**1. Problem Discussion**

You will be provided with an array containing n integers and you will also be given an integer say threshold , So you have to choose an integer or divisor such that if we divide all the array elements by that number and then sum all the elements and then the result should be less than or equal to threshold. So you have to return the smallest divisor.

**2. Approach**

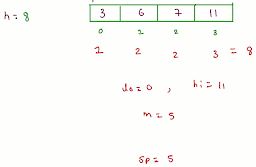
Here we will be solving this problem in O(n\*log(max)) time where ‘max’ is the maximum value in the array. Since, the value of ‘threshold’ could not be less than arr.length because in that case (element/divisor) < 1(which is not possible since we are taking the ceil here) . So, that case would be invalid. So, the value of ‘threshold’ would always be greater than or equal to arr.length i.e. (element/divisor would definitely give value>=1). Therefore, the maximum possible divisor would be ‘max’ because in that case (arr[i]/divisor) would give a value 1 for all the array elements and the minimum possible value could be any value in the range of 1 and max. So, in order to find the minimum sum of the array elements after division, we will use a Binary search approach between the possible divisor ranges that will take O(log(max)) time to find that divisor. And, for a particular divisor we need to check whether the sum of array elements after dividing all the elements by divisor is less than or equal to threshold or not. That will take O(n) time. Suppose we have an array ‘piles’ with given entries and the value of ‘threshold’ is 6. Now, we’ve to find the minimum divisor.

****

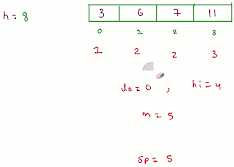
We will take 2 variables that define the possible divisor range as ‘lo’ and ‘hi’ representing the smallest and largest divisor possible. Initially, ‘lo’ will have value 1 and ‘hi’ will have the maximum value in the array.

****

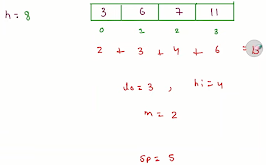
Now, instead of a linear approach to find the minimum divisor in this range, we will use a binary search approach because that will be more time-efficient. We will be taking a ‘sp’ variable that stores the minimum divisor found till now. And, a variable ‘mid’ that stores the divisor for which we are checking the possibility to be minimum divisor.

****

If a ‘mid’ divisor is possible then update the final divior ‘sp’ with ‘mid’ value and then check for another possible divisor in the left range by updating the ‘hi’ with ‘mid’-1.

****

Else if ‘mid’ is not a possible divisor then we need to increase the divisor for which to update the range by updating ‘lo’ with ‘mid’+1.

****

In this way, iterate until ‘lo’ gets greater or equal to ‘hi’. And, finally ‘sp’ will have the minimum divisor.

**3. Code**

ConsoleJava

import java.util.\*;

import java.io.\*;

public class Main {

public static int findSmallestDivisor(int[]nums,int th) {

//write your code here

int max = 0;

for(int val : nums) {

max = Math.max(val,max);

}

if(th == nums.length) {

return max;

}

int lo = 1;

int hi = max;

int divisor = 0;

while(lo <= hi) {

int div = lo + (hi-lo)/2;

boolean temp = isPossible(nums,div,th);

if(temp == true) {

divisor = div;

hi = div-1;

}

else {

lo = div+1;

}

}

return divisor;

}

public static boolean isPossible(int[]nums,int div,int th) {

int ans = 0;

for(int i=0; i < nums.length ; i++) {

ans = ans + (int)Math.ceil(nums[i]\*1.0/div);

}

return ans <= th;

}

public static void main(String[]args) {

Scanner scn = new Scanner(System.in);

//input work

int n = scn.nextInt();

int[]nums = new int[n];

for(int i=0 ; i < n ; i++) {

nums[i] = scn.nextInt();

}

int th = scn.nextInt();

int speed = findSmallestDivisor(nums,th);

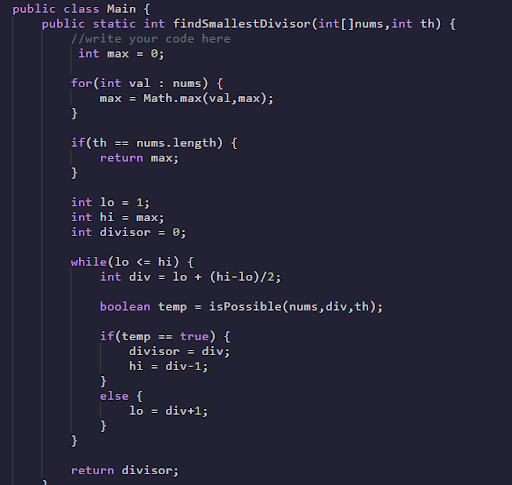
System.out.println(speed);

}

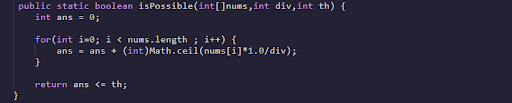
}

**4. Code Expalanation**

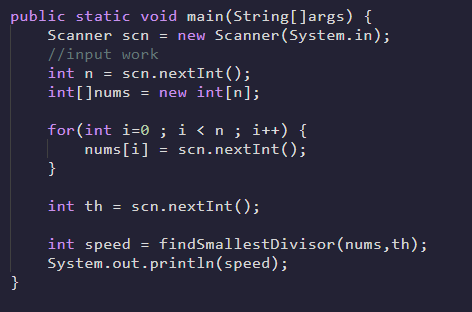
In the findSmallestDivisor() function, first we are calculating the maximum value in the array and storing it in the ‘max’ variable. After defining a possible divisor range with the help of ‘lo’ and ‘hi’ variables and finally using binary search approach in this range to find the minimum divisor for dividing array elements and storing it in ‘divisor’ variable.

****

In the isPossible() function, we are checking whether the sum of the array after dividing by div all the array elements is less than or equal to ‘th’ . And, the formula used for calculating the quotient of a single element is given as : ceil( nums[i] / div ).

****

Now, in the main() function, we are taking input of ‘nums’ array the ‘th’ variable that represents the number of threshold. Finally, calling our findSmallestDivisor() function to calculate the minimum divisor and printing it.

****

**5. Analysis**

Time Complexity

O(n\*log(max)) Here the time Complexity is O(n\*log(max)) because we are applying a binary search over a range(1,max) and for which O(log(max)) and for each speed we are traversing over all the array elements to calculate the sum in O(n).

Space Complexity

Constant Here we are not using any extra space so complexity is constant.