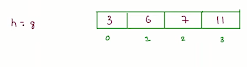
**1. Problem Discussion :**

In this problem, there is a boy Koko and Koko loves to eat bananas. There are ‘n’ piles of bananas in the garden, the ith pile has ‘piles[i]’ bananas. The guards have gone and will come back in ‘h’ hours. Koko can decide her bananas-per-hour eating speed(‘k’). Each hour, she chooses some pile of bananas and eats ‘k’ bananas from that pile, if available. If the pile has less than ‘k’ bananas, she eats all of them instead and will not eat any more bananas during this hour. Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return. You are supposed to return the minimum integer ‘k’ such that she can eat all the bananas within ‘h’ hours. For Example: There are 4 piles of bananas with 3, 6, 7 and 11 numbers of bananas in each pile respectively. Guards are going to return in 8 hours. So, we have to find the minimum speed for Koko to eat all the bananas within 8 hours. The minimum speed would be 4 bananas per hour because with this speed, first pile would take an hour, second pile would take 2 hours, third pile would take 2 hours and the fourth pile would take 3 hours i.e total hours taken are 8 (1+2+2+3). Easy right !

**2. Approach**

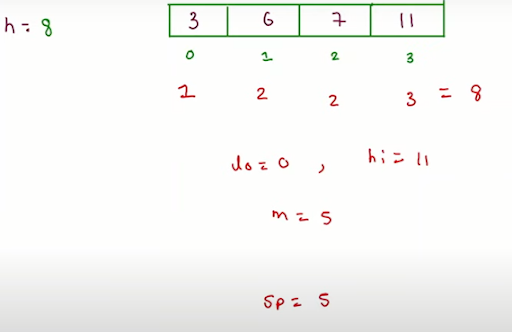
Here we will be solving this problem in O(n\*log(max)) time where ‘max’ is the maximum value in the ‘piles’ array. Since, the value of ‘h’ could not be less than piles.length because in that case we cannot give the minimum required single hour to all the piles. So, that case would be invalid. So, the value of ‘h’ would always be greater than or equals to piles.length i.e. Koko would definitely be giving an hour to each pile. Therefore, the maximum possible speed of eating bananas would be ‘max’ because in that case Koko will take an hour to eat all bananas from all the piles and the minimum possible value could be any value in the range of 0 and max. So, in order to find the minimum speed, we will use a Binary search approach between the possible speeds range that will take O(log(max)) time to find that speed. And, for a particular speed we need to check whether Koko is able to eat all the bananas in ‘h’ hours or not. That will take O(n) time. Suppose we have an array ‘piles’ with given entries and the value of ‘h’ is 8. Now, we’ve to find the minimum speed of eating bananas.

****

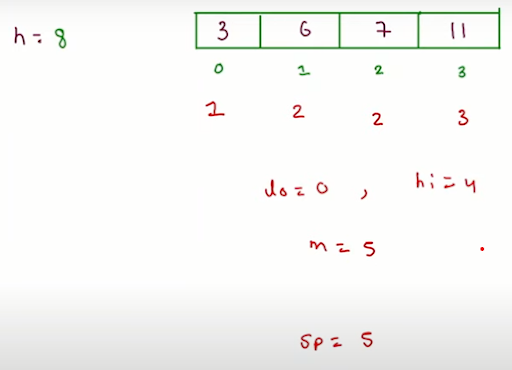
We will take 2 variables that define the possible speed range as ‘lo’ and ‘hi’ representing the lowest and highest speeds possible. Initially, ‘lo’ will have value 0 and ‘hi’ will have the maximum value in the piles array.

****

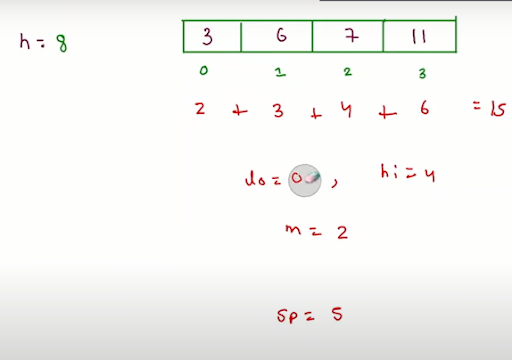
Now, instead of a linear approach to find the minimum speed in this range, we will use a binary search approach because that will be more time-efficient. We will be taking a ‘sp’ variable that stores the minimum speed found till now. And, a variable ‘mid’ that stores the speed for which we are checking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

****

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

****

Else if ‘mid’ is not a possible speed then Koko needs to increase its speed for which to update the range by updating ‘lo’ with ‘mid’+1.

****

**3. Code**

ConsoleJava

import java.util.\*;

import java.io.\*;

public class Main {

public static int minEatingSpeed(int[]piles,int h) {

//write your code here

int max = 0;

for(int val : piles) {

max = Math.max(max,val);

}

if(h == piles.length) {

return max;

}

int lo = 0;

int hi = max;

int speed = 0;

while(lo <= hi) {

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

boolean temp = isPossible(piles,sp,h);

if(temp == true) {

speed = sp;

hi = sp - 1;

}

else {

lo = sp + 1;

}

}

return speed;

}

public static boolean isPossible(int[]piles,int sp,int h) {

int ans = 0;

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

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

}

if(ans <= h) {

return true;

}

else {

return false;

}

}

public static void main(String[]args) {

Scanner scn = new Scanner(System.in);

//input work

int n = scn.nextInt();

int[]piles = new int[n];

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

piles[i] = scn.nextInt();

}

int h = scn.nextInt();

int speed = minEatingSpeed(piles,h);

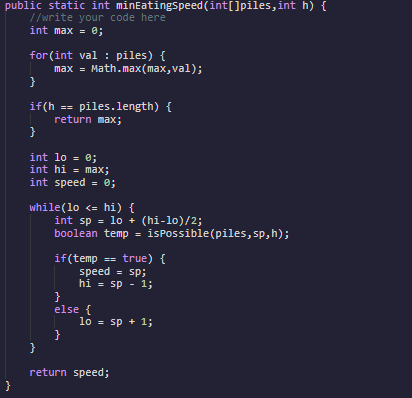
System.out.println(speed);

}

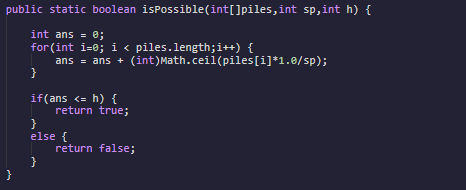
}

**4. Code Expalnation**

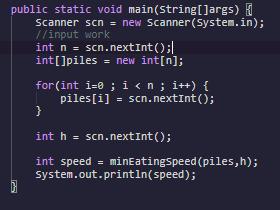
This code consists of 3 functions.

****

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

****

In the isPossible() function, we are checking whether Koko is able to eat all the bananas in ‘h’ hours with the speed of ‘sp’ bananas per hour. And, the formula used for calculating the no. of hours required to eat all bananas of a single pile is given as : ceil( piles[i] / sp ).

****

Now, in the main() function, we are taking input of ‘piles’ array that represents the no. of bananas in each pile and the ‘h’ variable that represents the number of hours Koko has. Finally, calling our minEatingSpeed() function to calculate the minimum speed and printing it.

**5. Analysis**

Time Complexity

O(n\*log(max)) Here the time complexity is O(n\*log(max)) because we are applying binary search over a range (0,max) for which O(log(max)) and for each speed we are traversing over all the array elements to calculate the number of hours required to eat all the bananas in all the piles for which O(n).

Space Complexity

Constant Here the space complexity is constant as no extra space is used.