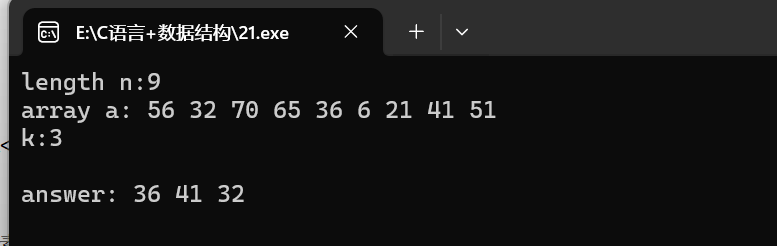
    cout<<endl;

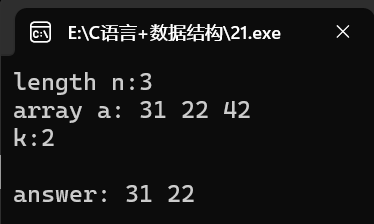
    delete[] a;

    delete[] d;

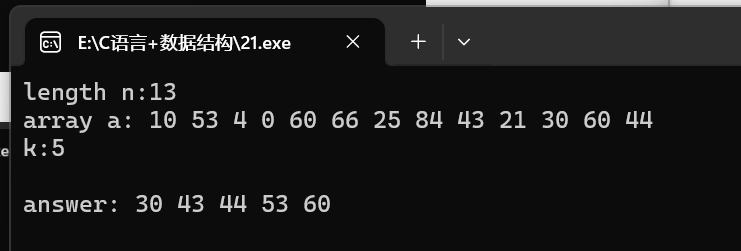
    return 0;

}









#### 性能分析

对RandomSelect算法，该算法划分的基准是随机的，最好情况下的时间复杂度为O ( n ) ，而最坏情况下的时间复杂度为O ( n^2 ) ，但这种情况在随机划分的过程中发生的的概率极小，因此RandomSelect算法的平均时间复杂度为O ( n )。

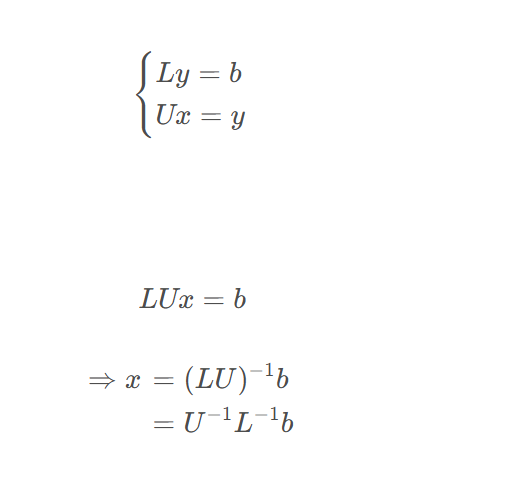
使用该方法找中位数、计算绝对值数组，找绝对值数组中第k小的数，输出符合要求的k个数的时间复杂度均为O ( n )，因此求邻近中位数的k个数的整个算法的时间复杂度同样为O ( n )。

## LU分解

LU分解是矩阵分解的一种，将一个矩阵分解为一个**下三角矩阵和一个上三角矩阵的乘积**，有时需要再乘上一个置换矩阵。

LU分解可以被视为**高斯消元法**的矩阵形式。在数值计算上，LU分解经常被用来解线性方程组、且在求逆矩阵和计算行列式中都是一个关键的步骤。

LU主要用来加快求解方程组的速度，常用于二次型矩阵求解。LU分解是接近中小规模、稠密矩阵的最好方法。

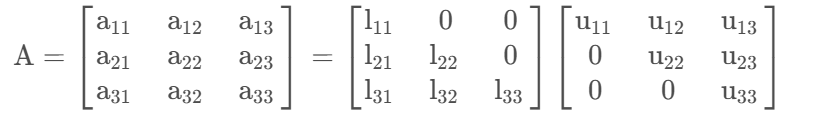


实际应用中，矩阵A 往往是固定的，而右侧向量b经常会替换。当矩阵A被分解后，L，U可充分利用，提高效率。

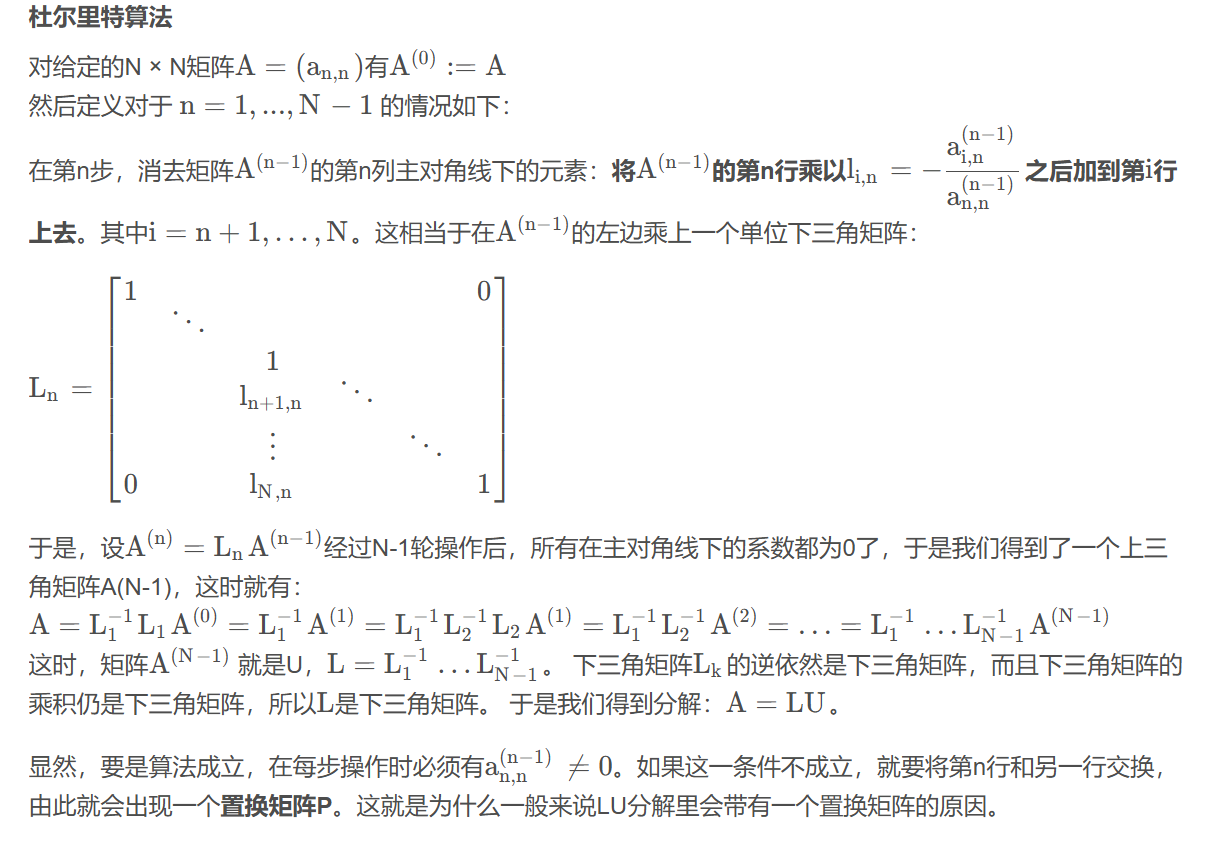
#### 1.理解问题

在数值线性代数中，LU分解广泛用于求解线性方程、计算矩阵行列式以及求逆矩阵等问题。调研学习LU分解方法以及实现算法。

#### **2.**算法设计

对于方阵 A，A的LU分解是将它分解成一个下三角矩阵 L 与上三角矩阵 U 的乘积，也就是 A = L U。

LU分解在本质上是高斯消元法的一种表达形式。实质上是将**A通过初等行变换变成一个上三角矩阵，其变换矩阵就是一个单位下三角矩阵**。这正是所谓的杜尔里特算法（Doolittle algorithm），从下至上地对矩阵A做初等行变换，将对角线左下方的元素变成零，然后再证明这些行变换的效果等同于左乘一系列单位下三角矩阵，这一系列单位下三角矩阵的乘积的逆就是L矩阵，它也是一个单位下三角矩阵。



推广：

**LDU 分解**

A=◣×⋱×◥

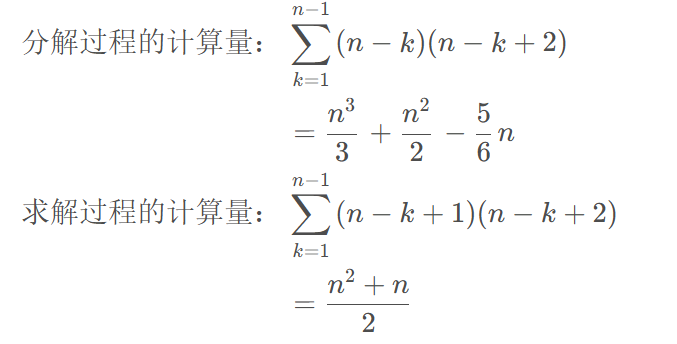
方阵 A 的LDU分解是是将它分解成一个单位下三角矩阵 L、对角矩阵 D 与单位上三角矩阵 U 的乘积，即A = L D U

其中单位上、下三角矩阵是指对角线上全是 1 的上、下三角矩阵。事实上，LDU 分解可以推广到 A 是一般的矩阵，而非方阵。

#### 性能分析

LU分解与高斯分解的对比：

高斯分解法的乘法计算量为



LU分解法的乘法计算量为：

