二 分

1、二分查找

1236. 二分查找

请在一个有序递增数组中（不存在相同元素），采用二分查找，找出值 x 的位置，如果 x 在数组中不存在，请输出 -1 ！

输入

第一行，一个整数 n ，代表数组元素个数（$n \leqslant 10^6$）

第二行，n 个数，代表数组的 n 个递增元素（$1 \leqslant$ 数组元素值 $\leqslant 10^8$）

第三行，一个整数 x ，代表要查找的数（$0 \leqslant x \leqslant 10^8$）

输出

x 在数组中的位置，或者 -1。

输入

10

1 3 5 7 9 11 13 15 17 19

3

输出

2

说明

请尝试采用递归和非递归两种方式来实现二分查找

```cpp
#include <bits/stdc++.h>//1-1236-1 解法一：闭区间 [l, r]  视频教程
using namespace std;
const int N = 1000100; // 数据量 // 常量
int a[N]; // 全局数组，存储有序数据
int main() {
	int n, x;
	scanf("%d", &n);
	for (int i = 1; i <= n; i++) {
		scanf("%d", &a[i]);
	}

	scanf("%d", &x);	// 输入要查找的值

	int l = 1, r = n, mid;		// 二分查找

	while (l <= r)
	{
		mid = (l + r) / 2; // 中间元素的下标

		if (x < a[mid])		r = mid - 1;		// 在左半部分查找
		else if (x > a[mid])	l = mid + 1;		// 在右半部分查找
		else if (x == a[mid])
		{
			printf("%d", mid);		// 找到目标值
			return 0;
		}
	}


    printf("-1");	// 找不到
    return 0;
}
```

```cpp
#include <bits/stdc++.h>//1-1236-2  解法二：左闭右开区间（l,r）视频教程
using namespace std;
const int N = 1000100; // 数据量 // 常量
int a[N];
int main() {
	int n, x;
	scanf("%d", &n);
	for (int i = 1; i <= n; i++) {
		scanf("%d", &a[i]);
	}

	scanf("%d", &x);	// 输入要查找的值

	int l = 1, r = n + 1, mid;

	while (l < r)
	{
		mid = (l + r) >> 1; // 中间元素下表

		if (x < a[mid])		r = mid;
		else if (x > a[mid])	l = mid + 1;		// 左边界跳过 mid
		else if (x == a[mid])
		{
			printf("%d", mid);			// 找到目标
			return 0;
		}
	}

	printf("-1");	// 未找到
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//1-1236-3    二分查找    wintereta 仅供参考
using namespace std;
int a[1000005];
int n,x;
int bs(int x)
{
	int left=1;
	int right=n;
	while(left<=right)
	{
		int mid=(left+right)/2;
		if(a[mid]==x)	return mid;
		if(a[mid]>x)	right=mid-1;
		if(a[mid]<x)	left=mid+1;
	}
	return -1;
}
int main()
{
	cin>>n;
	for(int i=1;i<=n;i++)
	{
		cin>>a[i];
	}
	cin>>x;
	cout<<bs(x);
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//1-1236-4  二分查找    R_S   二分循环 仅供参考
using namespace std;
int a[1000005];// 声明数组
int n,x;
int main()
{
	cin >> n;   // 完成输入过程
	for(int i=0;i<n;i++)
	{
		cin >> a[i];
	}
	cin >> x;

	int left,right,mid;     // 二分查找
	left = 0;
	right = n-1;
	mid = (left+right)/2;


	while(left <= right)    //left <= right   // 什么情况下查找呢?
	{
		if(a[mid]==x)
		{
			cout << mid+1;
			return 0;
		}
		else if(a[mid]>x)       right = mid-1;    // 证明要查找的数在左半边

		else if(a[mid]<x)       left = mid+1;// 证明要查找的数在右半边

		mid = (left+right)/2;          // 更新中心点
	}
	cout << -1;
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//1-1236-5    二分查找    R_S  二分递归    仅供参考
using namespace std;
int a[1000005];// 声明数组
int n,x;
void fun(int l,int r){
	int mid = (l+r)/2;
	if(l>r)      // 递归出口
	{
		cout << -1;
		return;
	}
	if(a[mid] == x)    // 比较
	{
		cout << mid+1;
		return;
	}
	else if(a[mid]>x)
	{
		fun(l,mid-1);
	}
	else if(a[mid]<x)
	{
		fun(mid+1,r);
	}
}
int main(){
	cin >> n;    // 完成输入过程
	for(int i=0;i<n;i++)
	{
		cin >> a[i];
	}
	cin >> x;
	fun(0,n-1); // 递归二分
	return 0;
}
```

1894. 二分查找左侧边界

请在一个有序不递减的数组中（数组中有相等的值），采用二分查找，找到值 x 第 1 次出现的位置，如果不存在 x 请输出 -1。

请注意：本题要求出 q 个 x ，每个 x 在数组中第一次出现的位置。

比如有 6 个数，分别是：1 2 2 2 3 3，那么如果要求 3 个数：3 2 5，在数组中第一次出现的位置，答案是：5 2 -1。

输入

第一行，一个整数 n，代表数组元素个数（$n \leqslant 10^5$）

第二行，n 个整数，用空格隔开，代表数组的 n 个元素（$1 \leqslant$ 数组元素的值 $\leqslant 10^8$）

第三行，一个整数 q，代表有要求出 q 个数首次出现的位置（$q \leqslant 10^5$）

第四行，q 个整数，用空格隔开，代表要找的数（$1 \leqslant$ 要找的数 $\leqslant 10^8$）

输出

输出 1 行，含 q 个整数，按题意输出要找的每个数在数组中首次出现的位置，如果不存在这样的数，请输出 -1。

输入

6

1 2 2 2 3 3

3

3 2 5

输出

5 2 -1

注意

由于本题读入、输出的数据较多，C++ 选手请使用 scanf 和 printf 替代 cin 和 cout 提升读写效率。

```cpp
#include <bits/stdc++.h>//2-1894-1  二分查找左侧边界 视频教程
using namespace std;
const int N = 100100;
int a[N];
int n, q; // n 个数，q 次查询
// 求元素 x 在数组 a 中首次出现的位置
int fun(int x) {
	int l = 1, r = n, mid;
	while (l <= r)
	{
		mid = l + r>> 1;
		if (x < a[mid])			r = mid - 1;
		else if (x > a[mid])		l = mid + 1;
		else if (x == a[mid])		r = mid - 1;

	}

	if ( a[l] == x)   return l;
	else		return -1;
}

int main()
{
	scanf("%d",&n);
	for (int i = 1; i <= n; i++)		scanf("%d",&a[i]);
	scanf("%d",&q);
	int x;
	for (int i = 1; i <= q; i++)
	{
		scanf("%d",&x);
		printf("%d ",fun(x));// 求元素 x 在数组 a 中首次出现的位置
	}
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//2-1894-2    二分查找左侧边界    zhangjin  二分，递归。
using namespace std;
int a[1000001],n,x;
int Find(int low,int high)
{
	if(a[low]>x||a[high]<x||low>high)
		return -1;

	int mid=(low+high)/2;
	if(a[mid]==x)
	{
		if(low==high) // 虽然找到 a[mid]=x，但它不一定是首次出现，还要继续找
			return mid;
		else
			return Find(low,mid);
	}
	else if(a[mid]<x)
		return Find(mid+1,high);
	else
		return Find(low,mid-1);
}
int main()
{
	int i,q;
	scanf("%d",&n);
	for(i=1;i<=n;i++)
		scanf("%d",&a[i]);
	scanf("%d",&q);
	while(q--)
	{
		scanf("%d",&x);
		printf("%d ",Find(1,n));
	}
	return 0;
}
```

1895. 二分查找右侧边界

请在一个有序不递减的数组中（数组中的值有相等的值），采用二分查找，找到最后 1 次出现值 x 的位置，如果不存在 x 请输出 -1。

请注意：本题要求出 q 个 x，每个 x 在数组中最后一次出现的位置。

比如有 6 个数，分别是：1 2 2 2 3 3，那么如果要求 3 个数：3 2 5，在数组中最后一次出现的位置，答案是：6 4 -1。

输入

第一行，一个整数 n，代表数组元素个数（$n \leqslant 10^5$）

第二行，n 个整数，用空格隔开，代表数组的 n 个元素（1 ≤ 数组元素的值 ≤ 10^8）

第三行，一个整数 q ，代表有要求出 q 个数最后一次出现的位置（$q \leqslant 10^5$）

第四行，q 个整数，用空格隔开，代表要找的数（1 ≤ 要找的数 ≤ 10^8）

输出

按题意输出位置或者 -1。

输入

6

1 2 2 2 3 3

3

3 2 5

输出

6 4 -1

注意

由于本题读入、输出的数据较多，C++ 选手请使用 scanf 和 printf 替代 cin 和 cout 提升读写效率。

```cpp
#include <bits/stdc++.h>//3-1895-1 二分查找右侧边界    视频教程
using namespace std;
const int N = 100100;
int a[N];
int n, q;
int fun(int x)
{    // 找右边界
    int l = 1, r = n, mid;// 查找的范围
    while (l <= r)
      {
        mid = (l + r) >> 1;
        if (x < a[mid])             r = mid - 1;
        else if (x > a[mid])        l = mid + 1;
        else if(x == a[mid])        l = mid + 1;// 相等向右看

    }
    if ( a[l-1] == x) return l-1;
      else        return -1;
}

int main()
{
    scanf("%d",&n);
    for (int i = 1; i <= n; i++) {
        scanf("%d",&a[i]);
    }
    scanf("%d",&q);     // q 次查询
    int x;

    for (int i = 1; i <= q; i++) {
        scanf("%d",&x);
        printf("%d ",fun(x));
    }
    return 0;
}
```

```
#include<bits/stdc++.h>//3-1895-2    二分查找右侧边界    zhangjin  二分，非递归
using namespace std;
int a[1000001],n,x;
int main()
{
	int i,q;
	scanf("%d",&n);
	for(i=1;i<=n;i++)		scanf("%d",&a[i]);
	scanf("%d",&q);
	int low,mid,high;
	while(q--)
	{
		scanf("%d",&x);
		low = 1;
		high = n;
		while(low+1<high&&a[low]<=x&&a[high]>=x)  //请特别留意这个地方，意思是 low 和 high 之间还有 1 个数
		{
			mid=(low+high)/2;
			if(a[mid]==x)
				low =mid;
			else if(a[mid]<x)
				low = mid+1;
			else
				high = mid-1;
		}
		if(a[high]==x)			//还剩下 low 和 high 两个数，分别枚举
			printf("%d ",high);
		else if(a[low]==x)
			printf("%d ",low);
		else
			printf("-1 ");
	}
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//3-1895-3  二分查找右侧边界    a623483487
using namespace std;
int a[100005],n,m;
int Search(int a[],int n,int key)
{
	int low = 1;
	int high = n;
	while(low <= high)
	{
		int mid =(high + low) / 2;
		if(key < a[mid])  high = mid - 1;
		else              low = mid + 1;
	}
	if(low <= n + 1 && key == a[low - 1])
	{
		return low - 1;
	}
	else
	{
		return -1;
	}
}
int main()
{
	scanf("%d",&n);
	for(int i = 1;i <= n;i ++)          cin >> a[i];
	scanf("%d",&m);
	for(int i = 1;i <= m;i ++)
	{
		int x;
		scanf("%d",&x);
		cout << Search(a,n,x) << " ";
	}
	return 0;
}
```

1896. 二分查找满足条件的数

请在一个有序不递减的数组中（数组中的值有相等的值），采用二分查找，找到第 1 个大于或等于元素 x 的位置，如果不存在，请输出 -1。

请注意：本题要求出 q 个 x ，每个 x 在数组找到第 1 个大于或等于 x 的元素的位置。

比如有 6 个数，分别是：1 2 2 2 6 6，那么如果要求 3 个数：5 8 2，在数组中找到第 1 个大于或等于他们的位置，答案是：5 -1 2。

输入

第一行，一个整数 n，代表数组元素个数（$n \leqslant 10^5$）

第二行，n 个整数，用空格隔开，代表数组的 n 个元素（$1 \leqslant$ 数组元素的值 $\leqslant 10^8$）

第三行，一个整数 q，代表有要查询 q 个数（$q \leqslant 10^5$）

第四行，q 个整数，用空格隔开，代表查询的数（$1 \leqslant$ 要找的数 $\leqslant 10^8$）

输出

按题意输出位置或者 -1。

输入

6

1 2 2 2 6 6

3

5 8 2

输出

5 -1 2

注意

由于本题读入、输出的数据较多，C++ 选手请使用 scanf 和 printf 替代 cin 和 cout 提升读写效率。

```cpp
#include<bits/stdc++.h>//4-1896    二分查找满足条件的数    a623483487
using namespace std;
int a[100005],n,m;
int Search(int a[],int n,int key)
{
	int low = 1;
	int high = n;
	while(low <= high)
	{
		int mid =(high + low) / 2;
		if(key <= a[mid]) high = mid - 1;
		else              low = mid + 1;
	}
	if(low <= n)
	{
		return low;
	}
	else
	{
		return -1;
	}
}
int main()
{
	scanf("%d",&n);
	for(int i = 1;i <= n;i ++)          scanf("%d",&a[i]);
	scanf("%d",&m);
	for(int i = 1;i <= m;i ++)
	{
		int x;
		scanf("%d",&x);
		cout << Search(a,n,x) << " ";
	}
	return 0;
}
```

2078. 起止位置

有 n 位同学按照年龄从小到大排好队。

王老师想要查询，年龄为 x 的同学，在队伍中首次出现的位置和最后一次出现的位置；如果队伍中不存在年龄为 x 的同学，请输出 -1。

由于人数太多，一个一个数，太慢啦，请你编程求解。

请注意：本题中王老师查询年龄 x 出现的起止位置，不是查询了 1 次，是查询了 q 次。

比如：假设有 6 位同学的年龄为：1 2 2 2 3 3，王老师查询了 4 个年龄，分别是 2 1 3 8 ，那么：

年龄为 2 的同学首次和最后一次出现的位置分别是：2 4；

年龄为 1 的同学首次和最后一次出现的位置分别是：1 1；

年龄为 3 的同学首次和最后一次出现的位置分别是：5 6；

年龄为 8 的同学首次和最后一次出现的位置分别是：-1 -1；

输入

第一行包含整数 n 和 q ，表示队伍中的总人数和询问个数。

第二行包含 n 个整数（整数的值均在 1 \sim 10000 范围内），表示队伍中每个人的年龄。

接下来 q 行，每行包含一个整数 x ，表示一次询问的值。

输出

共 q 行，每行包含两个整数，表示所求年龄在队伍中的起始位置和终止位置。

如果数组中不存在该元素，则返回 -1 -1。

输入

6 3

1 2 2 2 3 3

2

1

8

输出

2 4

1 1

-1 -1

数据范围

$1 \leqslant n \leqslant 100000, 1 \leqslant q \leqslant 10000, 1 \leqslant x \leqslant 10000$。

```cpp
#include<bits/stdc++.h>//5-2078      起止位置    javacn
using namespace std;
const int N = 1e5 + 10;
int a[N];
int n,q,x;
int fun1(int x){                //找左侧边界
      int l = 1,r = n,mid;
      while(l <= r)      {
            mid = l + r >> 1;
            if(x <= a[mid]) r = mid - 1;
            else l = mid + 1;
      }
      if(l>=1&&l<=n&&a[l] == x) return l;
      else return -1;
}
int fun2(int x){                //找右侧边界
      int l = 1,r = n,mid;
      while(l <= r)
      {
            mid = l + r >> 1;
            if(x < a[mid]) r = mid - 1;
            else l = mid + 1;
      }
      if(l-1>=1&&l-1<=n&&a[l-1] == x) return l-1;
      else return -1;
}
int main() {
      scanf("%d%d",&n,&q);     //读入 n 个问题
      for(int i = 1;i <= n;i++)            scanf("%d",&a[i]);
      while(q--)  {
            scanf("%d",&x);
            printf("%d %d\n",fun1(x),fun2(x));
      }
      return 0;
}
```

1898. 同时出现的数

Medusa 同学拿到了 2 组数字，老师请你编程帮他找出，第 2 组数中的哪些数，在第 1 组数中出现了，从小到大输出所有满足条件的数。

比如：

第 1 组数有：8 7 9 8 2 6 3

第 2 组数有：9 6 8 3 3 2 10

那么应该输出：2 3 3 6 8 9

输入

第一行两个整数 n 和 m ，分别代表 2 组数的数量。

第二行 n 个整数。

第三行 m 个整数。

对于 60% 的数据 1 ≤ n,m ≤ 1000，每个数 ≤ 2×10^9。

对于 100% 的数据 1 ≤ n, m ≤ 100000 ，每个数 ≤ 2×10^9。

输出

按照要求输出满足条件的数，数与数之间用空格隔开。

输入

7 7

8 7 9 8 2 6 3

9 6 8 3 3 2 10

输出

2 3 3 6 8 9

/*

有 10^5 个数如果顺序查找，每个数最多找 10^5 次能找到

那么循环总次数：10^{10}

如果二分找，大概找 17 次

总循环次数：17 * 10^6

思路：

将第二组数排序，能够实现从小到大找

将第一组数排序，能够实现二分查找

逐个判断第 2 组数的每个数在第 1 组数中是否出现过

*/

```cpp
#include <bits/stdc++.h>//6-1898-1
using namespace std;
const int N = 100010;
int a[N], b[N];
int n, m;
// 在 a 数组中判断元素 x 是否存在
bool fun(int x) {
	int l = 1, r = n, mid;
	while (l <= r)
	{
		mid = (l + r) >> 1;
		if (x < a[mid])
			r = mid - 1;
		else if (x > a[mid])
			l = mid + 1;
		else
			return true;
	}
	return false;
}

int main()
{
	cin >> n >> m;
	for (int i = 1; i <= n; i++)		cin >> a[i];
	for (int i = 1; i <= m; i++)		cin >> b[i];
	sort(a + 1, a + n + 1);
	sort(b + 1, b + m + 1);
	// 循环 b 数组的每个数，逐个判断在 a 数组中是否存在
	for (int i = 1; i <= m; i++)
	{
		if (fun(b[i]))	cout << b[i] << " ";
	}
	return 0;
}
```

视频教程

/*

第 2 组数中的哪些数，在第 1 组数中出现了

从小到大输出所有满足条件的数

思路：采用一分查找第 2 个数组中哪些数在第 1 个数组中出现了

1. 对 a 数组排序：方柜二分查找
2. 对 b 数组排序：方便结果从小到大输出
3. 循环 b 数组的每个数，在 a 数组中二分查找是否存在

*/

```cpp
#include <bits/stdc++.h>//6-1898-2 解法二：map 求解    视频教程
using namespace std;

/*
第 2 组数中的哪些数，在第 1 组数中出现了
从小到大输出所有满足条件的数
*/
const int N = 100010;
int a[N], b[N];
int n, m;
map<int, int> ma;

int main() {
	cin >> n >> m;
	for (int i = 1; i <= n; i++)
	{
		cin >> a[i];
		ma[a[i]] = 1; // 标记出现过的数
	}
	for (int i = 1; i <= m; i++)
	{
		cin >> b[i];
	}
	sort(b + 1, b + m + 1);
	// 循环 b 数组的每个数，逐个判断在 a 数组中是否存在
	for (int i = 1; i <= m; i++)
	{
		if (ma[b[i]] == 1)
		{
			cout << b[i] << " ";
		}
	}
	return 0;
}
```

1899. 最满意的方案

高考结束了，同学们要开始了紧张的填写志愿的过程，大家希望找一个自己最满意的大学填报方案，请你编程帮忙实现。 现有 m (m ≤ 100000) 所学校，每所学校预计分数线是 a_i ($a_i \leqslant 10^6$) 。有 n(n ≤ 100000) 位学生，估分分别为 b_i ($b_i \leqslant 10^6$)。

根据 n 位学生的估分情况，分别给每位学生推荐一所学校，要求学校的预计分数线和学生的估分相差最小（可高可低，毕竟是估分嘛），这个最小值为不满意度。求所有学生不满意度和的最小值。

输入

第一行读入两个整数 m,n，m 表示学校数， n 表示学生数。

第二行共有 m 个数，表示 m 个学校的预计录取分数。

第三行有 n 个数，表示 n 个学生的估分成绩。

输出

一行，为最小的不满意度之和。（数据保证计算结果≤ 10^9）

输入

4 3

513 598 567 689

500 600 550

输出

32

```cpp
#include <bits/stdc++.h>//7-1899-1    视频教程
using namespace std;
const int N = 100100;
int sc[N], x;
int m,n, s = 0; // s: 最小差值的和
int main() {
	cin >> m >> n;    // m个学校
	for (int i = 1; i <= m; i++)		cin >> sc[i];

	sort(sc + 1, sc + m + 1);    // 对学校分数排序，方便二分

	int l,r,mid;// n个同学
	for (int i = 1; i <= n; i++)
	{
		cin >> x; // x: 同学分数
		if	(x <= sc[1])	s = s + sc[1] - x;  // 特判不需要二分的情况
		else if (x >= sc[m])	s = s + x - sc[m];
		else
		{
			l = 1;
			r = m;   // 找左边界
			while (l <= r)
			{
				mid = (l + r) >> 1;
				if (x <= sc[mid])		r = mid - 1;
				else				l = mid + 1;
			}
			// 看 l 和 l-1 哪个位置的分数差更小
			s = s + min(sc[l] - x, x - sc[l - 1]);
		}
	}
	cout << s;
	return 0;
}
```

```
// 二分，查找最后一个小于等于 bi，和第一个大于等于 bi 的数
#include<bits/stdc++.h>//7-1899-2    最满意的方案    zhangjin
using namespace std;
int a[1000001],b,m,n;
int Find1(int low,int high,int x)
{
	if(low+1==high||low==high)     // 找最后一个小于等于 x 的数
	{
		if(a[high]<=x)
			return a[high];
		else
			return a[low];
	}

	int mid=(low+high)/2;
	if(a[mid]<=x)
		return Find1(mid,high,x);
	else
		return Find1(low,mid-1,x);
}
int Find2(int low,int high,int x)
{
	if(low+1==high||low==high)     // 找第一个大于等于 x 的数
	{
		if(a[low]>=x)
			return a[low];
		else
			return a[high];
	}
	int mid=(low+high)/2;
	if(a[mid]>=x)
		return Find2(low,mid,x);
	else
		return Find2(mid+1,high,x);
}
```

```
int main()
{
	int i,ans=0,t1,t2,MIN;
	scanf("%d%d",&m,&n);
	for(i=1;i<=m;i++) // 学校的预计分数
		scanf("%d",&a[i]);
	sort(a+1,a+1+m);

	for(i=1;i<=n;i++) // 学生的估计分数
	{
		scanf("%d",&b);
		if(a[1]>=b)
			ans += (a[1]-b);
		else if(b>a[m])
			ans += (b-a[m]);
		else
		{
			// 来到这里，a 数组里面肯定存在一个数是小于等于 b，又肯定存在一个数是大于等于 b 的
			t1 = Find1(1,m,b); // 最后一个小于等于 b 的数
			t2 = Find2(1,m,b); // 第一个大于等于 b 的数

			MIN = min(b-t1,t2-b);
			ans += MIN;
		}
	}

	printf("%d",ans);

	return 0;
}
```

1542. 小 X 算排名

小 X 很关心自己在学校的表现。

班主任手上有一本“个人得分记录本”，如果一位同学表现好就会加分，表现差则会扣分。学期结束，每位同学都得知了自己的个人得分。小 X 想知道其他同学情况如何，但由于排名不公布，他只好一个个去问班里的其他同学。

现在，小 X 手上有班里共 N 位同学的个人得分，他想知道每位同学的排名 （得分相同则排名相同，见样例），可并不知道该如何计算，希望你帮帮他。

输入

第一行包含一个整数 N 。

接下来 N 行，第 i 行包含一个整数 A_i，表示第 i 位同学的得分。

输出

N 行，第 i 行包含一个整数，表示第 i 位同学的排名。

输入

5

95

100

99

99

96

输出

5

1

2

2

4

数据范围

对于 30% 的数据，N ≤ 10。

对于 60% 的数据，N ≤ 1000。

对于 100% 的数据，1 ≤ N ≤ 100000，0 ≤ A_i ≤ 100000。

注意

由于本题读入、输出的数据较多，C++ 选手请使用 scanf 和 printf 替代 cin 和 cout 提升读写效率。

```cpp
// 二分查找
// 对分数排序，再二分查找每个分数的排名。

#include <bits/stdc++.h>//8-1542-1      小 X 算排名    javacn
using namespace std;
const int N = 1e5 + 10;
int a[N],b[N];
int n;

bool cmp(int x,int y)
{
	return x > y;
}

int main()
{
	scanf("%d",&n);
	for(int i = 1;i <= n;i++)
	{
		scanf("%d",&a[i]);
		b[i] = a[i];
	}

	sort(b+1,b+n+1,cmp);

	// 找 a 数组每个数第一次出现的位置
	for(int i = 1;i <= n;i++)
	{
		int p = lower_bound(b+1,b+n+1,a[i],cmp) - b;// 获取下标
		printf("%d\n",p);
	}

	return 0;
}
```

// 统计 >= 每个分数的人数，从而统计每个分数的排名。

由于本题中每个分数的值都≤ 100000，因此可以先统计每个分数出现的次数，从而统计出 >= 每个分数的人数，从而计算排名。

```
#include <bits/stdc++.h>//8-1542-2      小 X 算排名    javacn
using namespace std;
95  96  97  98  99  100
每个分数的人数：
 1    1    0     0   2      1
>= 每个分数的人数：
 5    4    3      3    3    1
96 分的排名 =  ≥ 97 分的总人数 + 1
const int N = 1e5 + 10;
int a[N],b[N];
int n;
int main()
{
	scanf("%d",&n);
	for(int i = 1;i <= n;i++)
	{
		scanf("%d",&a[i]);
		b[a[i]]++;// 求每个分数的人数
	}
	// 求 >= 每个分数的总人数，也就是从右向左求前缀和
	for(int i = 100000;i >= 1;i--)
	{
		b[i] = b[i+1] + b[i];//>=i 分的人数 = i 分的人数 + ≥ i+1 分的人数
	}

	// 求每个分数的排名
	for(int i = 1;i <= n;i++)
	{
		printf("%d\n",b[a[i]+1] + 1);
	}
	return 0;
}
```

```cpp
//map 求解：
#include <bits/stdc++.h>//8-1542-3      小 X 算排名     javacn
using namespace std;
// 存放分数（key）以及分数出现的次数（value）
map<int,int,greater<int> > m;
int n,i,c = 0,a[100100],t;
int main()
{
	cin>>n;
	for(i = 0;i < n;i++)
	{
		cin>>a[i];
		// 判断该分数在 map 中是否存在
		if(m.count(a[i]) == 0)
		{
			m[a[i]] = 1;    // 将分数存入 map，记录值为 1（分数出现的次数）
		}
		else
		{
			m[a[i]]++;
		}
	}
	map<int,int>::iterator it;     // 修正 map 的 value 为排名
	for(it = m.begin();it != m.end();it++)
	{
		t = it->second;
		m[it->first] = c + 1;// 修正排名
		c = c + t;// 统计每个分数之前有几个人
	}
	for(i = 0;i < n;i++)     // 输出每个分数的排名
	{
		cout<<m[a[i]]<<endl;
	}
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//8-1542-4      小 X 算排名    w2016010182
using namespace std;
int n;
map<int,int> m;
int main()
{
	scanf("%d",&n);
	int a[n+5],b[n+5];
	for(int i=1;i<=n;i++)
	{
		scanf("%d",&a[i]);
		b[i]=a[i];
	}
	sort(b+1,b+n+1);
	for(int i=1;i<=n;i++)
	{
		m[b[i]]=n-i+1;
	}
	for(int i=1;i<=n;i++)
	{
		printf("%d\n",m[a[i]]);
	}
	return 0;
}
```

1893. 最长上升子序列 LIS（2）

给定一个长度为 N 的数列，求数值严格单调递增的子序列的长度最长是多少。

输入

第一行包含整数 N 。

第二行包含 N 个整数，表示完整序列。

$1 \leqslant N \leqslant 100000$，$-10^9 \leqslant$ 数列中的数 $\leqslant 10^9$

输出

输出一个整数，表示最大长度。

输入

6

1 3 2 8 5 6

输出

4

将原来的 dp 数组的存储以每个数结尾的 LIS 序列的长度，修改为存储上升子序列长度为 i 的上升子序列的最小末尾数值。

原理：LIS 长度如果已经确定，那么如果这种长度的子序列的结尾元素越小，后面可能续的元素会更多！

```cpp
#include <bits/stdc++.h>//9-1893  最长上升子序列LIS（2）    javacn
using namespace std;
int a[100100],dp[100100];//dp:长度为i的LIS的最后一位最小值是多少
int i,n,l,r,mid;
int main()
{
	scanf("%d",&n);			//读入
	for(i = 1; i <= n; i++)			scanf("%d",&a[i]);
	dp[1] = a[1];			//边界
	int len = 1;//LIS的长度

	for(i = 2; i <= n; i++)		//从第2个数开始求解
	{		//如果a[i]比dp最后一位大,a[i]直接续上去，增加LIS的长度
		if(a[i] > dp[len])
		{
			len++;
			dp[len] = a[i];
		}
		else
		{
	//二分查找到dp数组中第1个>=a[i]的元素下标，替换（dp数组一定是递增的）
			l = 1;
			r = len;
			while(l <= r)
			{
				mid = (l + r) / 2;
				if(a[i] <= dp[mid]) r = mid - 1;
				else l = mid + 1;
			}
			dp[l] = a[i];//替换
		}
	}
	printf("%d",len);
	return 0;
}
```

1821. 最长公共子序列 (LCS)(1)

给出 1 ~ n 的两个排列 P_1 和 P_2 ，求它们的最长公共子序列。

输入

第一行是一个数 n ；（ n 是 51000 之间的整数）

接下来两行，每行为 n 个数，为自然数 1n 的一个排列（1n 的排列每行的数据都是 1n 之间的数，但顺序可能不同，比如 1 ~ 5 的排列可以是：1 2 3 4 5 ，也可以是 2 5 4 3 1 ）。

输出

一个整数，即最长公共子序列的长度。

输入

5

3 2 1 4 5

1 2 3 4 5

输出

3

我们可以用 dp[i][j] 来表示第一个串的前 i 位，第二个串的前 j 位的 LCS 的长度，那么递推出状态转移方程：

如果当前的 a [i] 和 b[j] 相同（即是有新的公共元素） 这说明该元素一定位于公共子序列中。因此，现在只需要找：a 数组 1~i-1 和 b 数组 1~j-1 的最长公共子序列：

```
dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
```

如果不相同，说明最后一个元素肯定不是公共子序列中的元素，那么考虑找 a 数组 1~i-1 和 b 数组 1~j 的 LCS，或者找：a 数组的 1~i 和 b 数组的 1~j-1 的 LCS，那么，状态转义方程如下：

```
dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
```

```cpp
#include <bits/stdc++.h>//10-1821    最长公共子序列(LCS)(1)    javacn
using namespace std;

/*
a[i]==b[j], 方程：dp[i-1][j-1]+1
a[i]!=b[j], 方程：max(dp[i][j-1],dp[i-1][j])
*/
const int N = 1010;// 常量, 表示数组大小
int a[N],b[N],dp[N][N];
int n,i,j;
int main()
{
	cin>>n;
	for(i = 1;i <= n;i++) cin>>a[i];
	for(i = 1;i <= n;i++) cin>>b[i];

	// 递推
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= n;j++)
		{
			if(a[i] == b[j]) dp[i][j] = dp[i-1][j-1] + 1;
			else dp[i][j] = max(dp[i-1][j],dp[i][j-1]);
		}
	}

	cout<<dp[n][n];
	return 0;
}
```

1822. 最长公共子序列 (LCS)(2)

给出 1~n 的两个排列 P_1 和 P_2，求它们的最长公共子序列。

和最长公共子序列 (LCS)(1) 问题不同的是，本题的 n 在 5100000 之间。

输入

第一行是一个数 n ；（n 是 5100000 之间的整数）

接下来两行，每行为 n 个数，为自然数 1n 的一个排列（1n 的排列每行的数据都是 1n 之间的数，但顺序可能不同，比如 1 ~ 5 的排列可以是：1 2 3 4 5，也可以是 2 5 4 3 1）。

输出

一个整数，即最长公共子序列的长度。

输入

5

3 2 1 4 5

1 2 3 4 5

输出

3

数据范围

对于 50% 的数据，n ≤ 1000；

对于 100% 的数据，n ≤ 100000。

1、因为两个序列都是 1~n 的全排列，那么两个序列元素互异且相同，也就是说只是位置不同；

2、通过 c 数组将 b 序列的数字在 a 序列中的位置求出；

3、如果 b 序列每个元素在 a 序列中的位置递增，说明 b 中的这个数在 a 中的这个数整体位置偏后，可以考虑纳入 LCS； 4、从而就可以转变成求用来记录新的位置的 c 数组中的 LIS。

```
#include <bits/stdc++.h>//11-1822      最长公共子序列 (LCS)(2)     javacn
using namespace std;
const int N = 100100;
int a[N],b[N],c[N],dp[N];
int n,i;
```

```
int main()
{
	cin>>n;
	for(i = 1; i <= n; i++)
	{
		scanf("%d",&a[i]);
		c[a[i]] = i;// 求出 a 数组的每个数的位置
	}
	for(i = 1; i <= n; i++) scanf("%d",&b[i]);
	dp[1] = c[b[1]];// 边界    // 求 b 数组的每个数在 a 数组的位置 (c[b[i]]) 的 LIS
	int len = 1;
	int l,r,mid;
	for(i = 2; i <= n; i++)        // 从第 2 个数开始讨论
	{
		if(c[b[i]] > dp[len])          // 增加 LIS 的长度
		{
			len++;
			dp[len] = c[b[i]];
		}
		else
		{
			l = 1;
			r = len;
			while(l <= r)
			{
				mid = (l + r) / 2;
				if(c[b[i]] <= dp[mid]) r = mid - 1;
				else l = mid + 1;
			}
			dp[l] = c[b[i]];
		}
	}
	cout<<len;
	return 0;
}
```

1902. 最少的修改次数

现有整数 $A_1, A_2, ... A_n$，修改最少的数字为实数（整数或者小数），使得数列严格单调递增。

输入

第一行，一个整数 n 。（$n \leqslant 10^5$）

第二行，n 个整数 A_i 。（$A_i \leqslant 10^9$）

输出

1 个整数，表示最少修改的数字的数量。

输入

3

1 3 2

输出

1

思路：求最少的修改次数， 那就是要找出需要修改的数字，而且越少也好。逆向思维， 找最长的上升子序列（LIS）。然后，用总个数减去上升的，即需要修改的数字。

```
#include<bits/stdc++.h>//12-1902    最少的修改次数  dragoncatter
using namespace std;
const int N = 1e5 + 10 ;
int n, cnt = 1;
int a[N], dp[N];
int main()
{
	cin >> n;
	for (int i = 1; i <= n; i++)
	{
		cin >> a[i];
	}
	dp[1] = a[1];
	for (int i = 2; i <= n; i++)
	{
		if (a[i] > dp[cnt])
		{
			dp[++cnt] = a[i];
		}
		else
		{
			int k = lower_bound(dp+1, dp + cnt+1, a[i], less<int>()) - dp;
			dp[k] = a[i];
		}
	}
	cout << n - cnt;
	return 0;
}
```

2、二分答案

1908. 伐木工

伐木工人米尔科需要砍倒 M 米长的木材。这是一个对米尔科来说很容易的工作，因为他有一个漂亮的新伐木机，可以像野火一样砍倒森林。不过，米尔科只被允许砍倒单行树木。

米尔科的伐木机工作过程如下：米尔科设置一个高度参数 H（米），伐木机升起一个巨大的锯片到高度 H，并锯掉所有的树比 H 高的部分（当然，树木不高于 H 米的部分保持不变）。米尔科就得到树木被锯下的部分。

例如，如果一行树的高度分别为 20，15，10 和 17 ，米尔科把锯片升到 15 米的高度，切割后树木剩下的高度将是 15，15，10 和 15，而米尔科将从第 1 棵树得到 5 米，从第 4 棵树得到 2 米，共得到 7 米木材。

米尔科非常关注生态保护，所以他不会砍掉过多的木材。这正是他为什么尽可能高地设定伐木机锯片的原因。帮助米尔科找到伐木机锯片的最大的整数高度 H ，使得他能得到木材至少为 M 米。换句话说，如果再升高 1 米，则他将得不到 M 米木材。

输入：第 1 行：2 个整数 N 和 M，N 表示树木的数量（$1 \leqslant N \leqslant 10^6$），M 表示需要的木材总长度（$1 \leqslant M \leqslant 2 \times 10^9$）。

第 2 行：N 个整数表示每棵树的高度，值均不超过 10^9。所有木材长度之和大于 M ，因此必有解。

输出：1 个整数，表示砍树的最高高度。

输入

5 20

4 42 40 26 46

输出

36

```
/*
本题：读入的数据在 int 范围，但求和可能会超过 int
因此定义 long long。
思考：找左边界，还是右边界；
本题：在高度为 mid 的情况下，如果能得到 >=m 米的木材，升高高度
因此本题求右边界； */
```

```cpp
#include <bits/stdc++.h>//1-1908  伐木工  javacn  视频教程
using namespace std;
const int N = 1000010;
long long a[N];
long long n,m,l = 1,r,mid;
bool check(long long x)// 检验锯片高度为 x 的情况下，能够得到的木材量是否 >=m
{
	long long s = 0;// 能够锯到的木材量
	for(int i = 1;i <= n;i++)
	{
		if(x < a[i]) s = s + a[i] - x;		// 如果 x< 树的高度，才能锯到木材
		if(s >= m) return true; // 如果得到 >=m 米的木材，就可以停止返回
	}
	return false;
}
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)		// 读入木材的高度
	{
		cin>>a[i];
		r = max(a[i],r);//r 的值应该是：所有树的最高高度
	}

	while(l <= r)		// 二分答案
	{
		mid = l + r >> 1;
		// 如果高度为 mid 的情况下，能够得到 >=m 的木材
		if(check(mid)) l = mid + 1;
		else r = mid - 1;
	}

	cout<<l - 1;// 右边界
	return 0;
}
```

1909. 跳石头

一年一度的“跳石头”比赛又要开始了！

这项比赛将在一条笔直的河道中进行，河道中分布着一些巨大岩石。组委会已经选择好了两块岩石作为比赛起点和终点。在起点和终点之间，有 N 块岩石（不含起点和终点的岩石）。在比赛过程中，选手们将从起点出发，每一步跳向相邻的岩石，直至到达终点。

为了提高比赛难度，组委会计划移走一些岩石，使得选手们在比赛过程中的最短跳跃距离尽可能长。由于预算限制，组委会至多从起点和终点之间移走 M 块岩石（不能移走起点和终点的岩石）。

输入

第一行包含三个整数 L,N,M ，分别表示起点到终点的距离，起点和终点之间的岩石数，以及组委会至多移走的岩石数。保证 L ≥ 1 且 N ≥ M ≥ 0 。

接下来 N 行，每行一个整数，第 i 行的整数 Di(0 < Di < L)， 表示第 i 块岩石与起点的距离。这些岩石按与起点距离从小到大的顺序给出，且不会有两个岩石出现在同一个位置。

输出

一个整数，即最短跳跃距离的最大值。

输入

25 5 2

2

11

14

17

21

输出

4

```
/*
在起点和终点之间，有 N 块岩石
最短跳跃距离越大，搬走的石块就越多！
考虑极端情况，如果跳跃距离为 L，都要搬走！
如果跳跃距离为 1，都不用搬！
1. 二分对象：跳跃距离，在 [1, l] 之间
2. 最短跳跃距离为 mid 的情况下，如何检验？
   检验搬走石头的数量：c
   如果 c <= m，说明最短跳跃距离太小，要放大
   否则，要缩小！
3. 求的是右边界
*/
```

说明

输入输出样例 1 说明：将与起点距离为 2 和 14 的两个岩石移走后，最短的跳跃距离为 4（从与起点距离 17 的岩石跳到距离 21 的岩石，或者从距离 21 的岩石跳到终点）。

另：对于 20% 的数据， 0 ≤ M ≤ N ≤ 10 。

对于 50% 的数据，0 ≤ M ≤ N ≤ 100 。

对于 100% 的数据， 0 ≤ M ≤ N ≤ 50,000, 1 ≤ L ≤ 1,000,000,000 。

```cpp
#include <bits/stdc++.h>//2-1909-1    视频教程
using namespace std;
const int N = 50100;
int a[N];
int n, m, l; // n 个石头，搬走 m 个，河道长 l
// 求：最短跳跃距离为 mid 的情况下，搬走的石块数量是否 <=m
bool check(int mid) {
	int c = 0; // 搬走的数量
	int p = 0; // 人的位置

	for (int i = 1; i <= n; i++)	// 讨论所有的石块
	{
		// 跳跃距离比 mid 小，不符合要求，移走石块，人不动
		if (a[i] - p < mid)	c++;
		else		p = a[i]; // 人跳过去

	}
	if (l - p < mid)		c++;	// 判断最后一次跳跃
	return c <= m;
}
int main()
{
	cin >> l >> n >> m;	// 读入石头
	for (int i = 1; i <= n; i++)		cin >> a[i];
	int left = 1, right = l,mid ;  // 二分
	 while (left <= right)
	{
		mid = (left + right) >> 1;
		// 如果：最短跳跃距离为 mid 的情况下，搬走的石块数量 <=m，放大距离
		if (check(mid)) left = mid + 1;
		else		right = mid - 1;
	}
	cout << left - 1; // 右边界
	return 0;
}
```

```cpp
#include <iostream>//2-1909-2  跳石头  kevinh
using namespace std;
int l,n,m,a[505000],i;
bool check(int mid)
{
	int p=0,c=0;
	for(int i=0;i<n;i++)
	{
		if(a[i]-p<mid)	c++;
		else		p = a[i];
	}
	if(l-p<mid) c++;
	return c<=m;
}
int main()
{
	cin>>l>>n>>m;
	for(i=0;i<n;i++)		cin>>a[i];

	int left=0,right=l,mid;
	while(left<=right)
	{
		mid = (left+right)/2;
		if(check(mid))
		{
			left = mid+1;
		}
		else
		{
			right = mid-1;
		}
	}
	cout<<left-1<<endl;
	return 0;
}
```

1561. 买木头

有 n 个木材供应商，每个供货商有长度相同一定数量的木头。长木头可以锯短，但短木头不能接长。有一个客人要求 m 根长度相同的木头。要求计算出，此时供货商提供的木头满足客人要求的最长的长度是多少。

例如 n=2,m=30，两个供货商的木头为

12,10 第 1 个供货商的木头长度为 12 ，共有 10 根；

5,10 第 2 个供货商的木头长度为 5 ，共有 10 根。

计算的结果为 5 ，即长度为 12 的木头一根可锯出两根长度为 5 的木头，多余的无用，长度为 5 的木头不动，此时可得到 30 根长度为 5 的木头。

输入

整数 n,m,L_1,S_1 （$1 \leqslant n \leqslant 10000, 1 \leqslant m \leqslant 1000000, 1 \leqslant L_1 \leqslant 10000, 1 \leqslant S_1 \leqslant 100$）

其中 L_1 是第一个供货商木头的长，S_1 是第一个供货商木头数量。其他供货商木头的长度和数量 L_i 和 S_i（$i \geqslant 2$），由下面的公式给出：

$L_i=((L_{i-1} \times 37011+10193) \bmod 10000)+1$

$S_i=((S_{i-1} \times 73011+24793) \bmod 100)+1$

输出

一个整数，即满足要求的 m 根长度相同的木头的最大长度。

本题测试数据保证一定有解。

输入

10 10000 8 20

输出

201

#include <bits/stdc++.h>//3-1561-1 买木头 javacn 二维数组求解：

```
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;
    a[1][1] = l_1;     // 第一个商家木头长度和数量
    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];
		}
	}
    // 输出每个商家的木头的长度和数量
//  for(i = 1;i <= n;i++){
//      cout<<a[i][1]<<"   "<<a[i][2]<<endl;
//  }

    // 计算在每个长度下各个商家能供多少根木头
	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++)		// 循环每个供应商
		{
			c += a[j][1] / i * a[j][2];
		}

		if(c >= m)		// 如果根数够
		{
			r = i;
			break;
		}
	}
	cout<<r<<endl;
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//3-1561-2    买木头    javacn          二分求解：
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;
		if(check(mid)) l = mid + 1;// 如果长度为 mid 能切出 >=m 个木头 // 放长
		else r = mid - 1;
	}

	cout<<l-1;
	return 0;
}
```

1910. 愤怒的奶牛

Farmer John 建造了一个有 N(2 ≤ N ≤ 100000) 个隔间的牛棚，这些隔间分布在一条直线上，坐标是 $x_1,...,x_N$ (0 ≤ x_i ≤ 1000000000)。

他的 C(2 ≤ C ≤ N) 头牛不满于隔间的位置分布，它们为牛棚里其他的牛的存在而愤怒。为了防止牛之间的互相打斗，Farmer John 想把这些牛安置在指定的隔间，所有牛中相邻两头的最近距离越大越好。那么，这个最大的最近距离是多少呢？

输入

第 1 行：两个用空格隔开的数字 N 和 C。

第 2 ~ N+1 行：每行一个整数，表示每个隔间的坐标。

输出

输出只有一行，即相邻两头牛最大的最近距离。

输入

5 3

1

2

8

4

9

输出

3

二分两头牛最大距离的最小值，如果两头牛的最大距离为 mid，能够容纳 >=c 头牛，说明检验成功，将答案放大一点再试试。否则，将答案缩小。

```cpp
#include<bits/stdc++.h>//4-1910-1  愤怒的奶牛    javacn
using namespace std;
const int N = 1e5 + 10;
int a[N],l,r,mid,n,c;
bool check(int mid) //测试距离为mid，可以放几头牛
{
	int tot = 1;//默认能放1头
	int last = 1;//第1个牛栏肯定有牛   //last:表示上1个牛栏
	for(int i = 2; i <= n; i++)
	{
		if(a[i] - a[last] >= mid)
		{
			tot++;//牛栏数自增
			last = i;
		}
	}
	if(tot >= c) return true;
	else return false;
}
int main()
{
	cin>>n>>c;  //n个牛栏，c头要分隔的奶牛
	for(int i = 1; i <= n; i++)		cin>>a[i];
	sort(a+1,a+1+n);  //排序
	l=1,r=a[n]-a[1];  //在[l,r]之间查找最大的最近距离
	while(l <= r)		//找右边界
	{
		mid = (l + r) >> 1;
		if(check(mid)) l = mid + 1;
		else r = mid - 1;
	}

	cout<<l-1;
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//4-1910-2    hasome
using namespace std;
int n,c,a[100010],l,r=INT_MIN,mid;
bool check(int x){
	int sum=c-1;
	int j=2;
	int i=1;
	while(sum>0)
	{
		while(a[j]-a[i]<x)
		{	j++;
			if(j==n+1) return false;
		}
		i=j;
		sum--;
	}
	return true;
}
int main(){
	cin>>n>>c;
	for(int i=1;i<=n;i++)
	{	cin>>a[i];
		r=max(a[i],r);
	}
	sort(a+1,a+n+1);
	l=1;
	r=r-1;
	while(l<=r)
	{	mid=l+r>>1;
		if(check(mid))	l=mid+1;
		else		r=mid-1;
	}
	cout<<l-1<<endl;
	return 0;
}
```

1912. 最小的空旷指数

A 市和 B 市之间有一条长长的高速公路，这条公路的某些地方设有路标，但是大家都感觉路标设得太少了，相邻两个路标之间往往隔着相当长的一段距离。为了便于研究这个问题，我们把公路上相邻路标的最大距离定义为该公路的“空旷指数”。

现在政府决定在公路上增设一些路标，使得公路的“空旷指数”最小。他们请求你设计一个程序计算能达到的最小值是多少。请注意，公路的起点和终点保证已设有路标，公路的长度为整数，并且原有路标和新设路标都必须距起点整数个单位距离。

输入

第 1 行包括三个数 L、N、K，分别表示公路的长度，原有路标的数量，以及最多可增设的路标数量。

第 2 行包括 N 个整数（这 N 个数并未排序），分别表示原有的 N 个路标的位置。路标的位置用距起点的距离表示，且一定位于区间 [0,L] 内。

输出

输出 1 行，包含一个整数，表示增设路标后能达到的最小“空旷指数”值。

输入

101 2 1

0 101

输出

51

说明

公路原来只在起点和终点处有两个路标，现在允许新增一个路标，应该把新路标设在距起点 50 或 51 个单位距离处，这样能达到最小的空旷指数 51。

数据范围：

50% 的数据中，2 ≤ N ≤ 100，0 ≤ K ≤ 100。

100% 的数据中，2 ≤ N ≤ 100000，0 ≤ K ≤ 100000。

100% 的数据中，0 < L ≤ 10000000。

二分答案，检验如果最小的空旷指数为 x 的情况下，路标数量是否 <= 规定的 k 。

```cpp
#include<bits/stdc++.h>//5-1912    最小的空旷指数    javacn
using namespace std;
int l,n,k;
int a[100010];
bool check(int x) // 检验如果最小的空旷指数为 x 的情况下 // 路标数量是否 <= 规定的 k
{
	int cnt=0;
	for(int i=2; i<=n; i++)
	{
		int d=a[i]-a[i-1];
		if(d>x)
		{
			cnt+=d/x;
			if(d % x == 0) cnt--;
		}
	}
	return cnt<=k;
}
int main()
{
	scanf("%d%d%d",&l,&n,&k);
	for(int i=1; i<=n; i++)		scanf("%d",&a[i]);
	sort(a+1,a+1+n);  // 读入无序，排序
	// 二分答案，如果 left、right 定义在 main 的外面
	// 记得不能用 left、right 这样的函数名
	int left=1,right=l;
	while(left<=right)
	{
		int mid=(left+right)>>1;
		if(check(mid)) right=mid-1;
		else left=mid+1;
	}

	cout<<left; // 输出左边界
}
```

1916. 防御迷阵

一队士兵来到了敌军城外，准备进攻敌城。敌人在城外布置一个防御迷阵，要进入城池首先必须通过城池外的防御迷阵。

迷阵由 n × m 个相同的小房间组成，每个房间与相邻四个房间之间有门可通行。而第 1 行的 m 个房间有 m 扇向外打开的门，是迷阵的入口。除了第 1 行和第 n 行的房间外，每个房间都安装了激光杀伤装置，将会对进入房间的人造成一定的伤害。第 i 行第 j 列造成的伤害值为 a_{ij}。 （第 1 行和第 n 行的 a_{ij} 值全部为 0）。

现在士兵打算以最小伤害代价通过迷阵，显然，他们可以选择任意多的人从任意的门进入，但必须到达第 n 行的房间。一个士兵受到的伤害值为他在经过的路径上所有房间的伤害值中的最大值。现在，士兵们掌握了迷阵的情况，他们需要提前知道怎么安排士兵的行进路线可以使得伤害值最小。

输入

第一行有两个整数 n,m 表示迷阵的大小。

接下来 n 行，每行 m 个数，第 i 行第 j 列的数表示 a_{ij} 。

数据范围：$2 \leqslant n,m \leqslant 1000$，$1 \leqslant a_{ij} \leqslant 1000$。

输出

输出一个数，表示最小伤害代价。

输入

4 2

0 0

3 5

2 4

0 0

输出

3

```
/*
第 1 行是入口，第 n 行是出口
伤害值：经过所有点的伤害值的最大
求：最小的伤害代价（求最大值的最小的问题）
二分答案：
1. 答案具有单调性——可以二分
2. 思考二分的范围：答案在什么范围内
3. 思考左边界、右边界 */
```

二分可能的伤害值，如果伤害值为 mid 的情况下，能够通过迷阵，则降低伤害值继续尝试。

判断伤害值为 mid 的情况下，能否通过迷阵的方法是：广搜，如果某个点的伤害值值 <=mid，则认为该点可行。

```cpp
#include <bits/stdc++.h>//6-1916   视频教程
using namespace std;
int a[1010][1010];
int n, m;
int l = INT_MAX, r = INT_MIN, mid;
int q[1000100][3];// 广搜变量准备
int fx[5] = {0, 0, 1, 0, -1};
int fy[5] = {0, 1, 0, -1, 0};
bool f[1010][1010]; // 标记是否走过
// 检验方法：采用广搜，从 1,1 出发，走 a[i][j]<=mid 的点，看能否走到第 n 行
bool check(int v) {// check(mid)：判断伤害值为 mid 的情况下，能否通过迷阵
      memset(f, false, sizeof(f));    // 重设标记数组的值，因为要广搜多次
      int h = 1, t = 1; // 指针
      q[1][1] = 1;    // 交代起始点
      q[1][2] = 1;
      f[1][1] = true; // 标记走过
      int tx, ty; // 要去的点
      while (h <= t) {
            for (int i = 1; i <= 4; i++)        // 尝试四方向
            {
                  tx = q[h][1] + fx[i];
                  ty = q[h][2] + fy[i];
   if (tx >= 1 && tx <= n && ty >= 1 && ty <= m && !f[tx][ty] && a[tx][ty] <= v)
                  {      // 如果该点可行    //!表示取反
                        t++;
                        q[t][1] = tx;
                        q[t][2] = ty;
                        f[tx][ty] = true; // 走过标记
                        if (tx == n) return true;     // 判断是否出迷阵
                  }
            }
            h++;
      }
      return false;
}
```

```cpp
int main() {
	cin >> n >> m;
	for (int i = 1; i <= n; i++)
	{
		for (int j = 1; j <= m; j++)
		{
			cin >> a[i][j];

			if (i != 1 && i != n)  // 如果不是第 1 行和最后一行，才是伤害值
			{
				l = min(l, a[i][j]);
				r = max(r, a[i][j]);
			}
		}
	}
	// 二分
	while (l <= r)
	{
		mid = (l + r) >> 1;
		// 如果伤害值为 mid，能通过，减少伤害值，再尝试
		if (check(mid)) r = mid - 1;
		else l = mid + 1;
	}
	cout << l; // 左边界
	return 0;
}
```