贪心

```
// 需要安排几位师傅加工零件
#include<bits/stdc++.h>//1-1326-1  liuchunhui001  推荐学习
using namespace std;
int n,m,a[105],sum,ct;//sum 加工几件，ct 最少需要几位师傅
int main()
{
	cin>>m>>n;
	for(int i=1; i<=n; i++)
		cin>>a[i];
	sort(a+1,a+n+1,greater<int>());// 最少，师傅工作效率要大，采取从大到小排序
	for(int i=1; i<=n; i++)
	{
		if(sum<m)
		{
			ct++;
			sum+=a[i];// 累加件数
		}
		else
		{
			cout<<ct;// 师傅够，不够就不输出
			return 0;// 结束程序
		}
	}
	cout<<"NO";// 如果在 for 那里没 return 0; 的话，这里就会输出 NO
	return 0;
}
```

由于题目要求的是最少需要多少师傅来加工零件，因此我们优先挑选加工能力强的师傅来加工零件。

```
#include <bits/stdc++.h>//1-1326-2     javacn
using namespace std;
// 辅助降序排序的函数
bool cmp(int a,int b){
	if(a > b)   return true;
	else  return false;
}
int a[110],i,s,n,m;
int main()
{
	cin>>m>>n;
	// 读入 n 个师傅的加工能力
	for(i = 1;i <= n;i++){
		cin>>a[i];
	}

	// 对 n 个师傅的加工能力进行排序
	sort(a+1,a+n+1,cmp);

	for(i = 1;i <= n;i++)    // 逐个求和
	{
		s = s + a[i];
		// 人数是否足够
		if(s >= m)
		{
			cout<<i;
			break;
		}
	}
	// 如果所有师傅都来加工人数也不够
	if(s < m)   cout<<"NO";
	return 0;
}
```

排队打水问题

每个人的总打水时间 = 该用户打水时间 + 排队等待时间，因此只有打水快的人先打，总的排队时间才是最少的。

```
#include <bits/stdc++.h>//2-1228  javacn
using namespace std;

int n,a[510],r,i,s = 0;
int main()
{
        cin>>n>>r;
        // 读入每个人的打水时间
        for(i = 1;i <= n;i++){
             cin>>a[i];
        }

        // 排序：从小到大，打的快的先打
        sort(a+1,a+n+1);

        // 从第 r+1 个人开始修正每个人的总打水时间
        // 求和
        for(i = 1;i <= n;i++)
       {
             if(i >= r + 1) a[i] += a[i-r]; // 相当于 a[i]=a[i]+a[i-r];
             s = s + a[i];
        }

        cout<<s;

        return 0;
}
```

拦截导弹的系统数量求解

1、当导弹来袭，假设导弹高度为 x，在 a 数组中找能拦截的系统中最矮的系统进行拦截

2、如果找不到这样的系统：开新系统拦截，将新系统的高度设为 x

3、如果能找到（找到能拦截的第 1 套系统，就是最矮）：因为导弹拦截系统的高度一定是递增的．用能拦截的第 1 套系统拦截，并将该系统的高度设为 x，最后看要几套系统

```
#include <bits/stdc++.h>//3-1229-1     javacn  推荐学习
using namespace std;//p：代表找到的能够拦截系统的下标 //k：代表有几套系统
int a[1100],n,i,j,x,p,k = 0;//x：代表每套系统的高度
int main() {
    cin>>n;
    for(i = 1;i <= n;i++)     // 循环导弹的数量
    {
        cin>>x;
        p = -1;// 假设没找到
        for(j = 1;j <= k;j++)          // 在 k 套系统中找第 1 套能拦截的系统
        {
            if(a[j] >= x)// 找到第 1 套能拦截的系统
            {
                p = j;
                break;
            }
        }
        if(p != -1)        // 如果找到系统拦截
        {
            a[p] = x;
        }
        else
        {
            k++;              // 开新系统
            a[k] = x;// 更新系统高度
        }
    }
    cout<<k;
    return 0;
}
```

```
#include<bits/stdc++.h>//3-1229-2    jiangyf70  仅供参考
using namespace std;
int a[1005];
int main()
{
    int n, b, k = 1; //k 导弹系统套数

    cin >> n;
    cin >> a[k]; // 先读入一个高度
    for(int i = 2; i <= n; i++)
    {
        cin >> b;
        bool f = 0;// 是否能拦截
        for(int j = 1; j <= k; j++) // 扫描导弹套数是否能拦截
        {
            if(a[j] >= b)
            {
                a[j] = b; // 能拦截就降低导弹高度
                f = 1;
                break;
            }
        }
        if(!f) a[++k] = b;// 不能拦截再开一套  完整写法:
    }
    cout << k;
    return 0;
}
```

```
if(f==0)
{
    k++;
    a[k] = b;
}
```

解法一：使用结构体解题

```
#include <bits/stdc++.h>//4-1372-1     javacn  推荐学习
using namespace std;
// 每个时间对
struct node {
	int begin;
	int end;
};
struct node a[110];
int n;
// 按结束时间升序排序
bool cmp(node n1,node n2) {
	return n1.end < n2.end?true:false;
}
int main() {
	int i;
	cin>>n;
	for(i = 0; i < n; i++) {
		cin>>a[i].begin>>a[i].end;
	}

	sort(a,a+n,cmp);
	int ans=1;  // 第 1 个活动必选
	int t = a[0].end;
	for(int i=1; i<n; i++)  // 在剩余活动中选择
	{
		// 如果当前活动与之前最后结束的活动不冲突，就接受当前活动。
		if(a[i].begin>=t)
		{
			ans++;
			t=a[i].end;
		}
	}
	cout<<ans;
}
```

贪心的策略：优先选择结束时间尽可能早的活动，因为结束时间越早，剩余的时间就越多，那么剩余的时间可以排更多的活动．解法：按照结束时间升序排序，然后逐个统计哪个活动的开始时间 >= 上一个被选中活动的结束时间　　解法二：使用整数数组排序解题

```
#include <bits/stdc++.h>//4-1372-2    javacn
using namespace std;
int n;// 活动数量
int s[110],e[110];// 活动起止时间
int i,j,c,etime;
int main() {
    cin>>n;
    for(i = 0;i < n;i++)    {    cin>>s[i]>>e[i];  }
    for(i = 1;i <= n - 1;i++)    // 按照结束时间升序
    {
        for(j = 0;j <= n - i - 1;j++)
        {
            if(e[j] > e[j+1])
            {
                swap(e[j],e[j+1]);
                swap(s[j],s[j+1]);
            }
        }
    }
    etime = e[0];    // 第 1 个活动必选 // 存储第 1 个活动的结束时间
    c = 1;// 已经有 1 个选中的活动了
    for(i = 1;i < n;i++)    // 在剩余活动中继续选
    {
        if(s[i] >= etime)// 如果一活动的开始时间 >= 上一个选中活动的结束时间
        {
            c++;
            etime = e[i];
        }
    }
    cout<<c;
    return 0;
}
```

淘淘捡西瓜　贪心：从最轻的西瓜开始依次尝试，直到装不下。

```
#include <bits/stdc++.h>//5-1456    javacn
using namespace std;

int n,x,a[110],s = 0;//s 代表总共装了多重的西瓜
int main()
{
	cin>>n>>x;
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
	}
	// 排序
	sort(a+1,a+n+1);
	// 从最轻的开始尝试
	int c = 0;// 装了多少个
	for(int i = 1;i <= n;i++)
	{
		// 背包已经装了 s 斤西瓜，如果再加这个西瓜的重量，总和 <=x
		if(s + a[i] <= x)
		{
			c++;
			s = s + a[i];
		}
		else
		{
			break;
		}
	}
	cout<<c;
	return 0;
}
```

// 删数问题　　优先删除第 1 个递减的位置，如果不存在，说明递增，删除最后一个：

```
#include <bits/stdc++.h>//6-1373     javacn
using namespace std;
string nn;
int s,m;
int main()
{
      cin>>nn;
      cin>>s;
      // 删除 s 个数
      for(int i=1; i<=s; i++)
      {
            // 默认删除最后一个
            m=nn.size()-1;
            for(int j = 0; j < nn.size() - 1; j++)
            {
                  // 如果找到第一个递减位置
                  if(nn[j]>nn[j+1])
                  {
                        m=j;
                        break;
                  }
            }
            nn.erase(m,1);
      }
      // 删除前导 0
      while(nn[0] == '0') {
        nn.erase(0,1);
      }

      cout<<nn<<endl;
      return 0;
}
```

接水问题

由于本题不能修改接水的顺序，必须按照读入的顺序接水

那么最短时间应该是：每个人都找当前打水时间最短的水龙头排队（也就是将第 i 个人的打水时间直接加到数组的最小数的下标对应的位置）

最后：求数组的最大值，也就是打水时间。

```
#include <bits/stdc++.h>//7-1485    javacn
using namespace std;
int a[110];// 存储每个水龙头的打水时间
int n,m,i,j,ma,mi,x;
int main()
{
	cin>>n>>m;
	// 读入每个人的打水时间
	for(i = 1;i <= n;i++)
	{
		cin>>x;
		// 找数组最小数下标
		mi = 1;
		for(j = 2;j <= m;j++)
		{
			if(a[j] < a[mi])
			{
				mi = j;
			}
		}
		a[mi] = a[mi] + x;
	}
	ma = a[1];// 找最大数
	for(i = 2;i <= m;i++)
	{
		if(a[i] > ma)	ma = a[i];
	}
	cout<<ma;
	return 0;
}
```

```
#include<bits/stdc++.h>//8-1730-1    javacn  推荐学习
using namespace std;
struct node{
        int money,num;
};
node a[1010];
int m,n;
//按单价升序排序
bool cmp(node n1,node n2){
        return n1.money<n2.money;
}
int main()
{
        cin>>m>>n;
        for(int i=1;i<=n;i++){
                cin>>a[i].money>>a[i].num;
        }
        sort(a+1,a+n+1,cmp);
        int c=0,s=0;//c,贺卡数量; s,贺卡价格
        for(int i=1;i<=n;i++)
        {
                if(c+a[i].num<m)//这家店都买完不够
                {
                        c=c+a[i].num;
                        s=s+a[i].num*a[i].money;
                }
                else
                {
                        s=s+(m-c)*a[i].money; //买够m
                        break;
                }
        }
        cout<<s;
}
```

解法一：使用结构体求解

购买贺年卡

本题显然是使用贪心的思想，既然要花最少的钱购买贺卡，那么必然优先购买价格低的贺卡。将所有店铺的贺卡按价格降序，但要注意数据上要保证价格和数量还是要对应的。

然后从第一家店，依次购买，直到数量满足m。

解法二：使用数组直接排序。

```
#include <bits/stdc++.h>//8-1730-2    javacn
using namespace std;
int m,n;
int price[1100],num[1100],i,j,s,c;
int main() {
	cin>>m>>n;//m:要买的数量 n:有多少商家

	for(i = 0;i < n;i++)	cin>>price[i]>>num[i];  //读入每家的单价和存货量
	//对单价升序排序，交换单价的同时交换数量
	for(i = 1;i <= n - 1;i++)//排序的次数
	{
		for(j = 0;j <= n - i - 1;j++)
		{
			if(price[j] > price[j+1])
			{
				swap(price[j],price[j+1]);
				swap(num[j],num[j+1]);
			}
		}
	}
	c = 0; //买了多少
	s = 0;//花了多少钱
	for(i = 0;i < n;i++)//从第1家开始买
	{
		c = c + num[i];		//先买后退
		s = s + num[i] * price[i];
		if(c >= m)//判断要不要退（多买或者正好，循环停止）
		{
			s = s - (c - m) * price[i];//退掉多买的
			break;
		}
	}
	cout<<s;
	return 0;
}
```

```
#include<bits/stdc++.h>//9-1375-1    jiangyf70    推荐学习
using namespace std;
int a[1005][1005];
int main(){
    int n, h, k = 0;;
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> h;
        bool f = 0;
        for(int j = 1; j <= k; j++)
        {
            if(a[j][a[j][0]] >= h)
            {
                a[j][0]++;
                a[j][a[j][0]] = h;
                f = 1;
                break;
            }
        }
        if(f == 0)
        {
            k++;
            a[k][0] = 1;
            a[k][1] = h;
        }
    }
    cout << k << endl;
    for(int i = 1; i <= k; i++)
    {   cout << i << ':';
        for(int j = 1; j <= a[i][0]; j++)       cout << a[i][j] <<' ';
        cout << endl;
    }
    return 0;
}
```

用二维数组的第 0 位保存导弹的个数比如：

系统 1：拦截 5 389 207 175 170 158

系统 2：拦截 3 300 299 165

```
#include <bits/stdc++.h>//9-1375-2    javacn
using namespace std;
int n,x;//n: 代表导弹数量,x: 代表每个导弹的高度
int i,j,p;
int r[1010][1010];// 存储第 i 套系统拦截的导弹的高度
int k;// 代表开了几套系统，也就是 r 数组的下标
int main()
{
        cin>>n;
        for(i = 1;i <= n;i++)// 读入导弹的高度
        {
                cin>>x;
                p = -1;// 代表能拦截的系统的下标 // 假设没有系统能拦截

                for(j = 1;j <= k;j++)    // 循环所有的系统，查找能拦截的第一套系统
                {
                        if(r[j][r[j][0]] >= x)
                        {
                                p = j;          1229: 拦截导弹的系统数量求解问题的升级版。
                                break;          数组：存储每套系统拦截的所有导弹的高度
                        }                       r[i][0]：第 i 套系统，拦截了几个导弹
                }                               r[i][r[i][0]]：存储第 i 套系统的最低高度

                if(p != -1) // 判断如果有系统能拦截
                {
                        r[p][0]++;// 拦截数量 +1
                        r[p][r[p][0]] = x; // 将第 i 个导弹的高度存储到 r[p] 这一行
                }
                else
                {
                        k++;
                        r[k][0] = 1;
                        r[k][1] = x;//r[k][r[k][0]] = x;
                }
        }
```

```
    // 系统数量
    cout<<k<<endl;
    for(i = 1;i <= k;i++)
    {
        cout<<i<<":";// 输出系统编号
        // 输出导弹的高度
        for(j = 1;j <= r[i][0];j++)
        {
            cout<<r[i][j]<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

拦截一次记录一次并更行系统拦截高度，不能拦截就新增加一套系统

```
#include<iostream>//9-1375-3    anselxu     仅供参考
#include<algorithm>
using namespace std;
int a[1010][1010];
/* 模拟进程：
二维数组 a[x][y], x 标记第几套导弹防御系统，
y=0 记录当前最高高度，拦截一次更新一次，y++ 记录每次拦截的高度。
*/
int main(){
	int n,h,k=0;
	cin>>n;
	for(int i=1;i<=n;i++)
	{
		cin>>h;
		bool f=0;
		for(int j=1;j<=k;j++)
		{
			int t=0;
			if(a[j][t]>=h)
			{
				a[j][t]=h;// 更新能拦截的最大高度
				f=1;
				while(a[j][t]!=0) t++;// 找到记录本次拦截的高度的位置
				a[j][t]=h;// 增加本次拦截的高度
				break;
			}
		}
		if(!f)
		{
			a[++k][0]=h;// 新增加一套系统，记录能拦截的高度
			a[k][1]=h;// 记录本次拦截的高度
		}
	}
	// 输出
```

```
    //输出
    cout<<k<<endl;
    for(int i=1;i<=k;i++)
    {
        int t=1;
        cout<<i<<':';
        while(a[i][t]!=0)
        {
            cout<<a[i][t++]<<' ';
        }
        cout<<endl;
    }
    return 0;
}
```

纪念品分组

```
#include <bits/stdc++.h>//10-1484    javacn    推荐学习
using namespace std;
/*
思路：最大值和最小值尝试分一组
1. 如果最大值 + 最小值 <=w，则分到一组
2. 否则，最大值单独分一组
*/
int a[30010],w,n,c = 0;
int main()
{
	cin>>w>>n;
	for(int i = 1;i <= n;i++){
		cin>>a[i];
	}
	sort(a+1,a+n+1);
	int h = 1,t = n;// 表示头尾在哪里
	while(h <= t)
	{
		if(a[h] + a[t] <= w)	// 头尾分一组
		{
			c++;
			h++;
			t--;
		}
		else
		{
			c++;	// 尾单独分一组
			t--;
		}
	}
	cout<<c;
	return 0;
}
```

数据量不大的情况下，可以贪心：

```
#include <bits/stdc++.h>//11-1413-1     javacn     推荐学习
using namespace std;
//1. 从最长可能切出的长度（max(a[i])），逐个向下尝试（最小是 1）
//2. 如果在长度为 i 的情况下能够切割出 >=m 根绳子，这个值 i 就是最大值
int n,a[110],m,c;
int ma = INT_MIN;//ma 代表最长的绳子的长度

int main() {
	cin>>n;
	for(int i = 1; i <= n; i++) {
		cin>>a[i];
		ma = max(ma,a[i]);
	}
	cin>>m;
	// 从最长的绳子长度向下尝试，找到第 1 个满足条件的长度
	for(int i = ma; i >= 1; i--)
	{
		// 计算在长度为 i 的情况下，能切割出多少根绳子
		c = 0;
		// 循环所有绳子的长度
		for(int j = 1; j <= n; j++)
		{
			c = c + a[j] / i;
			// 如果切割出来的绳子的数量 >=m，找到解了
			if(c >= m)
			{
				cout<<i;
				return 0;// 函数遇到 return 0 立刻结束
			}
		}
	}
	cout<<"Failed";
	return 0;
}
```

数据量大的情况下，可以二分答案：

```
#include <bits/stdc++.h>//11-1413-2    javacn
using namespace std;
// 本题满足单调性：切割长度越长，得到的数量一定越少。
const int N = 110;
int a[N];
int l = 1,r = 0,mid;
int n,m;
// 检验函数：检验切割长度为 mid，能否切割出 >=m 根绳子
bool check(int mid){
	int cnt = 0;
	for(int i = 1;i <= n;i++){
		cnt += a[i] / mid;
	}
	return cnt >= m;
}
int main()
{
	cin>>n;
	for(int i = 1;i <= n;i++){
		cin>>a[i];
		r = max(r,a[i]);// 找出最大数
	}
	cin>>m;
	while(l <= r)	// 二分答案
	{
		mid = l + r >> 1;
		if(check(mid))	l = mid + 1;// 如果检验通过：得到了 >=m 根绳子
		else		r = mid - 1;
	}

	if(l - 1 == 0) cout<<"Failed";
	else cout<<l-1;
	return 0;
}
```

```
#include <bits/stdc++.h>//12-1561-1  买木头  二维数组求解:    javacn
using namespace std;
int main()
{
	int n,m,l_1,s_1,i,j;
	// 记录 n 个商家每个商家的木头的长度和数量
	int a[10010][2];
	int r;// 最长木头长度
	cin>>n>>m>>l_1>>s_1;
	// 第一个商家木头长度和数量
	a[1][1] = l_1;
	a[1][2] = s_1;
	int max = a[1][1];// 最长木头的长度

	for(i = 2;i <= n;i++)
	{
		a[i][1] = ((a[i - 1][1] * 37011 + 10193) % 10000) + 1;
		a[i][2] = ((a[i - 1][2] * 73011 + 24793) % 100) + 1;

		if(a[i][1] > max)	max = a[i][1];
	}
	int c;	// 在每个长度下，各个供应商能供多少根木头
	for(i = max;i >= 1;i--)
	{
		c = 0;
		for(j = 1;j <= n;j++)	c += a[j][1] / i * a[j][2];// 循环供应商
		if(c >= m)// 如果根数够
		{
			r = i;
			break;
		}
	}
	cout<<r<<endl;
	return 0;
}
```

二分求解：

```
#include <bits/stdc++.h>//12-1561-2    javacn
using namespace std;
const int N = 10010;
int len[N],cnt[N];
int n,m;
bool check(int mid)// 检验切割长度为 mid，能否切出 m 根以上的木头
{
	int c = 0;
	for(int i = 1;i <= n;i++)	c += len[i]/mid*cnt[i];
	return c >= m;
}
int main()
{
	cin>>n>>m>>len[1]>>cnt[1];
	int l = 1,r = len[1],mid;

	for(int i = 2;i <= n;i++)	// 递推出每个供货商的木头长度和数量
	{
		len[i]=((len[i-1]*37011+10193)%10000)+1;
		cnt[i]=((cnt[i-1]*73011+24793)%100)+1;
		r = max(len[i],r);
	}

	while(l <= r)	// 二分可能的长度
	{
		mid = l + r >> 1;
		// 如果长度为 mid 能切出 >=m 个木头，继续找更长的木头
		if(check(mid))	l = mid + 1;
		else  r = mid - 1;
	}
	cout<<l-1;
	return 0;
}
```

任务调度

本题注意用 long long。

```
#include <bits/stdc++.h>//13-1551    javacn
using namespace std;
/*
贪心
  策略：由于每个任务完成的总时间 = 任务本身时间 + 排队时间
  因此不难想到，优先完成消耗时间少的任务
  1. 对所有任务升序排序
  2. 计算出每个任务完成的总时间
  总时间 = 当前任务消耗时间 + 上一个任务完成的总时间
*/
long long a[100010],n,r,s = 0;
int main()
{
        cin>>n>>r;
        for(int i = 1;i <= n;i++)
        {
                a[i] = (r % 100) + 1;
                r = (r * 6807 + 2831) % 20170;
        }

        // 排序
        sort(a+1,a+n+1);

        // 计算每个任务完成的总时间
        for(int i = 1;i <= n;i++)
        {
                a[i] = a[i] + a[i-1];
                s = s + a[i];
        }

        cout<<s;
        return 0;
}
```