算法：二分答案 讲义

一、 引言

在基础二分查找中，我们的搜索域局限于已有的物理数组（如在一个有序序列中查找特定元素）。而二分答案是一种高阶的算法思维，它将二分的搜索域拓展到了抽象的**解空间** 中，用于锁定满足特定条件的最优解。

1. 经典最优化问题：砍树问题 (EKO)

问题描述：
给定 N 棵树的高度集合，需要收集至少 M 米的木材。设定一个全局锯片高度阈值 H，凡是高度大于 H 的树，其超出 H 的部分将被截取。
优化目标：在保证收集木材总量 $\geq M$ 的前提下，求锯片高度 H 的**最大可能值。**

2. 最优化问题

日常算法中常见的“最大化最小值”、“最小化最大值”等最优化问题，直接正向求解往往极其困难，甚至缺乏解析解。
二分答案的思想提供了一种思路：它将**“求出最优解是多少”**转化为了**“验证某一个给定的猜测解是否合法”**。这种视角的转换，能够将指数级或多项式级的求解时间，降维至对数时间复杂度 $O(\log N)$。

二、 理论基础：单调性与复杂度突破

1. 暴力穷举的性能瓶颈

若采用暴力枚举，从高度 0 逐一递增尝试高度 H：

- **状态空间**：最高树高若为 10^9 米，则最坏需要尝试 10^9 次。
- **状态验证**：每次尝试需遍历 N 棵树验证总木材量，耗时 $O(N)$。
- **总时间复杂度**：$O(N \times \text{MaxHeight})$。当 $N \approx 10^6$ 且 $\text{MaxHeight} \approx 10^9$ 时，运算量达 10^{15} 级别，必定导致 TLE。

2. 二分充要条件：单调性 (Monotonicity)

观察砍树问题的内在物理规律：

- 阈值 H **递增** $\rightarrow$ 被截取部分缩短 $\rightarrow$ 木材总量**单调递减。**
  这种严格的单调递减特性，构成了二分答案的充要条件。在解空间 $[0, \text{MaxHeight}]$ 中，必然存在一个明确的布尔值临界点（即 `true` 与 `false` 的分界线）。我们无需线性遍历，而是可以直接通过折半查找（二分法）快速逼近该临界点。
  优化后的总时间复杂度将骤降至：$O(N \log(\text{MaxHeight}))$。

三、 标准求解模板

解决此类问题的标准步骤如下：

1. 界定解空间 $[left, right]$

明确答案的数学下界与上界。在砍树问题中：

- `left` $= 0$（地平线）。
- `right` $= \max(\text{trees})$（树的理论最高值）。

2. 构造验证函数 `check(mid)`

设计一个返回布尔值的函数，负责回答判定性问题："若猜测答案为 mid，当前方案是否合法？"
在砍树问题中，仅需计算高度为 mid 时的总木材截取量，与需求 M 进行比较即可。

3. 在二分中收敛答案

在 $[left, right]$ 区间内不断取中值 mid 交由 `check` 验证，并基于验证结果收缩边界，最终逼近极限即为我们的答案。

四、 算法推演与代码实现

1. 状态转移演示

假设 $N = 4, M = 7$，树高集合为 $\{20, 15, 10, 17\}$。
初始状态：`left` $= 0$, `right` $= 20$, `ans` $= 0$。

迭代	$left$	$right$	mid	`check(mid)` 验证	判定结果	边界更新策略
1	0	20	10	截取量 $= 10 + 5 + 0 + 7 = 22 \geq 7$	`true`	木材充足，尝试抬高锯片：$ans = 10, left = 11$
2	11	20	15	截取量 $= 5 + 0 + 0 + 2 = 7 \geq 7$	`true`	木材刚好，尝试极值：$ans = 15, left = 16$
3	16	20	18	截取量 $= 2 + 0 + 0 + 0 = 2 < 7$	`false`	木材不足，必须降低锯片：$right = 17$
4	16	17	16	截取量 $= 4 + 0 + 0 + 1 = 5 < 7$	`false`	木材不足，必须降低锯片：$right = 15$

循环终止：此时 $left(16) > right(15)$，退出循环。最优解 $ans = 15$。

2. C++ 实现

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

using namespace std;

// 验证器：检测在高度 h 时，木材总量是否达标
bool check(long long h, long long m, const vector<int>& trees) {
    long long sum = 0; // 使用 long long 预防大数值累加溢出
    for (int tree_height : trees) {
```

```
        if (tree_height > h) {
            sum += (long long)(tree_height - h);
            if (sum >= m) return true; // 达标即刻返回
        }
    }
    return sum >= m;
}

int main() {
    int n;
    long long m;
    cin >> n >> m;

    vector<int> trees(n);
    int max_h = 0;
    for (int i = 0; i < n; ++i) {
        cin >> trees[i];
        max_h = max(max_h, trees[i]);
    }

    // 解空间初始化
    long long left = 0, right = max_h;
    long long ans = 0;

    // 二分求解主逻辑
    while (left <= right) {
        // 防止 left+right 数值溢出的写法
        long long mid = left + (right - left) / 2;

        if (check(mid, m, trees)) {
            ans = mid;        // 记录合法解
            left = mid + 1;  // 持续逼近上界（求最大值）
        } else {
            right = mid - 1; // 方案非法，收缩上界
        }
    }

    cout << ans << "\n";
    return 0;
}
```

五、进击的奶牛 (Aggressive Cows) —— 最大化最小值模型

1. 数学模型

- **物理场景**：给定一条直线上 `N` 个不重复的坐标点（牛舍），需要从中挑出 `M` 个坐标（安置奶牛）。
- **最优化目标**：最大化这 `M` 个坐标中，任意相邻两个坐标之间距离的**最小值**。
- **术语定义**：这类问题在算法竞赛中被称为**“最大化最小值”**问题。

2. 单调性与解空间界定

要使用二分查找，必须先证明问题在解空间上存在单调性。

- **问题**：能否在给定的坐标中选出 `M` 个点，使得任意两点间距 $\geq d$。
- **推导**：如果在间距要求极其苛刻（极大）的情况下 $P(d)$ 为 `true`，那么放宽要求，对于任意更小的间距 $d-1$，$P(d-1)$ 必然也为 `true`。
- **解空间映射**：随着距离 d 的不断递增，命题 $P(d)$ 的返回值序列必然呈现 `[true, true, ..., true, false, false, ...]` 的阶跃分布。
- **边界界定**：
  - 下界 `left = 0`（理论上的最小距离）。
  - 上界 `right = max_pos - min_pos`（理论上的最大物理跨度）。
  - **目标**：寻找使 $P(d)$ 为 `true` 的**右侧极限边界**（即最后一个 `true`）。

3. 贪心验证（`check` 函数）的正确性证明

在 `check(d)` 函数中，我们使用了"**排序 + 顺序贪心**"的策略。很多同学会产生疑问：为什么第一头牛必须强行放在第一个牛舍（最左侧）？

- **贪心选择性质证明**：
  假设存在一种最优的安置方案，第一头牛没有放在第一个牛舍，而是放在了第 `k` 个牛舍（$k > 1$）。此时，如果我们把这第一头牛强行向左移动，挪到第 1 个牛舍，它与第二头牛的距离只会**增加**，而绝不会减少。
  因此，将第一头牛紧贴最左侧放置，是为后续的牛预留最大生存空间的**绝对安全选择**（不劣于其他任何选择）。
- **线性扫描**：在固定了第一头牛之后，我们只需从左向右进行一次 $O(N)$ 的线性扫描。只要当前牛舍与上一头牛的相对距离 $\geq d$，就立即安置下一头牛。这种"能放就立刻放"的局部最优策略，最终能推导出全局的最大放置数量。

4. C++ 核心代码

```
// 当规定的绝对安全距离为 d 时，能否成功安置 m 头牛？
bool check(int d, int m, const vector<int>& stalls) {
    // 前提：stalls 数组在传入前必须已经过升序排序！
    int cows_placed = 1;       // 直接在最左端安置第一头牛
    int last_pos = stalls[0]; // 记录上一头牛的坐标

    for (int i = 1; i < stalls.size(); ++i) {
        // 只要物理距离达标，立即执行安置（局部最优解）
        if (stalls[i] - last_pos >= d) {
            cows_placed++;
            last_pos = stalls[i]; // 更新
        }
    }
    // 若实际安置总数大于等于目标需求 m，则当前距离 d 合法
    return cows_placed >= m;
}
```

六、两类求解最值的模板

二分答案在算法竞赛中高度模式化，主要分化为两大结构。

1. 最大化合法解 (求满足条件的最大值)

解空间的布尔分布呈：`[T, T, ..., T, F, F, ...]`
目标：寻找最后一个 `true`。

- **边界更新**：`check` 成功时记录 `ans`，并执行 `left = mid + 1` 贪心向右探索。

2. 最小化合法解 (求满足条件的最小值)

解空间的布尔分布呈：`[F, F, ..., F, T, T, ...]`
目标：寻找第一个 `true`。

- **边界更新**：`check` 成功时记录 `ans`，并执行 `right = mid - 1` 贪心向左探索。