Longest subarray with absolute difference between elements less than or equal to K using Heaps

Difficulty Level : Hard • Last Updated : 28 Sep, 2021

Given an **array arr[]** of **N** integers and an integer **K**, our task is to find the length of the longest subarray such that for all possible pairs in the subarray absolute difference between elements is less than or equal to K.

Examples:

> ***Input:*** *arr[] = {2, 4, 5, 5, 5, 3, 1}, K = 0*
> ***Output:*** *3*
> ***Explanation:***
> *The possible subarray with difference in elements as 0 is {5, 5, 5} whose length is 3. Hence the output is 3.*
>
> ***Input:*** *arr[] = {1, 2, 3, 6, 7}, K = 2*
> ***Output:*** *3*
> ***Explanation:***
> *The possible subarray with difference in elements at most 2 is {1, 2, 3} whose length is 3. Hence the output is 3.*

Recommended: Please try your approach on {IDE} first, before moving on to the solution.

Naive Approach:
To solve the problem mentioned above the naive method is to use The Brute Force approach that is to generate all the possible subarray of the given array and check if the difference between the maximum and minimum element of the subarray is at most **K** or not. If it is, then update the length of the current subarray with the maximum length. Print the maximum length of the subarray after all the operations.

Below is the implementation of the above approach:

C++

```
// C++ implementation to find the Longest subarray
// of the given array with absolute difference between
// elements less than or equal to integer K
#include <bits/stdc++.h>
using namespace std;

int computeLongestSubarray(int arr[], int k, int n)
{

    // maxLength is 1 because k >= 0,
    // a single element, subarray will always
    // have absolute difference zero
    int maxLength = 1;

    // Check for all possible subarrays
    for(int i = 0; i < n; i++)
    {

        // Initialization of minimum &
        // maximum of current subarray
        int minOfSub = arr[i];
```

```
            {

                // Update the values for minimum & maximum
                if (arr[j] > maxOfSub)
                    maxOfSub = arr[j];

                if (arr[j] < minOfSub)
                    minOfSub = arr[j];

                // Check if current subarray satisfies
                // the given condition
                if ((maxOfSub - minOfSub) <= k)
                {
                    int currLength = j - i + 1;

                    // Update the value for maxLength
                    if (maxLength < currLength)
                        maxLength = currLength;
                }
            }
        }

        // Return the final result
        return maxLength;
    }

    // Driver Code
    int main()
    {
        int arr[] = { 1, 2, 3, 6, 7 };

        int k = 2;
        int n = sizeof(arr) / sizeof(arr[0]);

        int maxLength = computeLongestSubarray(arr, k, n);

        cout << (maxLength);
    }

    // This code is contributed by chitranayal
```

```
// Java implementation to find the Longest subarray
// of the given array with absolute difference between
// elements less than or equal to integer K

class GFG {
    public static int computeLongestSubarray(int arr[],
                                              int k)
    {
        // maxLength is 1 because k >= 0,
        // a single element, subarray will always
        // have absolute difference zero
        int maxLength = 1;

        // Check for all possible subarrays
        for (int i = 0; i < arr.length; i++) {
            // Initialization of minimum &
            // maximum of current subarray
            int minOfSub = arr[i];
            int maxOfSub = arr[i];

            for (int j = i + 1; j < arr.length; j++) {

                // Update the values for minimum & maximum
                if (arr[j] > maxOfSub)
                    maxOfSub = arr[j];

                if (arr[j] < minOfSub)
                    minOfSub = arr[j];

                // Check if current subarray satisfies
                // the given condition
                if ((maxOfSub - minOfSub) <= k) {
                    int currLength = j - i + 1;

                    // Update the value for maxLength
                    if (maxLength < currLength)
                        maxLength = currLength;
                }
            }
        }

        // Return the final result
    // Driver Code
    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 3, 6, 7 };

        int k = 2;

        int maxLength = computeLongestSubarray(arr, k);
        System.out.println(maxLength);
    }
}
```

Python3

```
# Python3 implementation to find the
# Longest subarray of the given array
# with absolute difference between
# elements less than or equal to integer K
def computeLongestSubarray (arr, k, n):

    # maxLength is 1 because k >= 0,
    # a single element, subarray will always
    # have absolute difference zero
    maxLength = 1

    # Check for all possible subarrays
    for i in range(n):

        # Initialization of minimum &
        # maximum of current subarray
        minOfSub = arr[i]
        maxOfSub = arr[i]

        for j in range(i + 1, n):

            # Update the values for
            # minimum & maximum
            if (arr[j] > maxOfSub):
                maxOfSub = arr[j]

            if (arr[j] < minOfSub):
```

```
            if ((maxOfSub - minOfSub) <= k):
                currLength = j - i + 1

                # Update the value for maxLength
                if (maxLength < currLength):
                    maxLength = currLength

    # Return the final result
    return maxLength

# Driver Code
if __name__ == '__main__':

    arr = [ 1, 2, 3, 6, 7 ]
    k = 2
    n = len(arr)

    maxLength = computeLongestSubarray(arr, k, n)

    print(maxLength)

# This code is contributed by himanshu77
```

C#

```
// C# implementation to find the longest subarray
// of the given array with absolute difference between
// elements less than or equal to integer K
using System;
class GFG
{
    public static int computelongestSubarray(int []arr,
                                             int k)
    {

        // maxLength is 1 because k >= 0,
        // a single element, subarray will always
        // have absolute difference zero
        int maxLength = 1;
```

```
            // Initialization of minimum &
            // maximum of current subarray
            int minOfSub = arr[i];
            int maxOfSub = arr[i];

            for (int j = i + 1; j < arr.Length; j++)
            {

                // Update the values for minimum & maximum
                if (arr[j] > maxOfSub)
                    maxOfSub = arr[j];

                if (arr[j] < minOfSub)
                    minOfSub = arr[j];

                // Check if current subarray satisfies
                // the given condition
                if ((maxOfSub - minOfSub) <= k)
                {
                    int currLength = j - i + 1;

                    // Update the value for maxLength
                    if (maxLength < currLength)
                        maxLength = currLength;
                }
            }
        }

        // Return the readonly result
        return maxLength;
    }

    // Driver Code
    public static void Main(String[] args)
    {
        int []arr = { 1, 2, 3, 6, 7 };
        int k = 2;
        int maxLength = computelongestSubarray(arr, k);
        Console.WriteLine(maxLength);
    }
}
```

Javascript

```
<script>

// JavaScript implementation to find the Longest subarray
// of the given array with absolute difference between
// elements less than or equal to integer K

function computeLongestSubarray(arr,k)
{
    // maxLength is 1 because k >= 0,
        // a single element, subarray will always
        // have absolute difference zero
        let maxLength = 1;

        // Check for all possible subarrays
        for (let i = 0; i < arr.length; i++) {
            // Initialization of minimum &
            // maximum of current subarray
            let minOfSub = arr[i];
            let maxOfSub = arr[i];

            for (let j = i + 1; j < arr.length; j++) {

                // Update the values for minimum & maximum
                if (arr[j] > maxOfSub)
                    maxOfSub = arr[j];

                if (arr[j] < minOfSub)
                    minOfSub = arr[j];

                // Check if current subarray satisfies
                // the given condition
                if ((maxOfSub - minOfSub) <= k) {
                    let currLength = j - i + 1;

                    // Update the value for maxLength
                    if (maxLength < currLength)
                        maxLength = currLength;
                }
        return maxLength;
}

 // Driver Code
let arr=[1, 2, 3, 6, 7];
let  k = 2;
let maxLength = computeLongestSubarray(arr, k);
document.write(maxLength);


// This code is contributed by avanitrachhadiya2155

</script>
```

Output:

```
3
```

Time Complexity: O(n^2)

Efficient Approach:
To optimize the above approach the idea is to use Heap Data Structure. Initialize a **minHeap** that will store the indices of the current subarray such that the elements are in ascending order, where the smallest appears at the top and a **maxHeap** that will store the indices of the current subarray such that the elements are in descending order, where the largest element appears at the top. Then iterate over the entire array and for each iteration check if:

- All the subarray elements satisfy the condition of **maxOfSub-minOfSub <= k**, then we **compare maxLength** so far to the length of current subarray and update maxLength to maximum of either maxLength or current subarray length.

which are not included in the new subarray.

- After each iteration, we increase our subarray length by incrementing the end pointer.

Below is the implementation of the above approach:

Java

```java
// Java implementation to find the Longest
// subarray of the given array with absolute
// difference between elements less than or equal
// to integer K using Heaps
import java.util.*;

class GFG {
    public static int computeLongestSubarray(int arr[],
                                             int k)
    {
        // Stores the maximum length subarray so far
        int maxLength = 0;

        Deque<Integer> maxHeap = new LinkedList<>();
        Deque<Integer> minHeap = new LinkedList<>();

        // Marks to the beginning and end
        // pointer for current subarray
        int beg = 0, end = 0;

        while (end < arr.length) {

            // Stores the current element being
            // added to the subarray
            int currEl = arr[end];

            // Remove indices of all elements smaller
            // than or equal to current from maxHeap
            while (maxHeap.size() > 0 &&
                       arr[maxHeap.peekLast()] <= currEl)
                maxHeap.removeLast();
```

```
// than or equal to current from minHeap
while (minHeap.size() > 0 &&
           arr[minHeap.peekLast()] >= currEl)
    minHeap.removeLast();

// Add current element's index to minHeap
minHeap.addLast(end);

// Index of maximum of current subarray
int maxOfSub = arr[maxHeap.peekFirst()];

// Index of minimum of current subarray
int minOfSub = arr[minHeap.peekFirst()];

// check if the largest possible difference
// between a pair of elements <= k
if (maxOfSub - minOfSub <= k) {
    // Length of current subarray
    int currLength = end - beg + 1;

    // Update maxLength
    if (maxLength < currLength)
        maxLength = currLength;
}

else {
    // If current subarray doesn't satisfy
    // the condition then remove the starting
    // element from subarray that satisfy
    // increment the beginning pointer
    beg++;

    // Remove elements from heaps that
    // are not in the subarray anymore
    while (minHeap.size() > 0 &&
                   minHeap.peekFirst() < beg)
        minHeap.removeFirst();

    while (maxHeap.size() > 0 &&
                   maxHeap.peekFirst() < beg)
        maxHeap.removeFirst();
}
```

```
        // Return the final answer
        return maxLength;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 3, 6, 7 };

        int k = 2;

        int maxLength = computeLongestSubarray(arr, k);
        System.out.println(maxLength);
    }
}
```

Python3

```
# Python3 implementation to find the Longest
# subarray of the given array with absolute
# difference between elements less than or equal
# to integer K using Heaps
from collections import deque

def computeLongestSubarray(arr, k):
    # Stores the maximum length subarray so far
    maxLength = 0

    maxHeap = []
    minHeap = []

    # Marks to the beginning and end
    # pointer for current subarray
    beg = 0
    end = 0

    while (end < len(arr)):
        # print(end)
        # Remove indices of all elements smaller
        # than or equal to current from maxHeap
        while (len(maxHeap) > 0 and arr[maxHeap[-1]] <= currEl):
            del maxHeap[-1]

        # Add current element's index to maxHeap
        maxHeap.append(end)

        # Remove indices of all elements larger
        # than or equal to current from minHeap
        while (len(minHeap) > 0 and arr[minHeap[-1]] >= currEl):

            # print(minHeap[-1])
            del minHeap[-1]

        # Add current element's index to minHeap
        minHeap.append(end)

        # Index of maximum of current subarray
        maxOfSub = arr[maxHeap[0]]

        # Index of minimum of current subarray
        minOfSub = arr[minHeap[0]]

        # check if the largest possible difference
        # between a pair of elements <= k
        if (maxOfSub - minOfSub <= k):

            # Length of current subarray
            currLength = end - beg + 1

            # Update maxLength
            if (maxLength < currLength):
                maxLength = currLength
        else:
            # If current subarray doesn't satisfy
            # the condition then remove the starting
            # element from subarray that satisfy
            # increment the beginning pointer
            beg += 1

            # Remove elements from heaps that
            while (len(maxHeap) > 0 and maxHeap[0] < beg):
                del maxHeap[0]

        end += 1

    # Return the final answer
    return maxLength

    # Driver code
if __name__ == '__main__':
    arr = [1, 2, 3, 6, 7]

    k = 2

    maxLength = computeLongestSubarray(arr, k)
    print(maxLength)

# This code is contributed by mohit kumar 29
```

Javascript

```
<script>

// JavaScript implementation to find the Longest
// subarray of the given array with absolute
// difference between elements less than or equal
// to integer K using Heaps

function computeLongestSubarray(arr,k)
{
    // Stores the maximum length subarray so far
        let maxLength = 0;

        let maxHeap = [];
        let minHeap = [];

        // Marks to the beginning and end
        // pointer for current subarray
        let beg = 0, end = 0;
        // added to the subarray
        let currEl = arr[end];

        // Remove indices of all elements smaller
        // than or equal to current from maxHeap
        while (maxHeap.length > 0 &&
                   arr[maxHeap[maxHeap.length-1]] <= currEl)
            maxHeap.pop();

        // Add current element's index to maxHeap
        maxHeap.push(end);

        // Remove indices of all elements larger
        // than or equal to current from minHeap
        while (minHeap.length > 0 &&
                   arr[minHeap[minHeap.length-1]] >= currEl)
            minHeap.pop();

        // Add current element's index to minHeap
        minHeap.push(end);

        // Index of maximum of current subarray
        let maxOfSub = arr[maxHeap[0]];

        // Index of minimum of current subarray
        let minOfSub = arr[minHeap[0]];

        // check if the largest possible difference
        // between a pair of elements <= k
        if (maxOfSub - minOfSub <= k) {
            // Length of current subarray
            let currLength = end - beg + 1;

            // Update maxLength
            if (maxLength < currLength)
                maxLength = currLength;
        }

        else {
            // If current subarray doesn't satisfy
            // the condition then remove the starting
            // element from subarray that satisfy
```

```
                // Remove elements from heaps that
                // are not in the subarray anymore
                while (minHeap.length > 0 &&
                               minHeap[0] < beg)
                    minHeap.shift();

                while (maxHeap.length > 0 &&
                               maxHeap[0] < beg)
                    maxHeap.shift();
            }

            end++;
        }

        // Return the final answer
        return maxLength;
}

// Driver code

let arr=[ 1, 2, 3, 6, 7 ];
let  k = 2;
let maxLength = computeLongestSubarray(arr, k);
document.write(maxLength);


// This code is contributed by rag2127

</script>
```

Output:

```
3
```

Time Complexity: O(n) because every element of the array is added and removed from the heaps only once.

Like 3

Previous

Longest subarray in which absolute difference between any two element is not greater than X

Next

Maximum length subarray with difference between adjacent elements as either 0 or 1

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Length of longest subarray in which elements greater than K are more than elements not greater than K**
14, Aug 19

02 **Find maximum number of elements such that their absolute difference is less than or equal to 1**
30, Oct 18

05 **Remove Minimum coins such that absolute difference between any two piles is less than K**
10, Jul 19

06 **Number of elements less than or equal to a number in a subarray : MO's Algorithm**
30, Apr 20

any two element is not greater than X
06, May 20

04 **Longest non-decreasing subsequence having difference between adjacent elements less than D**
19, Oct 21

Sorted Array
15, Jun 21

08 **Maximum sum subarray having sum less than or equal to given sum using Set**
22, Apr 20

Article Contributed By :

shradha_k
@shradha_k

Vote for difficulty

Current difficulty : Hard

Easy Normal Medium Hard Expert

Improved By : ukasp, himanshu77, mohit kumar 29, shikhasingrajput, avdnitrachhadiya2155, rag2127, varshagumber28, simmytarika5

Article Tags : subarray, Arrays, Competitive Programming, Heap

Practice Tags : Arrays, Heap

Improve Article Report Issue

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

Load Comments