题目描述

一个工厂有m条流水线，来并行完成n个独立的作业，该工厂设置了一个调度系统，在安排作业时，总是优先执行处理时间最短的作业。

现给定流水线个数m，需要完成的作业数n，每个作业的处理时间分别为t1,t2…tn。请你编程计算处理完所有作业的耗时为多少？

当n>m时，首先处理时间短的m个作业进入流水线，其他的等待，当某个作业完成时，依次从剩余作业中取处理时间最短的进入处理。

输入描述

第一行为2个整数（采用空格分隔），分别表示流水线个数m和作业数n；

第二行输入n个整数（采用空格分隔），表示每个作业的处理时长t1,t2…tn。

0< m,n<100，0<t1,t2…tn<100。

注：保证输入都是合法的。

输出描述

输出处理完所有作业的总时长。

用例

输入	3 5 8 4 3 2 10
输出	13
说明	1、先安排时间为2、3、4的3个作业。 2、第一条流水线先完成作业，然后调度剩余时间最短的作业8。 3、第二条流水线完成作业，然后调度剩余时间最短的作业10。 4、总工耗时就是第二条流水线完成作业的时间13（3+10）。

题目解析

简单的逻辑题。解题思路如下：

题目说"总是优先执行处理时间最短的作业"，因此我们可以将8 4 3 2 10 进行升序排序变为2 3 4 8 10，然后依次将排序后元素投入对应流水线中，如下图所示

计算每条流水线的时间总和，最大的个就是题解。

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

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

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

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

    System.out.println(getResult(m, n, times));
  }

  public static int getResult(int m, int n, int[] times) {
    Arrays.sort(times);

    int[] mArr = new int[m];
    for (int i = 0; i < n; i++) {
      mArr[i % m] += times[i];
    }

    return Arrays.stream(mArr).max().orElse(0);
  }
}
```

JS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) {
    let [m, n] = lines[0].split(" ").map((ele) => parseInt(ele));
    let times = lines[1]
      .split(" ")
      .slice(0, n)
      .map((ele) => parseInt(ele));

    times.sort((a, b) => a - b);

    let mArr = new Array(m).fill(0);

    times.forEach((time, idx) => {
      mArr[idx % m] += time;
    });

    console.log(Math.max(...mArr));

    lines.length = 0;
  }
});
```

Python算法源码

```python
# 输入获取
m, n = map(int, input().split())
times = list(map(int, input().split()))


# 算法入口
def getResult():
    times.sort()

    mArr = [0]*m

    for i in range(len(times)):
        mArr[i % m] += times[i]

    return max(mArr)


# 算法调用
print(getResult())
```