Heaters

Winter is coming! Your first job during the contest is to design a standard heater with fixed warm radius to warm all the houses.

Now, you are given positions of houses and heaters on a horizontal line, find out minimum radius of heaters so that all houses could be covered by those heaters.

So, your input will be the positions of houses and heaters seperately, and your expected output will be the minimum radius standard of heaters.

Note:

1. Numbers of houses and heaters you are given are non-negative and will not exceed 25000.
2. Positions of houses and heaters you are given are non-negative and will not exceed 10^9.
3. As long as a house is in the heaters' warm radius range, it can be warmed.
4. All the heaters follow your radius standard and the warm radius will the same.

Example 1:

```
Input: [1,2,3],[2]
Output: 1
Explanation: The only heater was placed in the position 2, and if we use the radius
 1 standard, then all the houses can be warmed.
```

Example 2:

```
Input: [1,2,3,4],[1,4]
Output: 1
Explanation: The two heater was placed in the position 1 and 4. We need to use radi
us 1 standard, then all the houses can be warmed.
```

Solution 1

The idea is to leverage decent `Arrays.binarySearch()` function provided by Java.

1. For each `house`, find its position between those `heaters` (thus we need the `heaters` array to be sorted).
2. Calculate the distances between this `house` and left `heater` and right `heater`, get a `MIN` value of those two values. Corner cases are there is no left or right heater.
3. Get `MAX` value among distances in step 2. It's the answer.

Time complexity: max(O(nlogn), O(mlogn)) - m is the length of houses, n is the length of heaters.

```java
public class Solution {
    public int findRadius(int[] houses, int[] heaters) {
        Arrays.sort(heaters);
        int result = Integer.MIN_VALUE;

        for (int house : houses) {
         int index = Arrays.binarySearch(heaters, house);
         if (index < 0) {
          index = -(index + 1);
         }
         int dist1 = index - 1 >= 0 ? house - heaters[index - 1] : Integer.MAX_VA
LUE;
         int dist2 = index < heaters.length ? heaters[index] - house : Integer.MA
X_VALUE;

         result = Math.max(result, Math.min(dist1, dist2));
        }

        return result;
    }
}
```

written by shawngao original link here

Solution 2

Based on 2 pointers, the idea is to find the nearest heater for each house, by comparing the next heater with the current heater.

```
public class Solution {
    public int findRadius(int[] houses, int[] heaters) {
        Arrays.sort(houses);
        Arrays.sort(heaters);

        int i = 0, j = 0, res = 0;
        while (i < houses.length) {
            while (j < heaters.length - 1
                && Math.abs(heaters[j + 1] - houses[i]) <= Math.abs(heaters[j] -
houses[i])) {
                j++;
            }
            res = Math.max(res, Math.abs(heaters[j] - houses[i]));
            i++;
        }

        return res;
    }
}
```

Updated solution inspired by @StefanPochmann

```
public class Solution {
    public int findRadius(int[] houses, int[] heaters) {
        Arrays.sort(houses);
        Arrays.sort(heaters);

        int i = 0, res = 0;
        for (int house : houses) {
            while (i < heaters.length - 1 && heaters[i] + heaters[i + 1] <= house
* 2) {
                i++;
            }
            res = Math.max(res, Math.abs(heaters[i] - house));
        }

        return res;
    }
}
```

written by lufangjianle original link here

Solution 3

Go through houses and heaters in ascending order. My `i` points to the current closest heater. Go to the next heater if the current house coordinate is larger than or equal to the middle between the current and the next heater.

```
def findRadius(self, houses, heaters):
    heaters = sorted(heaters) + [float('inf')]
    i = r = 0
    for x in sorted(houses):
        while x >= sum(heaters[i:i+2]) / 2.:
            i += 1
        r = max(r, abs(heaters[i] - x))
    return r
```

I btw started with `while abs(heaters[i+1] - x) <= abs(heaters[i] - x):`, the straight-forward check whether the next heater is closer than the current. Then I thought I probably don't need `abs` if I just use `while heaters[i+1] - x <= x - heaters[i]:`. That's obviously correct if `x` is between the heaters, because then that's the correct distances of `x` to the two heaters. Less obviously (but imho not surprisingly) it's also correct if `x` isn't between them. Finally, after rewriting it to `while heaters[i] + heaters[i+1] <= 2 * x:` I realized what that meant :-)

Update: Another solution by using binary search, inspired by others:

```
def findRadius(self, houses, heaters):
    heaters.sort()
    return max(min(abs(house - heater)
                   for i in [bisect.bisect(heaters, house)]
                   for heater in heaters[i-(i>0):i+1])
               for house in houses)
```

written by StefanPochmann original link here