题目描述

小明有 n 块木板，第 i (1 ≤ i ≤ n) 块木板长度为 ai。
小明买了一块长度为 m 的木料，这块木料可以切割成任意块，拼接到已有的木板上，用来加长木板。
小明想让最短的模板尽量长。请问小明加长木板后，最短木板的长度可以为多少?

输入描述

输入的第一行包含两个正整数， n (1 ≤ n ≤ 10^3), m (1 ≤ m ≤ 10^6)，n 表示木板数， m 表示木板长度。
输入的第二行包含 n 个正整数， a1, a2,...an (1 ≤ ai ≤ 10^6)。

输出描述

输出的唯一一行包含一个正整数，表示加长木板后，最短木板的长度最大可以为多少?

用例

输入	5 3 4 5 3 5 5
输出	5
说明	给第1块木板长度增加1，给第3块木板长度增加2后， 这5块木板长度变为[5,5,5,5,5]，最短的木板的长度最大为5。

输入	5 2 4 5 3 5 5
输出	4
说明	给第3块木板长度增加1后，这5块木板长度变为[4,5,4,5,5]，剩余木料的长度为1。此时剩余木料无论给哪块木板加长，最短木料的长度都为4。

题目解析

本题的题意是比较明确的，我的解题思路如下：

要想让最短的木板尽可能长，那么我们就要不停地递进式补足最短板，比如用例输入有5个板：4 5 3 5 5，可用材料m=3

最短的板长度是3，只有一个，那么我们就将他补足到4，此时消耗了一单位长度的材料，m=2

这样的话，只剩下两种长度的板4，5，

且4长度有两个，5长度有三个，最短板是长度4.

接下来我们应该尽量将最短板4长度的板补足到5长度，而刚好剩余材料m=2，可以将所有4长度的板补足到5长度，此时所有板都是5长度，且材料耗尽。

我们还需要考虑一种特殊情况，那就是m还有值，但是只剩下一种长度的板，此时我们应该平分材料到每一个板，

假设只剩一种长度的板有count个，则平均分的话，每个板能分得 m / count 长度，这个值有可能是小数，我们举个例子：

5个一样长度x的板，m = 13，则 13 / 5 = 2...3，因此最短板长度就是x+2，

再比如

5个一样长度x的板，m = 15，则 13 / 5 = 3，因此最短板长度就是x+3。

本题有点 贪心 思维，即优先分配m的长度给最短板。

关于算法的时间复杂度，由于板子数量最多1000个，因此统计相同长度板子数量的时间复杂度很低。

然后m的长度达到了10^6，这就比较大了，但是我们通过不断递进式累计最短板数量，最终不会受到m长度的影响，只会受到板子数量的影响。

因此下面整体复杂度是O(N)，且N取值最多是1000。

JavaScript算法源码

题目解析

本题的题意是比较明确的，我的解题思路如下：

要想让最短的木板尽可能长，那么我们就要不停地递进式补足最短板，比如用例输入有5个板：4 5 3 5 5，可用材料m=3

最短的板长度是3，只有一个，那么我们就将他补足到4，此时消耗了一单位长度的材料，m=2

这样的话，只剩下两种长度的板4，5，

且4长度有两个，5长度有三个，最短板是长度4.

接下来我们应该尽量将最短板4长度的板补足到5长度，而刚好剩余材料m=2，可以将所有4长度的板补足到5长度，此时所有板都是5长度，且材料耗尽。

我们还需要考虑一种特殊情况，那就是m还有值，但是只剩下一种长度的板，此时我们应该平分材料到每一个板，

假设只剩一种长度的板有count个，则平均分的话，每个板能分得 m / count 长度，这个值有可能是小数，我们举个例子：

5个一样长度x的板，m = 13，则 13 / 5 = 2...3，因此最短板长度就是x+2，

再比如

5个一样长度x的板，m = 15，则 13 / 5 = 3，因此最短板长度就是x+3。

本题有点 贪心 思维，即优先分配m的长度给最短板。

关于算法的时间复杂度，由于板子数量最多1000个，因此统计相同长度板子数量的时间复杂度很低。

然后m的长度达到了10^6，这就比较大了，但是我们通过不断递进式累计最短板数量，最终不会受到m长度的影响，只会受到板子数量的影响。

因此下面整体复杂度是O(N)，且N取值最多是1000。

JavaScript算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const [n, m] = lines[0].split(" ").map(Number);
    const a = lines[1].split(" ").map(Number);
    console.log(getResult(m, a));

    lines.length = 0;
  }
});

function getResult(m, a) {
  // 统计每种长度板的数量，记录到count中，属性是板长度，属性值是板数量
  const count = {};
  for (let ai of a) {
    count[ai] ? count[ai]++ : (count[ai] = 1);
  }

  // 将统计到的板，按板长度升序
  const arr = [];
  for (let ai in count) {
    arr.push([ai - 0, count[ai]]);
  }
  arr.sort((a, b) => a[0] - b[0]);

  // 只要还有剩余的m长度，就将他补到最短板上
  while (m > 0) {
    // 如果只有一种板长度，那么就尝试将m平均分配到各个板上
    if (arr.length === 1) {
      const [len, count] = arr[0];
      return len + Math.floor(m / count);
    }

    // 如果有多种板长度
    // min1是最短板
    let min1 = arr.shift();
    // min2是第二最短板
    let min2 = arr[0];

    // diff是最短板和第二最短板的差距
    let diff = min2[0] - min1[0];

    // 将所有最短板补足到第二短板的长度，所需要总长度total
    let total = diff * min1[1];

    // 如果m的长度不够补足所有最短板，那么说明此时最短板的长度就是题解
    if (total > m) {
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const [n, m] = lines[0].split(" ").map(Number);
    const a = lines[1].split(" ").map(Number);
    console.log(getResult(m, a));

    lines.length = 0;
  }
});

function getResult(m, a) {
  // 统计每种长度板的数量，记录到count中，属性是板长度，属性值是板数量
  const count = {};
  for (let ai of a) {
    count[ai] ? count[ai]++ : (count[ai] = 1);
  }

  // 将统计到的板，按板长度升序
  const arr = [];
  for (let ai in count) {
    arr.push([ai - 0, count[ai]]);
  }
  arr.sort((a, b) => a[0] - b[0]);

  // 只要还有剩余的m长度，就将他补到最短板上
  while (m > 0) {
    // 如果只有一种板长度，那么就尝试将m平均分配到各个板上
    if (arr.length === 1) {
      const [len, count] = arr[0];
      return len + Math.floor(m / count);
    }

    // 如果有多种板长度
    // min1是最短板
    let min1 = arr.shift();
    // min2是第二最短板
    let min2 = arr[0];

    // diff是最短板和第二最短板的差距
    let diff = min2[0] - min1[0];

    // 将所有最短板补足到第二短板的长度，所需要总长度total
    let total = diff * min1[1];

    // 如果m的长度不够补足所有最短板，那么说明此时最短板的长度就是题解
    if (total > m) {
      return min1[0] + Math.floor(m / min1[1]);
    }
    // 如果m的长度刚好可以补足所有最短板，那么说明最短板可以全部升级到第二短板，且刚好用完m，因此第二短板的长度就是题解
    else if (total === m) {
      return min2[0];
    }
    // 如果m的长度足够长，能补足所有最短板到第二短板，还能有剩余，则将最短的数量加到第二短板的数量上，继续下轮循环
    else {
      m -= total;
      min2[1] += min1[1];
    }
  }

  return arr[0][0];
}
```

Java算法源码

```java
import java.util.HashMap;
import java.util.PriorityQueue;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    int m = sc.nextInt();

    int[] a = new int[n];
    for (int i = 0; i < n; i++) {
      a[i] = sc.nextInt();
    }
```

Java算法源码

```java
import java.util.HashMap;
import java.util.PriorityQueue;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    int m = sc.nextInt();

    int[] a = new int[n];
    for (int i = 0; i < n; i++) {
      a[i] = sc.nextInt();
    }

    System.out.println(getResult(m, a));
  }

  public static int getResult(int m, int[] a) {
    // 统计每种长度板的数量，记录到woods中，key是板长度，val是板数量
    HashMap<Integer, Integer> woods = new HashMap<>();
    for (Integer ai : a) {
      if (woods.containsKey(ai)) {
        Integer val = woods.get(ai);
        woods.put(ai, ++val);
      } else {
        woods.put(ai, 1);
      }
    }

    // 将统计到的板，按板长度排优先级，长度越短优先级越高，这里使用优先队列来实现优先级
    PriorityQueue<Integer[]> pq = new PriorityQueue<>((b, c) -> b[0] - c[0]);
    for (Integer wood : woods.keySet()) {
      pq.offer(new Integer[] {wood, woods.get(wood)});
    }

    // 只要还有剩余的m长度，就将他补到最短板上
    while (m > 0) {
      // 如果只有一种板长度，那么就尝试将m平均分配到各个板上
      if (pq.size() == 1) {
        Integer[] info = pq.poll();
        int len = info[0];
        int count = info[1];
        return len + m / count;
      }

      // 如果有多种板长度
      // min1是最短板
      Integer[] min1 = pq.poll();
      // min2是第二最短板
      Integer[] min2 = pq.peek();

      // diff是最短板和第二最短板的差距
      int diff = min2[0] - min1[0];
      // 将所有最短板补足到第二短板的长度，所需要总长度total
      int total = diff * min1[1];

      // 如果m的长度不够补足所有最短板，那么说明此时最短板的长度就是题解
      if (total > m) {
        return min1[0] + m / min1[1];
      }
      // 如果m的长度刚好可以补足所有最短板，那么说明最短板可以全部升级到第二短板，且刚好用完m，因此第二短板的长度就是题解
      else if (total == m) {
        return min2[0];
      }
      // 如果m的长度足够长，能补足所有最短板到第二短板，还能有剩余，则将最短的数量加到第二短板的数量上，继续下轮循环
      else {
        m -= total;
        min2[1] += min1[1];
      }
    }

    return pq.peek()[0];
  }
}
```

Python算法源码

```python
# 输入获取
import math

n, m = map(int, input().split())
a = list(map(int, input().split()))


# 算法入口
def getResult(m, a):
    # 统计每种长度板的数量，记录到count中，属性是板长度，属性值是板数量
    count = {}
```

Python算法源码

```python
# 输入获取
import math

n, m = map(int, input().split())
a = list(map(int, input().split()))


# 算法入口
def getResult(m, a):
    # 统计每种长度板的数量，记录到count中，属性是板长度，属性值是板数量
    count = {}
    for ai in a:
        if count.get(ai) is None:
            count[ai] = 1
        else:
            count[ai] += 1

    # 将统计到的板，按板长度升序
    arr = []
    for ai in count.keys():
        arr.append([int(ai), count[ai]])

    arr.sort(key=lambda x: x[0])

    # 只要还有剩余的m长度，就将他补到最短板上
    while m > 0:
        # 如果只有一种板长度，那么就尝试将m平均分配到各个板上
        if len(arr) == 1:
            lenV, count = arr[0]
            return lenV + math.floor(m / count)

        # 如果有多种板长度
        min1 = arr.pop(0) # min1是最短板
        min2 = arr[0] # min2是第二最短板

        # diff是最短板和第二最短板的差距
        diff = min2[0] - min1[0]

        # 将所有最短板补足到第二短板的长度，所需要总长度total
        total = diff * min1[1]

        # 如果m的长度不够补足所有最短板，那么说明此时最短板的长度就是题解
        if total > m:
            return min1[0] + math.floor(m / min1[1])
        # 如果m的长度刚好可以补足所有最短板，那么说明最短板可以全部升级到第二短板，且刚好用完m，因此第二短板的长度就是题解
        elif total == m:
            return min2[0]
        # 如果m的长度足够长，能补足所有最短板到第二短板，还能有剩余，则将最短的数量加到第二短板的数量上，继续下轮循环
        else:
            m -= total
            min2[1] += min1[1]

    return arr[0][0]


# 算法调用
print(getResult(m, a))
```