Previous (/articles/contains-duplicate-iii/) Next (/articles/summary-ranges/)

274. H-Index (/problems/h-index/)

March 30, 2016 | 20.5K views

Average Rating: 4.58 (40 votes)

Given an array of citations (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia (https://en.wikipedia.org/wiki/H-index): "A scientist has index *h* if *h* of his/her *N* papers have **at least** *h* citations each, and the other *N* − *h* papers have **no more than** *h* citations each."

Example:

```
Input: citations = [3,0,6,1,5]
Output: 3
Explanation: [3,0,6,1,5] means the researcher has 5 papers in total and each
 of them had
             received 3, 0, 6, 1, 5 citations respectively.
             Since the researcher has 3 papers with at least 3 citations eac
h and the remaining
             two with no more than 3 citations each, her h-index is 3.
```

Note: If there are several possible values for *h*, the maximum one is taken as the h-index.

Summary

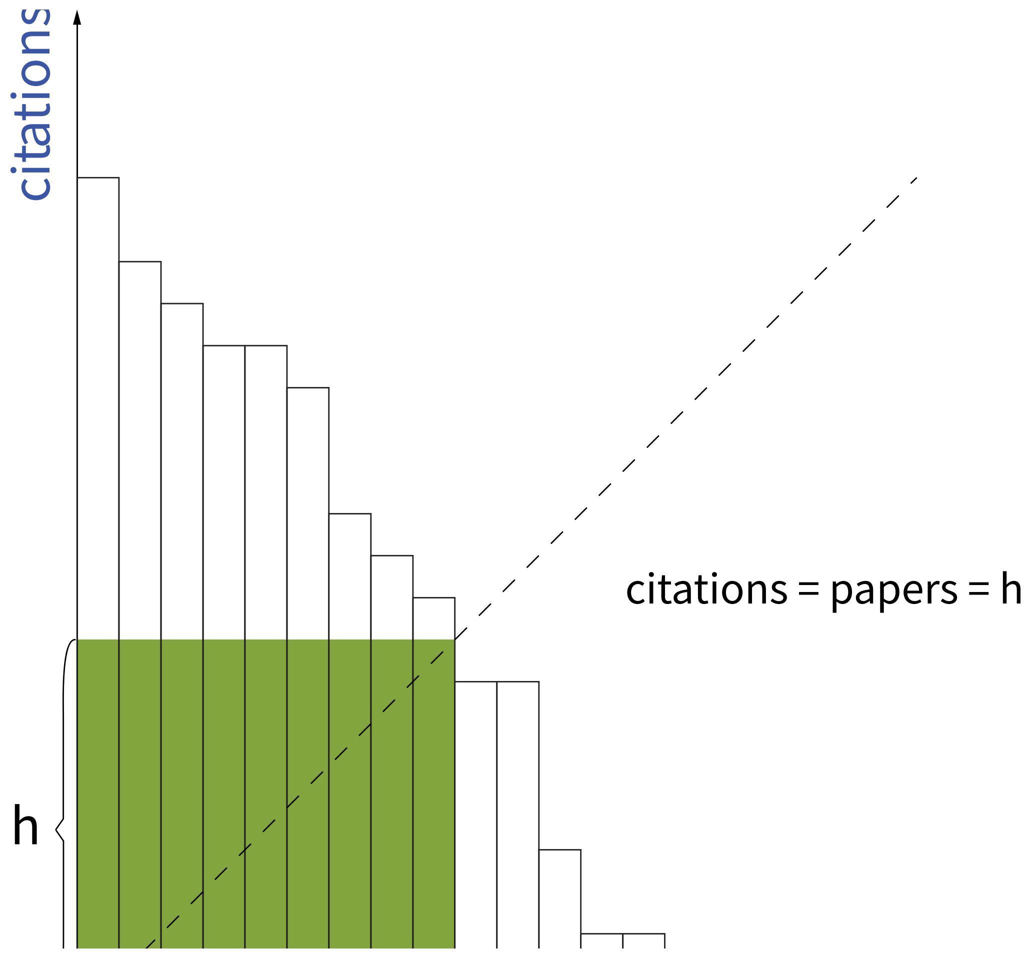
This article is for intermediate readers. It introduces the following ideas: Comparison Sort and Counting Sort.

Solution

Approach #1 (Sorting) [Accepted]

Intuition

Think geometrically. Imagine plotting a histogram where the y-axis represents the number of citations for each paper. After sorting in *descending* order, h-index is the length of the largest **square** in the histogram.



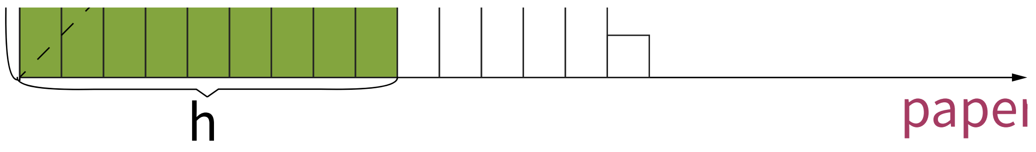


Figure 1. h-index from a plot of decreasing citations for papers

Algorithm

To find such a square length, we first sort the citations array in *descending* order. After sorting, if $\mathrm{citations}[i] > i$, then papers 0 to i all have at least $i + 1$ citations.

Thus, to find h-index, we search for the largest i (let's call it i') such that

$$\mathrm{citations}[i] > i$$

and therefore the h-index is $i' + 1$.

For example:

i	0	1	2	3	4	5	6
sorted citations	10	9	5	3	3	2	1
$\mathrm{citations}[i] > i?$	true	true	true	false	false	false	false

In this example, we know that the largest i with $\mathrm{citations}[i] > i$ is $i' = 2$. Thus

$$h = i' + 1 = 3$$

Because $\mathrm{citations}[i'] > i'$, $i' + 1$ papers (from paper 0 to paper i') have citations at least $i' + 1$ and $n - i' - 1$ papers (from paper $i' + 1$ to paper $n - 1$) have citations no more than $i' + 1$. By the definition of h-index, $h = i' + 1$.

It is also possible to find i' through binary search after sorting. However, since comparison sorting has a time complexity of $O(n \log n)$ which dominates the performance of entire algorithm (linear search is $O(n)$). Using a binary search ($O(\log n)$) instead of linear search won't change the asymptotic time complexity.

Also note that, we deduced the algorithm in descending for simplicity. Usually the sort function provided by default is in ascending order. The same principles applies to both ascending order and descending order. In the case of ascending order, we just scan it from backward.

Java

Copy

```java
public class Solution {
    public int hIndex(int[] citations) {
        // sorting the citations in ascending order
        Arrays.sort(citations);
        // finding h-index by linear search
        int i = 0;
        while (i < citations.length && citations[citations.length - 1 - i] > i) {
            i++;
        }
        return i; // after the while loop, i = i' + 1
    }
}
```

Complexity Analysis

- Time complexity : $O(n \log n)$. Comparison sorting dominates the time complexity.
- Space complexity : $O(1)$. Most libraries using `heap sort` which costs $O(1)$ extra space in the worst case.

Approach #2 (Counting) [Accepted]

Intuition

Comparison sorting algorithm has a lower bound of $O(n \log n)$. To achieve better performance, we need non-comparison based sorting algorithms.

Algorithm

From Approach #1, we sort the citations to find the h-index. However, it is well known that comparison sorting algorithms such as `heapsort`, `mergesort` and `quicksort` have a lower bound of $O(n \log n)$. The most commonly used non-comparison sorting is `counting sort`.

> Counting sort operates by counting the number of objects that have each distinct key value, and using arithmetic on those tallies to determine the positions of each key value in the output sequence. Its running time is linear in the number of items and the difference between the maximum and minimum keys, so it is only suitable for direct use in situations where the variation in keys is not significantly greater than the number of items.
>
> ---by Wikipedia

However, in our problem, the keys are the citations of each paper which can be much larger than the number of papers n. It seems that we cannot use `counting sort`. The trick here is the following observation:

> Any citation larger than n can be replaced by n and the h-index will not change after the replacement

The reason is that h-index is upper bounded by total number of papers n, i.e.

$$h \leq n$$

In the diagram, replacing citations greater than n with n is equivalent to cutting off the area where $y > n$.

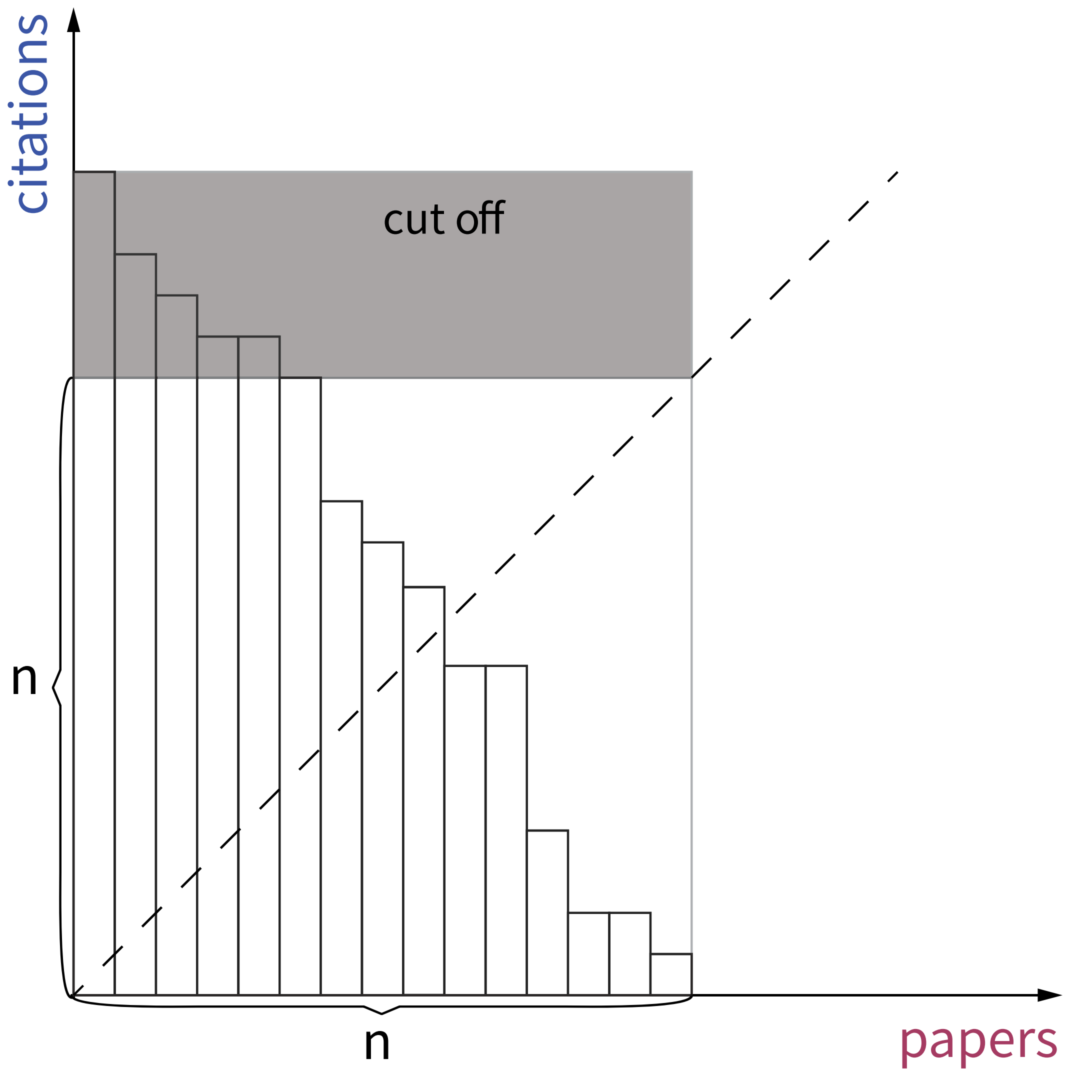


Figure 2. cutting off the area with citations more than n

Apparently, cutting that area off will not change the largest **square** and the h-index.

After we have the counts, we can get a sorted citations by traversing the counts array. And the rest is the same as Approach #1.

But we can do even better. The idea is that we don't even need to get sorted citations. We can find the h-index by using the paper counts directly.

To explain this, let's look at the following example:

$$\text{citations} = [1, 3, 2, 3, 100]$$

The counting results are:

k	**0**	**1**	**2**	**3**	**4**	**5**
count	0	1	1	2	0	1
s_k	5	5	4	3	1	1

The value s_k is defined as "the sum of all counts with citation $\geq k$" or "the number of papers having, at least, k citations". By definition of the h-index, the largest k with $k \leq s_k$ is our answer.

After replacing 100 with $n = 5$, we have $\text{citations} = [1, 3, 2, 3, 5]$. Now, we count the number of papers for each citation number 0 to 5. The counts are $[0, 1, 1, 2, 0, 1]$. The first k from right to left (5 down to 0) that have $k \leq s$ is the h-index 3.

Since we can calculate s_k on the fly when traverse the count array, we only need one pass through the count array which only costs $O(n)$ time.

Java   Copy

```java
public class Solution {
    public int hIndex(int[] citations) {
        int n = citations.length;
        int[] papers = new int[n + 1];
        // counting papers for each citation number
        for (int c: citations)
            papers[Math.min(n, c)]++;
        // finding the h-index
        int k = n;
        for (int s = papers[n]; k > s; s += papers[k])
            k--;
        return k;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. There are two steps. The counting part is $O(n)$ since we traverse the `citations` array once and only once. The second part of finding the h-index is also $O(n)$ since we traverse the `papers` array at most once. Thus, the entire algorithm is $O(n)$
- Space complexity : $O(n)$. We use $O(n)$ auxiliary space to store the counts.

Further Thoughts

> Is it possible to have multiple h-values?

The answer is **NO**. One can find this intuitively from Figure 1. The dashed line $y = x$ crosses the histogram once and only once, because the sorted bars are monotonic. It can also be proven from the definition of the h-index.

Rate this article:

Previous (/articles/contains-duplicate-iii/)

Next (/articles/summary-ranges/)

Comments: 12

Sort By

Type comment here... (Markdown is supported)

Preview   Post

(/yaopiupiupiu) yaopiupiupiu (/yaopiupiupiu) ★21 April 20, 2017 12:55 AM

Figure 1 is brilliant!

13 ∧ ∨ Share Reply

(/jaggi1234) jaggi1234 (/jaggi1234) ★19 December 3, 2019 11:39 PM

Can also be done by doing a binary search for the answer.
First step is sorting the array in O(nlogn).
Then start doing binary search:

Let low = 1 and high = n.

Read More

5 ∧ ∨ Share Reply

(/anmingyu11) anmingyu11 (/anmingyu11) ★419 June 6, 2019 11:49 AM

i wonder what this idoit question really mean for?

11 ∧ ∨ Share Reply

SHOW 1 REPLY

(/pachanta)

pachanta (/pachanta) ★3 March 17, 2018 5:24 AM

//SImple java solution (Accepted)

class Solution {
public int hIndex(int[] citations) {

Read More

2 ⌃ ⌄ Share Reply

(/tinapatil)

tinapatil (/tinapatil) ★9 September 24, 2019 10:03 AM

can someone help understand this

for (int s = papers[n]; k > s; s += papers[k])
k--;

Read More

1 ⌃ ⌄ Share Reply

SHOW 1 REPLY

(/nevsanev)

Nevsanev (/nevsanev) ★1096 April 11, 2019 5:31 AM

My Java sorting solution:

```
class Solution {
    public int hIndex(int[] citations) {
        int n = citations.length;
```

Read More

1 ⌃ ⌄ Share Reply

(/10000tb)

10000tb (/10000tb) ★398 May 11, 2018 3:25 PM

Definition define that there is one and only (In fact, for any valid h values, `1 ... h-1` are all valid, but definition of h-index demands a largest one, so only h is the valid `h-index`). No proof needed :).

0 ⌃ ⌄ Share Reply

(/jadenpan)

JadenPan (/jadenpan) ★260 March 21, 2017 11:58 PM

Nice and detailed explanation, I've learned a lot.

0 ⌃ ⌄ Share Reply

(/cissy27)

cissy27 (/cissy27) ★0 June 20, 2020 8:16 AM

for (int s = papers[n]; k > s; s += papers[k])
k--;

This look smart. But just wasting time of code readers.

0 ⌃ ⌄ Share Reply

(/newbiecoder1)

newbiecoder1 (/newbiecoder1) ★117 June 19, 2020 1:55 PM

I am confused about the definition of H-index provided in the problem description: "the other N − h papers have **no more than** h citations each."

I think it should be "the other N-h papers have **less than** h citations each."

0 ⌃ ⌄ Share Reply

‹ 1 2 ›