Eating Mangoes slowly

koko eating banana

Problem Description

You are given **N** piles of mangoes represented by an array of Integers **A**. The ith pile has **A[i]** mangoes. You need to finish eating all the mangoes in **B** hours.
You can decide the mangoes-per-hour eating speed of **k**. Each hour, you can choose some pile and eat **k** mangoes from that pile. If the pile has less than **k** mangoes, you have to eat all of them instead and will not eat any more mango during this hour.

Return the minimum integer **k** such that you can eat all the mangoes within **B** hours.

A = [3 6 7 11] (0 1 2 3) B = 8

K → mangoes / hour
eating speed

find min value of K.

A = [30 11 23 4 20] (0 1 2 3 4) B = 5

K = 30 { B = N }

we have to spend atleast 1 hour on a pile.

if (no. of piles = B)

→ you are allowed to stay on a single pile for just 1 hour.

A = {1 2 6 5 3} B = 4

K = 6

k : lo = 1 hi = max of array

A = [3 6 7 11] (0 1 2 3)

B = 8

ans = ~~8~~ 4

Is it possible to eat all mangoes within B hours if eating speed = mid

lo = 1	hi = 11	mid = 6	yes, hi = mid - 1
lo = 1	hi = 5	mid = 3	no, lo = mid + 1
lo = 4	hi = 5	mid = 4	yes, hi = mid - 1

lo = 4 hi = 3

Is it possible to eat all mangoes within B hours if eating speed = mid → isPossible ( )

A = [3 6 7 11] (0 1 2 3) (l)

B = 8

hrs 1 2 2 3

sp = 5

$hrs += ceil\left(\frac{A[i] * 1.0}{sp}\right)$

if (hrs <= B)
→ return true

else
→ return false

```java
public class Solution {
    public int solve(int[] A, int B) {

        //A -> piles of mangoes
        //B -> number of hours allowed
        //to find -> min eating speed


        int max = A[0];
        for(int i=0; i < A.length;i++) {
            max = Math.max(max,A[i]);
        }

        int lo = 1, hi = max;
        int ans = 0;

        while(lo <= hi) {
            int mid = (lo + hi)/2;

            if(isPossible(A,B,mid) == true) {
                ans = mid;
                hi = mid-1;
            }
            else {
                lo = mid+1;
            }
        }

        return ans;
    }


    /*
        Is it possible to eat all manoges within B hours,
        if mango eating speed = sp
    */
    boolean isPossible(int[]A, int B, int sp) {
        int hrs = 0;

        for(int i=0; i < A.length;i++) {
            int temp = (int)Math.ceil(A[i]*1.0/sp);
            hrs += temp;

            if(hrs > B) {
                return false;
            }
        }

        return true;
    }
}
```

K → min eating speed

=> mangoes / hr

A = [2 8 3 13] B = 8

ans = 4

isPossible

lo = 1 hi = 13 mid = 7 true

lo = 1 hi = 6 mid = 3 false

lo = 4 hi = 6 mid = 4 true

lo = 4 hi = 3

Sort the Coordinates

Problem Description

Given a **2D** Array **A** of size **Nx2** in which ith element is a point with **A[i][0]** and **A[i][1]** as **x** and **y** coordinates, respectively, sort the **2D** Array in ascending order of distance of points from origin.

If two points have the same distance from the origin, then the one with a smaller value of **x** should come first.
If two points have the same distance from the origin and also have same **x** coordinate, then the one with a smaller value of **y** should come first.

A = [[2,4] , [-1,3], [-1,2] , [0,1], [2,1], [-1,-3]]

dist 20 10 5 1 5 10

distance of (x,y) from (0,0) $\Rightarrow \sqrt{x^2+y^2}$

for comparison can we use only x^2+y^2.

A = [[2,4] , [-1,3], [-1,2] , [0,1], [2,1], [-1,-3]]

dist 20 10 5 1 5 10

A = [[0,1] [-1,2] [2,1] [-1,-3] [-1,3] [2,4]]

1 5 5 10 10 20

(1st) (2nd)

x_1, y_1 x_2, y_2

int d1 = $x_1^2 + y_1^2$; // distance of 1st from origin

int d2 = $x_2^2 + y_2^2$; // distance of 2nd from origin

```
if (d1 != d2) {
        return d1-d2;
}
else if (x1 != x2) {
        return x1 - x2;
}
else {
        return y1 - y2;
}
```

$x_1 = 2, y_1 = -1$

$x_2 = -1, y_2 = -2$

$d_1 = 5, d_2 = 5$

compare (a, b)
- -ve (keep a first)
- +ve (keep b first)
- 0 (doesn't matter)

```java
public class Solution {
    public int[][] solve(int[][] A) {
        int n = A.length;

        //converting int[][] to Integer[][]
        Integer[][]arr = new Integer[n][2];
        for(int i=0; i < n;i++) {
            arr[i][0] = A[i][0];
            arr[i][1] = A[i][1];
        }


        //sort arr
        Arrays.sort(arr, new Comparator<Integer[]>(){
            public int compare(Integer[]a,Integer[]b) {
                int x1 = a[0], y1 = a[1];
                int x2 = b[0], y2 = b[1];

                int d1 = x1*x1 + y1*y1;
                int d2 = x2*x2 + y2*y2;

                if(d1 != d2) {
                    return d1-d2;
                }
                else if(x1 != x2) {
                    return x1-x2;
                }
                else {
                    return y1-y2;
                }
            }
        });



        //converting Integer[][] to int[][]
        for(int i=0; i < n;i++) {
            A[i][0] = arr[i][0];
            A[i][1] = arr[i][1];
        }

        return A;
    }
}
```

Q.3 **Max product of 3 elements.**

A[ ] = [1, 2, 4, 5, 6]

Product = 4 * 5 * 6 = 120

A[ ] = [-2, -1, 2, 5, 8]

Product = 2 * 5 * 8

A[ ] = [-3, -2, 0, 1, 2, 8]

Product = -3 * -2 * 8

multiplying 3 largest no.

OR

multiplying 2 smallest no. with largest no.

```java
public class Solution {
    public int solve(int[] A) {
        int n  = A.length;
        Arrays.sort(A);

        //multiplying 3 largest numbers
        int v1 = A[n-1] * A[n-2] * A[n-3];

        //multiplying 2 smallest numbers with 1 largest
        int v2 = A[0] * A[1] * A[n-1];

        return Math.max(v1,v2);
    }
}
```