区间问题
- 区间选点
- 最大不相交区间数量
- 区间分组
- 区间覆盖
  - GPT的贪心做法：（改天自己写写）
  - 贪心做法

Huffman树
- 合并果子

排序不等式
- 排队打水

绝对值不等式
- 仓库选址

推公式
- 耍杂技的牛

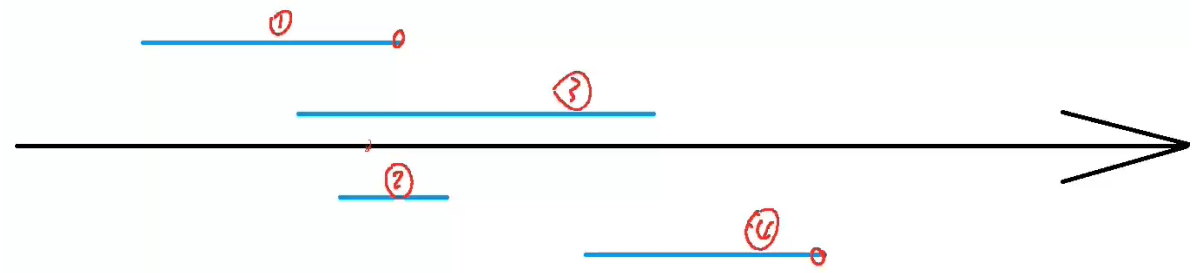
区间问题

区间选点

补充：贪心，就是每次选局部最优，在单峰状况下可以达到全局最优

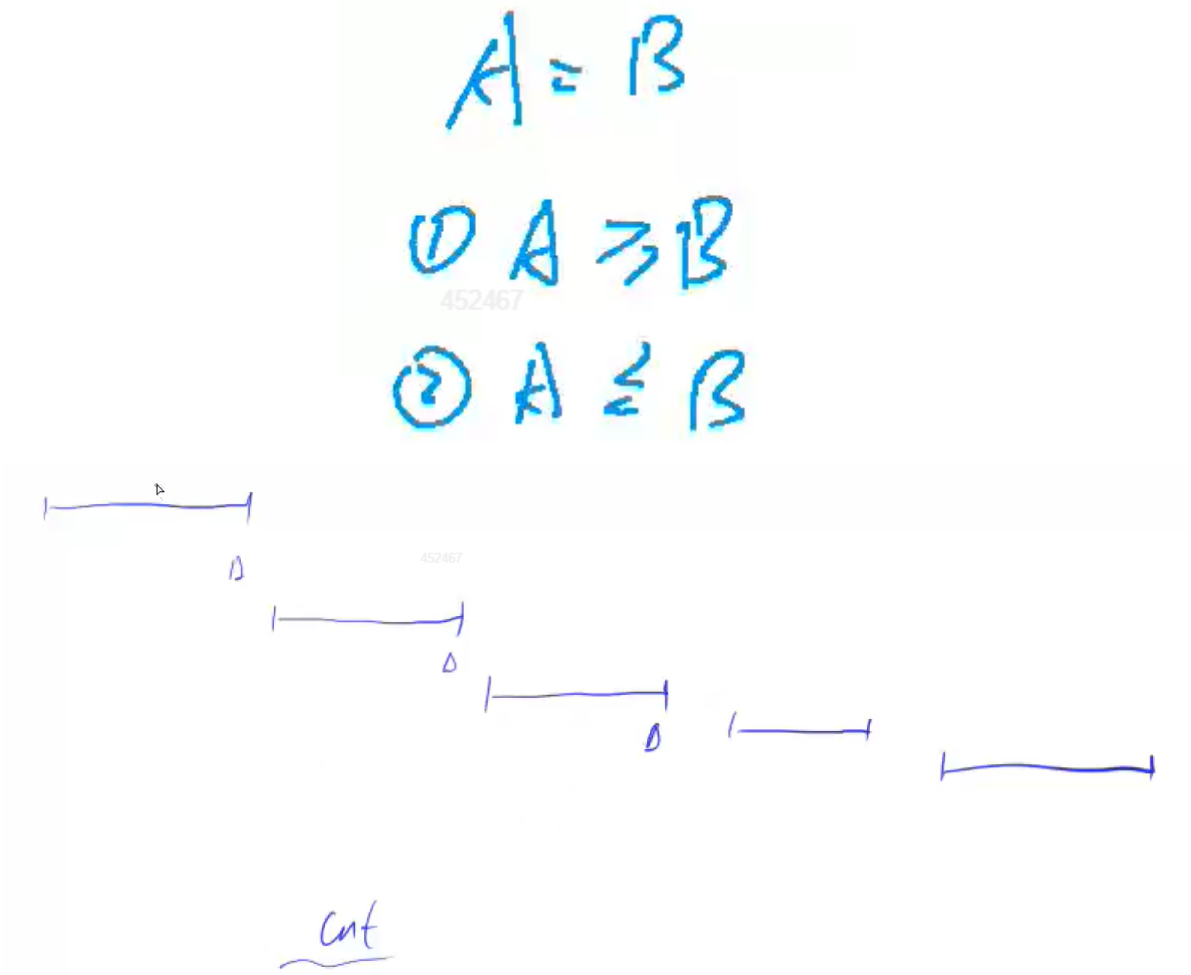
思路：先去尝试一些做法，区间问题无外乎就是排序

要么按左端点，要么右端点，要么双关键字

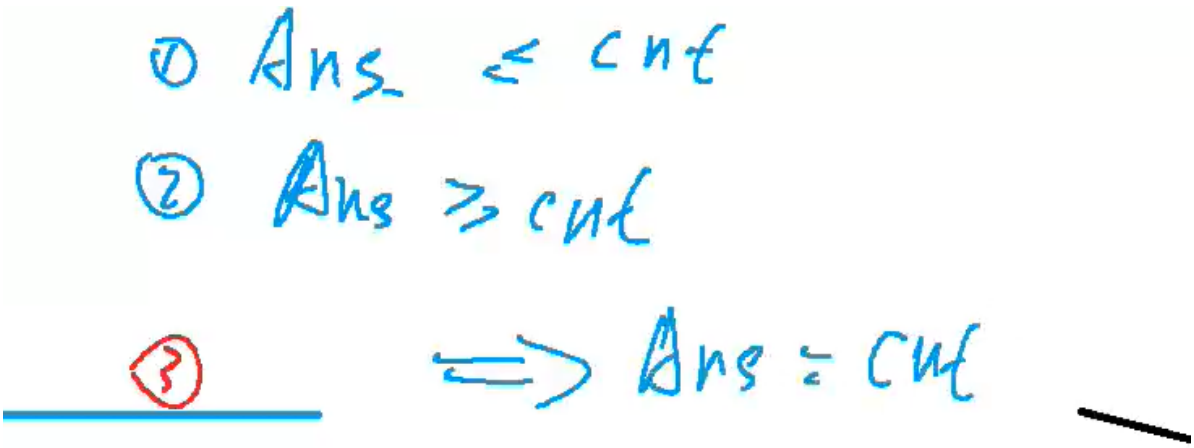


1. 将每个区间按右端点从小到大排序
2. 从前往后依次枚举每个区间
   如果当前区间中已经包含点，则直接pass
   否则，选择当前区间的右端点

在这个题里，怎么证明贪心是可行的呢？也就是要证明两个值相等，下面介绍数学中的常用套路



对于第二种情况，如果有cnt个不相交的区间，就意味着至少ans需要取cnt个点才能覆盖所有区间




```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n;
struct Range
{
    int l, r;
    // 由于要排序，所以需要重载一下我们的小于号
    bool operator < (const Range &W) const
```

```
    {
        return r < W.r;
    }
}range[N];

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i ++)
    {
        int l, r;
        scanf("%d%d", &l, &r);
        range[i] = {l, r};
    }

    sort(range, range + n); //默认是升序排序

    int res = 0; // 表示当前选择的点数
    int ed = -2e9; // 表示上一个点的下标，由于一开始哪一个点都没选，所以赋为负无穷
    for (int i = 0; i < n; i ++) // 枚举所有的区间
        if (range[i].l > ed)
        {
            res ++;
            ed = range[i].r;
        }

    printf("%d\n", res);

    return 0;
}
```

如果不适用运算符重载：

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n;
struct Range
{
    int l, r;
}range[N];

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i ++)
    {
        int l, r;
        scanf("%d%d", &l, &r);
        range[i] = {l, r};
    }
```

```
    sort(range, range + n, [](const Range &a, const Range &b) {
    return a.r < b.r;});


    int res = 0; // 表示当前选择的点数
    int ed = -2e9; // 表示上一个点的下标，由于一开始哪一个点都没选，所以赋为负无穷
    for (int i = 0; i < n; i ++) // 枚举所有的区间
        if (range[i].l > ed)
        {
            res ++;
            ed = range[i].r;
        }

    printf("%d\n", res);

    return 0;
}
```

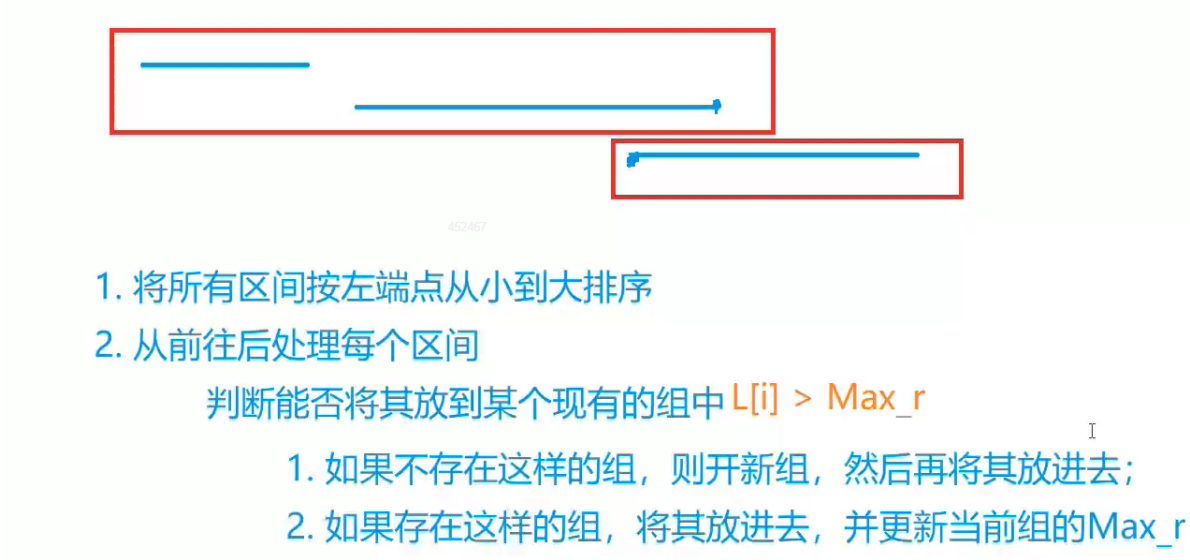
最大不相交区间数量

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n;

struct Range
{
    int l, r;
}range[N];


int main()
{
    cin >> n;
    for (int i = 0; i < n; i ++)
    {
        int l, r;
        cin >> l >> r;
        range[i] = {l, r};
    }

    sort(range, range + n, [](const Range &a, const Range &b){
        return a.r < b.r;
    });

    int res = 0;
    int ed = -2e9;
    for (int i = 0; i < n; i ++)
```

```
        if (range[i].l > ed)
        {
            res ++;
            ed = range[i].r;
        }

    cout << res << endl;

    return 0;

}
```

和上一题一模一样的代码过了

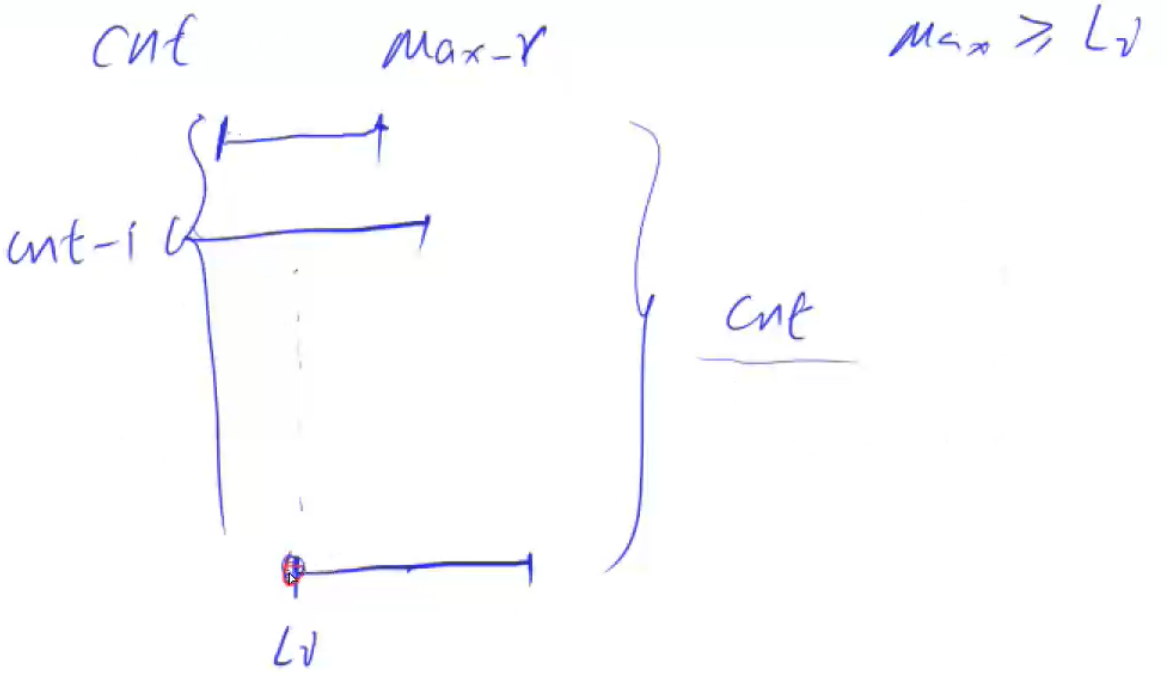
区间分组

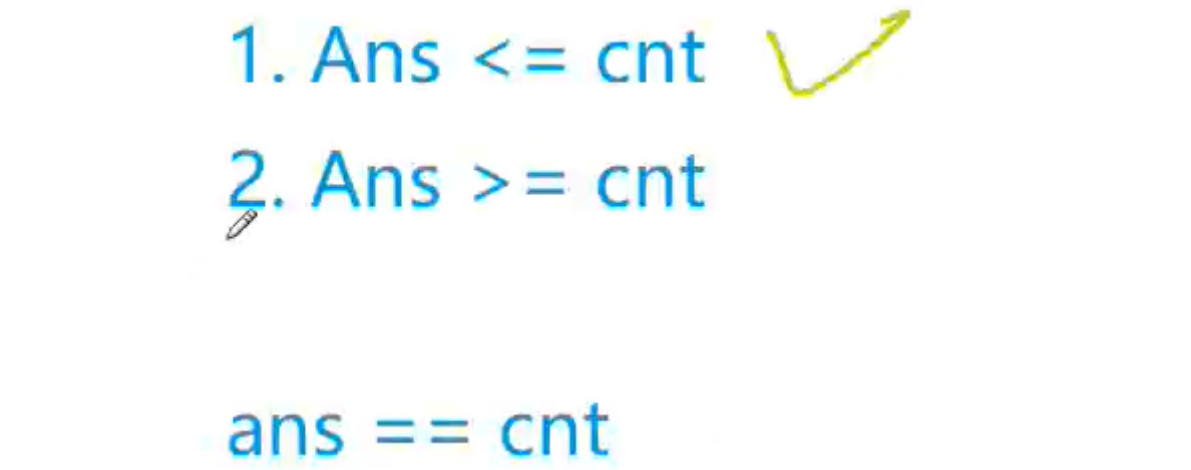


1. 将所有区间按左端点从小到大排序
2. 从前往后处理每个区间

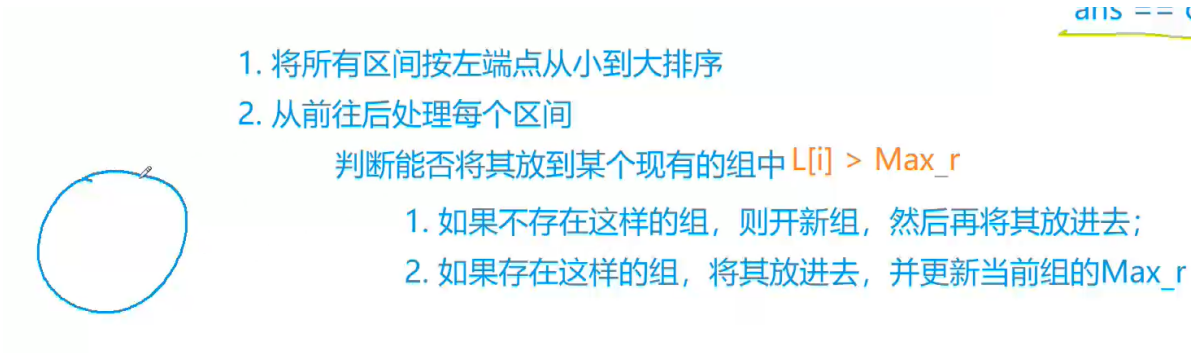
   判断能否将其放到某个现有的组中 L[i] > Max_r

   1. 如果不存在这样的组，则开新组，然后再将其放进去；
   2. 如果存在这样的组，将其放进去，并更新当前组的Max_r





问题来了，如何判断max_r <= li？这个问题可以用小根堆解决



所以这里用堆来维护每个组的max_r

```
#include <iostream>
#include <algorithm>
#include <queue> // 用到小根堆，引入queue

using namespace std;

const int N = 100010;

int n;
struct Range
{
    int l, r;
}range[N];

int main()
{
    scanf("%d", &n);

    for (int i = 0; i < n; i ++)
    {
        int l, r;
        scanf("%d%d", &l, &r);
        range[i] = {l, r};
    }

    sort(range, range + n, [](const Range &a, const Range &b){
        return a.l < b.l;
```

```
    });

    priority_queue<int, vector<int>, greater<int>> heap; // 小根堆的语法
    for (int i = 0; i < n; i ++)
    {
        auto r = range[i];
        if (heap.empty() || heap.top() >= r.l) heap.push(r.r);
        else
        {
            heap.pop();
            heap.push(r.r);
        }
    }

    printf("%d\n", heap.size());

    return 0;
}
```

注意:

特别解释一下这个地方为什么要用小根堆：因为要动态维护最小值，判断一下最小的max_r是不是小于li的

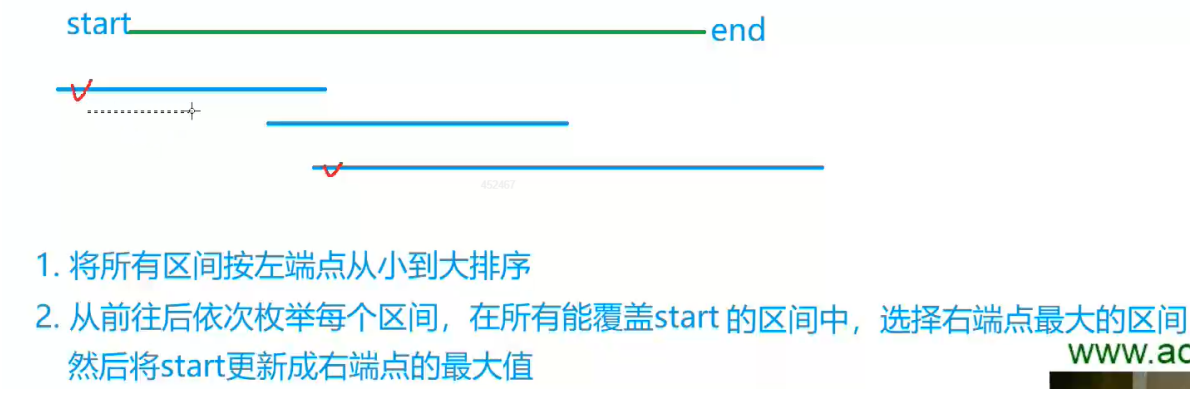
区间覆盖

GPT的贪心做法：（改天自己写写）

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Interval {
    int start, end;
};

// 比较函数：按照左端点升序排列，如果左端点相同则按照右端点降序排列
bool compareIntervals(const Interval &a, const Interval &b) {
    if (a.start == b.start) return a.end > b.end;
    return a.start < b.start;
}

int minIntervalsToCover(int s, int t, vector<Interval> &intervals) {
    // 特例处理：当 s == t 时，目标区间是一个点
    if (s == t) {
        for (const auto& interval : intervals) {
            if (interval.start <= s && interval.end >= s) {
                return 1;
            }
        }
        return -1;  // 如果没有区间覆盖这个点
    }
```

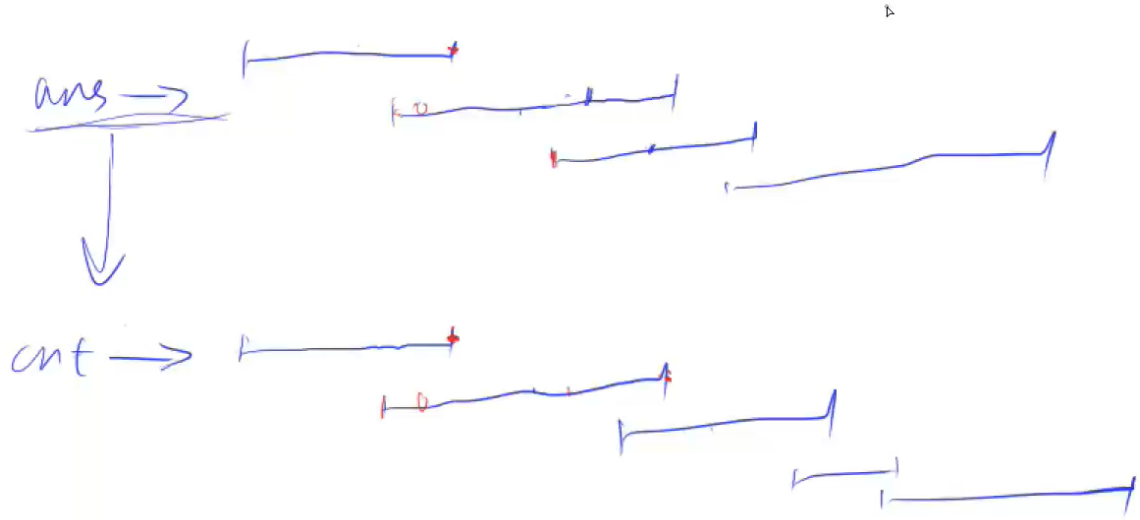
```
    // 按照自定义的比较函数对区间进行排序
    sort(intervals.begin(), intervals.end(), compareIntervals);

    int count = 0;        // 选择的区间数量
    int currentEnd = s;  // 当前覆盖到的最远右端点
    int i = 0;
    int n = intervals.size();

    // 遍历所有区间
    while (currentEnd < t) {
        int farthest = currentEnd;

        // 找到所有能够覆盖 currentEnd 的区间，并选择最远的区间
        while (i < n && intervals[i].start <= currentEnd) {
            farthest = max(farthest, intervals[i].end);
            i++;
        }

        // 如果没有找到更远的区间，说明无法覆盖
        if (farthest == currentEnd) {
            return -1;
        }

        // 更新当前覆盖范围，并增加计数
        currentEnd = farthest;
        count++;

        // 如果当前覆盖到或超过 t，结束循环
        if (currentEnd >= t) {
            break;
        }
    }

    // 返回使用的最少区间数
    return currentEnd >= t ? count : -1;
}

int main() {
    int s, t, N;
    cin >> s >> t;
    cin >> N;

    vector<Interval> intervals(N);
    for (int i = 0; i < N; i++) {
        cin >> intervals[i].start >> intervals[i].end;
    }

    int result = minIntervalsToCover(s, t, intervals);
    cout << result << endl;

    return 0;
}
```

贪心做法



替换法证明

通过替换调整的方式可以把最优解替换为当前的方案




```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n;
struct Range
{
    int l, r;
    bool operator< (const Range & W)const
    {
        return l < W.l;
    }
}range[N];

int main()
{
    int st, ed;
    scanf("%d%d", &st, &ed);
    scanf("%d", &n);
    for (int i = 0; i < n; i ++)
```

```
    {
        int l, r;
        scanf("%d%d", &l, &r);
        range[i] = {l, r};
    }

    sort(range, range + n);

    int res = 0;
    bool success = false;
    for (int i = 0; i < n; i ++)
    {
        int j = i, r = -2e9; // 双指针算法

        // 遍历所有左端点在start的左边的所有区间里面，右端点的最大值是多少
        while(j < n && range[j].l <= st)
        {
            r = max(r, range[j].r);
            j ++;
        }

        if (r < st)
        {
            res = -1;
            break;
        }

        res ++;
        if (r >= ed)
        {
            success = true;
            break;
        }

        st = r;
        i = j - 1;
    }
    if (!success) res = -1;
    printf("%d\n", res);

    return 0;
}
```

Huffman树

合并果子

我一开始的思路是对果子按照重量排序，再从前往后相加，但是这种方式并不一定能保证最优，比如：

按照排序并累加：

1. 先排序得到 `[5, 6, 7, 8]`。
2. 合并 `5` 和 `6`，得到新堆，重量为 `11`，消耗体力 `11`。
3. 合并 `11` 和 `7`，得到新堆，重量为 `18`，消耗体力 `18`。
4. 最后合并 `18` 和 `8`，得到新堆，重量为 `26`，消耗体力 `26`。
5. 总体力消耗为 `11 + 18 + 26 = 55`。

使用贪心策略（最小堆）：

1. 先合并 `5` 和 `6`，得到新堆，重量为 `11`，消耗体力 `11`。
2. 然后合并 `7` 和 `8`，得到新堆，重量为 `15`，消耗体力 `15`。
3. 最后合并 `11` 和 `15`，得到新堆，重量为 `26`，消耗体力 `26`。
4. 总体力消耗为 `11 + 15 + 26 = 52`。

因此，我应该关注的是每次合并重量最小的两堆，这个用数组如何来实现呢?

step1: 当然还是排序，排完序我再找最小的两堆只要从前往后找就行了（当这样需要每次更新完都排一次序）

算了，还是用小根堆做吧：

```
#include <iostream>
#include <queue>
using namespace std;

int main() {
    int n;
    cin >> n;
    priority_queue<int, vector<int>, greater<int>> pq; // 小根堆

    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        pq.push(x);
    }

    int total_cost = 0;

    while (pq.size() > 1) {
        int a = pq.top();
        pq.pop();
        int b = pq.top();
        pq.pop();
        int combined = a + b;
        total_cost += combined;
```

```
        pq.push(combined);
    }

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

y总视频里讲的和我的实现一样，就不放讲解截图了，题目难度可见一斑。

排序不等式

排队打水

我的代码：

```
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 100010;

int q[N];

int main()
{
    int n;
    cin >> n;
    for (int i = 0; i < n; i ++) cin >> q[i];

    sort(q, q + n);

    LL res = 0;
    for (int i = 0; i < n - 1; i ++)
    {
        res += q[i] * (n - i - 1);
    }

    cout << res << endl;
}
```

下面康康y总的做法：

乐，和我的做法一模一样，那此题难度也可见一斑，略了。

绝对值不等式

仓库选址

我自己的思路写的：

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main()
{
    int N;
    cin >> N;
    vector<int> A(N);
    for (int i = 0; i < N; i ++) cin >> A[i];

    sort(A.begin(), A.end());

    int res = 0;
    for (int i = 0; i < N; i ++)
    {
        if (i < N / 2) res += A[N / 2] - A[i];
        else res += A[i] - A[N / 2];
    }

    cout << res << endl;
}
```

看一个加强版，一维变二维：

3167. 星星还是树

题目 | 提交记录 | 讨论 | 题解 | 视频讲解

在二维平面上有 n 个点，第 i 个点的坐标为 (x_i, y_i)。

请你找出一个点，使得该点到这 n 个点的距离之和最小。

该点可以选择在平面中的任意位置，甚至与这 n 个点的位置重合。

输入格式

第一行包含一个整数 n。

接下来 n 行，每行包含两个整数 x_i, y_i，表示其中一个点的位置坐标。

输出格式

输出最小距离和，答案四舍五入取整。

数据范围

$1 \le n \le 100$,
$0 \le x_i, y_i \le 10000$

输入样例：

```
4
0 0
0 10000
10000 10000
10000 0
```

难度：中等

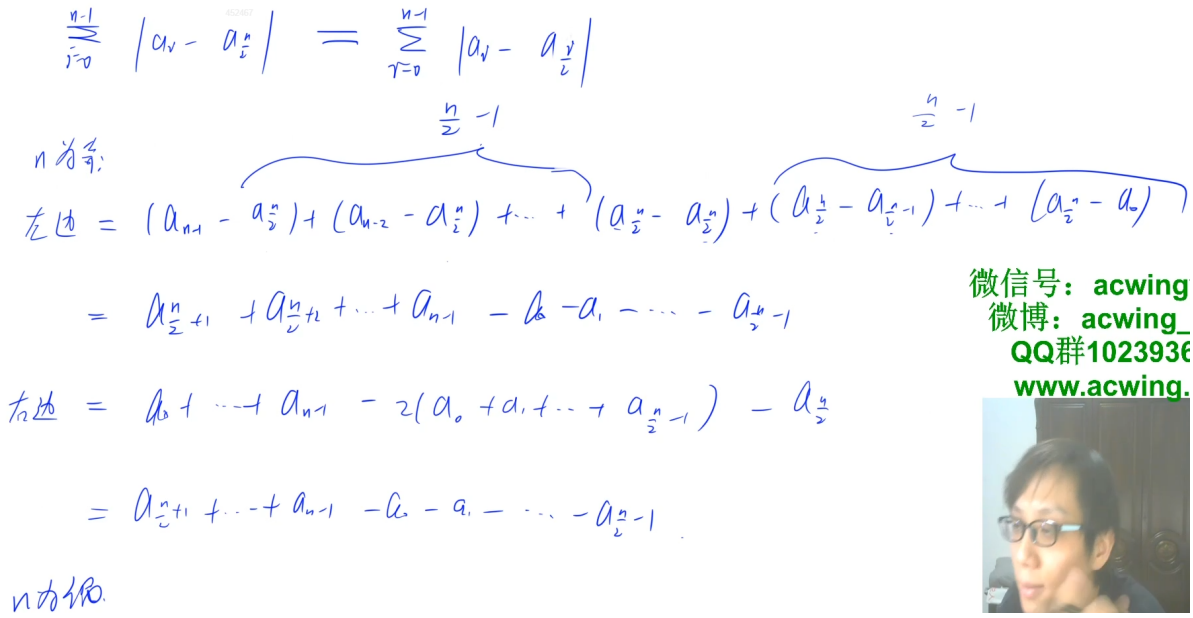
时/空限制：1s / 64MB

总通过数：

总尝试数：

来源：POJ2420

算法标签



贴了个证明，下面展示写法2

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main()
{
    int N;
    cin >> N;
    vector<int> A(N);
    for (int i = 0; i < N; i ++) cin >> A[i];

    sort(A.begin(), A.end());

    int res = 0;
    for (int i = 0; i < N; i ++) res += A[i] - A[i / 2];
    cout << res << endl;
}
```

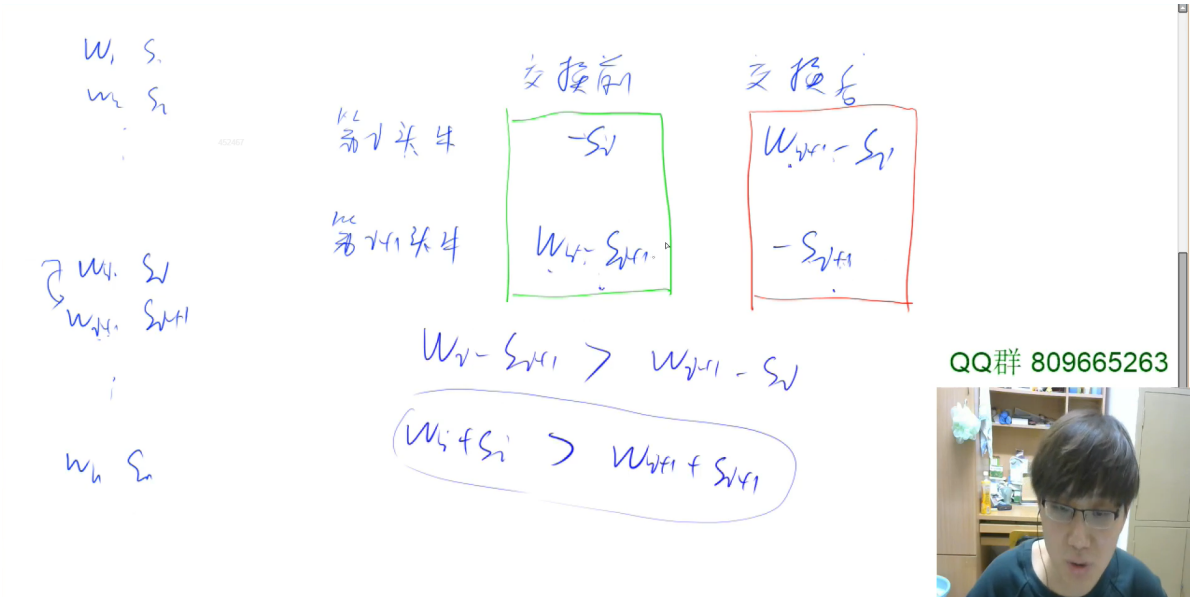
当然也可以用快速选择算法降低一点复杂度

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n;
int a[N];

int main()
{
    cin >> n;
    for (int i = 0; i < n; i ++ ) cin >> a[i];
    nth_element(a, a + n / 2, a + n);

    int res = 0;
    for (int i = 0; i < n; i ++ ) res += abs(a[i] - a[n / 2]);
    cout << res << endl;
    return 0;
}
```


调试代码 提交

推公式

耍杂技的牛




```cpp
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 50010;

struct Cow
{
    int w, s;
    bool operator < (const Cow & a)const{
        return w + s < a.w + a.s;
    }
}cow[N];
```

```cpp
int main()
{
    int n;
    cin >> n;
    for (int i = 0; i < n; i ++)
    {
        cin >> cow[i].w >> cow[i].s;
    }

    sort(cow, cow + n);

    LL res = -0x3f3f3f, tmp = 0;;
    for (int i = 0; i < n; i ++)
    {
        tmp += cow[i].w;
        res = max(res, tmp - cow[i].s - cow[i].w);
    }

    cout << res << endl;

    return 0;
}
```