POTW Gold 9.16.19

Analysis by Aaron Truong

Initially, it may seem like a greedy solution is viable. However, when considering the case ...a, b... where a > b, it is not immediately obvious whether it is more efficient to lower a and necessary preceding values or increase b and necessary subsequent values.

Thus, the solution is dynamic programming. Let a be the array of interest and dp(i, j) be the minimum cost to make a[0]...a[i] nondecreasing such that a[i] <= j.

It follows that dp(i, j) = min(dp(i, j-1), dp(i-1, j) + |j-a[i]|) with the solution being dp(n-1, a\_max).

This runs in O(nm), where m is the range of the array. However, this can be optimized to O(n2) by noting that every element always takes on an existing element in the array after an optimal change. Thus, instead of looping through the entire range, we can simply sort the array in ascending order (denoted by a\_s) and iterate j through this new array.

It now follows that dp(i, j) = min(dp(i, j-1), dp(i-1, j) + |a\_s[j]-a[i]|) with the solution being dp(n-1, n-1).

// C++ implementation

#include <cstdio>

#include <algorithm>

using namespace std;

int n;

long long a[10000], a\_s[10000], dp[10001][10000];

int main() {

scanf("%d", &n);

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

scanf("%lld", &a[i]);

a\_s[i]=a[i];

}

sort(a\_s, a\_s+n);

for(int i=1; i<=n; i++) {

dp[i][0]=dp[i-1][0]+abs(a\_s[0]-a[i-1]);

for(int j=1; j<n; j++) {

dp[i][j]=min(abs(a\_s[j]-a[i-1])+dp[i-1][j], dp[i][j-1]);

}

}

printf("%lld\n", dp[n][n-1]);

}

// Java implementation

import java.io.\*;

import java.util.\*;

public class ClassName {

static int n;

static int a[] = new int[10000], a\_s[] = new int[10000], dp[][] = new int[10001][10000];

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

StringTokenizer tk = new StringTokenizer(br.readLine());

n = Integer.parseInt(tk.nextToken());

tk = new StringTokenizer(br.readLine());

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

a[i] = Integer.parseInt(tk.nextToken());

a\_s[i] = a[i];

}

Arrays.sort(a\_s, 0, n);

for(int i = 1; i <= n; i++) {

dp[i][0] = dp[i - 1][0] + Math.abs(a\_s[0] - a[i - 1]);

for(int j = 1; j < n; j++) {

dp[i][j] = Math.min(Math.abs(a\_s[j] - a[i - 1]) + dp[i - 1][j], dp[i][j - 1]);

}

}

System.out.println(dp[n][n - 1]);

}

}