**4.2**

T(n)=2\* T(n/2)+c

= 2\* (2\*T(n/4)+c)+c

=(2^2)\* T(n/(2^2))+c(2+1)

=(2^k)T(n/(2^k))+c((2^k-1)+…+1)

=c((2^(k+1))-1)

=c(n-1)

=O(n)

**4.5 （编程题）**

算法时间复杂度为O(3/2n)

算法 MaxMin(A[*l..r*],Max,Min)

输入：数值数组A[*l..r*]

输出：最大值Max

if(*r*=*l*) Max=A[*l*]；

else

if *r*－*l*=1

if A[*l*]≤A[r]

Max=A[r];

              else

Max=A[*l*];

else //r－*l*>1

Max(A[*l*,(*l*+r)/2],Max1);

Max(A[(*l*+r/)2..r],Max2);

if Max1＜Max2   Max= Max2

#include <stdio.h>

void getmax(int a[],int \*max,int left,int right)  {

    if(a == NULL || left > right){

        return ;

    }

    if(left == right){

        \*max = MAX(\*max,a[left]);

        return ;

    }

    else if(right-left == 1){

        if(a[right] > a[left]){

            \*max = MAX(\*max,a[right]);

            return ;

        }

        else{

            \*max = MAX(\*max,a[left]);

            return ;

        }

    }

    getmax(a,max,left,(left+right)/2);

    getmax(a,max,(left+right)/2+1,right);

}

int main(int argc,char \*argv[])

{

    int a[] = {1,9,4,7,3,8,4};

    int temp1 = a[0];

    int \*max = &temp1;

    get\_min\_max(a,max,0,sizeof(a)/sizeof(a[0])-1);

    printf("max = %d",\*max);

    return 0;

}

**4.11（编程题）**

分析：因为矩阵的行和列都是递增的，所以整个矩阵的对角线上的数字也是递增的，故可以在对角线上进行二分查找，即以该元素在所在列和行为鞍点，即行最大列最小或者列最大行最小，每次与最右上角数字比较，若小之，排除次列；若大之，排除此行，实现快速查找 ,比如排除掉左上和右下的两个矩形中的元素，而在左下和右上的两个矩形继续递归查找

输入：n\*n的矩阵a[i][j] ，比较的数字为k

输出：是否存在k在矩阵中，true或者false

1. 取最左下角的元素并记为a，初始点为a[n-1][0]
2. if a>k,表明a所在的行都大于a，肯定不在此行上，只能往列去寻找，i--
3. if a<k, 表明a所在的列都小于k，只能往行去寻找，j—-
4. if a=k ，return true；
5. if i<0 or j>n，return false;

最差情况时间复杂度为2n-1

实现：

#include "stdio.h"  
int M[n][n]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};  
void getx(int x){  
  int i,j,flag=0;  
  if(x<M[0][0] || x>M[n-1][n-1]) printf("The number x is error!");  
  for(i=0;i<n;i++){  
  if(flag==1) break;  
  if(x<=M[i][n-1]){  
    for(j=0;j<n;j++)  
      if(x==M[i][j]){  
 flag=1;printf("在 M[%d][%d]中找到x/n",i,j);break;}  
 }  
  }  
}  
int main(){  
 int i,j,a;  
  printf("/nEnter the number x:");scanf("%d",&a);  
  for(i=0;i<n;i++)  
  {  for(j=0;j<n;j++)  printf("%4d",M[i][j]);  
    printf("/n");  
  }  
}

**5.3（编程题）**

**时间复杂度O(nlogn)**

//三个函数说明RangeBinarySearch 、BinarySearchL、BinarySearchR

#include<iostream>   
using namespace std;

void RangeBinarySearch(int a[], int l, int u, int b,int e){  
 if(b <= e){  
  if(l > u) {  
   swap(l,u);  
  }  
  int mid = (b+e)/2;  
  if(a[mid] < l ) {  
   RangeBinarySearch(a,l,u,mid+1,e);  
  }  
  else if(a[mid] > u ) {  
   RangeBinarySearch(a,l,u,b,mid-1);  
  }  
  else{  
   int b1 = BinarySearchL(a,b,mid,l);  //左边界  
   int e1 = BinarySearchR(a,mid,e,u);  //右边界  
   for(int i=b1; i<=e1; ++i) {  
    cout<<a[i]<<' ';  
   }  
   cout<<endl;  
   return;  
  }  
 }  
 else{  
  cout<<"there is no values fall between "<<l<<"and "<<u<<endl;  
  return;  
 }  
}

int BinarySearchL(int a[],int b,int e,int k) {  
 if(b <= e) {  
  int mid = (b+e)/2;  
  if(a[mid] == k) {  
   while((mid>=b) && (a[mid] == k)) {  
     mid--;  
   }  
   return ++mid;  
  }  
  if(a[mid] > k) {  
   if(mid == b)  
    return mid;  
   else if(a[mid-1]<k)  
    return mid;  
   else  
    return BinarySearchL(a,b,mid-1,k);

  }  
  if(a[mid] < k) {  
   if(a[mid+1]>k)  
    return mid+1;  
   else  
    return BinarySearchL(a,mid+1,e,k);  
  }  
 }  
 else{  
  return b;  
 }  
}

int BinarySearchR(int a[],int b,int e,int k) {  
 if(b <= e) {  
  int mid = (b+e)/2;  
  if(a[mid] == k){  
   while((mid<=e) && (a[mid] == k))  
   {  
     mid++;  
   }  
   return --mid;  
  }  
  if(a[mid] < k) {  
   if(mid == e)  
    return mid;  
   else if(a[mid+1]>k)  
    return mid;  
   else  
    return BinarySearchR(a,mid+1,e,k); }  
  if(a[mid] > k) {  
   if(a[mid-1]<k)  
    return mid-1;  
   else  
    return BinarySearchR(a,b,mid-1,k);  
  }  
 }  
 else{  
  return e;  
 }  
}  
int main(){  
 int a[10] = {1,3,4,5,6,7,9,13,15,17,19};  
 RangeBinarySearch(a,27,29,0,9);  
 return 0;  
}

**5.8 （编程题）**

我想推导一下此类问题通解：

若桌上有n个火柴棒,甲乙两人按照如下规则轮流取走这些火柴棒:

1)每次只允许取走1~m个火柴棒;

2)最后一次取走火柴棒的人获胜;

1. **：若甲先取,且甲想取胜**

甲先取若干个火柴棒,之后乙取,若乙取1个火柴棒,那么之后甲就取m个火柴棒;若乙取2个火柴棒,则甲取m-1个火柴棒......总之就是保证甲+乙=m+1,那么,就产生了如下m+1种情况:

1. **： n=(m+1)\*k+1,k是整数;**

甲先取1个火柴棒,那么还剩下(m+1)\*k个火柴棒,之后采取策略B,这样,就能保证每回合都取走m+1个,且甲取后一次.以此类推,甲就一定能取胜.

1. **： n=(m+1)\*k+2,k是整数**

甲先取2个火柴棒,那么还剩下(m+1)\*k个火柴棒,之后采取策略B,这样,就能保证每回合都取走m+1个,且甲取后一次.以此类推,甲就一定能取胜.

......

**(m)： n=(m+1)\*k+m,k是整数:**

甲先取m个火柴棒,那么还剩下(m+1)\*k个火柴棒,之后采取策略B,这样,就能保证每回合都取走m+1个,且甲取后一次.以此类推,甲就一定能取胜.

**(m+1)：n=(m+1)\*k,k是整数:**

这种情况下,甲要么先取1个,要么先取2个,要么先取3个,......要么先取m个,那么分别还剩下:(m+1)\*k-1,(m+1)\*k-2,(m+1)\*k-3,......,(m+1)\*k-m个火柴棒,亦即[(m+1)\*k+m]-(m+1),......,[(m+1)\*k+1]-(m+1)个火柴棒.这相当于上面的1)~m)这m种情况,只不过先取的是乙,这种情况下,乙必胜.

1. **：甲先取,且甲想取胜**

**(1)若n=(m+1)\*k+1,(m+1)\*k+2,......,(m+1)\*k+m**

采取上述策略,甲一定能取得胜利

**（2)若n=(m+1)\*k**

乙采取上述策略,乙一定能取得胜利.

总而言之，只有一种情况下,乙是必胜的,那就是n=(m+1)\*k,其余都是甲必胜,可见,甲必胜的概率随着m的增加而不断变大,同时,也可以看出,先手者胜率高.

代码实现：现有若干根火柴，两人轮流取，每人每次可取走1-4根，不可多取，也不能不取，谁取最后最后一根赢，设计算法玩家永远赢

#include <iostream>

using namespace std;

int main()  {

    int nSticks = 21;

    int nManTaken = 0;

    int nMerchineTaken = 0;

    nSticks -= 1;

    while (nSticks)  {

        cout << "How many stick do you wish to take(1~4)?";

        do {

            cin >> nManTaken;

            if (nManTaken >4 || nManTaken < 1)

            {

                cout << "input 1~4 : ";

            }

        } while (nManTaken >4 || nManTaken < 1);

        nSticks -= nManTaken;

        cout << (nSticks + 1) << " stick left in the pile." << endl;

        if (nSticks >= 5) {

            nMerchineTaken = 1;

        }

        else {

            switch (nSticks) {

            case 1:

                nMerchineTaken = 1;

                break;

            case 2:

                nMerchineTaken = 2;

                break;

            case 3:

                nMerchineTaken = 3;

                break;

            case 4:

                nMerchineTaken = 4;

                break;

            }

        }

        nSticks -= nMerchineTaken;

        cout << "compute take " << nMerchineTaken << " stick." << endl;

        if (nSticks > 0){

            cout << (nSticks + 1) << " stick left in the pile." << endl;

        }

        else {

            cout << " 1 stick left in the pile." << endl;

            cout << "How many stick do you wish to take(1~1)?";

            do  {

                cin >> nManTaken;

            } while (nManTaken != 1);

            cout << " You have taken the last stick." << endl;

            cout << " \* \* \* You lose!" << endl;

        }

    }      return 0;

}