最长连续不下降子序列

```cpp
#include<iostream>

using namespace std;

#define MAXN 10000
int main(){
    int n;
    int a[MAXN];
    cin>>n;
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    int now=1,ans=1;
    for(int i=2;i<=n;i++){
        if(a[i]>a[i-1]) now++;
        else now = 1;
        ans = max(now,ans);
    }
    cout<<ans<<endl;
}
```

最长不下降子序列

```cpp
#include<iostream>
#include<cstring>
using namespace std;

#define MAXN 1000
#define INF 0x3f3f3f3f

int main(){
    int n;
    int a[MAXN];//存数组
    int dp[MAXN];//dp[i]即长度为i的子序列的结尾数字
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
    }
    memset(dp,INF,sizeof(dp));
    for(int i=1;i<=n;i++){
        int *px=lower_bound(dp+1,dp+1+n,a[i]);//找到等于它或者小于它的位置
        *px=a[i];//插入，如果原位置有数据则覆盖，因为同样长度则留结尾数字较小的那个
    }
    int now=1;
    while(dp[now]!=INF) now++;
    cout<<now-1<<endl;
}
```

最长公共子序列

```cpp
#include<iostream>
#include<algorithm>
```

```
#include<cstring>
using namespace std;

#define MAXN 1000

int dp[MAXN][MAXN];

int main() {
    dp[0][0]=dp[0][1]=dp[1][0]=0;
    string a,b;
    cin>>a>>b;
    int la=a.size()-1;
    int lb=b.size()-1;
    for(int i=0; i<la; i++) {
        for(int j=0; j<lb; j++) {
            if(a[i]==b[j]) {
                dp[i+1][j+1]=dp[i][j]+1;
            } else {
                dp[i+1][j+1]=max(dp[i+1][j],dp[i][j+1]);
            }
        }
    }
    cout<<dp[la][lb]<<endl;
}

```

数位*DP*

```
typedef long long ll;
int a[20];
ll dp[20][state];//不同题目状态不同
ll dfs(int pos,/*state变量*/,bool lead/*前导零*/,bool limit/*数位上界变量*/)//不是每个题都要判断前导零
{
    //递归边界，既然是按位枚举，最低位是0，那么pos==-1说明这个数我枚举完了
    if(pos==-1) return 1;/*这里一般返回1，表示你枚举的这个数是合法的，那么这里就需要你在枚举时必须每一位都要满足题目条件，也就是说当前枚举到pos位，一定要保证前面已经枚举的数位是合法的。不过具体题目不同或者写法不同的话不一定要返回1 */
    //第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
    if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
    /*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应，具体为什么是有条件的记忆化后面会讲*/
    int up=limit?a[pos]:9;//根据limit判断枚举的上界up;这个的例子前面用213讲过了
    ll ans=0;
    //开始计数
    for(int i=0;i<=up;i++)//枚举，然后把不同情况的个数加到ans就可以了
    {
        if() ...
        else if()...
        ans+=dfs(pos-1,/*状态转移*/,lead && i==0,limit && i==a[pos]) //最后两个变量传参都是这样写的
        /*这里还算比较灵活，不过做几个题就觉得这里也是套路了
        大概就是说，我当前数位枚举的数是i，然后根据题目的约束条件分类讨论
        去计算不同情况下的个数，还有要根据state变量来保证i的合法性，比如题目
        要求数位上不能有62连续出现,那么就是state就是要保存前一位pre,然后分类，
        前一位如果是6那么这意味就不能是2，这里一定要保存枚举的这个数是合法*/
    }
```

```
    //计算完，记录状态
    if(!limit && !lead) dp[pos][state]=ans;
    /*这里对应上面的记忆化，在一定条件下时记录，保证一致性，当然如果约束条件不需要考虑lead，这里就是lead就完全不用考虑了*/
    return ans;
}
ll solve(ll x)
{
    int pos=0;
    while(x)//把数位都分解出来
    {
        a[pos++]=x%10;//个人老是喜欢编号为[0,pos),看不惯的就按自己习惯来，反正注意数位边界就行
        x/=10;
    }
    return dfs(pos-1/*从最高位开始枚举*/,/*一系列状态 */,true,true);//刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为0嘛
}
int main()
{
    ll le,ri;
    while(~scanf("%lld%lld",&le,&ri))
    {
        //初始化dp数组为-1,这里还有更加优美的优化,后面讲
        printf("%lld\n",solve(ri)-solve(le-1));
    }
}
```

HDOJ2089 不要62

统计区间 [a,b] 中不含 4 和 62 的数字有多少个。

```
#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;

int l,r;
const int MAXN = 25;
int dp[MAXN][2];//第二维记录前面一个是否为6
int a[MAXN];
int n,m;

int dfs(int pos,bool stat,int pre,bool limit){
    //limit记录之前是否都在上界，state记录之前的是不是6(记忆化作第二维)，pre记录上一个数字
    if(pos==-1) return 1;
    if(!limit&&dp[pos][stat]!=-1) return dp[pos][stat];
    int up = limit?a[pos]:9;//如果前面的都在上界则只能到a[pos]
    int ans = 0;
    for(int i=0;i<=up;i++){
        if(pre==6&&i==2) continue;//62的情况
        if(i==4) continue;
        ans += dfs(pos-1,i==6,i,limit&&i==a[pos]);
    }
    if(!limit) dp[pos][stat] = ans;
    return ans;
```

```
}

int solve(int x){
    //先按位转化到数组
    int px = 0;
    while(x){
        a[px++] = x%10;
        x/=10;
    }
    memset(dp,-1,sizeof(dp));//清空记忆化数组
    return dfs(px-1,0,-1,1);
}

int main(){
    while(cin>>l>>r){
        if(l==0&&r==0) break;
        //cout<<solve(r)<<' '<<solve(l-1)<<endl;
        cout<<solve(r)-solve(l-1)<<endl;
    }
}
```

sum of log

```
+#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;
typedef long long LL;
int l,r;
const int MAXN = 32;
int dp[MAXN][2][2];//第二维记录前面一个是否为2
int a[MAXN],b[MAXN];
int n,m;
const int mod = 1e9 + 7;

int dfs(int pos,bool limitx, bool limity){
    //limit记录之前是否都在上界，state记录之前的是不是6(记忆化作第二维)，pre记录上一个数字
    if(pos==-1) return 1;
    if(dp[pos][limitx][limity]!=-1) return dp[pos][limitx][limity];
    int upa = limitx ? a[pos]:1;//如果前面的都在上界则只能到a[pos]
    int upb = limity ? b[pos]:1;//如果前面的都在上界则只能到b[pos]
    LL ans = 0;
    for (int i=0; i<=upa; ++i) {
        for (int j=0; j<=upb; ++j) {
            if (i && j) continue;
            ans=(ans+dfs(pos-1,limitx&&i==upa, limity&&j==upb)) % mod;
        }
    }
    dp[pos][limitx][limity] = ans;
    return ans;
}

LL solve(int x,int y){
    //先按位转化到数组
```

```
    memset(dp,-1,sizeof(dp));//清空记忆化数组
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    int cntx = 0; int cnty = 0;
    while(x){ a[cntx++] = x&1; x>>=1;}
    while(y){ b[cnty++] = y&1; y>>=1;}

    LL ans= 0;
    for (int i=0; i<max(cntx,cnty); ++i ){
        LL res=0;
        if (i<cntx)
            res=(res+dfs(i-1,i==cntx-1,i>=cnty)) % mod;
        if (i<cnty)
            res=(res+dfs(i-1,i>=cntx,i==cnty-1)) % mod;
        ans=(ans+res*(i+1)) % mod;
    }
    return ans;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    int _;
    cin >> _;
    while (_--) {
        int x,y;
        cin >> x >> y;
        cout << solve(x,y) << "\n";
    }
}
```

# 背包

```
//01背包
//一个价值为v,重量为w的物品最多取一个,
// C++ Version
for (int i = 1; i <= n; i++)
  for (int l = W; l >= w[i]; l--) f[l] = max(f[l], f[l - w[i]] + v[i]);
```

# 并查集

```
#include <cstdio>
#define MAXN 5005
int fa[MAXN], rank[MAXN];
inline void init(int n)
{
    for (int i = 1; i <= n; ++i)
    {
        fa[i] = i;
        rank[i] = 1;
    }
}
```

```cpp
int find(int x)
{
    return x == fa[x] ? x : (fa[x] = find(fa[x]));
}
inline void merge(int i, int j) //按秩合并
{
    int x = find(i), y = find(j);
    if (rank[x] <= rank[y])
        fa[x] = y;
    else
        fa[y] = x;
    if (rank[x] == rank[y] && x != y)
        rank[y]++;
}
int main()
{
    int n, m, p, x, y;
    scanf("%d%d%d", &n, &m, &p);
    init(n);
    for (int i = 0; i < m; ++i)
    {
        scanf("%d%d", &x, &y);
        merge(x, y);
    }
    for (int i = 0; i < p; ++i)
    {
        scanf("%d%d", &x, &y);
        printf("%s\n", find(x) == find(y) ? "Yes" : "No");
    }
    return 0;
}
```

# 快速幂

## 矩阵快速幂求斐波那契数列

```cpp
#include <cstdio>
#define MOD 1000000007
typedef long long ll;

struct matrix
{
    ll a1, a2, b1, b2;
    matrix(ll a1, ll a2, ll b1, ll b2) : a1(a1), a2(a2), b1(b1), b2(b2) {}
    matrix operator*(const matrix &y)
    {
        matrix ans((a1 * y.a1 + a2 * y.b1) % MOD,
                   (a1 * y.a2 + a2 * y.b2) % MOD,
                   (b1 * y.a1 + b2 * y.b1) % MOD,
                   (b1 * y.a2 + b2 * y.b2) % MOD);
        return ans;
    }
};

```

```cpp
matrix qpow(matrix a, ll n)
{
    matrix ans(1, 0, 0, 1); //单位矩阵
    while (n)
    {
        if (n & 1)
            ans = ans * a;
        a = a * a;
        n >>= 1;
    }
    return ans;
}

int main()
{
    ll x;
    matrix M(0, 1, 1, 1);
    scanf("%lld", &x);
    matrix ans = qpow(M, x - 1);
    printf("%lld\n", (ans.a1 + ans.a2) % MOD);
    return 0;
}
```

# 莫队

## 小Z的袜子

```cpp
#include <bits/stdc++.h>
#define _debug1(x) cout<< #x <<"="<< x<<endl;
#define _debug2(x,y) cout<< #x <<"="<< x << " " << #y <<"="<< y<<endl;
#define _IOS ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
#define fileio freopen("data.in","r",stdin);freopen("data.out","w",stdout);


using namespace std;
typedef long long LL;
typedef pair<int,int> PII;
const int N = 5e4 + 7;
LL n,m;
LL a[N];
LL gcd(LL x,LL y) { return y==0? x:gcd(y,x % y) ;}
LL l=1, r=0;
LL sq;
struct  query {
    LL l;LL r;LL id;
    bool operator<(const query &o) const {
        if (l / sq != o.l /sq) //因为快的大小是sqrt(n)，所以可以直接求出块的位置
            return l<o.l;
        if (l / sq & 1) return r<o.r; else return r > o.r; //分奇偶排序
    }
}q[N];
LL cnt[N];
LL ans[N];
LL d[N];
```

```
LL cur=0;
inline void add(LL x) {
    int p=a[x];
    cur-=cnt[p]*(cnt[p]-1) / 2;
    cnt[p]++;
    cur+=cnt[p]*(cnt[p]-1) / 2;
}

inline void dec(LL x) {
    int p=a[x];
    cur-=cnt[p]*(cnt[p]-1) / 2;
    cnt[p]--;
    cur+=cnt[p]*(cnt[p]-1) / 2;
}

int main()
{
    _IOS;
    cin >> n >> m;
    sq=n/sqrt(m*2/3);
    for (int i=1; i<=n; ++i) cin >> a[i];
    for (int i=1; i<=m; ++i) {cin >> q[i].l >> q[i].r; q[i].id=i;}
    sort(q+1,q+m+1);
    cur=0;
    //cnt[a[1]]++;
    for (int i=1 ;i<=m; ++i) { //顺序不能轻易改变，原则是先扩大区间再减小区间
        while (l>q[i].l) add(--l);
        while (r<q[i].r) add(++r);
        while (l<q[i].l) dec(l++);
        while (r>q[i].r) dec(r--);
        if (l==r) ans[q[i].id]=0; else {ans[q[i].id]=cur; d[q[i].id]=
(q[i].r-q[i].l)*(q[i].r-q[i].l+1)/2;}
    }

    for (int i=1; i<=m; ++i) {
        if (ans[i]==0) {cout << "0/1" << "\n"; continue;}
        LL g=gcd(ans[i], d[i]);
        cout << ans[i]/g << "/" << d[i]/g << "\n";

    }
    return 0;
}

```

带修改的莫队

```
#include <bits/stdc++.h>
#define _debug1(x) cout << #x << "=" << x << endl;
#define _debug2(x, y) cout << #x << "=" << x << " " << #y << "=" << y <<
endl;

using namespace std;
typedef long long LL;
typedef pair<int, int> PII;
```

```
const int N = 1e6 + 33340;
inline int read(){
    int X=0; bool flag=1; char ch=getchar();
    while(ch<'0'||ch>'9') {if(ch=='-') flag=0; ch=getchar();}
    while(ch>='0'&&ch<='9') {X=(X<<1)+(X<<3)+ch-'0'; ch=getchar();}
    if(flag) return X;
    return ~(X-1);
}

LL n, m;
LL cntq = 0, cntr = 0;
LL a[N];
LL gcd(LL x, LL y) { return y == 0 ? x : gcd(y, x % y); }
LL l = 1, r = 0, t = 0;
LL sz;
LL bel[N];
struct query
{
    LL l,r,id,t;
    bool operator<(const query p1) const{
        if(bel[this->l] != bel[p1.l]) return bel[this->l] < bel[p1.l];
        if(bel[this->r] != bel[p1.r]){//注意对r进行分块后,令time相近的在一起
            return this->r < p1.r;
        }
        return this->t < p1.t;
    }

} q[N];
struct queryr
{
    LL x, y;
} qr[N];
LL cnt[N];
LL ans[N];
LL d[N];
LL cur = 0;          //当前答案

inline void add(LL p) //因为后面有修改所以直接传颜色而不是下标
{
    if (cnt[p] == 0)
        ++cur;
    ++cnt[p];
}

inline void dec(LL p)
{
    --cnt[p];
    if (cnt[p] == 0)
        --cur;
}

inline void upd(int i, int t)
{
    if (q[i].l <=qr[t].x && qr[t].x <= q[i].r) {
        dec(a[qr[t].x]);
        add(qr[t].y);
    }
    swap(a[qr[t].x],qr[t].y);
```

```
}

int main()
{
    //_IOS;
    n=read();
    m=read();
    sz= pow(m,(double)2/(double)3);
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
        bel[i]=i/sz+1;
    }
    for (int i = 1; i <= m; ++i)
    {
        char op;
        int x, y;
        cin >> op;x=read();y=read();
        if (op == 'Q')
        {
            ++cntq;
            q[cntq].l = x;
            q[cntq].r = y;
            q[cntq].id = cntq;
            q[cntq].t = cntr;
        }
        else
        {
            ++cntr;
            qr[cntr].x = x;
            qr[cntr].y = y;
        }
    }
    sort(q + 1, q + cntq + 1);
    l = 1;
    r = 0;
    t = 0;
    cur = 0;
    //cnt[a[1]]++;
    for (int i = 1; i <= cntq; ++i)
    {
        while (l > q[i].l) add(a[--l]);
        while (r < q[i].r) add(a[++r]);
        while (l < q[i].l) dec(a[l++]);
        while (r > q[i].r) dec(a[r--]);

        while (t < q[i].t) upd(i, ++t);
        while (t > q[i].t) upd(i, t--);

        ans[q[i].id] = cur;
    }

    for (int i = 1; i <= cntq; ++i)
    {
        printf("%d\n",ans[i]);
    }
    return 0;
}
```

# 前缀和与差分

一维前缀和

S[i] = a[1] + a[2] + ... a[i]
a[l] + ... + a[r] = S[r] - S[l - 1]
二维前缀和

S[i, j] = 第i行j列格子左上部分所有元素的和
以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵的和为：
S[x2, y2] - S[x1 - 1, y2] - S[x2, y1 - 1] + S[x1 - 1, y1 - 1]
一维差分
给区间[l, r]中的每个数加上c：B[l] += c, B[r + 1] -= c

二维差分 —— 模板题 AcWing 798. 差分矩阵
给以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵中的所有元素加上c：
S[x1, y1] += c, S[x2 + 1, y1] -= c, S[x1, y2 + 1] -= c, S[x2 + 1, y2 + 1] += c

# 匈牙利算法

二分图匹配

最大匹配数=最小覆盖点数

```
#include<bits/stdc++.h>
using namespace std;
#define _for(i, a, b) for(int i = (a); i <= (b); ++i)

const int N = 510;
int sum;
int n, m, k, t, x;
int vis[N], match[N];
vector<int> G[N];

bool dfs (int u) {
    for(auto v : G[u]) {
        if(match[v] == u) continue;
        if(!vis[v]) {
            vis[v] = 1;
            if(!match[v] || dfs(match[v])) {
                match[v] = u;
                return true;
            }
        }
    }
    return false;
}

void solve () {
    memset(match, 0, sizeof match);
    _for(i, 1, n) {
        memset(vis, 0, sizeof vis);
```

```
        if(dfs(i)) sum++;
    }
    for(int i = 1; i <= n && k; i++) {
        memset(vis, 0, sizeof vis);
        if(dfs(i)) sum++, k--;
    }
}

int main() {
    scanf("%d%d%d", &n, &m, &k);
    _for(i, 1, n) {
        scanf("%d", &t);
        while(t--) {
            scanf("%d", &x);
            G[i].push_back(x);
        }
    }
    solve();
    printf("%d\n", sum);
    return 0;
}

```

棋盘覆盖

```
#include <bits/stdc++.h>
#define _debug(x) {cout<< #x <<"="<< x<<endl;}
using namespace std;

typedef long long LL;
const int N = 100 + 7;
int step[4][2]={{-1,0},{0,1},{1,0},{0,-1}};
vector<int> g[10007];
int vis[10007];

int t,n;
int m[N][N];
int L[10007];
int ans=0;

int trans(int x ,int y) {
    return (x*n+y) / 2;
}

bool match(const int & from,int p) {
    for (auto i=0; i<g[p].size(); ++i) {
        int to=g[p][i];
        if ( vis[to]!=from ) {
           vis[to]=from;
           if (L[to]==-1 || match(from,L[to])) {

               L[to]=p;
               return true;
           }
        }
```

```
    }
    return false;
}

int hungary() {
    int ans=0;
    for (int i=0; i<n; ++i) {
        for (int j= i%2==0? 0:1; j<n; j+=2) {
            //memset(vis,0,sizeof(vis));
            if ( m[i][j]==0 && match(trans(i,j),trans(i,j)) ) {
                ++ans;
            }
        }
    }
    return ans;
}

int main()
{
    cin >> n;
    cin >> t;
    memset(L,-1,sizeof(L));
    memset(vis,-1,sizeof(vis));

    for (int i=0; i<t ; ++i) {
        int x,y;
        cin>> x >> y;
        x-=1;
        y-=1;
        m[x][y]=1;
    }
    for (int i=0; i<n; ++i) {
        for (int j= i%2==0 ? 0:1 ; j<n; j+=2) {
            if (m[i][j]==1) continue;
            for (int k=0; k<4; ++k) {
                int x=i+step[k][0];
                int y=j+step[k][1];
                if (x>=0 && x<n && y>=0 && y<n && m[x][y]==0) {
                    g[trans(i,j)].push_back(trans(x,y));
                }
            }
        }
    }
    // for (int i=0; i<n; ++i) {
    //     for (int j= i%2==0 ? 0:1 ; j<n; j+=2) {
    //         cout << g[trans(i,j)].size() << " ";
    //     }
    //     cout << endl;
    // }
    cout << hungary() << endl;
    return 0;
}
```

# 字符串

```
//马拉车的初始化 abcb->@a#b#c#b$
//返回新的长度
int init(char *st){
    int i, len = strlen(st);
    tmp[0] = '@';
    for(i = 1;i <= 2*len; i+=2){
        tmp[i] = '#';tmp[i+1] = st[i/2];
    }
    tmp[2*len+1] = '#';
    tmp[2*len+2] = '$';
    tmp[2*len+3] = 0;
    return 2*len+1;
}

```

```
//cnt【i】以Si字符为结尾时的回文个数
void manacher(char *st, int len){
    int mx = 0, ans = 0, po = 0;
    for(int i = 1;i <= len; i++){
        if(mx > i)Len[i] = min(mx - i, Len[2*po-i]);
        else Len[i] = 1;
        while(st[i-Len[i]] == st[i+Len[i]])Len[i]++;
        if(Len[i] + i > mx){mx = Len[i] + i; po = i;}
        ans = max(ans, Len[i]);
    }
    int now = 0;
    for(int i = 1;i <= len; i++){
        cha[i]++;
        cha[i+Len[i]]--;
    }
    for(int i = 1;i <= len; i++){
        now += cha[i];
        if(i%2 == 0)cnt[i/2] = now;
    }
}
```

# GarsiaWachs算法

这个算法简述来说就是：加入有a,b,c*a*,*b*,*c*三个石子堆

先合并左边两个的代价是2a+2b+c2*a*+2*b*+*c*，而先合并右边两个的代价是a+2b+2c*a*+2*b*+2*c*，所以只用看a*a*和c*c*，a<c*a*<*c*

于是先合并a<c*a*<*c*的吧，完了之后如果还有剩就从右边开始合并bc*b***c*

合并ab*a***b*的时候，合并之后插入从左边数第一个大于他的数的后面开始，究竟为什么这么做我也不怎么理解。。。

但是如果这道题就这样按照所说的话会TLE*TLE*，因为插入删除都是O(n)*O*(*n*)的，所以只好想个办法让这个n*n*变得比较小

新开了一个数组，一个一个加石头进去，满足a b*a***b*的合并条件就合并，这样会保持n*n*比较小

```

```

```cpp
#include<cmath>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<algorithm>
#include<queue>
#include<vector>
#include<iostream>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef double db;
#define maxn 50010
#define inf 0x3f3f3f3f
const int mod=100003;

void read(int &x){
    int f=1;x=0;
    char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-') f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9') {x=x*10+ch-'0';ch=getchar();}
    x*=f;
}
int n,w; long long ans=0;
vector<int> l;
int solu(){
    int k=l.size()-2,q=-1;
    for(int i=0;i<l.size()-2;++i) if (l[i]<=l[i+2]) {k=i;break;}
    int t=l[k]+l[k+1];
    l.erase(l.begin()+k);l.erase(l.begin()+k);
    for(int i=k-1;i>=0;--i) if (l[i]>t) {q=i; break;}
    l.insert(l.begin()+q+1,t);
    return t;
}
int main(){
    read(n);
    for(int i=1;i<=n;++i) read(w),l.push_back(w);
    for(int i=0;i<n-1;i++) ans+=solu();
    printf("%lld",ans);
    return 0;
}
```

# 二分

lower_bound( )和upper_bound( )都是利用二分查找的方法在一个排好序的数组中进行查找的。

在从小到大的排序数组中，

lower_bound( begin,end,num)：从数组的begin位置到end-1位置二分查找第一个**大于或等于num**的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

upper_bound( begin,end,num)：从数组的begin位置到end-1位置二分查找第一个**大于num**的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

在从大到小的排序数组中，重载lower_bound()和upper_bound()

lower_bound( begin,end,num,greater() ):从数组的begin位置到end-1位置二分查找第一个**小于或等于num**的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

upper_bound( begin,end,num,greater() ):从数组的begin位置到end-1位置二分查找第一个**小于num的数字**，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

1.__builtin_popcount(unsigned int n)
该函数时判断n的二进制中有多少个1

```
int n = 15; //二进制为1111
cout<<__builtin_popcount(n)<<endl;//输出4

```

2.__builtin_parity(unsigned int n)
该函数是判断n的二进制中1的个数的奇偶性

```
int n = 15;//二进制为1111
int m = 7;//111
cout<<__builtin_parity(n)<<endl;//偶数个，输出0
cout<<__builtin_parity(m)<<endl;//奇数个，输出1
```

3.__builtin_ffs(unsigned int n)
该函数判断n的二进制末尾最后一个1的位置，从一开始

```
int n = 1;//1
int m = 8;//1000
cout<<__builtin_ffs(n)<<endl;//输出1
cout<<__builtin_ffs(m)<<endl;//输出4
```

4.__builtin_ctz(unsigned int n)
该函数判断n的二进制末尾后面0的个数，当n为0时，和n的类型有关

```
int n = 1;//1
int m = 8;//1000
cout<<__builtin_ctzll(n)<<endl;//输出0
cout<<__builtin_ctz(m)<<endl;//输出3
```

5. __builtin_clz (unsigned int x)
   返回前导的0的个数。