EKO-SPOJ SPOJ

Lumberjack Mirko needs to cut down M meters of wood using his woodcutting machine. The machine has a sawblade that can be raised to a certain height H (in meters) and cuts off all tree parts higher than H. Mirko wants to set the sawblade as high as possible while still being able to cut off at least M meters of wood. Help Mirko find the maximum integer height of the sawblade.

Input:

The first line of input contains two space-separated positive integers, N and M, where N is the number of trees (1 ≤ N ≤ 1,000,000) and M is Mirko's required amount of wood (1 ≤ M ≤ 2,000,000,000).

The second line of input contains N space-separated positive integers, the heights of each tree (in meters). The heights will be less than 1,000,000,000, and the sum of all heights will exceed M.

Output:

The first and only line of output must contain the required height setting.

Input:

4 7

20 15 10 17

Output:

15

Explanation:

Mirko has 4 trees with heights 20, 15, 10, and 17 meters. He needs to cut off at least 7 meters of wood. By setting the sawblade to a height of 15 meters, the remaining tree heights will be 15, 15, 10, and 15 meters. Mirko will take 5 meters off the first tree and 2 meters off the fourth tree, totaling 7 meters of wood.

Input:

5 20

4 42 40 26 46

Output:

36

Explanation:

Mirko has 5 trees with heights 4, 42, 40, 26, and 46 meters. He needs to cut off at least 20 meters of wood. By setting the sawblade to a height of 36 meters, the remaining tree heights will be 4, 36, 36, 26, and 36 meters. Mirko will take 6 meters off the second tree, 4 meters off the third tree, and 10 meters off the fifth tree, totaling 20 meters of wood.

Approach 1: Using Binary Search Algorithm to find the maximum height of the sawblade

The isPossibleSolution() function checks if a certain height (mid) is a possible solution by iterating through the tree heights. It calculates the amount of wood that can be obtained from the trees at the given height and compares it with the required wood. The function returns true if the wood count is greater than or equal to the required wood, indicating that the height is a valid solution.

The maximumHeight() function finds the maximum height of the sawblade using a binary search approach. It first sorts the tree heights in ascending order. Then, it initializes low and high as the minimum and maximum possible heights, respectively, based on the sorted tree heights. It applies binary search by iteratively updating low and high until they converge. In each iteration, it calculates the mid height and checks if it is a possible solution using the isPossibleSolution() function. If it is a solution, it updates ans with mid and looks for larger heights by updating low. Otherwise, it looks for smaller heights by updating high. Finally, it returns the maximum height found (ans).

In the main() function, a test case with tree heights {26, 4, 42, 40, 46} and a required wood amount of 20 is used to demonstrate the algorithm. The maximumHeight() function is called with the test case parameters, and the result is printed.

The time complexity of the algorithm is O(N log M), where N is the number of trees and M is the difference between the minimum and maximum tree heights. The sort operation takes O(N log N) time, and the binary search loop performs O(log M) iterations.

The space complexity is O(1) since the algorithm uses a constant amount of extra space for variables.