码上回顾

二分答案

二分答案

此类问题查找的目标不明确，但**判断条件**是明确的。

需要把二分算法彻底执行完，才能确定最后的结果。

如：切割木材

切5段等长的木材，每段最长的长度，

此类问题称为**二分答案**。

码上回顾

回顾练习1

【NOIP2001】在顺序表(2，5，7，10，14，15，18，23，35，41，52)中，用二分法查找14，所需的关键码比较的次数为（　　）。注意：中间值=(l+r)/2

A．2

B．3

C．4

D．5

码上回顾

回顾练习1

【NOIP2001】在顺序表(2，5，7，10，14，15，18，23，35，41，52)中，用二分法查找14，所需的关键码比较的次数为（ C ）。注意：中间值=(l+r)/2

A. 2

B. 3

C. 4

D. 5

次数 \ 序号	1	2	3	4	5	6	7	8	9	10	11
1	2	5	7	10	14	15	18	23	35	41	52
2	2	5	7	10	14	15					
3			7	10	14	15					
4				10	14	15					

码上得知-1-复习贪心算法

码上得知

复习贪心算法

有4堆胡萝卜，兔子只能从每堆里拿走1根胡萝卜。目标：拿的4根胡萝卜总重最大。

每堆可以看做：阶段1　阶段2　阶段3　阶段4

每堆最重的胡萝卜就是每个阶段的最优解

阶段性的最优选择，
被称为：**局部最优解**

码上得知

复习贪心算法

有4堆胡萝卜，兔子只能从每堆里拿走1根胡萝卜。目标：拿的4根胡萝卜总重最大。

每堆可以看做：阶段1　阶段2　阶段3　阶段4

每堆最重的胡萝卜就是每个阶段的最优解

阶段性的最优选择，
被称为：**局部最优解**

通过**局部最优解**，形成一个最终的最优结果，称之为**全局最优解**。

通过局部最优解逐步推导出全局最优解的思想，称之为**贪心算法**。

码上得知

复习贪心算法题型

①选择类问题：

n堆胡萝卜，每堆选一根，要求选出的总重最大。

策略：每堆选最重。

码上得知

复习贪心算法题型

③部分背包问题：

n种商品，每种商品有总价、有数量，有一个可以装m数量商品的背包，要求背包装入商品价值最大。

5种商品

	商品1	商品2	商品3	商品4	商品5
总价	12	8	20	7	15
数量	6	2	4	7	5

可装载数量10

码上得知

复习贪心算法题型

③部分背包问题：

n种商品，每种商品有总价、有数量，有一个可以装m数量商品的背包，要求背包装入商品价值最大。

策略：每次选单价最贵。

5种商品

	商品1	商品2	商品3	商品4	商品5
总价	**12**	**8**	**20**	**7**	**15**
数量	**6**	**2**	**4**	**7**	**5**
单价	2	4	5	1	3

可装载数量10

商品3 ■■■■ 价值20 4件
商品2 ■■ 价值8 2件
商品5 ■■■■■ 价值12 4件
商品1 ■■■■■■
商品4 ■■■■■■■

最大装载价值为：40

码上得知

复习贪心算法题型

④分组乘船问题：

n个人上船，每艘船的载重都是一样的，一艘船最多上两人，怎么分组，船用的最少。

体重：15　17　102　70　90　68

码上得知

复习贪心算法题型

④分组乘船问题：

n个人上船，每艘船的载重都是一样的，一艘船最多上两人，怎么分组，船用的最少。

前面乘船问题得到一个结论，要想上人多，从轻的开始上。(目的是让第二名船员有更大的机会上船)
所以当前问题可以**假设**从最轻的开始上。

体重：15　17　102　70　90　68

码上得知

复习贪心算法题型

④分组乘船问题：

n个人上船，每艘船的载重都是一样的，一艘船最多上两人，怎么分组，船用的最少。

策略：每次尝试最小体重与最大体重同组，否则最大体重单独一组。

前面乘船问题得到一个结论，要想上人多，从轻的开始上。(目的是让第二名船员有更大的机会上船)
所以当前问题可以**假设**从最轻的开始上。

体重：15　17　102　70　90　68

可作为第2名上船的人员

第二人不需要节省空间，所以尽量选最重。
(这样可让剩下人的体重尽量小，为减少船只数量做准备)

码上得知

复习贪心算法题型

⑤排队等待问题：

n个人排队，1位老师答疑，每个人都有一个答疑时长，每个人花的时间等于自己的答疑时间加等前面人的时间，怎样排队，达到所有人所花时间总和最小。

答疑顺序

	第1次答疑	第2次答疑	第3次答疑	第4次答疑
小童3分钟 	3			
大熊1分钟 	3	1		
小鹿2分钟 	3	1	2	
小美6分钟 	3	1	2	6

总答疑时间：$3\times4 + 1\times3 + 2\times2 + 6\times1$

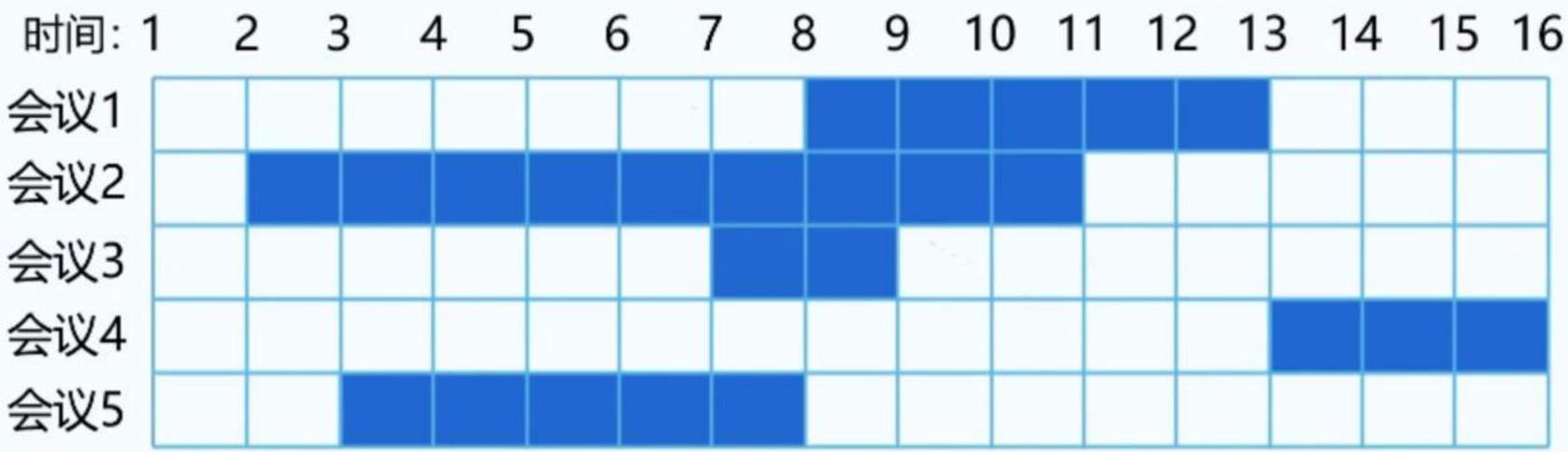
固定值

码上得知

复习贪心算法题型

⑥区间问题：

n个会议，每个会议有开始时间、结束时间，在一个固定的时间段里，会议不冲突的前提下开尽量多的会议。

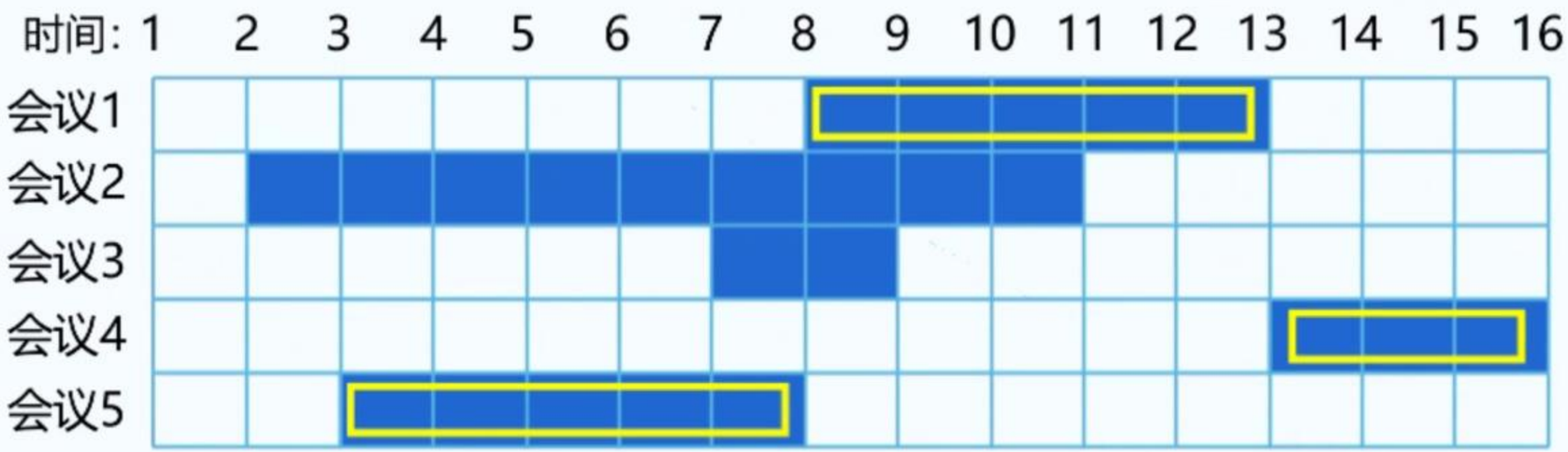


码上得知

复习贪心算法题型

⑥区间问题：

n个会议，每个会议有开始时间、结束时间，在一个固定的时间段里，会议不冲突的前提下开尽量多的会议。

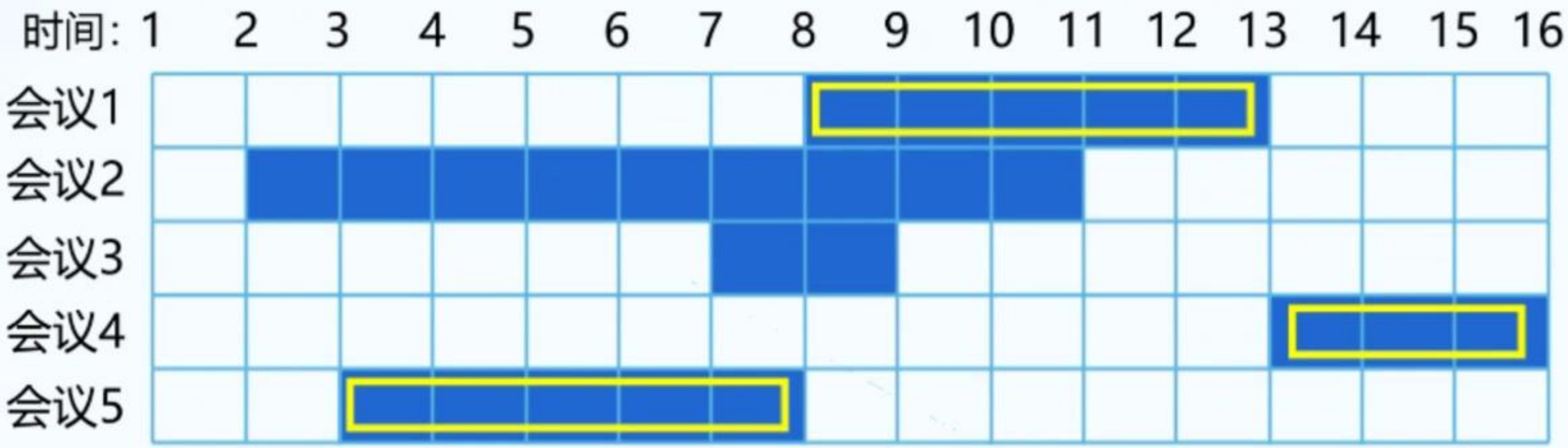


观察一下，这里最多可以开**3场**会议。

逻辑：先安排最早结束的会议

码上得知

复习贪心算法题型

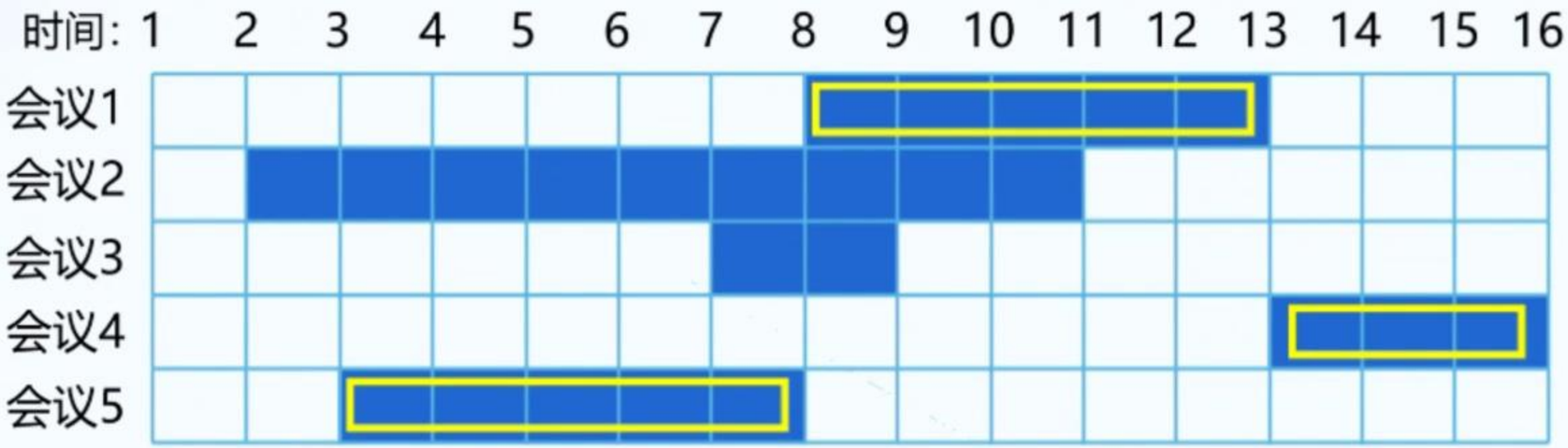


证明：
结束时间最早，证明在它之前不可能有完整会议了，可选的会议都会在它之后，那就需要让后续的时间越多越好，而结束时间最早的会议后方的时间最充裕，所以优先选结束时间早的会议安排。

策略：先选择最早结束的会议，后面所选会议的开始时间不小于上一个所选会议的结束时间。

码上得知

复习贪心算法题型



证明：
结束时间最早，证明在它之前不可能有完整会议了，可选的会议都会在它之后，那就需要让后续的时间越多越好，而结束时间最早的会议后方的时间最充裕，所以优先选结束时间早的会议安排。

策略：先选择最早结束的会议，后面所选会议的开始时间不小于上一个所选会议的结束时间。

会议这道题目属于**区间问题**的一个分支：**最大不相交区间数量问题**

码上得知

进阶题型

区间问题

区间选点问题

排队等待问题

n个人排队，**m位**老师答疑。

导弹拦截问题

解码时刻

问题分析

【输入样例】

4	4名学生
2 3 1 2	每位同学的答疑时间

假设按照这个序列，依次选择老师

【输出样例】

11

解码时刻

问题分析

【输入样例】

4	4名学生
2 3 1 2	每位同学的答疑时间

假设按照这个序列，依次选择老师

【输出样例】

11

解码时刻

问题分析

【输入样例】

4	4名学生
2 3 1 2	每位同学的答疑时间

假设按照这个序列，依次选择老师

【输出样例】

11

两位老师时

1号老师	
第1次答疑	第2次答疑
2	
2	1

2号老师	
第1次答疑	第2次答疑
3	
3	2

先选的，尽可能小

答疑第1位: 2×2 → 3×2

答疑第2位: 1×1 → 2×1 → 固定值

总答疑时间：13

策略：答疑时间短的先答疑，按照老师顺序依次选择。

解码时刻

问题分析

【输入样例】

4	4名学生
2 3 1 2	每位同学的答疑时间

【输出样例】

11

答疑时间少的先答疑，
依次选择老师

解码时刻

问题分析

为什么是依次选择老师?

等待时间也算入总花费时间内，所以每次选择等待时间少的老师答疑。

解码时刻

问题分析

为什么是依次选择老师?

等待时间也算入总花费时间内，所以每次选择等待时间少的老师答疑。

第一位同学选择1号老师或2号老师都可以，

后面学生每次选择等待时间少的老师答疑，

从而形成**依次**选择老师的顺序。

解码时刻

问题分析

为什么是依次选择老师?

等待时间也算入总花费时间内，所以每次选择等待时间少的老师答疑。

第一位同学选择1号老师或2号老师都可以，
后面学生每次选择等待时间少的老师答疑，
从而形成**依次**选择老师的顺序。

1号老师需要等待3分钟，2号老师需要等待5分钟，
选择1号老师更优，依然是**依次**选择老师。

解码时刻

问题分析

假设为学生进行编号

i为学生编号时：
当i≤2时：
第i个人**答疑完成时间**=自己的**答疑时间**
当i>2时：
第i个人**答疑完成时间**=第i个人的**答疑时间** + **i-2号**同学的**答疑完成时间**

解码时刻

解题步骤

1. 创建需要的数组及变量。

```
int a[401]={}; //每位同学的答疑时间
int n; //学生人数
```

2. 读入数据。

```
cin>>n;
for(int i=1;i<=n;i++){
    cin>>a[i];
}
```

解题步骤

4. 计算每个学生的答疑完成时间。

解码时刻

解题步骤

4. 计算每个学生的答疑完成时间。

第3位同学开始计算

```
for(int i=3;i<=n;i++){ //计算学生答疑完成时间

}
```

解题步骤

4. 计算每个学生的答疑完成时间。

```
for(int i=3;i<=n;i++){ //计算学生答疑完成时间
    a[i]=a[i]+a[i-2];
}
```

4. 统计所有学生的答疑完成时间。

```
int t=0;
for(int i=1;i<=n;i++){
    t+=a[i]; //统计所有学生的答疑完成时间
}
cout<<t;
```

完整代码

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int a[401]={};
    int n,t=0;
    cin>>n;
    for(int i=1;i<=n;i++)
        cin>>a[i];
    sort(a+1,a+1+n); //升序排序
    for(int i=3;i<=n;i++){ //计算学生答疑完成时间
        a[i]=a[i]+a[i-2];
    }
    for(int i=1;i<=n;i++)
        t+=a[i]; //统计所有学生的答疑完成时间
    cout<<t;
    return 0;
}
```

亲自出码

问题分析

【输入样例】

6 3

9 5 3 7 10 5

【输出样例】

8.67

亲自出码

问题分析

【输入样例】

6 3	6名学生，3位老师
9 5 3 7 10 5	每位同学的答疑时间

【输出样例】

8.67 平均答疑时间,保留两位小数

1号老师	答疑完成时间	2号老师	答疑完成时间	3号老师	答疑完成时间
第1位同学	3	第2位同学	5	第3位同学	5
第4位同学	3+7=10	第5位同学	5+9=14	第6位同学	5+10=15

总答疑时间：3+10+5+14+5+15=52

平均答疑时间：总时间52/总人数6=8.67

策略：答疑时间短的先答疑，按照老师顺序依次选择。

问题分析

观察排队的顺序

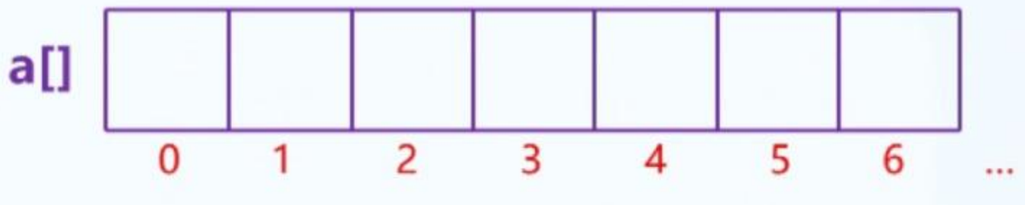
当有2位老师时，编号差为2。
当有3位老师时，编号差为3。
当有**r**位老师时，编号差为**r**。

亲自出码

解题步骤

1. 创建需要的数组及变量。

```
int a[401]={}; //每位同学的答疑时间
int n,r; //n:学生人数  r:老师人数
```



亲自出码

解题步骤

4. 计算每个学生的完成答疑时间。

亲自出码

解题步骤

4. 计算每个学生的完成答疑时间。

```
//计算学生答疑完成时间
for(int i= r+1;i<=n;i++){
}
```

第r+1位同学开始计算

亲自出码

解题步骤

4. 计算每个学生的完成答疑时间。

```
//计算学生答疑完成时间
for(int i= r+1;i<=n;i++){
    a[i]=a[i]+a[i-r];
}
```

5. 统计所有学生的答疑完成时间。

```
int t=0;
for(int i=1;i<=n;i++){
    t+=a[i]; //统计所有学生的答疑完成时间
}
```

亲自出码

完整代码

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int a[401] = {};
    int n,r;
    cin >> n >> r;
    for(int i = 1;i <= n; i++){
        cin >> a[i];
    }
    sort(a+1,a+1+n);
    for(int i = r+1; i <= n;i++){
        a[i] = a[i] + a[i-r];
    }
    int t = 0;
    for(int i = 1; i <= n; i++){
        t += a[i];
    }
    cout << fixed << setprecision(2) << double(t) / n;
    return 0;
}
```

解码时刻

区间选点

【问题描述】

给定N个闭区间[l,r]，请在数轴上选择尽量少的点，使得每个区间内至少包含一个选出的点。输出所选点的最小数量。注意：位于区间端点上的点也算作区间内。

【输入格式】

第一行一个整数N，表示闭区间数量（$1 \le N \le 10^5$）。

接下来N行，每行两个整数l，r，分别表示一个区间的左端点和右端点（$-10^9 \le l \le r \le 10^9$）。

【输出格式】

输出一个整数，表示所选点的最小数量。

【输入样例】

```
5
0 3
1 2
-1 2
0 1
4 5
```

【输出样例】

```
2
```

解码时刻

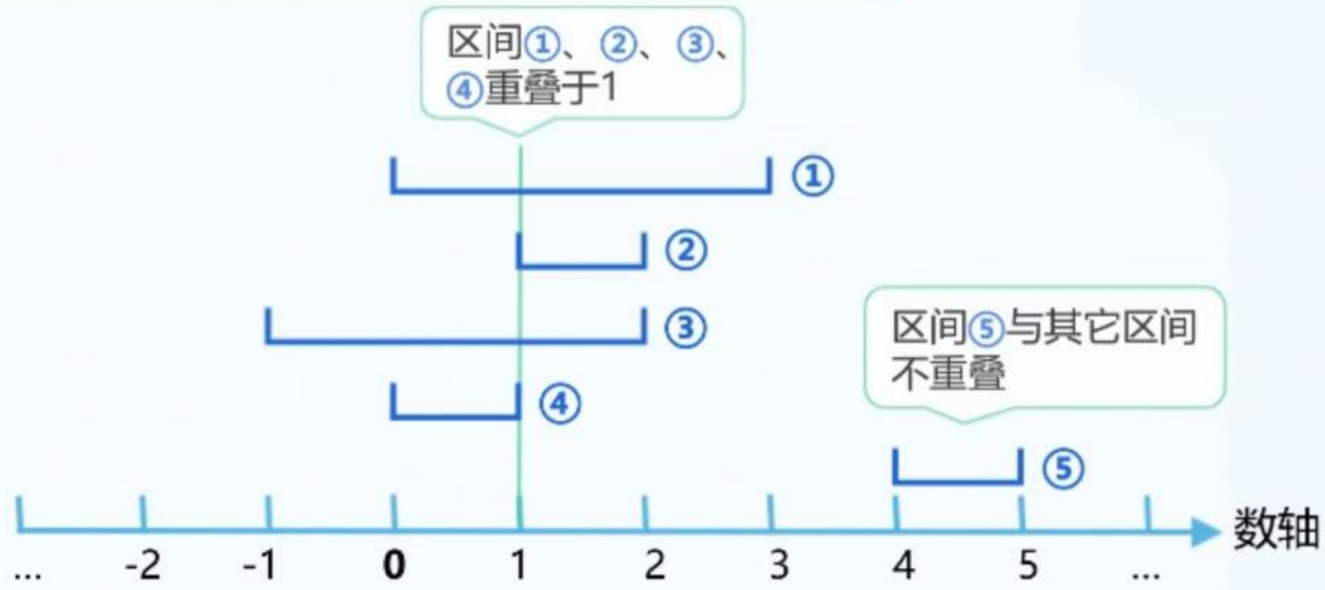
问题分析

【输入样例】

5	5个闭区间
0 3	[0,3]区间
1 2	[1,2]区间
-1 2	[-1,2]区间
0 1	[0,1]区间
4 5	[4,5]区间

【输出样例】

2



重叠区间选点

题目要求：数轴上选择尽量少的点，使得每个区间内至少包含一个选出的点

解码时刻

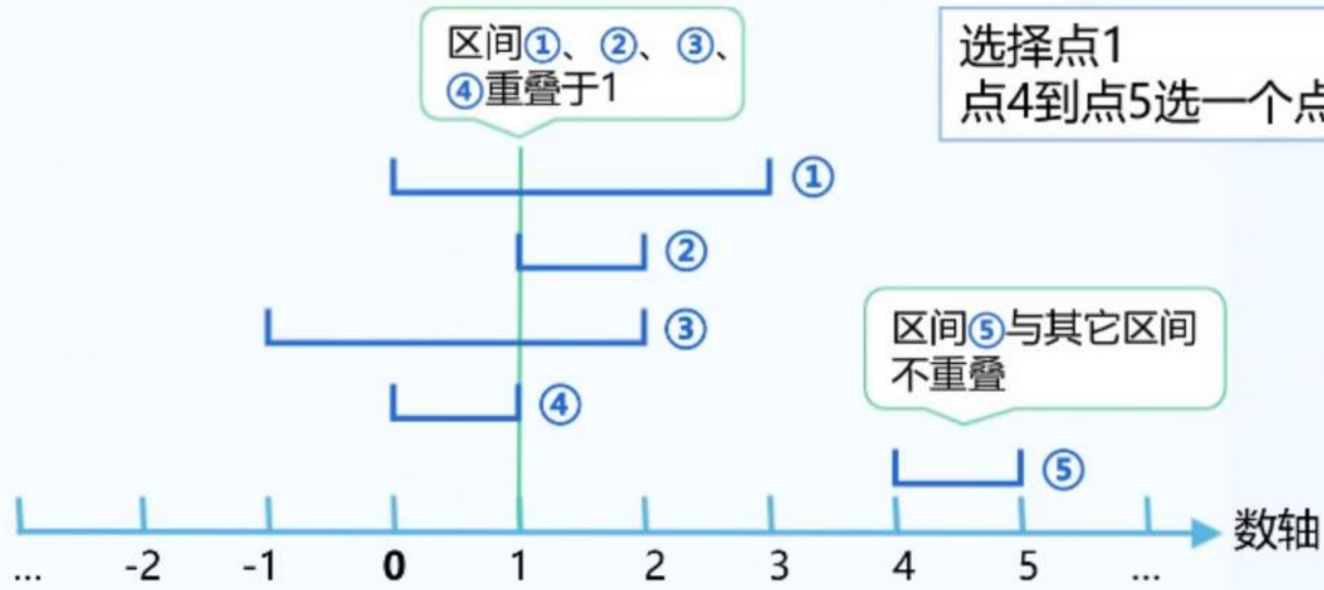
问题分析

【输入样例】

5	5个闭区间
0 3	[0,3]区间
1 2	[1,2]区间
-1 2	[-1,2]区间
0 1	[0,1]区间
4 5	[4,5]区间

【输出样例】

2 选点的最小数量



重叠区间选点

题目要求：数轴上选择尽量少的点，使得每个区间内至少包含一个选出的点

关键点： 重叠区间

两个区间为例：如果两个区间不重叠，则需选两个点；如果区间有重叠，则只需选一个点。

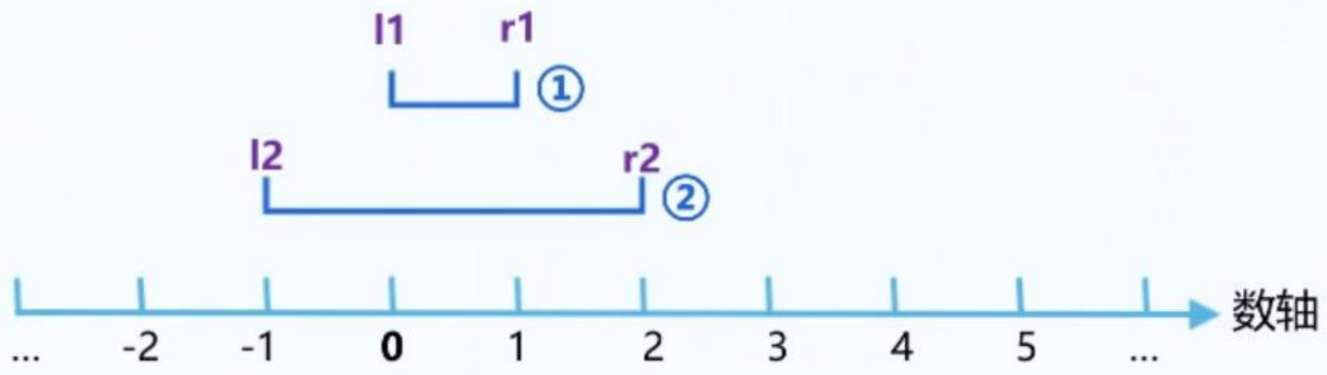
解码时刻

问题分析

如何判断两个区间是否有重叠?

假设：区间①的右端点 ≤ 区间②的右端点（**右端点升序**）

判断：区间①的**右端点** ≥ 区间②的**左端点** **证明有重叠**

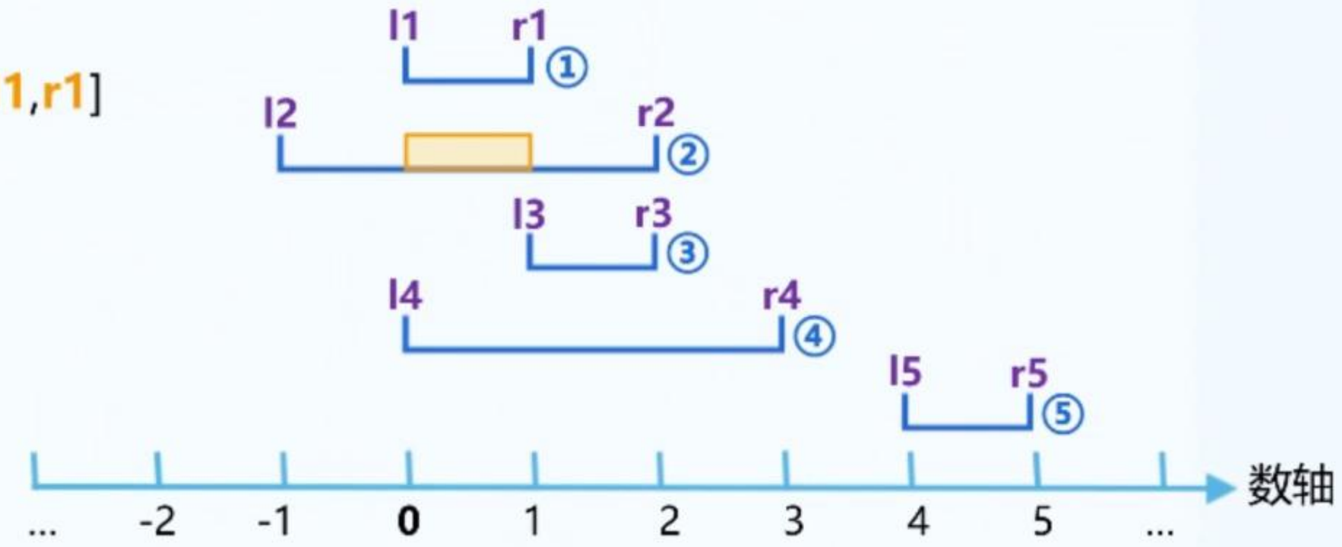


解码时刻

问题分析

判断多个区间是否有重叠，假设**右端点升序**

r1≥l2 区间①和区间②有重叠，重叠于[l1,r1]



解码时刻

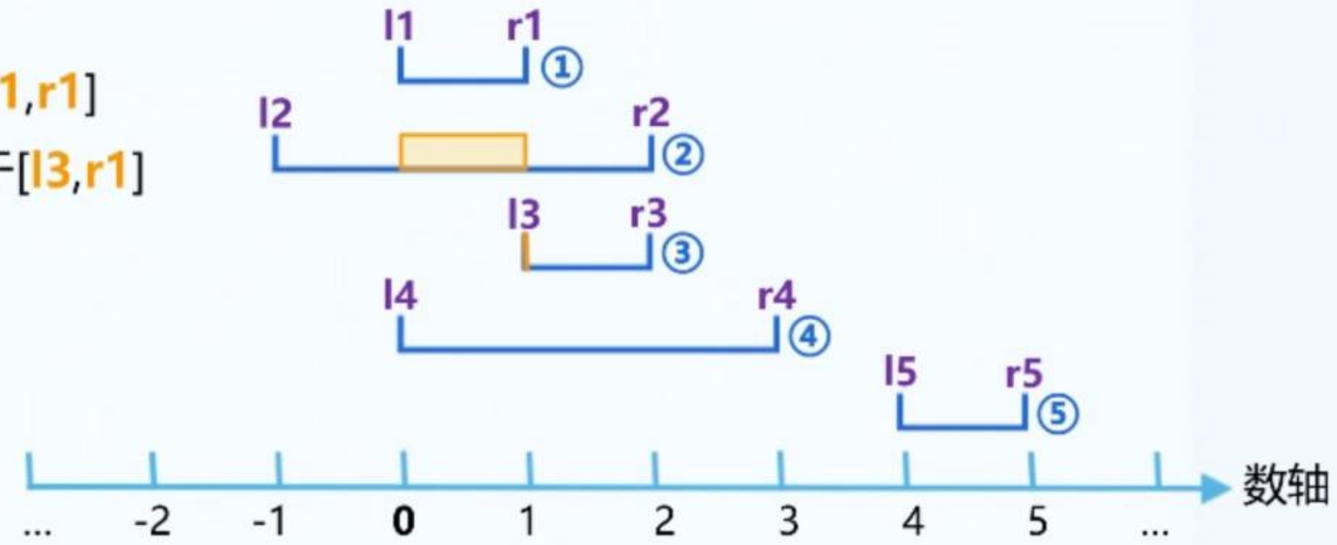
问题分析

判断多个区间是否有重叠，假设**右端点升序**

判断：**重叠区间**的**右端点** ≥ 下一个区间的**左端点** **证明有重叠**

r1≥l2 区间①和区间②有重叠，重叠于[l1,r1]

r1≥l3 重叠区间和区间③有重叠，重叠于[l3,r1]



解码时刻

问题分析

判断多个区间是否有重叠，假设**右端点升序**

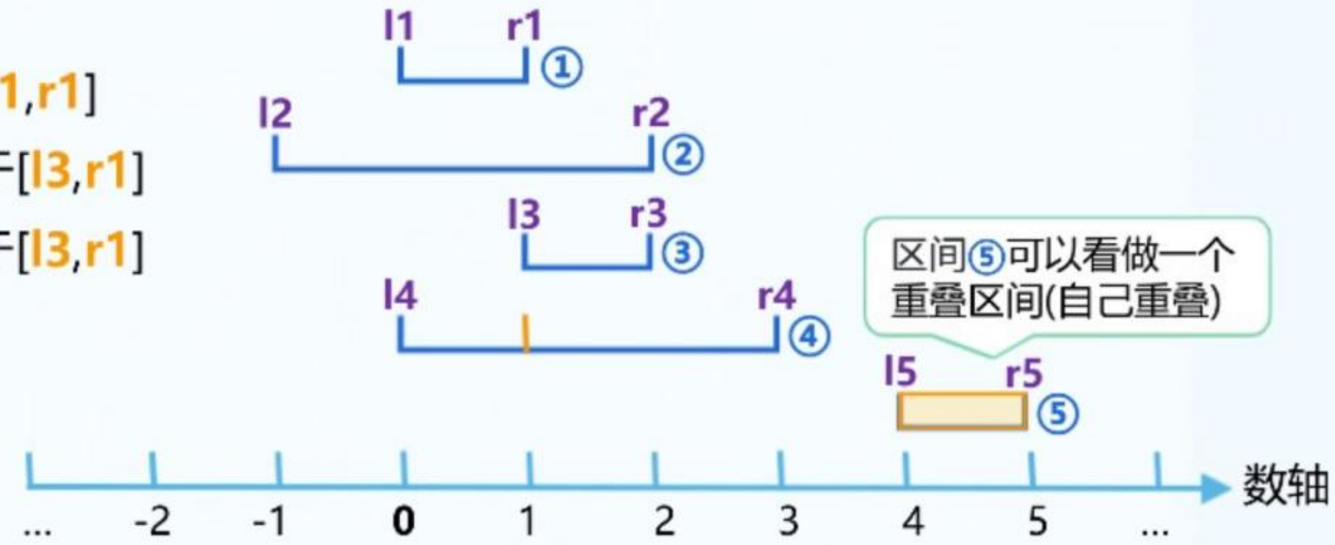
判断：**重叠区间**的**右端点** ≥ 下一个区间的**左端点** **证明有重叠**

r1≥l2 区间①和区间②有重叠，重叠于[l1,r1]

r1≥l3 重叠区间和区间③有重叠，重叠于[l3,r1]

r1≥l4 重叠区间和区间④有重叠，重叠于[l3,r1]

r1<l5 重叠区间和区间⑤**无**重叠



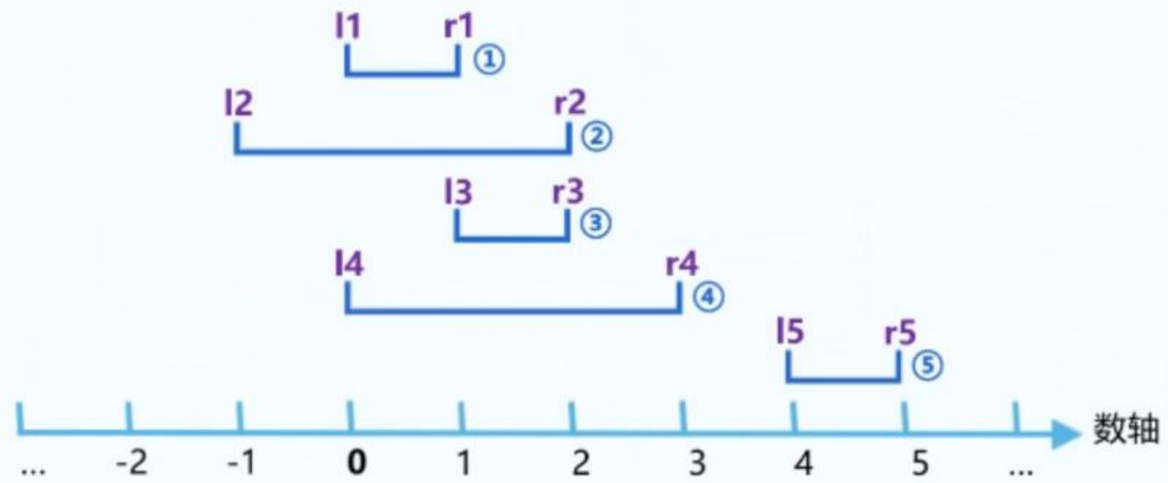
此时有两个不同的重叠区间，每个重叠区间都需要选一个点。

总结：重叠区间的数量就是选点的数量。

解码时刻

问题分析

寻找重叠区间步骤：



1. 区间按照右端点升序，第一个区间看做重叠区间
2. 从下一个区间开始比较，如果重叠区间的右端点≥当前区间的左端点，跳过当前区间(有重叠)；否则当前区间看做新的重叠区间，重复此过程。

解题步骤

右端点进行升序排序

	左端点	右端点
区间1	0	3
区间2	1	2
区间3	-1	2
区间4	0	1
区间5	4	5

	左端点	右端点
区间4	0	1
区间3	-1	2
区间2	1	2
区间1	0	3
区间5	4	5

解题步骤

右端点进行升序排序

	左端点	右端点
区间1	0	3
区间2	1	2
区间3	-1	2
区间4	0	1
区间5	4	5

	左端点	右端点
区间4	0	1
区间3	-1	2
区间2	1	2
区间1	0	3
区间5	4	5

左端点和右端点在排序时要同时移动，可以用结构体对数据进行存储。

1. 创建结构体及排序规则。

```
struct node{
    //l为左端点，r为右端点
    int l,r;
}a[100001];
```

```
//排序规则
bool cmp(node x,node y){
    //按照右端点升序排列
    return x.r<y.r;
}
```

解题步骤

2. 创建需要的变量并读入数据。

```
int n; //n个闭区间
cin>>n;
for(int i=1;i<=n;i++){
    cin>>a[i].l>>a[i].r;
}
```

	左端点	右端点
区间4	0	1
区间3	-1	2
区间2	1	2
区间1	0	3
区间5	4	5
	a[i].l	a[i].r

3. 按右端点升序排序。

```
sort(a+1,a+1+n,cmp);
```

解码时刻

解题步骤

4. 进行选点并输出结果。

```
int t=a[1].r; //t:存储重叠区间的右端点
int ans=1; //ans:存储重叠区间的数量
```

ans **1**

	左端点	右端点
区间4	0	1
区间3	-1	2
区间2	1	2
区间1	0	3
区间5	4	5
	a[i].l	a[i].r

t

重叠区间**右端点** ≥ 下一个区间**左端点**　跳过

解码时刻

解题步骤

4. 进行选点并输出结果。

```
int t=a[1].r; //t:存储重叠区间的右端点
int ans=1; //ans:存储重叠区间的数量
```

ans **1**

	左端点	右端点
区间4	0	1
区间3	-1	2
区间2	1	2
区间1	0	3
区间5	4	5
	a[i].l	a[i].r

t

重叠区间**右端点** ≥ 下一个区间**左端点**　跳过

重叠区间**右端点** ≥ 下一个区间**左端点**　跳过

重叠区间**右端点** ≥ 下一个区间**左端点**　跳过

解码时刻

解题步骤

4. 进行选点并输出结果。

```
int t=a[1].r; //t:存储重叠区间的右端点
int ans=1; //ans:存储重叠区间的数量
```

ans **2**

	左端点	右端点	
区间4	0	1	
区间3	-1	2	
区间2	1	2	
区间1	0	3	
区间5	4	5	**t**
	a[i].l	a[i].r	

重叠区间**右端点** ≥ 下一个区间**左端点** 跳过

重叠区间**右端点** ≥ 下一个区间**左端点** 跳过

重叠区间**右端点** ≥ 下一个区间**左端点** 跳过

重叠区间**右端点** < 下一个区间**左端点** **增加**

完整代码

```cpp
#include<bits/stdc++.h>
using namespace std;
struct node{
    int l,r;
}a[100001];
bool cmp(node x,node y){
    return x.r<y.r;
}
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i].l>>a[i].r;
    }
    sort(a+1,a+1+n,cmp);
    int t=a[1].r;
    int ans=1;
    for(int i=2;i<=n;i++){
        if(t<a[i].l){
            t=a[i].r;
            ans++;
        }
    }
    cout<<ans;
    return 0;
}
```

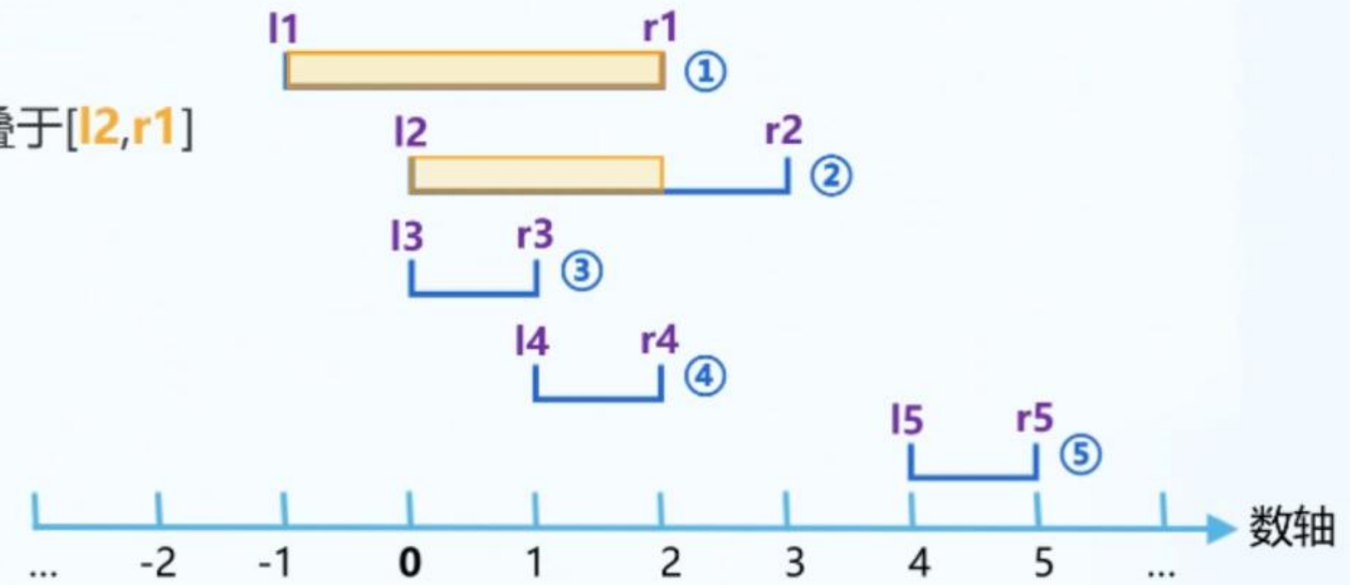
解码时刻

左端点升序

左端点升序，判断多个区间是否有重叠。

判断：**重叠区间**的**右端点** ≥ 下一个区间的**左端点**　　**证明有重叠**

r1≥l2 **重叠**区间和区间②有重叠，重叠于[l2,r1]



解码时刻

左端点升序

左端点升序，判断多个区间是否有重叠。

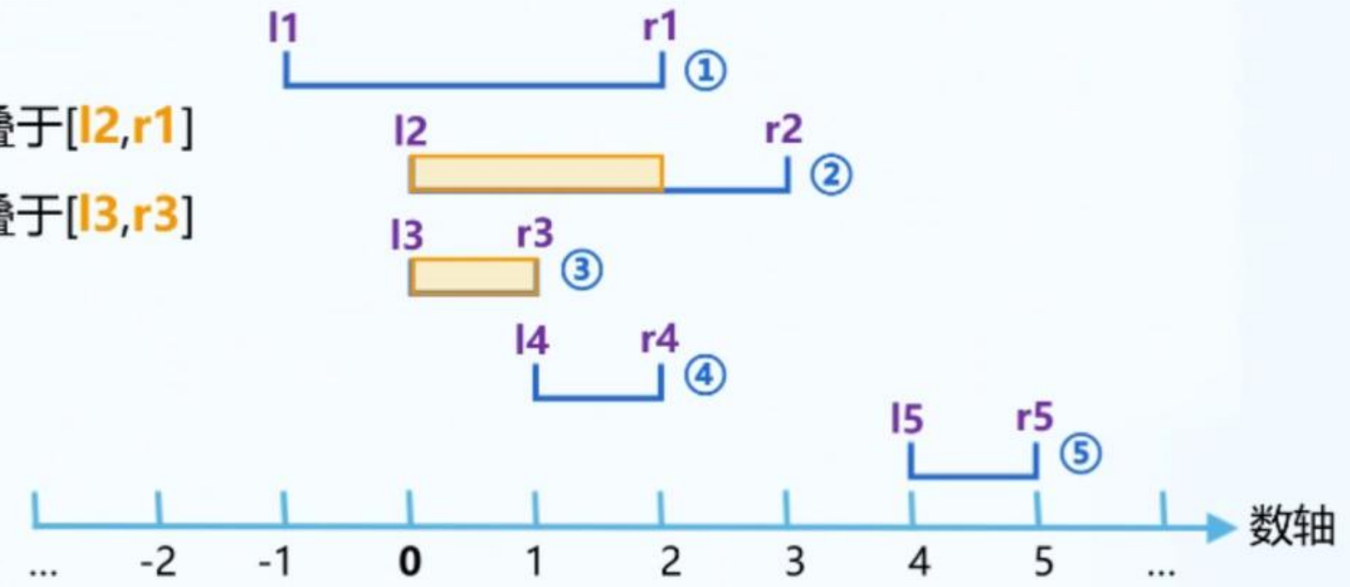
重叠区间的右端点发生变化时，更新右端点

(1) 判断：**重叠区间**的**右端点** ≥ 下一个区间的**左端点** **证明有重叠**

(2) 条件(1)成立，当前区间右端点和旧重叠区间右端点比较，较小值为新重叠区间右端点

r1≥l2 重叠区间和区间②有重叠，重叠于[l2,r1]

r1≥l3 重叠区间和区间③有重叠，重叠于[l3,r3]



解码时刻

左端点升序

左端点升序，判断多个区间是否有重叠。

重叠区间的右端点发生变化时，更新右端点

(1) 判断：**重叠区间**的**右端点** ≥ 下一个区间的**左端点** **证明有重叠**

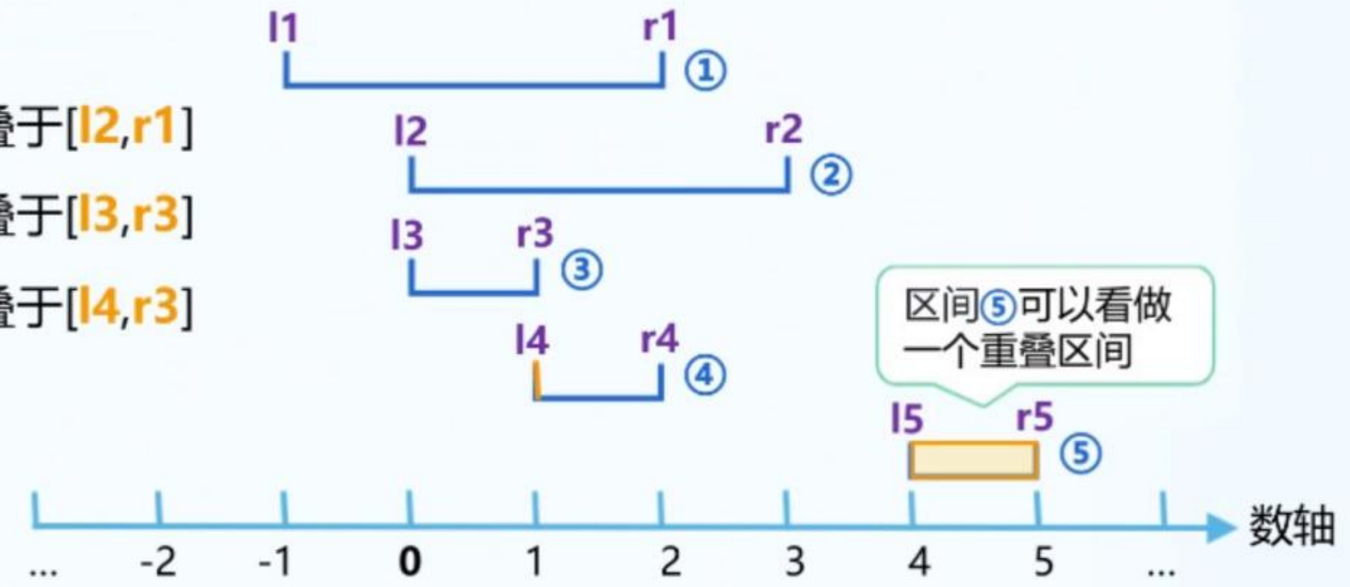
(2) 条件(1)成立，当前区间右端点和旧重叠区间右端点比较，较小值为新重叠区间右端点

r1≥l2 重叠区间和区间②有重叠，重叠于[l2,r1]

r1≥l3 重叠区间和区间③有重叠，重叠于[l3,r3]

r3≥l4 重叠区间和区间④有重叠，重叠于[l4,r3]

r3<l5 重叠区间和区间⑤**无**重叠

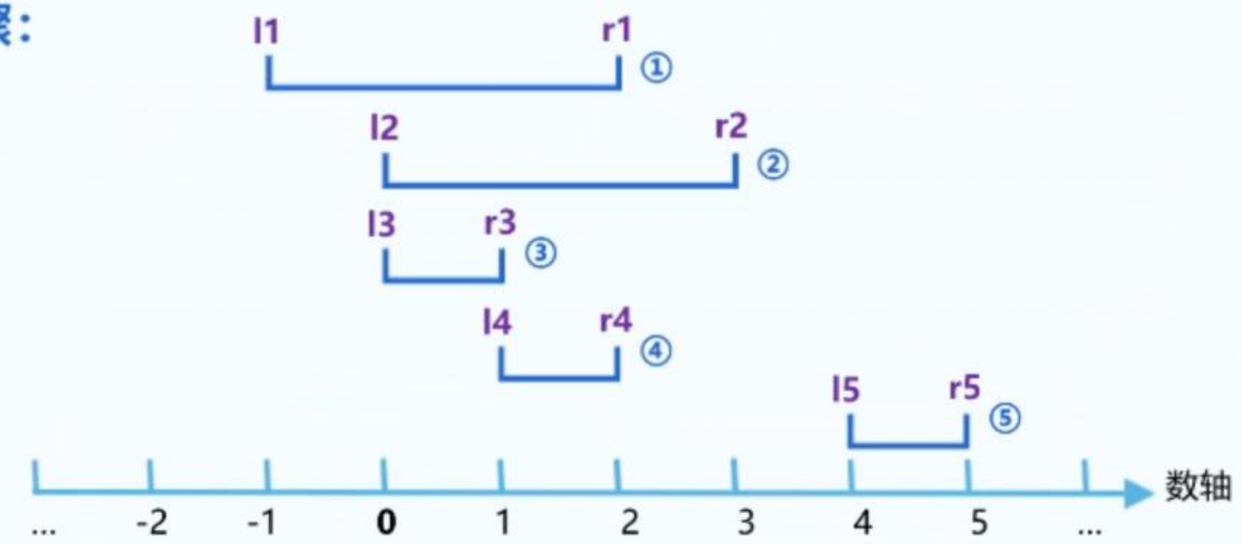


重叠区间的数量就是选点的数量。

解码时刻

问题分析

寻找重叠区间步骤：



1. 区间按照左端点升序，第一个区间看做重叠区间
2. 从下一个区间开始比较，如果重叠区间的右端点≥当前区间的左端点，**更新重叠区间右端点**，并跳过当前区间(有重叠)；否则当前区间看做新的重叠区间，重复此过程。

解题步骤

左端点进行升序排序。

	左端点	右端点
区间1	0	3
区间2	1	2
区间3	-1	2
区间4	0	1
区间5	4	5

	左端点	右端点
区间3	-1	2
区间1	0	3
区间4	0	1
区间2	1	2
区间5	4	5

左端点和右端点在排序时要同时移动，可以用结构体对数据进行存储。

1. 创建结构体及排序规则

```
struct node{
    //l为左端点，r为右端点
    int l,r;
}a[100001];
```

```
//排序规则
bool cmp(node x,node y){
    //按照左端点升序排列
    return x.l<y.l;
}
```

解码时刻

解题步骤

2. 创建需要的变量并读入数据。

```
int n; //n个闭区间
cin>>n;
for(int i=1;i<=n;i++){
    cin>>a[i].l>>a[i].r;
}
```

	左端点	右端点
区间3	-1	2
区间1	0	3
区间4	0	1
区间2	1	2
区间5	4	5
	a[i].l	a[i].r

3. 按左端点升序排序。

```
sort(a+1,a+1+n,cmp);
```

解码时刻

解题步骤

4. 进行选点并输出结果。

```
int t=a[1].r; //t:存储重叠区间的右端点
int ans=1; //ans:存储重叠区间的数量
```

ans 1

t

	左端点	右端点
区间3	-1	2
区间1	0	3
区间4	0	1
区间2	1	2
区间5	4	5
	a[i].l	a[i].r

解码时刻

解题步骤

4. 进行选点并输出结果。

```
int t=a[1].r; //t:存储重叠区间的右端点
int ans=1; //ans:存储重叠区间的数量
```

ans **1**

	左端点	右端点
区间3	-1	2
区间1	0	3
区间4	0	1
区间2	1	2
区间5	4	5
	a[i].l	a[i].r

t 更新重叠区间右端点

重叠区间**右端点≥**下一个区间**左端点** 更新重叠区间右端点(右端点较小值)

重叠区间**右端点≥**下一个区间**左端点** 更新重叠区间右端点(右端点较小值)

解码时刻

解题步骤

4. 进行选点并输出结果。

```
int t=a[1].r; //t:存储重叠区间的右端点
int ans=1; //ans:存储重叠区间的数量
```

ans 1

	左端点	右端点
区间3	-1	2
区间1	0	3
区间4	0	1
区间2	1	2
区间5	4	5
	a[i].l	a[i].r

t

重叠区间右端点≥下一个区间左端点 更新重叠区间右端点(右端点较小值)

重叠区间右端点≥下一个区间左端点 更新重叠区间右端点(右端点较小值)

重叠区间右端点≥下一个区间左端点 更新重叠区间右端点(右端点较小值)

重叠区间右端点<下一个区间左端点 增加

完整代码

```
#include<bits/stdc++.h>
using namespace std;
struct node{
    int l,r;
}a[100001];
bool cmp(node x,node y){
    return x.l<y.l;
}
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i].l>>a[i].r;
    }
    sort(a+1,a+1+n,cmp);
    int ans=1,t=a[1].r;
    for(int i=2;i<=n;i++){
        if(t>=a[i].l){
            t=min(t,a[i].r);
        }else{
            t=a[i].r;
            ans++;
        }
    }
    cout<<ans;
    return 0;
}
```

想一想

本题也可以使用**左端点降序**和**右端点降序**完成，核心思路还是找重叠区间的数量。
同学们课下思考一下如何解决。

问题分析

【输入样例】

6	6个导弹
389 207 300 200 310 65	每个导弹高度

【输出样例】

3 最少需要3套系统

问题分析

1. 导弹顺序是**固定**的。
2. 拦截系统的**第一发炮弹**可以是**任意的高度**，但是**以后拦截的每一发炮弹都不能高于前一发的高度**。

问题分析

1. 导弹顺序是**固定**的。
2. 拦截系统的**第一发炮弹**可以是**任意的高度**，但是**以后拦截的每一发炮弹都不能高于前一发的高度**。

207**小于**系统1拦截高度
可以拦截！

系统	系统拦截高度
系统1	207

问题分析

1. 导弹顺序是**固定**的。
2. 拦截系统的**第一发炮弹**可以是**任意的高度**，但是**以后拦截的每一发炮弹都不能高于前一发的高度**。

系统	系统拦截高度
系统1	207
系统2	300

遍历现有拦截系统，能拦截时，使用**最低**的系统拦截

问题分析

1. 导弹顺序是**固定**的。
2. 拦截系统的**第一发炮弹**可以是**任意的高度**，但是**以后拦截的每一发炮弹都不能高于前一发的高度**。

系统	系统拦截高度
系统1	200
系统2	300
系统3	310

根据题意已知，拦截高度越高越有利，所以我们优先选择**较小值**，保留其它系统的**更高拦截位置**！

遍历现有拦截系统，能拦截时，使用**最低**的系统拦截

不能拦截时，**增加**新系统拦截

问题分析

题目计算最少的拦截系统数量，则需要让每套拦截系统**尽量多次使用**，避免浪费。

策略：当出现新的导弹时，**优先**从**已有拦截系统**中，选取**高度最低**且**可成功拦截**的系统；如果没有，则增加一套新系统。

解题步骤

1. 创建所需数组及变量。

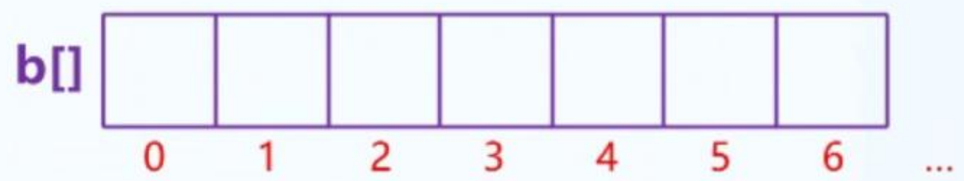
```
int n; //n枚导弹
int a[501]={}; //每枚导弹的高度
```

2. 读入数据。

解题步骤

3. 拦截第一枚导弹。

```
int b[501]={}; //系统的拦截高度
```



解题步骤

4. 拦截之后的导弹

遍历现有拦截系统，能拦截时，使用**最低**的系统拦截

不能拦截时，**增加**新系统拦截

解题步骤

4. 拦截之后的导弹

遍历现有拦截系统，能拦截时，使用**最低**的系统拦截

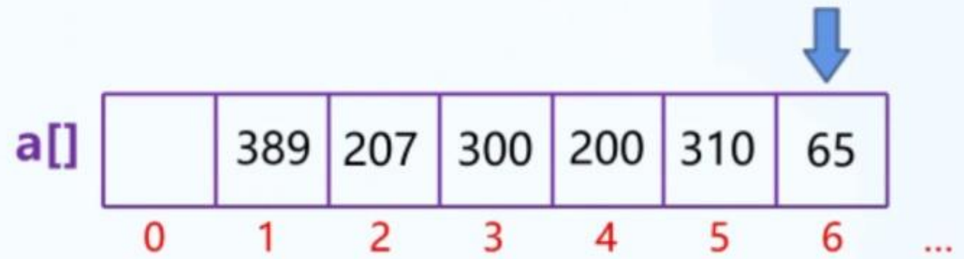
不能拦截时，**增加**新系统拦截

解题步骤

4. 拦截之后的导弹

```
for(int i=2;i<=n;i++){

    for(int j=1;j<=k;j++){ //遍历现有系统
        if(b[j]>=a[i]){ //是否可以拦截
            能拦截时，寻找最低的系统编号
        }
    }
    如果上面if没有执行，
        需要新增拦截系统
    否则
        更新能拦截且最低系统的高度
}
```



遍历现有拦截系统，能拦截时，使用**最低**的系统拦截

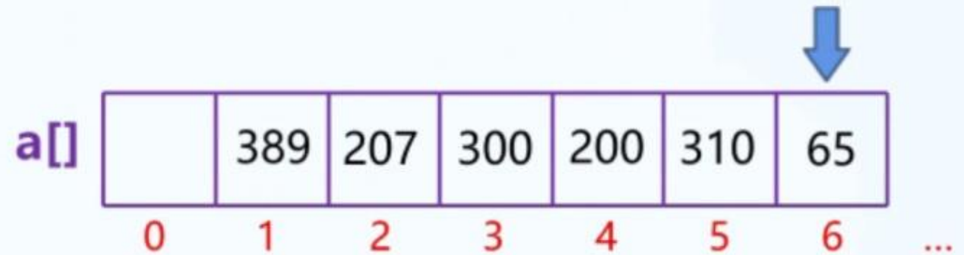
不能拦截时，**增加**新系统拦截

解题步骤

4. 拦截之后的导弹

```
for(int i=2;i<=n;i++){
    bool flag=false; //初始化现有系统不能拦截状态
    int m; //存储当前成功拦截导弹的最低值的编号
    for(int j=1;j<=k;j++){ //遍历现有系统
        if(b[j]>=a[i]){ //是否可以拦截
            if(flag==false){
                m=j; //第一套系统的编号
            }

            flag=true; //现有系统能拦截状态
        }
    }
    if(flag==false){ //新增拦截系统
        k++;
        b[k]=a[i];
    }else{ 更新能拦截且最低系统的高度} //更新系统拦截高度
}
```



遍历现有拦截系统，能拦截时，使用**最低**的系统拦截

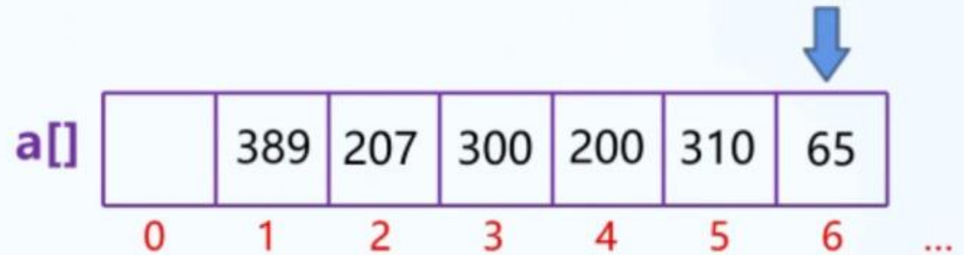
不能拦截时，**增加**新系统拦截

快码加鞭

解题步骤

4. 拦截之后的导弹

```
for(int i=2;i<=n;i++){
    bool flag=false; //初始化现有系统不能拦截状态
    int m; //存储当前成功拦截导弹的最低值的编号
    for(int j=1;j<=k;j++){ //遍历现有系统
        if(b[j]>=a[i]){ //是否可以拦截
            if(flag==false){
                m=j; //第一套系统的编号
            } else{
                if(b[j]<b[m])    m=j; //找到更小的拦截系统
            }
            flag=true; //现有系统能拦截状态
        }
    }
    if(flag==false){ //新增拦截系统
        k++;
        b[k]=a[i];
    }else{   b[m]=a[i];          } //更新系统拦截高度
}
```



遍历现有拦截系统，能拦截时，使用**最低**的系统拦截

不能拦截时，**增加**新系统拦截

完整代码

```
#include<bits/stdc++.h>
using namespace std;
int a[501]; //存储每枚导弹的高度
int main(){
    int n; //n枚导弹
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
    }
    int b[501]={}; //拦截系统
    b[1]=a[1]; //第一套系统拦截
    int k=1;
    for(int i=2;i<=n;i++){
        bool flag=false; //现有系统不能拦截状态
        int m;
        for(int j=1;j<=k;j++){ //遍历k套系统
            if(b[j]>=a[i]){ //可以拦截
                if(flag==false){ //找到第一个可拦截的系统
                    m=j; //第一个可拦截系统的编号
                    flag=true; //现有系统能拦截状态
                }else{
                    if(b[j]<b[m])    m=j; //找到更小的拦截系统
                }
            }
        }
        if(flag==false){ //新增拦截系统
            k++;    b[k]=a[i];
        }else    b[m]=a[i]; //m号系统拦截
    }
    cout<<k;
    return 0;
}
```

码上总结

总结题型

拦截导弹问题：

拦截系统，拦截的**第一发炮弹**可以是**任意的高度**，但是**以后拦截的每一发炮弹都不能高于前一发的高度。**某天，雷达捕捉到敌国的导弹来袭，告诉你这些来的导弹的高度，如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

策略：

当出现新的导弹时，优先从已有拦截系统中，选取高度最低且可成功拦截的系统；如果没有，则增加一套新系统。