合并果子
签到题，多种写法都可以解决此题。
枚举求解：枚举数轴上的每一个点，计算出所有果子与此点的距离之和，取一个枚举得到的最优解。时间复杂度 O(n^2)。

```
#include<bits/stdc++.h>
using namespace std;
const int inf=1e9;
char ch[105];
int main()
{
	ios::sync_with_stdio(false);
	cin.tie(0);
	int n,ans=inf; cin>>n;
	cin>>ch+1;
	for(int i=1;i<=n;i++)
	{
		int sum=0;
		for(int j=1;j<=n;j++) if(ch[j]=='*') sum+=abs(i-j);
		ans=min(ans,sum);
	}
	cout<<ans;
	return 0;
}
```

正解：若果子数为奇数取中间位置的果子作为合并点。若为偶数则取中间位置的两个果子之间的任意一点作为合并点。
以下是证明过程。

将两端的果子所在位置两两配对，处于当前颜色内部的所有点到包围此颜色的两端点距离之和相等，且处于当前颜色外部的点到包围此颜色的两端点距离之和比内部点的要大。
时间复杂度 O(n)。

```
#include<bits/stdc++.h>
using namespace std;
const int inf=1e9;
char ch[105];
int rec[105];
int main()
{
```

```
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n,ans=0,cnt=0; cin>>n;
    cin>>ch+1;
    for(int i=1;i<=n;i++) if(ch[i]=='*') rec[++cnt]=i;
    for(int i=1;i<=n;i++) if(ch[i]=='*') ans+=abs(rec[(cnt+1)/2]-i);
    cout<<ans;
    return 0;
}
```

括号匹配
进行栈操作的模拟即可。定义一个 top 栈顶变量，
枚举到'('进栈，top++,枚举到')',top--。如果 top 在过程中变成负数，
或者 top 最终不是 0，答案为 NO，否则为 YES。
时间复杂度为 O(n)

```
#include<bits/stdc++.h>
using namespace std;
string s;
int top;
main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin>>s;
    bool ok=1;
    for(auto u:s)
    {
        if(u=='(') top++;
        else if(u==')')
        {
            if(!top){ok=0;break;}
            top--;
        }
    }
    if(ok&&!top) cout<<"YES"<<'\n';
    else cout<<"NO"<<'\n';
}
```

小小 p 与耳机
贪心。容易证明按每个订单的截止时间升序排序处理最优。
反证法：假设某个订单贡献了最大延迟时间，我们交换他和其他订单的位置，答案会更优。
发现把它和截止时间比它大的订单交换，会使答案变坏，订单截止时间不变，
总耗时变大了。

如果把它和截止时间比它小的订单交换，更小的订单换到当前位置，订单总耗时不变，截止时间变小了，答案也是变坏。
因此，我们无法通过交换排序后的最大贡献订单的位置来使得答案更优，假设不成立。
综上，按截止时间排序最优，时间复杂度 O(nlogn)，大家可以自己造数据验证一下。

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=1e5+5;
int n;
struct node
{
	int tim,ed;
}nd[N];
bool cmp(node a,node b)
{
	if(a.ed!=b.ed) return a.ed<b.ed;
	return a.tim<b.tim;
}
main()
{
	ios::sync_with_stdio(false);
	cin.tie(nullptr);
	cin>>n;
	for(int i=1;i<=n;i++)
		cin>>nd[i].tim>>nd[i].ed;
	sort(nd+1,nd+1+n,cmp);
	int res=0,mx=0;
	for(int i=1;i<=n;i++)
	{
		res+=nd[i].tim;
		mx=max(mx,res-nd[i].ed);
	}
	cout<<mx;
}
```

鲨鱼泡泡
这是一道维护区间共点的裸题。有一种比较常用的套路，将每个区间断开成左右端点，按左右端点坐标排序。我们记左端点为入点，右端点为出点。
枚举到左端点时，代表左端点所在的区间加入，并和剩余未删除的区间共点；枚举到右端点时，则代表右端点所在的区间离开，我们需要删去。如果两个区间的加入和离开共点时，加入的优先级要大于离开。
维护区间时，我们用一个变量 cnt 记录当前有几个区间共点，再开两个 multiset 各自维护当前所有共点区间的左右端点最值即可。

由于一个大区间的左右端点的贡献最多来自两个小区间的左右端点，当选取区间 m>=2 时，显然我们选取 cnt>=m 时的情况进行统计就好了，
但是当 m==1 时，我们则需要特判一下。
时间复杂度 O(nlogn)

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
ios::sync_with_stdio(false);
cin.tie(0);
int n,m,len,ans=0; cin>>n>>m>>len;
multiset <pair<int,int>> s;
multiset <pair<int,int>> b;
map <int,pair<int,int>> a;
for(int i=1;i<=n;i++)
{
int l,r; cin>>l>>r;
s.insert({l,-i});
s.insert({r,i});
b.insert({l,r});
a[i]={l,r};
}
if(m==1)
{
for(int i=1;i<=n;i++) ans=max(ans,a[i].second-a[i].first+1);
cout<<len-ans;
return 0;
}
int maxr=-1,cnt=0;
while(!s.empty())
{
pair<int,int> g=*s.begin();
s.erase(s.begin());
if(g.second<0)
{
maxr=max(maxr,a[-g.second].second);
cnt++;
if(cnt<m) continue;
pair<int,int> h=*b.begin();
ans=max(ans,maxr-h.first+1);
}
else
{
```

```
cnt--;
pair<int,int> h=a[g.second];
b.erase(b.find(h));
}
}
cout<<len-ans;
return 0;
}
```