Minimum Moves to Equal Array Elements II

Given a **non-empty** integer array, find the minimum number of moves required to make all array elements equal, where a move is incrementing a selected element by 1 or decrementing a selected element by 1.

You may assume the array's length is at most 10,000.

Example:

```
Input:
[1,2,3]

Output:
2

Explanation:
Only two moves are needed (remember each move increments or decrements one element)
:

[1,2,3]  =>  [2,2,3]  =>  [2,2,2]
```

Solution 1

```
public class Solution {
    public int minMoves2(int[] nums) {
        Arrays.sort(nums);
        int i = 0, j = nums.length-1;
        int count = 0;
        while(i < j){
            count += nums[j]-nums[i];
            i++;
            j--;
        }
        return count;
    }
}
```

written by chnsht original link here

Solution 2

This solution relies on the fact that if we increment/decrement each element to the median of all the elements, the optimal number of moves is necessary. The median of all elements can be found in expected O(n) time using QuickSelect (or O(n) time using deterministic select).

'''

```
public int minMoves2(int[] nums) {
    int sum = 0;
    int median = findMedian(nums);
    for (int i=0;i<nums.length;i++) {
        sum += Math.abs(nums[i] - median);
    }
    return sum;
}

public int findMedian(int[] nums) {
    return getKth(nums.length/2+1, nums, 0, nums.length - 1);
}

public int getKth(int k, int[] nums, int start, int end) {
    int pivot = nums[end];
    int left = start;
    int right = end;

    while (true) {
        while (nums[left] < pivot && left < right) left++;
        while (nums[right] >= pivot && right > left) right--;
        if (left == right) break;
        swap(nums, left, right);
    }

    swap(nums, left, end);
    if (k == left + 1)  return pivot;
    else if (k < left + 1) return getKth(k, nums, start, left - 1);
    else return getKth(k, nums, left + 1, end);
}

public void swap(int[] nums, int n1, int n2) {
 int tmp = nums[n1];
 nums[n1] = nums[n2];
 nums[n2] = tmp;
}
```

'''

written by compton_scatter original link here

Solution 3

```
def minMoves2(self, nums):
    median = sorted(nums)[len(nums) / 2]
    return sum(abs(num - median) for num in nums)

def minMoves2(self, nums):
    nums.sort()
    return sum(nums[~i] - nums[i] for i in range(len(nums) / 2))
```

written by StefanPochmann original link here