2. Parallel Processing

A computer has a certain number of cores and a list of files that need to be executed. If a file is executed by a single core, the execution time equals the number of lines of code in the file. If the lines of code can be divided by the number of cores, another option is to execute the file in parallel using all the cores, in which case the execution time is divided by the number of cores. However, there is a limit as to how many files can be executed in parallel. Given the lengths of the code files, the number of cores, and the limit, what is the minimum amount of time needed to execute all the files?

For example, let's say that there are $n = 5$ files, where $files = [4, 1, 3, 2, 8]$ (indicating the number of lines of code in each file), $numCores = 4$, and $limit = 1$. Even though both the first and fifth files can be executed in parallel, you must choose only one of them because the limit is 1. The optimal way is to parallelize the last file, so the minimum execution time required is 4 + 1 + 3 + 2 + (8/4) = 12. Therefore, the answer is 12.

Function Description
Complete the function *minTime* in the editor below.

minTime has the following parameter(s):
int *files[n]*: an array of integers where *files[i] indicates the number of lines of code in the* i^{th} *file*
int *numCores:* the number of cores in the computer
int *limit*: the maximum number of files that can be executed in parallel
Returns:
long int: the minimum units of time needed to execute all the files

Constraints

- $1 \le n \le 10^5$
- $1 \le files[i] \le 10^9$
- $1 \le numCores \le 10^9$
- $1 \le limit \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of *files* on the computer.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains a long integer, *files[i]*, denoting the *number of lines of code in the* i^{th} *file.*
The next line contains a long integer, *numCores*, denoting the number of cores in the computer.
The last line contains a long integer, *limit*, denoting the max number of files that can be executed in parallel.

▼ Sample Case 0

Sample Input For Custom Testing

```
3
5
3
1
5
5
```

Sample Output

```
5
```

Explanation
Here, there are $n = 3$ files on the computer, where $files = [5, 3, 1]$, $numCores = 5$, and $limit = 5$. Even though we can parallelize up to 5 pieces of code, we only parallelize the first file because the lines of code can be divided between cores equally. So, the minimum time required is (5/5) + 3 + 1 = 5. Therefore, the answer is 5.

▼ Sample Case 1

Sample Input For Custom Testing

```
3
3
1
5
1
5
```

Sample Output

```
9
```

Explanation

Here, there are *n = 3* files on the computer, where *files = [3, 1, 5]*, *numCores = 1*, and *limit = 5*. Even though we can parallelize up to 5 pieces of code, we only have 1 core, so parallelization won't help us. So, the minimum time required is 3 + 1 + 5 = 9. Therefore, the answer is 9.