Finding out the minimum difference between elements in an array

Asked 9 years, 6 months ago Modified 2 years ago Viewed 77k times

30

13

I have an integer array with some finite number of values. My job is to find the minimum difference between any two elements in the array.

Consider that the array contains

```
4, 9, 1, 32, 13
```

Here the difference is minimum between 4 and 1 and so answer is 3.

What should be the algorithm to approach this problem. Also, I don't know why but I feel that using trees, this problem can be solved relatively easier. Can that be done?

algorithm

Share Follow

edited Oct 19, 2017 at 2:17

3 5

asked Sep 2, 2012 at 2:12

OneMoreError

7,012 17 67 107

3 en.wikipedia.org/wiki/Closest_pair_of_points_problem – Rsh Sep 2, 2012 at 6:30

2 You mean you are solving this codechef.com/SEP12/problems/HORSES – nikhil Sep 2, 2012 at 7:33

Yup.. I asked this question based on that !! – OneMoreError Sep 3, 2012 at 11:34

8 Answers

Sorted by: Highest Score (default)

47

The minimum difference will be one of the differences from among the consecutive pairs in sorted order. Sort the array, and go through the pairs of adjacent numbers looking for the smallest difference:

```
int[] a = new int[] {4, 9, 1, 32, 13};
Arrays.sort(a);
int minDiff = a[1]-a[0];
for (int i = 2 ; i != a.length ; i++) {
    minDiff = Math.min(minDiff, a[i]-a[i-1]);
}
System.out.println(minDiff);
```

This prints `3`.

Share Follow

answered Sep 2, 2012 at 2:17

Sergey Kalinichenko
693k 75 1043 1454

Okay.. I get what you are saying. But sorting itself will take O(n.log n) time. I'm just curious, but can we do without sorting !! – OneMoreError Sep 2, 2012 at 2:20

3 @CSSS if you do a radix sort it is *O(n)* – obataku Sep 2, 2012 at 2:23

@CSSS I don't think you can do it faster than `O(N*LogN)`. You have to go through elements of the array at least once, and for each element you'll need to find its best "counterpart" for subtraction. The best you can do there is `Log(N)` if you use a tree. – Sergey Kalinichenko Sep 2, 2012 at 2:24

5 @CSSS Walk the array, and build a binary search tree from its elements. Every time you add a node to your BST, check the difference between the newly added element and each of the nodes that you walk while finding the place of the new element in the tree. The counterpart with the smallest difference will be among one of these nodes. Inserting a node in a tree takes `Log(N)`, for a total of `O(N*Log(N))`. – Sergey Kalinichenko Sep 2, 2012 at 2:43

4 Yea, building a binary search tree is just another flavour of sorting in reality. – Stephen C Sep 2, 2012 at 3:58

13

You can take advantage of the fact that you are considering integers to make a linear algorithm:

1. First pass: compute the maximum and the minimum

2. Second pass: allocate a boolean array of length (max - min + 1), false initialized, and change the (value - min)th value to true for every value in the array
3. Third pass: compute the differences between the indexes of the true valued entries of the boolean array.

Share Follow

answered Aug 8, 2014 at 22:18

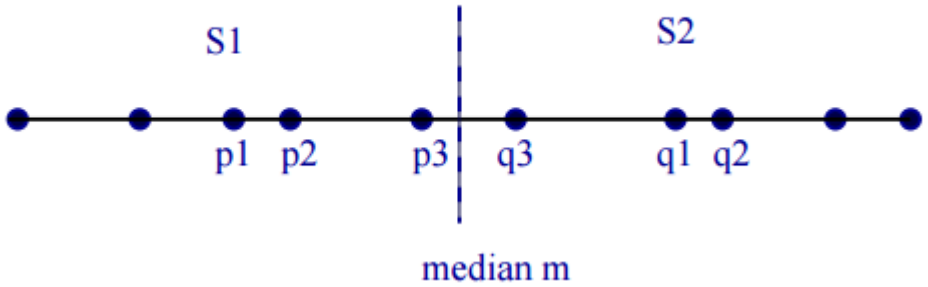
mookid
1,016 11 19

5 This is linear in `N`, but also gets a linear dependency on `max-min` which can make it pretty bad. – DarioP Nov 23, 2015 at 10:46

7

While all the answers are correct, I wanted to show the underlying algorithm responsible for `n log n` run time. The *divide and conquer* way of finding the minimum distance between the two points or finding the closest points in a 1-D plane.

The general algorithm:



- Let m = median(S).
- Divide S into S1, S2 at m.
- δ1 = Closest-Pair(S1).
- δ2 = Closest-Pair(S2).
- δ12 is minimum distance across the cut.
- Return δ = min(δ1, δ2, δ12).

Here is a sample I created in Javascript:

```
// Points in 1-D
var points = [4, 9, 1, 32, 13];

var smallestDiff;

function mergeSort(arr) {
  if (arr.length == 1)
    return arr;

  if (arr.length > 1) {
    let breakpoint = Math.ceil((arr.length / 2));
    // Left list starts with 0, breakpoint-1
    let leftList = arr.slice(0, breakpoint);
    // Right list starts with breakpoint, length-1
    let rightList = arr.slice(breakpoint, arr.length);

    // Make a recursive call
    leftList = mergeSort(leftList);
    rightList = mergeSort(rightList);

    var a = merge(leftList, rightList);
    return a;
  }
}

function merge(leftList, rightList) {
  let result = [];
  while (leftList.length && rightList.length) {
    // Sorting the x coordinates
    if (leftList[0] <= rightList[0]) {
      result.push(leftList.shift());
    } else {
      result.push(rightList.shift());
    }
  }

  while (leftList.length)
    result.push(leftList.shift());

  while (rightList.length)
    result.push(rightList.shift());

  let diff;
```

```
  if (result.length > 1) {
    diff = result[1] - result[0];
  } else {
    diff = result[0];
  }

  if (smallestDiff) {
    if (diff < smallestDiff)
      smallestDiff = diff;
  } else {
    smallestDiff = diff;
  }
  return result;
}

mergeSort(points);

console.log(`Smallest difference: ${smallestDiff}`);
```

Run code snippet Expand snippet

Share Follow

edited Mar 28, 2017 at 13:00

answered Mar 28, 2017 at 12:43

Suhail Gupta
20.6k 60 180 309

4

I would put them in a heap in `O(nlogn)` then pop one by one and get the minimum difference between every element that I pop. Finally I would have the minimum difference. However, there might be a better solution.

Share Follow

answered Sep 2, 2012 at 5:35

polerto
1,690 5 29 49

4

This is actually a restatement of the `closest-pair` problem in `one-dimension`.
https://en.wikipedia.org/wiki/Closest_pair_of_points_problem http://www.cs.umd.edu/~samir/grant/cp.pdf

As the Wikipedia article cited below points out, the best decision-tree model of this problem also runs in $\Omega(n \log n)$ time.

Share Follow

answered Jun 18, 2015 at 18:18

Saket Mittal
3,426 3 26 48

---

sharing the simplest solution.

2

```
function FindMin(arr) {

    //sort the array in increasing order

    arr.sort((a,b) => {
       return a-b;
    });

    let min = arr[1]-arr[0];

    let n = arr.length;

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

      let m = arr[i+1] - arr[i];
      if(m < min){
        m = min;
      }
    }

    return m; // minimum difference.
  }
```

Share Follow

edited Nov 27, 2019 at 8:44

greybeard
2,060 7 24 57

answered Oct 19, 2019 at 13:02

dinesh_malhotra
1,729 16 9

The given problem can easily be solved in O(n) time. Look at the following code that I wrote.

1

```
import java.util.Scanner;
public class Solution {
public static void main(String [] args) {
    Scanner input = new Scanner(System.in);
    int i, minDistance = 999999;
    boolean flag = false;
    int capacity = input.nextInt();
    int arr[] = new int[capacity];
    for (i = 0; i < capacity; i++) {
        arr[i] = input.nextInt();
    }

    int firstElement = input.nextInt();
    int secondElement = input.nextInt();

    int prev = 0;

    for (i = 0; i < capacity; i++) {
        if (arr[i] == firstElement || arr[i] == secondElement) {
            prev = i;
            break;
        }
    }

    for (; i < capacity; i++) {
        if(arr[i] == firstElement || arr[i] == secondElement) {
            if(arr[i] != arr[prev] && minDistance > Math.abs(i - prev)) {
                minDistance = Math.abs(i - prev);
                flag = true;
                prev = i;
            } else {
                prev = i;
            }
        }
    }

    if(flag)
        System.out.println(minDistance);
```

Share Follow

answered Feb 24, 2020 at 19:08

Pikachu

258 3 12

1 Your answer would have been much better if you had explained the logic behind your algorithm instead of providing the code without any

explanation. – Milos Mar 15 at 2:34

-1

In Python 3 this problem can be simplified by using the module **itertools** which gives the combinations available for a list. From that list we can find the sum of each combination and find the minimum of those values.

```
import itertools

arr = [4, 9, 1, 32, 13]

if len(arr) > 1:
    min_diff = abs(arr[0] - arr[1])
else:
    min_diff = 0

for n1, n2 in itertools.combinations(arr, 2): # Get the combinations of numbers
    diff = abs(n1-n2) # Find the absolute difference of each combination
    if min_diff > diff:
        min_diff = diff # Replace incase a least differnce found

print(min_diff)
```

Share Follow

edited Nov 27, 2019 at 9:04

answered Nov 26, 2019 at 17:43

Prakash S
444 4 9

1 While this might answer the question, a little explanation could improve the answer's long term value. – jrook Nov 26, 2019 at 19:32

@jrook Updated the answer to give an explanation. – Prakash S Nov 27, 2019 at 5:29

The comments mix `absolute difference` and `minimum of sum` . A list with about `len(arr)` ^{2} elements is constructed in a "non-pythonic" way. – greybeard Nov 27, 2019 at 8:00

@greybeard Edited the answer to avoid using list. What about this one ? – Prakash S Nov 27, 2019 at 8:59

1 What is the complexity? If you examine all combinations, then it is O(n^2), therefore much slower that other answers – Damien Nov 27, 2019 at 9:26

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect

this question from spam and non-answer activity.