Maximum Gap

Given an unsorted array, find the maximum difference between the successive elements in its sorted form.

Try to solve it in linear time/space.

Return 0 if the array contains less than 2 elements.

You may assume all elements in the array are non-negative integers and fit in the 32-bit signed integer range.

Credits:
Special thanks to @porker2008 for adding this problem and creating all test cases.

Solution 1

Suppose there are N elements in the array, the min value is***min*** and the max value is ***max***. Then the maximum gap will be no smaller than ceiling[(***max*** - ***min***) / (N - 1)].

Let gap = ceiling[(***max*** - ***min***) / (N - 1)]. We divide all numbers in the array into n-1 buckets, where k-th bucket contains all numbers in [***min*** + (k-1)gap, ***min*** + k*gap). Since there are n-2 numbers that are not equal ***min*** or ***max*** and there are n-1 buckets, at least one of the buckets are empty. We only need to store the largest number and the smallest number in each bucket.

After we put all the numbers into the buckets. We can scan the buckets sequentially and get the max gap. my blog for this problem

```
public class Solution {
public int maximumGap(int[] num) {
    if (num == null || num.length < 2)
        return 0;
    // get the max and min value of the array
    int min = num[0];
    int max = num[0];
    for (int i:num) {
        min = Math.min(min, i);
        max = Math.max(max, i);
    }
    // the minimum possibale gap, ceiling of the integer division
    int gap = (int)Math.ceil((double)(max - min)/(num.length - 1));
    int[] bucketsMIN = new int[num.length - 1]; // store the min value in that bu
cket
    int[] bucketsMAX = new int[num.length - 1]; // store the max value in that bu
cket
    Arrays.fill(bucketsMIN, Integer.MAX_VALUE);
    Arrays.fill(bucketsMAX, Integer.MIN_VALUE);
    // put numbers into buckets
    for (int i:num) {
        if (i == min || i == max)
            continue;
        int idx = (i - min) / gap; // index of the right position in the buckets
        bucketsMIN[idx] = Math.min(i, bucketsMIN[idx]);
        bucketsMAX[idx] = Math.max(i, bucketsMAX[idx]);
    }
    // scan the buckets for the max gap
    int maxGap = Integer.MIN_VALUE;
    int previous = min;
    for (int i = 0; i < num.length - 1; i++) {
        if (bucketsMIN[i] == Integer.MAX_VALUE && bucketsMAX[i] == Integer.MIN_VA
LUE)
            // empty bucket
            continue;
        // min value minus the previous value is the current gap
        maxGap = Math.max(maxGap, bucketsMIN[i] - previous);
        // update previous bucket value
        previous = bucketsMAX[i];
    }
    maxGap = Math.max(maxGap, max - previous); // updata the final max value gap
    return maxGap;
}
```

}

written by zkfairytale original link here

Solution 2

You can look at radix sort visualization here before reading the code: https://www.cs.usfca.edu/~galles/visualization/RadixSort.html

```java
public class Solution {
public int maximumGap(int[] nums) {
    if (nums == null || nums.length < 2) {
        return 0;
    }

    // m is the maximal number in nums
    int m = nums[0];
    for (int i = 1; i < nums.length; i++) {
        m = Math.max(m, nums[i]);
    }

    int exp = 1; // 1, 10, 100, 1000 ...
    int R = 10; // 10 digits

    int[] aux = new int[nums.length];

    while (m / exp > 0) { // Go through all digits from LSB to MSB
        int[] count = new int[R];

        for (int i = 0; i < nums.length; i++) {
            count[(nums[i] / exp) % 10]++;
        }

        for (int i = 1; i < count.length; i++) {
            count[i] += count[i - 1];
        }

        for (int i = nums.length - 1; i >= 0; i--) {
            aux[--count[(nums[i] / exp) % 10]] = nums[i];
        }

        for (int i = 0; i < nums.length; i++) {
            nums[i] = aux[i];
        }
        exp *= 10;
    }

    int max = 0;
    for (int i = 1; i < aux.length; i++) {
        max = Math.max(max, aux[i] - aux[i - 1]);
    }

    return max;
}
```

}

1. The first step is to find the maximum value in nums array, it will be the threshold to end while loop.

2. Then use the radix sort algorithm to sort based on each digit from Least Significant Bit (LSB) to Most Significant Bit (MSB), that's exactly what's showing in the link.
3. `(nums[i] / exp) % 10` is used to get the digit, for each digit, basically the digit itself serves as the index to access the count array. Count array stores the index to access aux array which stores the numbers after sorting based on the current digit.
4. Finally, find the maximum gap from sorted array.

Time and space complexities are both O(n). (Actually time is O(10n) at worst case for Integer.MAX_VALUE 2147483647)

written by Alexpanda original link here

Solution 3

Suppose you have n pigeons with labels and you put them into m holes based on their label with each hole of the same size. Why bother putting pigeons into holes? Because you want to disregard the distance between pigeons **within** each one hole.

Only when at least one hole is empty can we disregard the distance between pigeons within each one hole and compute the maximum gap solely by the distance between pigeons **in adjacent holes**. We make sure that at least one hole is empty by using m=n-1 (i.e. n-2 pigeons in n-1 holes => at least one hole is empty).

```
int maximumGap(vector<int>& nums) {
        const int n = nums.size();
        if(n<=1) return 0;
        int maxE = *max_element(nums.begin(),nums.end());
        int minE = *min_element(nums.begin(),nums.end());
        double len = double(maxE-minE)/double(n-1);
        vector<int> maxA(n,INT_MIN);
        vector<int> minA(n,INT_MAX);
        for(int i=0; i<n; i++) {
            int index = (nums[i]-minE)/len;
            maxA[index] = max(maxA[index],nums[i]);
            minA[index] = min(minA[index],nums[i]);
        }
        int gap = 0, prev = maxA[0];
        for(int i=1; i<n; i++) {
            if(minA[i]==INT_MAX) continue;
            gap = max(gap,minA[i]-prev);
            prev = maxA[i];
        }
        return gap;
    }
```

written by morrischen2008 original link here