<https://leetcode.ca/2018-01-12-774-Minimize-Max-Distance-to-Gas-Station/>

You are given an **integer** array **stations** that represents the positions of the gas stations on the **x-axis**. You are also given an integer k.

You should add k new gas stations. You can add the stations anywhere on the x-axis, and not necessarily on an integer position.

Let **penalty()** be the maximum distance between adjacent gas stations after adding the k new stations.

Return the **smallest possible value of penalty()**. Answers within 10^-6 of the actual answer will be accepted.

**Example 1:**

Input: stations = [1,2,3,4,5,6,7,8,9,10], k = 9

Output: 0.50000

**Example 2:**

Input: stations = [23,24,36,39,46,56,57,65,84,98], k = 1

Output: 14.00000

**Constraints:**

10 <= stations.length <= 2000

0 <= stations[i] <= 10^8

stations is sorted in a strictly increasing order.

1 <= k <= 10^6

**Attempt 1: 2024-12-08**

**Solution 1: Binary Search + Greedy (60 min)**

**Since "stations is sorted in a strictly increasing order.", no need sorting**

class Solution {

    public double minmaxGasDist(int[] stations, int k) {

        double lo = 0;

        double hi = stations[stations.length - 1] - stations[0];

        // Precision threshold for floating-point comparison

        double epsilon = 1e-6;

        while (lo < hi - epsilon) {

            double mid = lo + (hi - lo) / 2;

            // Cannot use hi = mid - 1 or lo = mid + 1 since not gap as integer

            if (canPlaceStations(stations, k, mid)) {

                hi = mid; // Try for a smaller maximum distance

            } else {

                lo = mid; // Increase the maximum distance

            }

        }

        return lo;

    }

    private boolean canPlaceStations(int[] stations, int k, double maxDist) {

        int requiredStations = 0;

        for (int i = 1; i < stations.length; i++) {

            double gap = stations[i] - stations[i - 1];

            // Number of additional stations needed for this gap

            requiredStations += (int) Math.ceil(gap / maxDist) - 1;

        }

        return requiredStations <= k;

    }

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

**Refer to chatGPT**

**Approach:**

This is a **binary search on the answer** problem. The task is to **minimize** the maximum distance by strategically placing additional gas stations.

**Key Idea:**

Binary Search:

The smallest possible maximum distance is 0 (ideal but impractical).

The largest possible maximum distance is the current largest gap between adjacent gas stations.

Feasibility Check:

For a given maximum distance d, determine if it is possible to place k additional gas stations such that no gap exceeds d.

To check, calculate the number of stations required to reduce all gaps to ≤ d.

class Solution {

public double minmaxGasDist(int[] stations, int k) {

// Binary search bounds

double left = 0;

double right = stations[stations.length - 1] - stations[0];

double epsilon = 1e-6; // Precision threshold for floating-point comparison

while (right - left > epsilon) {

double mid = left + (right - left) / 2;

if (canPlaceStations(stations, k, mid)) {

right = mid; // Try for a smaller maximum distance

} else {

left = mid; // Increase the maximum distance

}

}

return left;

}

private boolean canPlaceStations(int[] stations, int k, double maxDist) {

int requiredStations = 0;

for (int i = 1; i < stations.length; i++) {

double gap = stations[i] - stations[i - 1];

requiredStations += (int) Math.ceil(gap / maxDist) - 1; // Number of additional stations needed for this gap

}

return requiredStations <= k;

}

}

**Explanation:**

Binary Search:

Start with left=0 and right=max(stations) - min(stations).

Use the mid-point of the current range to test if it’s feasible to reduce all gaps to ≤mid.

canPlaceStations **Function**:

For each gap between adjacent stations, calculate how many additional stations are required to reduce the gap to ≤ maxDist.

Return true if the total required stations ≤ k, otherwise false.

Precision Handling:

The binary search stops when the difference between right and left is less than a small threshold epsilon (e.g., 10^−6).

**Refer to**

<https://grandyang.com/leetcode/774/>

这道题说给了我们n个加油站，两两之间相距不同的距离，然后我们可以在任意地方新加K个加油站，问能使得任意两个加油站之间的最大距离的最小值是多少。乍眼一看，感觉很绕，一会儿最大，一会儿最小的。其实我们可以换个场景，比如n个人站一队，每两个人之间距离不同，有的人之间距离可能很大，有的人可能挨得很近。我们现在需要再加入K个人到队列中，我们希望人与人之间的距离尽可能小，所以新人就应该加入到距离大的地方，然后问我们加入K个人后，求人与人之间的最大距离。这么一说，是不是清晰一点了呢。博主最开始看到这个加油站的题，以为跟之前那道[Gas Station](http://www.cnblogs.com/grandyang/p/4266812.html)有关联，结果发现二者并没有什么关系，只不过公用了加油站这个场景而已。对于这道题，我们还是抽离出本质，就是数组插数问题。博主最先考虑的是用贪婪算法，就是先算出每两个数字之间的距离，然后我们每次往距离最大的那两个数字之间插入一个数字，这种想法看似正确，但是会跪在这样一个test case：

[10, 19, 25, 27, 56, 63, 70, 87, 96, 97]，K = 3

其两两之间的距离为：

9，6，2，29，7，7，17，9，1

如果按照博主前面所说的方法，会先将29分开，变成两个14.5，然后会将17分开，变成两个8.5，还剩一个加油站，会将其中一个14.5分开，变成两个7.25。但是这样弄下来，最大的距离还是14.5，而实际上我们有更好的办法，我们用两个加油站将29三等分，会变成三个9.67，然后用剩下的一个去分17，得到两个8.5，此时最大距离就变成了9.67，这才是最优的解法。这说明了博主那种图样图森破的贪婪算法并不work，缺乏对Hard题目的尊重。

后来看了[官方解答贴](https://leetcode.com/problems/minimize-max-distance-to-gas-station/solution/)中的解法，发现用DP会内存超标MLE，用堆会时间超标TLE。其实这道题的正确解法是用二分搜索法，博主第一反应是，还有这种操作！！？？就是有这种操作！！！这道题使用的二分搜索法是博主归纳总结帖[LeetCode Binary Search Summary 二分搜索法小结](http://www.cnblogs.com/grandyang/p/6854825.html)中的第四种，即二分法的判定条件不是简单的大小关系，而是可以抽离出子函数的情况，下面我们来看具体怎么弄。如果光说要用二分法来做，那么首先就要明确的是二分法用来查找什么，难道是用来查找要插入加油站的位置吗？很显然不是，其实是用来查找那个最小的任意两个加油站间的最大距离。这其实跟之前那道[Kth Smallest Element in a Sorted Matrix](http://www.cnblogs.com/grandyang/p/5727892.html)非常的类似，那道题的二分搜索也是直接去折半定位所求的数，然后再来验证其是否真的符合题意。这道题也是类似的思路，题目中给了数字的范围[0, 10^8]，那么二分查找的左右边界值就有了，又给了误差范围10^-6，那么只要right和left差值大于这个阈值，就继续循环。我们折半计算出来的mid就是一个candidate，我们要去验证个candidate是否符合题意。验证的方法其实也不难，我们计算每两个加油站之间的距离，如果此距离大于candidate，则计数器累加1，如果大于candidate距离的个数小于等于k，则说明我们的candidate偏大了，那么right赋值为mid；反之若大于candidate距离的个数大于k，则说明我们的candidate偏小了，那么left赋值为mid。最后left和right都会收敛为所要求的最小的任意两个加油站间的最大距离，是不是很神奇呀！！Amazing！！参见代码如下：

class Solution {

// Finds the minimum possible distance between gas stations after adding k extra stations.

public double minmaxGasDist(int[] stations, int K) {

// Define the range for the possible solution (left is minimum distance, right is maximum distance)

double left = 0, right = 1e8;

// Continue searching while the precision is greater than 1e-6

while (right - left > 1e-6) {

// Take the middle of the current range as a guess

double mid = (left + right) / 2.0;

// If it's possible to place gas stations such that the maximum distance is less than or equal to 'mid',

// then we update right to the current 'mid' to see if we can find an even smaller maximum distance

if (isPossible(mid, stations, K)) {

right = mid;

} else {

// If it's not possible, we update left to be 'mid' to look for solutions greater than 'mid'

left = mid;

}

}

// Since the left and right converge to the point where right - left <= 1e-6, left is the most accurate answer

return left;

}

// Helper method to check if it's possible to have a maximum gas station distance less than x after adding K gas stations.

private boolean isPossible(double x, int[] stations, int K) {

int count = 0; // the number of gas stations we need to add to satisfy the condition

// Go through all pairs of adjacent stations

for (int i = 0; i < stations.length - 1; ++i) {

// Find the number of additional stations needed for this segment so that the distance between stations is <= x

count += (int) ((stations[i + 1] - stations[i]) / x);

}

// Check if the count of additional stations required is less than or equal to K (the count we can add)

return count <= K;

}

}

**Refer to**

<https://just4once.gitbooks.io/leetcode-notes/content/leetcode/binary-search/774-minimize-max-distance-to-gas-station.html>

**Thought Process**

Dynamic Programing (TLE)

Create a dp array to store the best answer, where dp[s][k] defines to be the answer for inserting k ending at jth station

We initialize dp[0][j] with the distance[0] / (j + 1), where j ranges from 0 to K

We start loop for all distance segments from 1 to N - 1 using i, and nested loop for inserting extra stations from 0 to K using j

For insert j stations for ith segment, we need to check all the previous k insertions ranges from 0 to j stations against the inserting the k stations in ith segment, the maximum of them is the candidate for our answer

The answer dp[i][j] is then the minimum of all the candiates

Time complexity O(nK^2)

Space complexity O(nK)

class Solution {

public double minmaxGasDist(int[] stations, int K) {

int n = stations.length;

double[][] dp = new double[n - 1][K + 1];

double[] distances = new double[n - 1];

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

distances[i - 1] = stations[i] - stations[i - 1];

}

for (int k = 0; k <= K; k++) {

dp[0][k] = distances[0] / (k + 1);

}

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

for (int j = 0; j <= K; j++) {

double ans = 99999999;

for (int k = 0; k <= j; k++) {

ans = Math.min(ans, Math.max(dp[i - 1][k], distances[i] / (j - k + 1)));

}

dp[i][j] = ans;

}

}

return dp[n - 2][K];

}

}

Greedy Approach with Heap (TLE)

Use heap to store each distance segments along with number of segments (initially all 1)

Every time, we pull a max distance segment out considering along with number of segment and increase the number and pull it back

After K operations, we have used up all the stations

Then the answer will be on top of the heap

Time complexity O(Klogn)

Space complexity O(n)

class Solution {

public double minmaxGasDist(int[] stations, int K) {

PriorityQueue<int[]> maxHeap = new PriorityQueue<>((a, b) -> a[0] \* 1.0 / a[1] > b[0] \* 1.0 / b[1] ? -1 : 1);

for (int i = 1; i < stations.length; i++) {

maxHeap.offer(new int[]{stations[i] - stations[i - 1], 1});

}

int[] node = null;

while (K-- > 0) {

node = maxHeap.poll();

node[1]++;

maxHeap.offer(node);

}

node = maxHeap.poll();

return node[0] \* 1.0 / node[1];

}

}

Binary Search

Using the boundaries of stations, the possible width will be in between 0 and 1e8

Perform binary search on the width within these boundaries, we will narrow down our search until these boundaries differ less than our desired delta

Now the conditions to narrow our search to left or right depends on a separate function possible(mi, stations, K), where it check if we can use less than or equal to K stations to achieve the mi distance

If possible return true, we set hi = mi to find better distance

Else we set lo = mi

Time complexity O(nlogW), where w = 1e8/1e-6 = 1e14

Space complexity O(1)

class Solution {

public double minmaxGasDist(int[] stations, int K) {

double delta = 1e-6;

double lo = 0, hi = 1e8;

while (hi - lo > delta) {

double mi = (lo + hi) / 2;

if (possible(mi, stations, K)) hi = mi;

else lo = mi;

}

return lo;

}

private boolean possible(double x, int[] stations, int K) {

int count = 0;

for (int i = 1; i < stations.length; i++) {

count += (int) ((stations[i] - stations[i - 1]) / x);

}

return count <= K;

}

}

**Refer to**

[L410.Split Array Largest Sum (Ref.L1011,L704,L1482)](note://WEB4d9d6f26a965381b137615128e2648be)

[L1011.Capacity To Ship Packages Within D Days (Ref.L410,L1482)](note://WEB87ca8f9dd9a14fb4535856a4e0f8f4b1)

[L1283.Find the Smallest Divisor Given a Threshold (Ref.L410,L1011,L1482)](note://76952362C4844CB3B68E28EA09F2EE58)

[L1482.Minimum Number of Days to Make m Bouquets (Ref.L410,L1011)](note://WEBd5ce394160da45a79e92dbbc806111ec)

[L1552.Magnetic Force Between Two Balls (Ref.L1802,L2226)](note://WEB4d72055c8eee671bf55dca2e0a3f951d)

[L1802.Maximum Value at a Given Index in a Bounded Array (Ref.L410)](note://WEB7d060d486a37a8c92696be41f18fb27c)

[L2064.Minimized Maximum of Products Distributed to Any Store (Ref.L410)](note://WEB2ca95ff77a9fee4bcced70f3e62a41df)

[L2226.Maximum Candies Allocated to K Children (Ref.L1802,L1552)](note://WEBbc250dc9fe15c478547298491ba2386f)