Minimum Moves to Equal Array Elements

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

Example:

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

Output:
3

Explanation:
Only three moves are needed (remember each move increments two elements):

[1,2,3]  =>  [2,3,3]  =>  [3,4,3]  =>  [4,4,4]
```

Solution 1

Add `1` to `n - 1` elements is the same as subtracting `1` from one element, w.r.t goal of making the elements in the array equal.
So, best way to do this is make all the elements in the array equal to the `min` element.
`sum(array) - n * minimum`

```
public class Solution {
    public int minMoves(int[] nums) {
        if (nums.length == 0) return 0;
        int min = nums[0];
        for (int n : nums) min = Math.min(min, n);
        int res = 0;
        for (int n : nums) res += n - min;
        return res;
    }
}
```

written by kdtree original link here

Solution 2

Incrementing all but one is equivalent to decrementing that one. So let's do that instead. How many single-element decrements to make all equal? No point to decrementing below the current minimum, so how many single-element decrements to make all equal to the current minimum? Just take the difference from what's currently there (the sum) to what we want (n times the minimum).

Python:

```
def minMoves(self, nums):
    return sum(nums) - len(nums) * min(nums)
```

Ruby:

```
def min_moves(nums)
  nums.inject(:+) - nums.size * nums.min
end
```

Java (ugh :-):

```
public int minMoves(int[] nums) {
    return IntStream.of(nums).sum() - nums.length * IntStream.of(nums).min().getA
sInt();
}
```

C++ (more ugh):

```
int minMoves(vector<int>& nums) {
    return accumulate(begin(nums), end(nums), 0) - nums.size() * *min_element(beg
in(nums), end(nums));
}
```

written by StefanPochmann original link here

Solution 3

First time, I try to `find the max num every time, and +1 to rest num`, code like below:

```
public int minMoves(int[] nums) {
    return helper(nums, 0);
}

private int helper(int[] nums, int count) {
    int max = 0;
    int total = 1;
    for (int i = 1; i < nums.length; i++) {
        if (nums[i] > nums[max]) max = i;
        else if (nums[i] == nums[max]) total++;
    }
    if (total == nums.length) return count;

    for (int i = 0; i < nums.length; i++) {
        if (i != max) nums[i]++;
    }
    return helper(nums, ++count);
}
```

but when the nums is `[1, 1, 2147483647]`, it will be `java.lang.StackOverflowError`.

So I try to improve in this way that `find the max and min num every time, and +(max - min) to rest num`, code like below:

```
public int minMoves(int[] nums) {
    return helper(nums, 0);
}

private int helper(int[] nums, int count) {
    int max = 0, min = 0;
    int total = 1;
    for (int i = 1; i < nums.length; i++) {
        if (nums[i] > nums[max]) max = i;
        else if (nums[i] < nums[min]) min = i;
        else if (nums[i] == nums[max]) total++;
    }
    if (total == nums.length) return count;

    int dis = nums[max] - nums[min];
    for (int i = 0; i < nums.length; i++) {
        if (i != max) nums[i] += dis;
    }
    return helper(nums, count + dis);
}
```

But when the num length is bigger to 10000, it wil be `Time Limit Exceeded`.

Then, I want to `implements it by no recursive way and use insert sort`

`every time in order to reduce unnecessary traversal operation`. code like below:

```
    public int minMoves(int[] nums) {
        int res = 0;
        int n = nums.length;
        Arrays.sort(nums);

        while (nums[n - 1] != nums[0]) {
            int dis = nums[n - 1] - nums[0];
            for (int i = 0; i < n - 1; i++) {
                nums[i] += dis;
            }
            res += dis;

            //insert sort
            int max = nums[n - 1];
            int i = n - 2;
            while (i >= 0) {
                if (nums[i] > max) nums[i + 1] = nums[i--];
                else break;
            }
            nums[i + 1] = max;
        }

        return res;
    }
```

But it still `Time Limit Exceeded`.

========== **The final solution is as follows** ==========

The final flash, I though that should we use `dynamic programming`?

- `[step] is The number of steps arrive at the state of [all equal]`
- `[finalNum] is The value of the state of [all equal]`

we can know that

- `step[i] = (step[i-1] + num[i]) - finalNum[i-1] + step[i-1]`
- `finalNum[i] = num[i] + step[i-1]`

```java
public int minMoves(int[] nums) {
    Arrays.sort(nums);

    int n = nums.length;
    int step = 0;
    int finalNum = nums[0];

    for (int i = 1; i < n; i++) {
        int tmp = finalNum;
        finalNum = nums[i] + step;
        if (finalNum == tmp) continue;   //attention!!
        step = finalNum - tmp + step;
    }

    return step;
}
```

written by zhangCabbage original link here