# 题目

给定一个整数数组 ribbons 和一个整数 k，数组每项 ribbons[i] 表示第 i 条绳子的长度。对于每条绳子，你可以将任意切割成一系列长度为正整数的部分，或者选择不进行切割。

例如，如果给你一条长度为 4 的绳子，你可以：

保持绳子的长度为 4 不变；

切割成一条长度为 3 和一条长度为 1 的绳子；

切割成两条长度为 2 的绳子；

切割成一条长度为 2 和两条长度为 1 的绳子；

切割成四条长度为 1 的绳子。

你的任务是最终得到 k 条完全一样的绳子，他们的长度均为相同的正整数。如果绳子切割后有剩余，你可以直接舍弃掉多余的部分。

对于这 k 根绳子，返回你能得到的绳子最大长度；如果你无法得到 k 根相同长度的绳子，返回 0。

示例 1:

输入: ribbons = [9,7,5], k = 3

输出: 5

解释:

- 把第一条绳子切成两部分，一条长度为 5，一条长度为 4；

- 把第二条绳子切成两部分，一条长度为 5，一条长度为 2；

- 第三条绳子不进行切割；

现在，你得到了 3 条长度为 5 的绳子。

示例 2:

输入: ribbons = [7,5,9], k = 4

输出: 4

解释:

- 把第一条绳子切成两部分，一条长度为 4，一条长度为 3；

- 把第二条绳子切成两部分，一条长度为 4，一条长度为 1；

- 把第二条绳子切成三部分，一条长度为 4，一条长度为 4，还有一条长度为 1；

现在，你得到了 4 条长度为 4 的绳子。

示例 3:

输入: ribbons = [5,7,9], k = 22

输出: 0

解释: 由于绳子长度需要为正整数，你无法得到 22 条长度相同的绳子。

提示:

1 <= ribbons.length <= 105

1 <= ribbons[i] <= 105

1 <= k <= 109

# 分析

## 方法一：二分查找

这道题想让我们把给出的一组带子剪成k个长度相等的带子，且可以扔掉一些不需要的带子，并寻找能够满足要求的最大的带子的长度。我们可以使用二分查找的方法来寻找最大的符合要求的长度。首先我们需要划定查找的范围，以left作为最小值，right作为最大值。容易得到left为0。为了满足题目要求，所有带子的长度的和至少为每根带子的长度乘要求的带子的数量，因此right应当为所有带子的长度和除以k。在二分查找的过程中，如果mid的值不满足要求，那么下一步的查找区间应当为[left,mid-1]。如果满足要求，以res作为需要返回的结果，则res=mid，并在[mid+1,right]区间内继续搜索，如此往复，直到left>right后退出，返回结果。

以下为本题二分查找的模板：

while(left<=right)

{

int mid=(left+right)/2;

if(canCut(mid,ribbons,k))

{

res=mid;

left=mid+1;

}

else right=mid-1;

}

代码中的canCut函数就是判断mid是否为满足题意的长度的函数。我们需要将每根带子剪出尽可能多的长为mid的带子，而每根带子能剪出的最多的符合要求的带子的数量为ribbons[i]/mid，因此我们只需要遍历ribbons数组，求出ribbons[i]/mid并累加，即可得到最多的长度为mid的带子的数量sum。若sum>k则返回true，否则返回false。

canCut函数的代码如下：

bool canCut(int mid,vector<int>&ribbons,int k)

{

if(mid==0)return true;

long long sum=0;

for(int i=0;i<ribbons.size();i++)sum+=(ribbons[i]/mid);

return sum>=k;

}

本题ribbons[i]与ribbons.size()的数量级都是10^5，在累加时或许会出现值超出INT\_MAX-1的情况，因此sum我用了long long的数据类型。

以下为完整代码：

class Solution {

public:

bool canCut(int mid,vector<int>&ribbons,int k)

{

if(mid==0)return true;

long long sum=0;

for(int i=0;i<ribbons.size();i++)sum+=(ribbons[i]/mid);

return sum>=k;

}

int maxLength(vector<int>& ribbons, int k)

{

int n=ribbons.size();

long long sum=0;

for(int i=0;i<n;i++)sum+=ribbons[i];

sum/=k;

long long left=0,right=sum;

long long res=right;

while(left<=right)

{

int mid=(left+right)/2;

if(canCut(mid,ribbons,k))

{

res=mid;

left=mid+1;

}

else right=mid-1;

}

return res;

}

};

复杂度分析：

二分查找的时间复杂度是O(logN)，执行canCut函数的时间复杂度是O(N)，每次查找都需要调用一次canCut函数，故整体时间复杂度为O(NlogN)。