Problem Statement

Fabian is in charge of a law firm working on an important case. For a case coming up, he needs a specific folder which is stored in one of the filing cabinets arranged in a line against the wall of the records room. He has assigned a number of workers to find the folder from the filing cabinets. He doesn't want the workers to get in each other's way, nor does he want folders from different filing cabinets getting mixed up, so he has decided to partition the cabinets, and assign a specific section to each worker. Each worker will have at least 1 cabinet to search through.

More specifically, Fabian wants to divide the line of filing cabinets into N sections (where N is the number of workers) so that every cabinet that the *i*th worker looks through is earlier in the line than every cabinet that the *j*th worker has to look through, for $i < j$.

His initial thought was to make all the sections equal, giving each worker the same number of filing cabinets to look through, but then he realized that the filing cabinets differed in the number of folders they contained. He now has decided to partition the filing cabinets so as to minimize the maximum number of folders that a worker would have to look through. For example, suppose there were three workers and nine filing cabinets with the following number of folders:

1. 10 20 30 40 50 60 70 80 90

He would divide up the filing cabinets into the following sections:

1. 10 20 30 40 50 | 60 70 | 80 90

The worker assigned to the first section would have to look through 150 folders. The worker assigned to the second section would have to search through 130 folders, and the last worker would filter through 170 folders. In this partitioning, the maximum number of folders that a worker looks through is 170. No other partitioning has less than 170 folders in the largest partition.

Write a class FairWorkload with a method getMostWork which takes a int[] **folders** (the number of folders for each filing cabinet) and an int **workers** (the number of workers). The method should return an int which is the maximum amount of folders that a worker would have to look through in an optimal partitioning of the filing cabinets. For the above example, the method would have returned 170.

Definition

Class:	FairWorkload
Method:	getMostWork
Parameters:	int[], int
Returns:	int
Method signature:	int getMostWork(int[] folders, int workers)

(be sure your method is public)

Constraints

- **folders** will contain between 2 and 15 elements, inclusive
- Each element of **folders** will be between 1 and 1000, inclusive
- **workers** will be between 1 and the number of elements in **folders**, inclusive

Examples

0)

```
{ 10, 20, 30, 40, 50, 60, 70, 80, 90 }
3
Returns: 170
```

This is the example from above.

1)

```
{ 10, 20, 30, 40, 50, 60, 70, 80, 90 }
5
Returns: 110
```

With the addition of two more workers, it makes more sense to partition the books as follows:

```
10 20 30 40 | 50 60 | 70 | 80 | 90
```

The most folders that any single worker must look through will be 110.

2)

```
{ 568, 712, 412, 231, 241, 393, 865, 287, 128, 457, 238, 98, 980, 23, 782 }
4
Returns: 1785
```

The filing cabinets should be partitioned as follows:

```
568 712 412 | 231 241 393 865 | 287 128 457 238 98 | 980 23 782
```

3)

```
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1000 }
2
Returns: 1000
```

The only fair partitioning gives the 14 cabinets with 1 folder to one worker, and the last cabinet with 1000 folders to the other wor ker.

4)

```
{ 50, 50, 50, 50, 50, 50, 50 }
2
Returns: 200
```

There are two valid partitions:

```
50 50 50 | 50 50 50 50
50 50 50 50 | 50 50 50
```

Both of these partitions result in a maximum of 200 folders for any worker.

5)

```
{1,1,1,1,100}
5
Returns: 100
```

With 5 workers, each worker gets a filing cabinet, and the most folders that any worker has to look through will be 100.

6)

```
{ 950, 650, 250, 250, 350, 100, 650, 150, 150, 700 }
6
Returns: 950
```

With 6 workers, the maximum amount of folders that any worker would have to look through is 950. There are 25 different ways of partitioning the filing cabinets to produce this maximum:

```
950 | 650 | 250 | 250 350 | 100 650 150 | 150 700
950 | 650 | 250 | 250 350 100 | 650 150 | 150 700
950 | 650 | 250 | 250 350 100 | 650 150 150 | 700
950 | 650 | 250 250 | 350 | 100 650 150 | 150 700
950 | 650 | 250 250 | 350 100 | 650 150 | 150 700
950 | 650 | 250 250 | 350 100 | 650 150 150 | 700
950 | 650 | 250 250 350 | 100 | 650 150 | 150 700
950 | 650 | 250 250 350 | 100 | 650 150 150 | 700
950 | 650 | 250 250 350 | 100 650 | 150 | 150 700
950 | 650 | 250 250 350 | 100 650 | 150 150 | 700
950 | 650 | 250 250 350 | 100 650 150 | 150 | 700
950 | 650 | 250 250 350 100 | 650 | 150 | 150 700
950 | 650 | 250 250 350 100 | 650 | 150 150 | 700
950 | 650 | 250 250 350 100 | 650 150 | 150 | 700
950 | 650 250 | 250 | 350 | 100 650 150 | 150 700
950 | 650 250 | 250 | 350 100 | 650 150 | 150 700
950 | 650 250 | 250 | 350 100 | 650 150 150 | 700
950 | 650 250 | 250 350 | 100 | 650 150 | 150 700
950 | 650 250 | 250 350 | 100 | 650 150 150 | 700
950 | 650 250 | 250 350 | 100 650 | 150 | 150 700
950 | 650 250 | 250 350 | 100 650 | 150 150 | 700
950 | 650 250 | 250 350 | 100 650 150 | 150 | 700
950 | 650 250 | 250 350 100 | 650 | 150 | 150 700
950 | 650 250 | 250 350 100 | 650 | 150 150 | 700
950 | 650 250 | 250 350 100 | 650 150 | 150 | 700
```