Minimize the maximum difference between the heights

Given heights of n towers and a value k. We need to either increase or decrease height of every tower by k (only once) where k > 0. The task is to minimize the difference between the heights of the longest and the shortest tower after modifications, and output this difference.

Examples:

```
Input  : arr[] = {1, 15, 10}, k = 6
Output : arr[] = {7, 9, 4}
         Maximum difference is 5.
Explanation : We change 1 to 6, 15 to
9 and 10 to 4. Maximum difference is 5
(between 4 and 9). We can't get a lower
difference.

Input : arr[] = {1, 5, 15, 10}
        k = 3
Output : arr[] = {4, 8, 12, 7}
Maximum difference is 8

Input : arr[] = {4, 6}
        k = 10
Output : arr[] = {14, 16} OR {-6, -4}
Maximum difference is 2

Input : arr[] = {6, 10}
        k = 3
Output : arr[] = {9, 7}
Maximum difference is 2

Input : arr[] = {1, 10, 14, 14, 14, 15}
        k = 6
Output: arr[] = {7, 4, 8, 8, 8, 9}
Maximum difference is 5

Input : arr[] = {1, 2, 3}
        k = 2
Output: arr[] = {3, 4, 5}
Maximum difference is 2
```

Source : Adobe Interview Experience | Set 24 (On-Campus for MTS)

The idea is to sort all elements increasing order. Below are steps.

1. Sort array in increasing order
2. Initialize maximum and minimum elements.
   maxe = arr[n-1], mine = arr[0]
3. If k is more than difference between maximum and minimum, add/subtract k to all elements as k cannot decrease the difference. Example {6, 4}, k = 10.
4. In sorted array, update first and last elements.
   arr[0] += k; // arr[0] is minimum and k is +ve
   arr[n-1] -= k; // arr[n-1] is maximum and k is -ve
5. Initialize mac and min of modified array (only two elements have been finalized)
   new_max = max(arr[0], arr[n-1]), new_min = min(arr[0], arr[n-1])
6. Finalize middle n-2 elements. Do following for every element arr[j] where j lies from 1 to n-2.
   - If current element is less than min of modified array, add k.
   - Else If current element is more than max of modified array, subtract k.
   - arr[j] is between new_min and new_max.
     1. If arr[j] is closer to new_max, subtract k
     2. Else add k to arr[j].

7. Update new_max and new_min if required
   new_max = max(arr[j], new_max), new_min = min(arr[j], new_min)
8. Returns difference between new_max and new_min
   return (new_max – new_min);

Below is C++ implementation of above idea.

```
// C++ program to find the minimum possible
// difference between maximum and minimum
// elements when we have to add/subtract
// every number by k
#include <bits/stdc++.h>
using namespace std;

// Modifies the array by subtracting/adding
// k to every element such that the difference
// between maximum and minimum is minimized
int getMinDiff(int arr[], int n, int k)
{
    // There should be at least two elements
    if (n <= 1)
        return 0;

    // Sort array in increasing order
    sort(arr, arr+n);

    // Initialize maximum and minimum
    int maxe = arr[n-1];
    int mine = arr[0];

    // If k is more than difference between maximum
    // and minimum, add/subtract k to all elements
    // as k cannot decrease the difference
    if (k >= maxe - mine)
    {
        for (int i=0; i<n; i++)
            arr[i] += k; // Subtract would also work
        return (maxe - mine);
    }

    // In sorted array, first element is minimum
    // and last is maximum, we must add k to minium
    // and subtract k from maximum
    arr[0]   += k;
    arr[n-1] -= k;

    // Initialize mac and min of modified array (only
    // two elements have been finalized)
    int new_max = max(arr[0], arr[n-1]);
    int new_min = min(arr[0], arr[n-1]);

    // Finalize middle n-2 elements
    for (int j=1; j<n-1; j++)
    {
        // If current element is less than min of
        // modified array, add k.
        if (arr[j] < new_min)
            arr[j] += k;

        // If current element is more than max of
        // modified array, subtract k.
        else if (arr[j] > new_max)
            arr[j] -= k;

        // arr[j] is between new_min and new_max

        // If arr[j] is closer to new_max, subtract k
        else if ((arr[j] - new_min) > (new_max - arr[j]))
            arr[j] -= k;

        // Else add k
```

```
        else
            arr[j] += k;

        // Update new_max and new_min if required
        new_max = max(arr[j], new_max);
        new_min = min(arr[j], new_min);
    }

    // Returns difference between new_max and new_min
    return (new_max - new_min);
}

// Driver function to test the above function
int main()
{
    int arr[] = {1, 15, 10} ;
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 6;
    int res = getMinDiff(arr, n, k);

    cout << "Modified array is \n";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";

    cout << "\nMaximum difference is " << res;
    return 0;
}
```

Output :

```
Modified array is
7 9 4
Maximum difference is 5
```

Time Complexity : O(n Log n)