7-1 合并果子

贪心

```
3
1 2 9
```

1. 合并1和2 -> 3 , 9
2. 合并3和9 -> 12

```
10
3 5 1 7 6 4 2 5 4 1
```

不再演示这个大样例。

总之，贪心策略是：每一次操作都选当前最小和次最小的两堆。

做n-1次策略，最终答案是最优的。

代码

priority_queue q 是大数优先的优先级队列--大根堆

priority_queue<int,vector,greater>q是小数优先的优先级队列--小根堆

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    int ans = 0;
    priority_queue<int , vector<int> , greater<int>> q;
    while(n--){
        int x;
        cin >> x;
        q.push(x);
    }
    while(q.size() > 1){
        auto t1 = q.top();
        q.pop();
        auto t2 = q.top();
        q.pop();
        ans += t1 + t2;
        q.push(t1 + t2);
    }
    cout << ans << "\n";
    return 0;
}
```

7-2 二分查找

```
数组：1 2 3 4 5
查找 3 的下标
```

数组做一个映射：

$$a[i] = \begin{cases} 1 & \text{if } a[i] \geq 3 \\ 0 & \text{if } a[i] < 3 \end{cases}$$

数组变成了**0 0 1 1 1**

1就是check函数的true，**0**就是check函数的false

我们为了查找第一个出现的`1`的下标，就需要在check函数返回true的时候`（说明mid~n都是1）`做 r = mid ，若返回false`（说明：1~mid都是0）`，让l = mid + 1。

```cpp
#include <bits/stdc++.h>
using namespace std;

bool check(vector<int> &a , int mid , int x)
{
    if(a[mid] >= x) return true;
    return false;
}
int main()
{
    int n , m;
    cin >> n >> m;
    vector<int> a(n + 1);
    for(int i = 1; i <= n; i++){
        cin >> a[i];
    }
    sort(a.begin() + 1 , a.end());
    vector<int> ans;
    while(m --){
        int x;
        cin >> x;
        int l = 1 , r = n;
        while(l < r){
            int mid = l + r >> 1;
            if(check(a , mid , x)){
                r = mid;
            }else{
                l = mid + 1;
            }
        }
        ans.push_back(l - 1);
    }
    for(int i = 0; i < ans.size(); i++){
        cout << ans[i] << " "[i == ans.size() - 1];
    }
    return 0;
}
```

7-3 【二分模板】自然对数

~~过子就行~~

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    double k;
    cin >> k;
    printf("%.6lf\n",log(k));
    return 0;
}
```

7-4 切木头

二分答案 + check函数--贪心

答案区间在1~1e9，不能遍历所有数**O（1e9）**，只能二分**O（log（1e9））**。二分答案是对答案区间做二分，所以本题二分的区间类似于**7-2 二分查找**的数组1~n。二分答案就是在[答案下限，答案上限]做二分。这个区间也满足00..01..111这样子的性质。原因是：如果最长长度L = 3可以在K刀内实现，那么最长长度L > 3都可以在K刀内实现。同**7-2 二分查找**做一个映射。**1**就是check函数返回true，**0**就是check函数返回false，我们要找第一个1出现的位置。

$$a[i] = \begin{cases} 1 & \text{if 切}k\text{刀，最长长度可以为}i \\ 0 & \text{if 切}k\text{刀，最长长度总是比}i\text{大} \end{cases}$$

最后一个问题是，写check函数。

check答案mid是true还是false <=> check 最长长度能否为mid。

由我们只能切K次，所以我们贪心地想最少切几刀就能让所有木头长度都小于等于K。、

举一个例子，现在有一根木头长度是9，要求切后所有木头小于等于3，那么你第一个刀切下长度为3的木头，第二刀再切一根长度为3的木头。 如果这个木头长度是10，要求切后所有木头小于等于3，那么策略是第一刀切下3，第二刀切下3，两刀过后有三根木头3 3 4，再切4，你可以切成1,3或者2,2都满足条件。

贪心策略:

$$\text{min\_刀数} = \begin{cases} \frac{l}{\text{mid}} - 1 & \text{if } l\%\text{mid} = 0 \\ \frac{l}{\text{mid}} & \text{if } l\%\text{mid} \neq 0 \end{cases}$$

一共有n根木头，对每一根木头都采取这个策略，累加cnt得到min_总刀数，如果cnt比k还大，意味着mid太小了，k刀不够用的，失败映射成0，返回false。否则返回true。

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long

bool check(vector<int> &a , int mid , int n , int k)
{
    int cnt = 0;
    for(int i = 1; i <= n; i++){
        if(a[i] % mid){
            cnt += a[i] / mid;
        }else{
            cnt += a[i] / mid - 1;
        }
```

```
    }
    if(cnt > k) return false;
    return true;
}
signed main()
{
    int n , k;
    cin >> n >> k;
    vector<int> a(n + 1);
    for(int i = 1; i <= n; i++){
        cin >> a[i];
    }
    int l = 1 , r = 1e9;
    while(l < r){
        int mid = l + r >> 1;
        if(check(a , mid , n , k)){
            r = mid;
        }else{
            l = mid + 1;
        }
    }
    cout << l << "\n";
    return 0;
}
```

7-5 今年12月不AC?

贪心 区间问题--最大不相交区间的数量

首先解释一下题意：有n个节目，每个节目都有自己的起止时间。有些节目的时间出现重叠，主人公看节目是从头看到尾，而且只看一个节目，不能同时看多个节目，主人公想知道他最多能看多少节目。举个例子，有三个节目。起止时间分别是 （1 ， 3） （3 ， 4） （1 ， 5）。那么主人公会选择在1--3看第一个节目，3--4看第二个节目，一共能看两个节目，如果看第三个节目不能看前两个节目了，总共只看一个节目，不是最优的策略。

想象一个数轴，把每个节目映射到一个线段，左右端点对应节目起止时间。选出一些线段组合，它们不相交（端点可以重合），**问题转化**：求组合里最大线段数量。

贪心策略：

我们想象自己是问题里的主人公，先给所有的节目**按照结束时间**从小到大排序，我们为了看尽量多的节目，所以首先看**最早结束**的节目，看完后，我们立刻在**还没有开始的节目中**（此时**可能**有节目已经开始了，我们只能放弃它）去找**最早结束**的节目，看这个节目是最优的。我们总是在选择**当下最省时**的节目，得到**局部最优解**，实际上它也是**全局最优解**，省去数学证明~~（本人也忘咋证了）~~

```
#include <bits/stdc++.h>
using namespace std;

#define int long long

struct node
{
    // l , r 是线段左右端点（对应节目起止时间）
    int l , r;
};

bool cmp(node a , node b)
```

```
{
    return a.r < b.r;
}

signed main()
{
    int n;
    cin >> n;
    vector<node> a(n + 1);
    for(int i = 1; i <= n; i++){
        //输入每一个节目的起止时间（对应线段的左右端点）
        cin >> a[i].l >> a[i].r;
    }
    //输入0
    int x; cin >> x;
    //按照线段右端点从小到大排序（按照节目的结束时间从小到大排序）
    sort(a.begin() + 1 , a.end() , cmp);
    //ans是答案，即看的节目数量
    int ans = 0;
    for(int i = 1; i <= n; ){
        //看第i个节目
        ans++;
        int R = a[i].r;
        int j = i + 1;
        // a[j].l < R <=> 节目的开始时间小于第i个节目的结束时间 <=>看完第i个节目后，第j个节目已经开始了，不选j
        // 找到第一个a[j].l >= R 的节目：j，由于数组已经按照结束时间从小到大排序，所以j是我们下一个要看的节目
        while(j <= n && a[j].l < R) j++;
        //下一个节目是j：把j赋给i
        i = j;
    }
    cout << ans << "\n";
    return 0;
}
```

7-6 h4176.赶作业

贪心

想让扣的分最低，肯定是先做那些扣分最多的作业，尽可能地保持剩下的作业即使没能完成也可以让扣的分最小化，那第一步就是对作业**按扣的分数从大到小排序**了，下一步要确定做作业的顺序了。

作为学生，我们一般来说都是什么时候完成作业的呢？就是**deadline**，即使扣分很大，我们保证在**最后一天**做完就行了。

贪心策略：排完序，我们从最高分的作业开始做，每次都在截止时间或者靠近截止时间完成作业就OK，这样保证扣分大的作业顺利完成，扣分小的作业也尽可能完成。

我们搞一个vis数组，如果是0，表示这一天还没有安排作业，可以使用；如果是1，表示已经有安排了，不能使用。我们对于第i个作业，在第**1**天到第**a[i].dd**（第i个作业的deadline）之间选距离**a[i].dd**最近的**j**，给**vis[j]**设置为**1**，表示这天已被安排；如果第第**1**天到第**a[i].dd**（第i个作业的deadline）的vis值都是1，意味第i个作业无法完成，需要扣分，**ans += a[i].score**

```
#include <bits/stdc++.h>
using namespace std;
```

```cpp
#define int long long

struct node
{
    int dd , score;
};

bool cmp(node a , node b)
{
    return a.score > b.score;
}
void solve()
{
    int n;
    cin >> n;
    vector<node> a(n + 1);
    for(int i = 1; i <= n; i++){
        //输入deadline
        cin >> a[i].dd;
    }
    for(int i = 1; i <= n; i++){
        //输入score
        cin >> a[i].score;
    }
    //按照socre从大到小排序
    sort(a.begin() + 1 , a.end() , cmp);
    //初始化答案ans
    int ans = 0;
    vector<int> f(n + 1 , 0);
    for(int i = 1; i <= n; i++){
        bool ok = false;
        //j <- a[i].dd to 1 ，找到第一个满足 f[j] == 0 的，把f[j]设为1：在第j天写第i个作业
        for(int j = a[i].dd; j >= 1; j--){
            if(!f[j]){
                //成功找到，ok 设置为 true
                ok = true;
                f[j] = 1;
                break;
            }
        }
        //如果ok还是false ，说明没有找到，第i个作业无法完成，扣分
        if(!ok) ans += a[i].score;
    }
    cout << ans << "\n";
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int _; cin >> _;
    while(_--) solve();
    return 0;
}
```

7-7 【USACO】进击的奶牛

与**7-4切木头**套路一样的**二分答案 + 贪心**

二分答案区间的思路和7-4是一样一样的。

不同的地方在于贪心：先给牛棚坐标排序。给定一个答案mid后，总是期望让每两头相邻的牛之间距离尽可能接近于mid，用cnt记录能放多少头牛，如果大于等于c头，说明mid距离还不够大，mid是可行的答案。（last是上一头牛在的牛棚的坐标）

（ps：如果cnt>=c 返回false 否则返回true。答案区间依旧是00..01..11，这次我们要找的答案是最后一个0而不是第一个1了。）

```
bool check(vector<int> &x , int mid , int n , int c)
{
    int cnt = 1;
    int last = 1;
    for(int i = 1; i <= n; i++){
        if(x[i] - x[last] >= mid){
            last = i;
            cnt++;
        }
    }
    if(cnt >= c) return false;
    else return true;
}
```

```
#include <bits/stdc++.h>
using namespace std;

#define int long long

bool check(vector<int> &x , int mid , int n , int c)
{
    int cnt = 1;
    int last = 1;
    for(int i = 1; i <= n; i++){
        if(x[i] - x[last] >= mid){
            last = i;
            cnt++;
        }
    }
    if(cnt >= c) return false;
    else return true;
}

signed main()
{
    int n , c;
    cin >> n >> c;
    vector<int> x(n + 1);
    for(int i = 1; i <= n; i++){
        cin >> x[i];
    }
    sort(x.begin() + 1 , x.end());
    int l = 0 , r = 1e9 + 1;
    while(l < r){
```

```
        int mid = l + r >> 1;
        if(check(x , mid , n , c)){
            r = mid;
        }else{
            l = mid + 1;
        }
    }
    cout << l - 1 << "\n";
    return 0;
}
```