

分治算法

是一种**自底向上**的算法，将大问题拆成若干个小问题，每个小问题与大问题类型一致而规模小很多，然后将若干个小问题的解合并成为大问题的答案。

分治算法所要求解的问题必须**具备最优子结构**。

通常求解子问题合并大问题使用dp，如果用分治则要做大量重复工作，对公共子问题求解多次。

通常将大问题分成大小相等的 k 份，**这样的划分几乎比其他情况的划分都要好**。

递归算法

递归算法即函数不断直接或间接调用自身。

递归函数两大要素：边界条件与递归方程。

递归算法耗用的时间和空间都比其对应的非递归算法要多。但是递归算法结构更加明确，编写算法简单。

排序问题

问题：

用递归得出一个序列的全排列

dfs。

整数划分的递归版本

将一个正整数划分成若干个正整数相加，有多少种方法。

将问题扩展为：求整数 n 的最大加数不超过 m 的划分方法数

考虑几种情况：

- 如果 $n = 1, m = 1, p(n, m) = 1$
- 如果 $n < m$ ，那么显然 $p(n, m) = p(n, n)$ (最大加数不可能超过总和本身)
- 如果 $n == m$ ，那么 $p(n, m) = p(n, n) = p(n, n - 1) + 1$
- 如果 $n > m$ ，那么显然 $p(n, m) = p(n, m - 1) + p(n - m, m)$

最后我们想要的答案就是 $p(n, n)$

算法复杂度的时间递推式

一般根据递归方程写出该算法的时间复杂度。注意两点：

- 注意边界条件，最后项数是多少。

- 计算复杂度时取无穷大优先级最高的。

计算结论：

$$T(n) = bT(n/a) + O(n)$$

- $= n \log_a(n), b == a$
- $= n^{\log_a(b)}, b > a$
- $= O(n), b < a$

二分

搜索的复杂度为 $O(\log n)$ 。取定中点，当目标小于中点向左搜索，大于目标向右搜索。

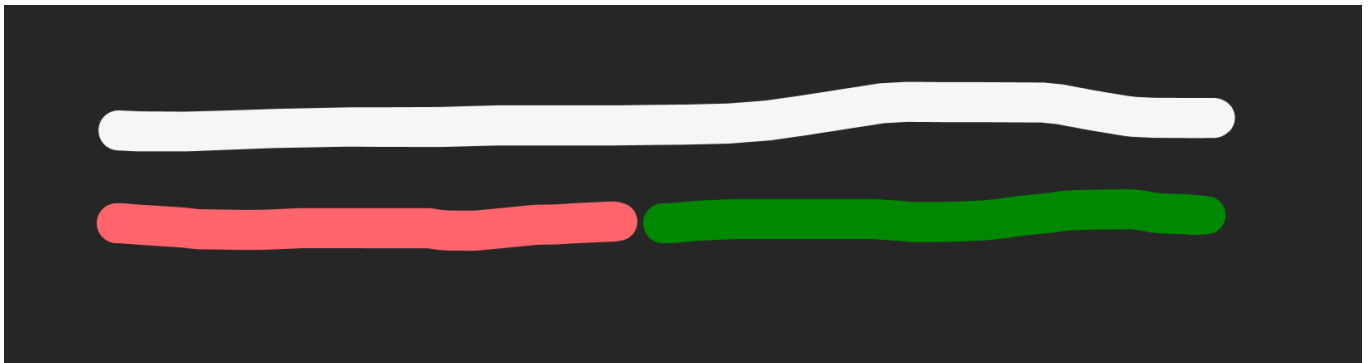
把二分板子拿过来：（考试不要求实现）

#整数二分

二分一共有两套模板，这里——介绍。

二分的本质是在有一定顺序（不一定升序或降序）的序列中查找一个边界位置。

当我们所求的位置的要求是，在这个边界右边的区间（包含这个边界）都满足条件，在这个左边的边界的区间都不满足条件，可用第一套模板。例如在下图中：



白色为我们所要求的数轴区间。如果我们的要求中右边绿色区间都满足条件，那么可得：
 当判断条件为true，mid同样有可能是答案，搜索区间更新为 $[l, mid]$ ；
 而如果不满足条件，则mid在左边，那么区间更新为 $[mid+1, r]$ ；
 这就是第一套模板。

同样的，当我们的条件是查找在这个边界左边区间中都满足条件（包含这个边界），而右边区间都不满足时，又可得：

当判断条件为true，则mid满足条件，区间更新为 $[mid, r]$ ；
 如果判断条件为false，则mid在右边，区间更新为 $[l, mid-1]$ ；

注意如果使用第二套模板（查找满足左区间条件的）则最初的mid定义为 $l+r+1 >> 1$ ；相反的，第一套模板定义为 $l+r >> 1$ ；

第一套模板：

在一个序列中查找第一次出现x的下标。

易得，在这个边界的右半区间都满足的条件是： $\geq x$ 。

```
//此处省略其他内容

l=0;
r=n-1;
mid=l+r>>1;
while(l<r)
{
    if(arr[mid]>=x)
    {
        r=mid;
    }
    else
        l=mid+1;
}
if(arr[l]==x)
{
    return l;
}
else
    return -1;
// 此处省略其他内容
```

第二套模板：

在一个序列中查找最后一次x出现的下标。

易得，在这个区间的左半部分满足的条件是 $\leq x$ 。用第二套模板。

```
//此处省略其他内容

l=0;
r=n-1;
mid=l+r>>1;
while(l<r)
{
    if(arr[mid]<=x)
    {
```

```

        l=mid;
    }
    else
        r=mid-1;

}
if(arr[l]==x)
{
    return l;
}
else
    return -1;
// 此处省略其他内容

```

例题:

[789. 数的范围 - AcWing题库](#)

```

# include<bits/stdc++.h>
using namespace std;
typedef long long ll;
ll n;
ll q;
ll arr[100005];

int main (void)
{
    scanf("%d",&n);
    scanf("%d",&q);
    for(int i=0;i<n;i++)
        scanf("%d",&arr[i]);

    while(q--)
    {
        ll x;
        scanf("%d",&x);
        ll l=0;
        ll r=n-1;
        ll mid;
        while(l<r)
        {

```

```

        mid=l+r>>1;

        if(arr[mid]>=x)
        {
            r=mid;
        }
        else
        l=mid+1;
    }
    if(arr[l]!=x)
    {
        cout<<"-1 -1"<<endl;
        continue;
    }
    else
    cout<<l<<" ";
    l=0;
    r=n-1;
    while(l<r)
    {
        mid=l+r+1>>1;
        if(arr[mid]<=x)
        {
            l=mid;
        }
        else
        r=mid-1;
    }
    cout<<l<<endl;

}

}

```

#浮点数二分

相较于整数，浮点数二分非常简单，无需考虑整除问题，因此左边界与右边界与mid的更新都可以直接进行。

[790. 数的三次方根 - AcWing题库](#)

```

# include<bits/stdc++.h>
# include<iomanip>

```

```

using namespace std;
typedef long long ll;

typedef long double ld;

int main (void)
{

    ld n;
    cin>>n;
    ld l=-10000;
    ld r=10000;
    ld mid=(l+r)/2;
    while(r-l>1e-8)
    {

        mid=(l+r)/2;
        if(mid*mid*mid>=n)
        {
            r=mid;
        }
        else
            l=mid;
    }
    cout<<fixed<<setprecision(6)<<l<<endl;

}

```

归并排序

归并排序的思想非常简单，即为将数组先分成若干个小序列，然后在每个小序列内对数组进行排序，每个小序列排序好之后再总体结合起来成为一个较大的序列，再对这些较大的序列进行排序，得到更大的序列，依次类推，直到得到完整的序列排序。

复杂度为 $O(n\log n)$

重点在实现（考试不要求）

```

# include<bits/stdc++.h>
using namespace std;
typedef long long ll;
ll temp[114514];

```

```

void merge_sort(ll arr[],ll l,ll r)
{
    if(l>=r)
        return;
    ll mid=l+r>>1;
    merge_sort(arr,l,mid);
    merge_sort(arr,mid+1,r);
    ll k=0,i=l,j=mid+1;
    while(i<=mid&&j<=r)
    {
        if(arr[i]<arr[j]) temp[k++]=arr[i++];
        else temp[k++]=arr[j++];
    }
    while(i<=mid) temp[k++]=arr[i++];
    while(j<=r) temp[k++]=arr[j++];
    for(int i=l,j=0;i<=r;i++,j++)
        arr[i]=temp[j];
}

int main (void)
{
    ll arr[114514];
    for(int i=0;i<=17;i++)
        arr[i]=rand()%(114514-i+1)+1;
    merge_sort(arr,0,17);
    for(int i=0;i<=17;i++)
        cout<<arr[i]<<endl;
}

```

快速排序

思想和归并排序很相似。只需记住复杂度最坏为 $O(n^2)$ ，平均为 $O(n\log n)$ 。重点在实现，考试不要求。

```

# include<bits/stdc++.h>
typedef long long ll;
using namespace std;
void quick_sort(int arr[],int l,int r)
{
    if(l>=r)    //注意这里的大小关系非常容易弄混
        return;
    int i=l-1;
    int j=r+1;

```

```

        int x=arr[(l+r)/2];
        while(i<j)
        {
            do i++;while(arr[i]<x);
            do j--;while(arr[j]>x);
            if(i<j)
                swap(arr[j],arr[i]);
        }
        quick_sort(arr,l,j);
        quick_sort(arr,j+1,r);
    }
    int main (void)
    {
        int arr[1155];
        for(int i=0;i<15;i++)
            arr[i]=rand()%(114514-i+1)+1;
        for(int i=0;i<15;i++)
            cout<<arr[i]<<" ";
        cout<<endl;

        quick_sort(arr,0,14);
        for(int i=0;i<15;i++)
            cout<<arr[i]<<" ";
        return 0;
    }
}

```

比赛日程表安排

设计一个满足以下要求的比赛日程表：

- (1) 每个选手必须与其他 $n-1$ 个选手各赛一次；
- (2) 每个选手一天只能赛一次；
- (3) 循环赛一共进行 $n-1$ 天。

本质为将所有选手先划分为一个个只有两个人的小子集，在子集内安排比赛；然后合并，直到最大的集合。

复杂度为 $O(n)$

