Maximum adjacent difference in an array in its sorted form

Difficulty Level : Hard • Last Updated : 24 Jun, 2021

Given an array, find the maximum difference between its two consecutive elements in its sorted form.

Examples:

```
Input: arr[] = {1, 10, 5}
Output: 5
Sorted array would be {1, 5, 10} and
maximum adjacent difference would be
10 - 5 = 5

Input: arr[] = {2, 4, 8, 11}
Output: 4
```

Recommended Practice

Maximum Gap

Try It!

Naive Solution:

First sort the array, then traverse it and keep track of the maximum difference between adjacent elements.

Efficient Solution:

This **solution** is based on the idea of Pigeonhole sorting. No need to sort the array, just have to fill the buckets and keep track of the maximum and minimum value of each bucket. If found an empty bucket, The maximum gap would be the difference of **maximum value in the previous bucket – minimum value in the next bucket**.

As we want to almost sort these so that, we can have maximum gap. Also for any ith element, the value of (arr[i]-min_value)/(max_value-min_value) keeps increasing as arr[i] keeps increasing and this value always varies from 0 to 1. As we want to put the sorted results in bucket of size n. We multiply this value by (n-1) hence make a variable **delta = (max_value – min_value)/(n-1)**. Now in maxBucket or minBucket, all the value at any index j before index any i will always less than the value at index i, minBucket[j]<minBucket[i] for j<i. It is possible that two different arr[i], might have same value of **(arr[i]-min_value)/delta**, therefore we are making 2 different buckets maxBucket and minBucket.

As we have find the max difference between **consecutive values**, we must consider the max possible value upto to previous index as prev_val and the minBucket[i] for current index i, and ans will be max of ans and minBucket[i]-prev_val.

Let us solve the above example by this approach.

Working Example:

> ***Input***: *arr[] = {1, 10, 5}*
>
> ***Output: 5***
>
> ***Step1***: *Find max_val and min_val*

delta = (max_val – min_val)/(n-1)

delta = (10-1)/(3-1) = 4.5

***Step3**: Initialize buckets, maxBucket={INT_MIN}, minBucket={INT_MAX}*

***Step4**: For any index i, calculate index arr[i] in bucket and update in buckets,*

in = (arr[i]-min_val)/delta

maxBucket[in]=max(maxBucket[in],arr[i])

minBucket[in]=min(minBucket[in],arr[i])

for all index in arr in values are => 0,2,0

maxBucket=[5,INT_MIN,10]

minBucket=[1,INT_MAX,10]

***Step5**: Hence ans is max of minBucket[i]-(max of value upto previous index)*

in this case for i=2: max_gap = max(max_gap,minBucket[2] – max(maxBucket[1],maxBucket[0]))

max_gap = 10-5=5

This is just for presenting the concept, all other basic validations are in the main code.

Below is the code for the above approach:

C++

```
using namespace std;

int maxSortedAdjacentDiff(int* arr, int n)
{
    // Find maximum and minimum in arr[]
    int maxVal = arr[0], minVal = arr[0];
    for (int i = 1; i < n; i++) {
        maxVal = max(maxVal, arr[i]);
        minVal = min(minVal, arr[i]);
    }

    // Arrays to store maximum and minimum values
    // in n-1 buckets of differences.
    int maxBucket[n - 1];
    int minBucket[n - 1];
    fill_n(maxBucket, n - 1, INT_MIN);
    fill_n(minBucket, n - 1, INT_MAX);

    // Expected gap for every bucket.
    float delta = (float)(maxVal - minVal) / (float)(n - 1);

    // Traversing through array elements and
    // filling in appropriate bucket if bucket
    // is empty. Else updating bucket values.
    for (int i = 0; i < n; i++) {
        if (arr[i] == maxVal || arr[i] == minVal)
            continue;

        // Finding index of bucket.
        int index = (float)(floor(arr[i] - minVal) / delta);

        maxBucket[index] = max(maxBucket[index], arr[i]);
           minBucket[index] = min(minBucket[index], arr[i]);
    }

    // Finding maximum difference between maximum value
    // of previous bucket minus minimum of current bucket.
    int prev_val = minVal;
    int max_gap = 0;
    for (int i = 0; i < n - 1; i++) {
        if (minBucket[i] == INT_MAX)
            continue;
    max_gap = max(max_gap, maxVal - prev_val);

    return max_gap;
}

int main()
{
    int arr[] = { 1, 10, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxSortedAdjacentDiff(arr, n) << endl;
    return 0;
}
```

Java

```
// Java program for the above approach
import java.util.Arrays;

// Java program to find maximum adjacent difference
// between two adjacent after sorting.
class GFG {

    static int maxSortedAdjacentDiff(int[] arr, int n)
    {
        // Find maximum and minimum in arr[]
        int maxVal = arr[0];
        int minVal = arr[0];
        for (int i = 1; i < n; i++) {
            maxVal = Math.max(maxVal, arr[i]);
            minVal = Math.min(minVal, arr[i]);
        }

        // Arrays to store maximum and minimum values
        // in n-1 buckets of differences.
        int maxBucket[] = new int[n - 1];
        int minBucket[] = new int[n - 1];
        Arrays.fill(maxBucket, 0, n - 1, Integer.MIN_VALUE);
        Arrays.fill(minBucket, 0, n - 1, Integer.MAX_VALUE);

        // Expected gap for every bucket.
        float delta
```

```
// filling in appropriate bucket if bucket
// is empty. Else updating bucket values.
for (int i = 0; i < n; i++) {
    if (arr[i] == maxVal || arr[i] == minVal) {
        continue;
    }

    // Finding index of bucket.
    int index = (int)(Math.round((arr[i] - minVal)
                                 / delta));

    // Filling/Updating maximum value of bucket
    if (maxBucket[index] == Integer.MIN_VALUE) {
        maxBucket[index] = arr[i];
    }
    else {
        maxBucket[index]
            = Math.max(maxBucket[index], arr[i]);
    }

    // Filling/Updating minimum value of bucket
    if (minBucket[index] == Integer.MAX_VALUE) {
        minBucket[index] = arr[i];
    }
    else {
        minBucket[index]
            = Math.min(minBucket[index], arr[i]);
    }
}

// Finding maximum difference between maximum value
// of previous bucket minus minimum of current
// bucket.
int prev_val = minVal;
int max_gap = 0;
for (int i = 0; i < n - 1; i++) {
    if (minBucket[i] == Integer.MAX_VALUE) {
        continue;
    }
    max_gap = Math.max(max_gap,
                       minBucket[i] - prev_val);
    prev_val = maxBucket[i];
```

```
        return max_gap;
    }

    // Driver program to run the case
    public static void main(String[] args)
    {

        int arr[] = { 1, 10, 5 };
        int n = arr.length;
        System.out.println(maxSortedAdjacentDiff(arr, n));
    }
}
```

Python3

```
# Python3 program to find maximum adjacent
# difference between two adjacent after sorting.

def maxSortedAdjacentDiff(arr, n):

    # Find maximum and minimum in arr[]
    maxVal, minVal = arr[0], arr[0]
    for i in range(1, n):
        maxVal = max(maxVal, arr[i])
        minVal = min(minVal, arr[i])

    # Arrays to store maximum and minimum
    # values in n-1 buckets of differences.
    maxBucket = [INT_MIN] * (n - 1)
    minBucket = [INT_MAX] * (n - 1)

    # Expected gap for every bucket.
    delta = (maxVal - minVal) // (n - 1)

    # Traversing through array elements and
    # filling in appropriate bucket if bucket
    # is empty. Else updating bucket values.
    for i in range(0, n):
        if arr[i] == maxVal or arr[i] == minVal:
            continue
        # Filling/Updating maximum value
        # of bucket
        if maxBucket[index] == INT_MIN:
            maxBucket[index] = arr[i]
        else:
            maxBucket[index] = max(maxBucket[index],
                                             arr[i])

        # Filling/Updating minimum value of bucket
        if minBucket[index] == INT_MAX:
            minBucket[index] = arr[i]
        else:
            minBucket[index] = min(minBucket[index],
                                             arr[i])

    # Finding maximum difference between
    # maximum value of previous bucket
    # minus minimum of current bucket.
    prev_val, max_gap = minVal, 0

    for i in range(0, n - 1):
        if minBucket[i] == INT_MAX:
            continue

        max_gap = max(max_gap,
                      minBucket[i] - prev_val)
        prev_val = maxBucket[i]

    max_gap = max(max_gap, maxVal - prev_val)

    return max_gap

# Driver Code
if __name__ == "__main__":

    arr = [1, 10, 5]
    n = len(arr)
    INT_MIN, INT_MAX = float('-inf'), float('inf')

    print(maxSortedAdjacentDiff(arr, n))

# This code is contributed by Rituraj Jain
```

```
// C# program to find maximum
// adjacent difference between
// two adjacent after sorting.
using System;
using System.Linq;

class GFG
{
static int maxSortedAdjacentDiff(int[] arr,
                                int n)
{
    // Find maximum and minimum in arr[]
    int maxVal = arr[0];
    int minVal = arr[0];
    for (int i = 1; i < n; i++)
    {
        maxVal = Math.Max(maxVal, arr[i]);
        minVal = Math.Min(minVal, arr[i]);
    }

    // Arrays to store maximum and
    // minimum values in n-1 buckets
    // of differences.
    int []maxBucket = new int[n - 1];
    int []minBucket = new int[n - 1];
    maxBucket = maxBucket.Select(i => int.MinValue).ToArray();
    minBucket = minBucket.Select(i => int.MaxValue).ToArray();

    // maxBucket.Fill(int.MinValue);
    // Arrays.fill(minBucket, 0, n - 1, Integer.MAX_VALUE);

    // Expected gap for every bucket.
    float delta = (float) (maxVal - minVal) /
                  (float) (n - 1);

    // Traversing through array elements and
    // filling in appropriate bucket if bucket
    // is empty. Else updating bucket values.
    for (int i = 0; i < n; i++)
    {
        if (arr[i] == maxVal || arr[i] == minVal)
```

```
            // Finding index of bucket.
            int index = (int) (Math.Round((arr[i] -
                                 minVal) / delta));

            // Filling/Updating maximum value of bucket
            if (maxBucket[index] == int.MinValue)
            {
                maxBucket[index] = arr[i];
            }
            else
            {
                maxBucket[index] = Math.Max(maxBucket[index],
                                                      arr[i]);
            }

            // Filling/Updating minimum value of bucket
            if (minBucket[index] == int.MaxValue)
            {
                minBucket[index] = arr[i];
            }
            else
            {
                minBucket[index] = Math.Min(minBucket[index],
                                                      arr[i]);
            }
        }

        // Finding maximum difference between
        // maximum value of previous bucket
        // minus minimum of current bucket.
        int prev_val = minVal;
        int max_gap = 0;
        for (int i = 0; i < n - 1; i++)
        {
            if (minBucket[i] == int.MaxValue)
            {
                continue;
            }
            max_gap = Math.Max(max_gap, minBucket[i] -
                                        prev_val);
            prev_val = maxBucket[i];
```

```
    return max_gap;
}

// Driver Code
public static void Main()
{
    int []arr = {1, 10, 5};
    int n = arr.Length;
    Console.Write(maxSortedAdjacentDiff(arr, n));
}
}

// This code contributed by 29AjayKumar
```

Javascript

```
<script>

// JavaScript program to find maximum adjacent difference
// between two adjacent after sorting.

function maxSortedAdjacentDiff(arr, n)
{
    // Find maximum and minimum in arr[]
    var maxVal = arr[0], minVal = arr[0];
    for (var i = 1; i < n; i++) {
        maxVal = Math.max(maxVal, arr[i]);
        minVal = Math.min(minVal, arr[i]);
    }

    // Arrays to store maximum and minimum values
    // in n-1 buckets of differences.
    var maxBucket = Array(n-1).fill(-1000000000);
    var minBucket = Array(n-1).fill(1000000000);

    // Expected gap for every bucket.
    var delta = (maxVal - minVal) / (n - 1);

    // Traversing through array elements and
    // filling in appropriate bucket if bucket
    // is empty. Else updating bucket values
```

```
            continue;

        // Finding index of bucket.
        var index = Math.floor((arr[i] - minVal) / delta);

        maxBucket[index] = Math.max(maxBucket[index], arr[i]);
           minBucket[index] = Math.min(minBucket[index], arr[i]);
    }

    // Finding maximum difference between maximum value
    // of previous bucket minus minimum of current bucket.
    var prev_val = minVal;
    var max_gap = 0;
    for (var i = 0; i < n - 1; i++) {
        if (minBucket[i] == 1000000000)
            continue;
        max_gap = Math.max(max_gap, minBucket[i] - prev_val);
        prev_val = maxBucket[i];
    }
    max_gap = Math.max(max_gap, maxVal - prev_val);

    return max_gap;
}

var arr = [1, 10, 5];
var n = arr.length;
document.write( maxSortedAdjacentDiff(arr, n));


</script>
```

Output

```
5
```

Time complexity: O(n)

Auxiliary Space: O(n)

Like 30

Next

Pigeonhole Sort

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Check if two sorted arrays can be merged to form a sorted array with no adjacent pair from the same array**
16, Nov 20

02 **Maximum array elements that can be removed along with its adjacent values to empty given array**
05, Nov 20

03 **Count triplets from a sorted array having difference between adjacent elements equal to D**
12, May 21

04 **Generate longest possible array with product K such that**

05 **Maximum number of partitions that can be sorted individually to make sorted**
28, Feb 18

06 **Count number of common elements between a sorted array and a reverse sorted array**
12, May 21

07 **Circularly Sorted Array (Sorted and Rotated Array)**
01, Jun 22

12, Mar 21

10, Sep 20

Article Contributed By :

Aashish Chauhan
@Aashish Chauhan

Vote for difficulty

Current difficulty : Hard

Easy Normal Medium Hard Expert

Improved By : Rajput-Ji, 29AjayKumar, rituraj_jain, nehal sharma, 2701gouravgoel, rrrtnx, khushboogoyal499

Article Tags : Pigeonhole Principle, Arrays, Searching

Practice Tags : Arrays, Searching

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Careers

In Media

Contact Us

Privacy Policy

Copyright Policy

Learn

Algorithms

Data Structures

SDE Cheat Sheet

Machine learning

CS Subjects

Video Tutorials

Courses

News

Top News

Technology

Work & Career

Business

Finance

Lifestyle

Knowledge

Languages

Python

Java

CPP

Golang

C#

SQL

Kotlin

Web Development

Web Tutorials

Django Tutorial

Contribute

Write an Article

Improve an Article

NodeJS

Do Not Sell My Personal Information