LeetCode Training Day 6 Binary Search II

Guess Number

Guess the number is a special form of **binary search**. Such problem normally asks you to find the most optimal number which satisfies some condition. Such problems may have some phases such as “find the maximum /minum number which can…” or “find the kth number…”. If you can not find other easy solutions, you can try binary search.

The idea is that you know the range of answer, for example from 1 to maximum of num[i], then you try a specific number, and test the condition, if condition is not satisified due to the number you guess is too big or too small, you can adjust your search range. If the condition met, then you know this is a candidate of answer, but may not the best one, then you adjust search again to get the minimum or maximum number which can satisfy the condition.

The code pattern is similar to binary search. The time complexity for such problem is normally N \* LOG(N)

## 378. Kth Smallest Element in a Sorted Matrix

Medium

Given a *n* x *n* matrix where each of the rows and columns are sorted in ascending order, find the kth smallest element in the matrix.

Note that it is the kth smallest element in the sorted order, not the kth distinct element.

**Example:**

matrix = [

[ 1, 5, 9],

[10, 11, 13],

[12, 13, 15]

],

k = 8,

return 13.

**Note:**  
You may assume k is always valid, 1 ≤ k ≤ n2.

### Analysis:

You guess a number between the left upper corner and right bottom corner and search the matrix from right top and check how many numbers are less than the guessed number.

The time complexity is O(NlogN), where N is the total number of the elements in the matrix.

/// <summary>

/// Leet code #378. Kth Smallest Element in a Sorted Matrix

/// </summary>

int LeetCodeBinarySearch::countNoGreaterValue(vector<vector<int>>& matrix, int value, int k)

{

int i = 0, j = matrix[0].size() - 1;

int count = 0;

while (i < (int)matrix.size() && j >= 0)

{

if (matrix[i][j] <= value)

{

i++;

count += j + 1;

if (count > k) return count;

}

else j--;

}

return count;

}

/// <summary>

/// Leet code #378. Kth Smallest Element in a Sorted Matrix

///

/// Given a n x n matrix where each of the rows and columns are sorted

/// in ascending order,

/// find the kth smallest element in the matrix. Note that it is the kth

/// smallest element in the sorted order, not the kth distinct element.

/// Example:

/// matrix =

/// [

/// [ 1, 5, 9],

/// [10, 11, 13],

/// [12, 13, 15]

/// ],

/// k = 8,

/// return 13.

/// </summary>

int LeetCodeBinarySearch::kthSmallest(vector<vector<int>>& matrix, int k)

{

int low = matrix[0][0];

int high = matrix[matrix.size() - 1][matrix[0].size() - 1];

int result = 0;

while (low <= high)

{

int mid = low + (high - low) / 2;

if (countNoGreaterValue(matrix, mid, k) < k)

{

low = mid + 1;

}

else

{

result = mid;

high = mid - 1;

}

}

return low;

}

## 410. Split Array Largest Sum

Hard

Given an array which consists of non-negative integers and an integer *m*, you can split the array into *m* non-empty continuous subarrays. Write an algorithm to minimize the largest sum among these *m* subarrays.

**Note:**  
If *n* is the length of array, assume the following constraints are satisfied:

* 1 ≤ *n* ≤ 1000
* 1 ≤ *m* ≤ min(50, *n*)

**Examples:**

Input:

**nums** = [7,2,5,10,8]

**m** = 2

Output:

18

Explanation:

There are four ways to split **nums** into two subarrays.

The best way is to split it into **[7,2,5]** and **[10,8]**,

where the largest sum among the two subarrays is only 18.

### Analysis:

You guess the number from 0 to total sum using binary search and see if you can use this number to split the array into 2 parts.

/// <summary>

/// Leet Code 410. Split Array Largest Sum

///

/// Hard

///

/// Given an integer array nums and an integer k, split nums into k

/// non-empty subarrays such that the largest sum of any subarray is

/// minimized.

///

/// Return the minimized largest sum of the split.

/// A subarray is a contiguous part of the array.

///

/// Example 1:

/// Input: nums = [7,2,5,10,8], k = 2

/// Output: 18

/// Explanation: There are four ways to split nums into two subarrays.

/// The best way is to split it into [7,2,5] and [10,8], where the

/// largest sum among the two subarrays is only 18.

///

/// Example 2:

/// Input: nums = [1,2,3,4,5], k = 2

/// Output: 9

/// Explanation: There are four ways to split nums into two subarrays.

/// The best way is to split it into [1,2,3] and [4,5], where the largest

/// sum among the two subarrays is only 9.

///

/// Constraints:

/// 1. 1 <= nums.length <= 1000

/// 2. 0 <= nums[i] <= 10^6

/// 3. 1 <= k <= min(50, nums.length)

/// </summary>

int LeetCodeBinarySearch::splitArray(vector<int>& nums, int k)

{

int first = 0;

int last = 0;

for (size\_t i = 0; i < nums.size(); i++)

{

last += nums[i];

first = max(first, nums[i]);

}

// the maximum slice size must be between the maximum number and the total

// sum of the slice, try all possible by binary search.

int result = 0;

while (first <= last)

{

// assume the mid is the sum of maximum slice

int mid = first + (last - first) / 2;

int count = 1;

int subTotal = 0;

for (size\_t i = 0; i < nums.size(); i++)

{

if (subTotal + nums[i] > mid)

{

count++;

if (count > k)

{

break;

}

else

{

subTotal = 0;

}

}

subTotal += nums[i];

}

// if count is greater than k, it means bursted

if (count > k)

{

first = mid + 1;

}

else // if not bursted, find the smaller sum greedily

{

result = mid;

last = mid - 1;

}

}

return result;

}

## 878. Nth Magical Number

Hard

A positive integer is *magical* if it is divisible by either A or B.

Return the N-th magical number.  Since the answer may be very large, **return it modulo**10^9 + 7.

**Example 1:**

**Input:** N = 1, A = 2, B = 3

**Output:** 2

**Example 2:**

**Input:** N = 4, A = 2, B = 3

**Output:** 6

**Example 3:**

**Input:** N = 5, A = 2, B = 4

**Output:** 10

**Example 4:**

**Input:** N = 3, A = 6, B = 4

**Output:** 8

**Note:**

1. 1 <= N <= 10^9
2. 2 <= A <= 40000
3. 2 <= B <= 40000

### Analysis:

Given a number you can check how many multiple of A and B and deduct the multiple of LCM of A and B. You guess the number and count it is k-th magical number, and use binary search to close the gap.

/// <summary>

/// Leet code #878. Nth Magical Number

///

/// A positive integer is magical if it is divisible by either A or B.

///

/// Return the N-th magical number. Since the answer may be very large,

/// return it modulo 10^9 + 7.

///

/// Example 1:

/// Input: N = 1, A = 2, B = 3

/// Output: 2

///

/// Example 2:

/// Input: N = 4, A = 2, B = 3

/// Output: 6

///

/// Example 3:

/// Input: N = 5, A = 2, B = 4

/// Output: 10

///

/// Example 4:

/// Input: N = 3, A = 6, B = 4

/// Output: 8

///

/// Note:

/// 1. 1 <= N <= 10^9

/// 2. 2 <= A <= 40000

/// 3. 2 <= B <= 40000

/// </summary>

int LeetCodeBinarySearch::nthMagicalNumber(int n, int a, int b)

{

int c = a \* b / (int)std::gcd(a, b);

int mod = 1000000007;

unsigned long long first = 1;

unsigned long long last = (unsigned long long)a \* n;

while (first < last)

{

unsigned long long middle = first + (last - first) / 2;

unsigned long long k = middle / a + middle / b - middle / c;

if (k < n) first = middle + 1;

else last = middle;

}

return (int)(first % mod);

}

## 1231. Divide Chocolate

Hard

You have one chocolate bar that consists of some chunks. Each chunk has its own sweetness given by the array sweetness.

You want to share the chocolate with your K friends so you start cutting the chocolate bar into K+1 pieces using K cuts, each piece consists of some **consecutive** chunks.

Being generous, you will eat the piece with the **minimum total sweetness** and give the other pieces to your friends.

Find the **maximum total sweetness** of the piece you can get by cutting the chocolate bar optimally.

**Example 1:**

**Input:** sweetness = [1,2,3,4,5,6,7,8,9], K = 5

**Output:** 6

**Explanation:** You can divide the chocolate to [1,2,3], [4,5], [6], [7], [8], [9]

**Example 2:**

**Input:** sweetness = [5,6,7,8,9,1,2,3,4], K = 8

**Output:** 1

**Explanation:** There is only one way to cut the bar into 9 pieces.

**Example 3:**

**Input:** sweetness = [1,2,2,1,2,2,1,2,2], K = 2

**Output:** 5

**Explanation:** You can divide the chocolate to [1,2,2], [1,2,2], [1,2,2]

**Constraints:**

* 0 <= K < sweetness.length <= 10^4
* 1 <= sweetness[i] <= 10^5

### Analysis:

You guess the sweetness from 1 to sum of the sweetness in the chocolate bar.

/// <summary>

/// Leet code #1231. Divide Chocolate

///

/// You have one chocolate bar that consists of some chunks. Each chunk

/// has its own sweetness given by the array sweetness.

///

/// You want to share the chocolate with your K friends so you start

/// cutting the chocolate bar into K+1 pieces using K cuts, each piece

/// consists of some consecutive chunks.

/// Being generous, you will eat the piece with the minimum total

/// sweetness and give the other pieces to your friends.

///

/// Find the maximum total sweetness of the piece you can get by cutting

/// the chocolate bar optimally.

///

/// Example 1:

/// Input: sweetness = [1,2,3,4,5,6,7,8,9], K = 5

/// Output: 6

/// Explanation: You can divide the chocolate to [1,2,3], [4,5], [6],

/// [7], [8], [9]

///

/// Example 2:

/// Input: sweetness = [5,6,7,8,9,1,2,3,4], K = 8

/// Output: 1

/// Explanation: There is only one way to cut the bar into 9 pieces.

///

/// Example 3:

///

/// Input: sweetness = [1,2,2,1,2,2,1,2,2], K = 2

/// Output: 5

/// Explanation: You can divide the chocolate to [1,2,2], [1,2,2], [1,2,2]

///

///

/// Constraints:

/// 1. 0 <= K < sweetness.length <= 10^4

/// 2. 1 <= sweetness[i] <= 10^5

/// </summary>

int LeetCodeBinarySearch::maximizeSweetness(vector<int>& sweetness, int K)

{

int last = 0;

for (size\_t i = 0; i < sweetness.size(); i++)

{

last += sweetness[i];

}

int first = 0;

int result = first;

while (first <= last)

{

int middle = first + (last - first) / 2;

int index = 0;

int sum = 0;

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

{

sum = 0;

while (index < (int)sweetness.size() && sum < middle)

{

sum += sweetness[index];

index++;

}

if (sum < middle) break;

}

if (sum < middle)

{

last = middle - 1;

}

else

{

result = middle;

first = middle + 1;

}

}

return result;

}

## 719. Find K-th Smallest Pair Distance

Hard

Given an integer array, return the k-th smallest **distance** among all the pairs. The distance of a pair (A, B) is defined as the absolute difference between A and B.

**Example 1:**

**Input:**

nums = [1,3,1]

k = 1

**Output: 0**

**Explanation:**

Here are all the pairs:

(1,3) -> 2

(1,1) -> 0

(3,1) -> 2

Then the 1st smallest distance pair is (1,1), and its distance is 0.

**Note:**

1. 2 <= len(nums) <= 10000.
2. 0 <= nums[i] < 1000000.
3. 1 <= k <= len(nums) \* (len(nums) - 1) / 2.

### Analysis:

First you need to sort the numbers, and then you guess the number from 0 to the maximum distance (the distance between the minimum and maximum number).

On every guessed number, you count how many distances are less than it. You start from first two element and make it as the first pointer and the last pointer for the window, keep on moving the last pointer to the end until the distance is more than what you guessed, then you move the first pointer one position to the right. Do you need to try the last pointer on left? The answer is no, all the left number to the first element is less than distance D then they are from the second element in less than D as well, you should only try the right side for the last pointer.

Given that you keep on moving the first pointer and last pointer of the sliding window to right, the whole algorithm it is a O(NlogN), such technique is also used in many merge sort algorithm.

/// <summary>

/// Leet Code 719. Find K-th Smallest Pair Distance

///

/// Hard

///

/// The distance of a pair of integers a and b is defined as the absolute

/// difference between a and b.

///

/// Given an integer array nums and an integer k, return the kth smallest

/// distance among all the pairs nums[i] and nums[j] where

/// 0 <= i < j < nums.length.

///

/// Example 1:

/// Input: nums = [1,3,1], k = 1

/// Output: 0

/// Explanation: Here are all the pairs:

/// (1,3) -> 2

/// (1,1) -> 0

/// (3,1) -> 2

/// Then the 1st smallest distance pair is (1,1), and its distance is 0.

///

/// Example 2:

/// Input: nums = [1,1,1], k = 2

/// Output: 0

///

/// Example 3:

/// Input: nums = [1,6,1], k = 3

/// Output: 5

///

/// Constraints:

/// 1. n == nums.length

/// 2. 2 <= n <= 10^4

/// 3. 0 <= nums[i] <= 10^6

/// 4. 1 <= k <= n \* (n - 1) / 2

/// </summary>

int LeetCodeBinarySearch::smallestDistancePair(vector<int>& nums, int k)

{

sort(nums.begin(), nums.end());

// Minimum absolute difference

int low = nums[1] - nums[0];

for (int i = 1; i < (int)nums.size() - 1; i++)

{

low = min(low, nums[i + 1] - nums[i]);

}

int high = nums[nums.size() - 1] - nums[0];

int result = 0;

while (low <= high)

{

int mid = (low + high) / 2;

int count = 0;

int first = 0;

int last = 1;

while (first < (int)nums.size())

{

while (last < (int)nums.size() && nums[last] - nums[first] <= mid)

{

last++;

}

count += last - first - 1;

if (count > k) break;

first++;

}

if (count >= k)

{

result = mid;

high = mid - 1;

}

else

{

low = mid + 1;

}

}

return result;

}

## 786. K-th Smallest Prime Fraction

Hard

A sorted list A contains 1, plus some number of primes.  Then, for every p < q in the list, we consider the fraction p/q.

What is the K-th smallest fraction considered?  Return your answer as an array of ints, where answer[0] = p and answer[1] = q.

**Examples:**

**Input:** A = [1, 2, 3, 5], K = 3

**Output:** [2, 5]

**Explanation:**

The fractions to be considered in sorted order are:

1/5, 1/3, 2/5, 1/2, 3/5, 2/3.

The third fraction is 2/5.

**Input:** A = [1, 7], K = 1

**Output:** [1, 7]

**Note:**

* A will have length between 2 and 2000.
* Each A[i] will be between 1 and 30000.
* K will be between 1 and A.length \* (A.length - 1) / 2.

### Analysis:

This problem is almost identical to problem 719, just change the distance to the fraction value which is a double. You guess the value in double, with controlling the precision and slowly close the gap.

/// <summary>

/// Leet code #786. K-th Smallest Prime Fraction

///

/// A sorted list A contains 1, plus some number of primes. Then, for

/// every p < q in the list, we consider the fraction p/q.

///

/// What is the K-th smallest fraction considered? Return your answer as

/// an array of ints, where answer[0] = p and answer[1] = q.

///

/// Examples:

/// Input: A = [1, 2, 3, 5], K = 3

/// Output: [2, 5]

/// Explanation:

/// The fractions to be considered in sorted order are:

/// 1/5, 1/3, 2/5, 1/2, 3/5, 2/3.

/// The third fraction is 2/5.

///

/// Input: A = [1, 7], K = 1

/// Output: [1, 7]

/// Note:

/// 1. A will have length between 2 and 2000.

/// 2. Each A[i] will be between 1 and 30000.

/// 3. K will be between 1 and A.length \* (A.length + 1) / 2.

/// </summary>

vector<int> LeetCodeBinarySearch::kthSmallestPrimeFraction(vector<int>& A, int K)

{

double min\_value = 0;

double abs\_min = (double)1 / (2 \* (double)A[A.size() - 1] \* (double)A[A.size() - 2]);

double max\_value = 1;

vector<int> result(2);

while (max\_value - min\_value >= abs\_min)

{

double middle = (min\_value + max\_value) / 2;

double current\_max = 0;

size\_t first = 0;

size\_t last = 1;

int count = 0;

while (last < A.size())

{

if (first == last)

{

count += first;

last++;

}

else

{

double current\_value = (double)A[first] / A[last];

if (current\_value <= middle)

{

if (current\_value > current\_max)

{

result[0] = A[first];

result[1] = A[last];

current\_max = current\_value;

}

first++;

}

else

{

count += first;

last++;

}

}

}

if (count == K) break;

else if (count > K) max\_value = middle;

else min\_value = middle;

}

return result;

}

## 875. Koko Eating Bananas

Medium

Koko loves to eat bananas. There are n piles of bananas, the ith pile has piles[i] bananas. The guards have gone and will come back in h hours.

Koko can decide her bananas-per-hour eating speed of k. Each hour, she chooses some pile of bananas and eats k bananas from that pile. If the pile has less than k bananas, she eats all of them instead and will not eat any more bananas during this hour.

Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

Return *the minimum integer* k *such that she can eat all the bananas within* h *hours*.

**Example 1:**

**Input:** piles = [3,6,7,11], h = 8

**Output:** 4

**Example 2:**

**Input:** piles = [30,11,23,4,20], h = 5

**Output:** 30

**Example 3:**

**Input:** piles = [30,11,23,4,20], h = 6

**Output:** 23

**Constraints:**

* 1 <= piles.length <= 104
* piles.length <= h <= 109
* 1 <= piles[i] <= 109

### Analysis:

You choose a number as eating speed, see if Koko can finish all bananas in the H hour, if yes, this can be the answer, then try to see if she can even eat slowly, if not, increase the eating speed.

/// <summary>

/// Leet code #875. Koko Eating Bananas

///

/// Koko loves to eat bananas. There are N piles of bananas, the i-th

/// pile has piles[i] bananas. The guards have gone and will come back

/// in H hours.

///

/// Koko can decide her bananas-per-hour eating speed of K. Each hour,

/// she chooses some pile of bananas, and eats K bananas from that pile.

/// If the pile has less than K bananas, she eats all of them instead,

/// and won't eat any more bananas during this hour.

///

/// Koko likes to eat slowly, but still wants to finish eating all the

/// bananas before the guards come back.

///

/// Return the minimum integer K such that she can eat all the bananas

/// within H hours.

///

/// Example 1:

/// Input: piles = [3,6,7,11], H = 8

/// Output: 4

///

/// Example 2:

/// Input: piles = [30,11,23,4,20], H = 5

/// Output: 30

///

/// Example 3:

/// Input: piles = [30,11,23,4,20], H = 6

/// Output: 23

///

/// Note:

/// 1. 1 <= piles.length <= 10^4

/// 2. piles.length <= H <= 10^9

/// 3. 1 <= piles[i] <= 10^9

/// </summary>

int LeetCodeBinarySearch::minEatingSpeed(vector<int>& piles, int H)

{

int first = 1;

int last = 1;

int result = 0;

for (size\_t i = 0; i < piles.size(); i++)

{

last = max(last, piles[i]);

}

while (first <= last)

{

int middle = first + (last - first) / 2;

int count = 0;

for (size\_t i = 0; i < piles.size(); i++)

{

count += piles[i] / middle;

if (piles[i] % middle != 0) count++;

if (count > H) break;

}

if (count > H)

{

first = middle + 1;

}

else

{

result = middle;

last = middle - 1;

}

}

return result;

}

## 1011. Capacity To Ship Packages Within D Days

Medium

A conveyor belt has packages that must be shipped from one port to another within days days.

The ith package on the conveyor belt has a weight of weights[i]. Each day, we load the ship with packages on the conveyor belt (in the order given by weights). We may not load more weight than the maximum weight capacity of the ship.

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within days days.

**Example 1:**

**Input:** weights = [1,2,3,4,5,6,7,8,9,10], days = 5

**Output:** 15

**Explanation:** A ship capacity of 15 is the minimum to ship all the packages in 5 days like this:

1st day: 1, 2, 3, 4, 5

2nd day: 6, 7

3rd day: 8

4th day: 9

5th day: 10

Note that the cargo must be shipped in the order given, so using a ship of capacity 14 and splitting the packages into parts like (2, 3, 4, 5), (1, 6, 7), (8), (9), (10) is not allowed.

**Example 2:**

**Input:** weights = [3,2,2,4,1,4], days = 3

**Output:** 6

**Explanation:** A ship capacity of 6 is the minimum to ship all the packages in 3 days like this:

1st day: 3, 2

2nd day: 2, 4

3rd day: 1, 4

**Example 3:**

**Input:** weights = [1,2,3,1,1], days = 4

**Output:** 3

**Explanation:**

1st day: 1

2nd day: 2

3rd day: 3

4th day: 1, 1

**Constraints:**

* 1 <= days <= weights.length <= 5 \* 104
* 1 <= weights[i] <= 500

### Analysis:

You can try the capacity from 1 to the total cargo size by using binary search and see if you can pack the cargo in D days. If yes, keep it as an answer and try smaller ship, if not adjust to bigger ship. Watch a single cargo can bust a ship.

/// <summary>

/// Leet code #1011. Capacity To Ship Packages Within D Days

///

/// A conveyor belt has packages that must be shipped from one port to another

/// within D days.

///

/// The i-th package on the conveyor belt has a weight of weights[i]. Each

/// day, we load the ship with packages on the conveyor belt (in the order

/// given by weights). We may not load more weight than the maximum weight

/// capacity of the ship.

///

/// Return the least weight capacity of the ship that will result in all the

/// packages on the conveyor belt being shipped within D days.

///

/// Example 1:

///

/// Input: weights = [1,2,3,4,5,6,7,8,9,10], D = 5

/// Output: 15

/// Explanation:

/// A ship capacity of 15 is the minimum to ship all the packages in 5 days

/// like this:

/// 1st day: 1, 2, 3, 4, 5

/// 2nd day: 6, 7

/// 3rd day: 8

/// 4th day: 9

/// 5th day: 10

///

/// Note that the cargo must be shipped in the order given, so using a ship

/// of capacity 14 and splitting the packages into parts like (2, 3, 4, 5),

/// (1, 6, 7), (8), (9), (10) is not allowed.

///

/// Example 2:

///

/// Input: weights = [3,2,2,4,1,4], D = 3

/// Output: 6

/// Explanation:

/// A ship capacity of 6 is the minimum to ship all the packages in 3 days like this:

/// 1st day: 3, 2

/// 2nd day: 2, 4

/// 3rd day: 1, 4

///

/// Example 3:

///

/// Input: weights = [1,2,3,1,1], D = 4

/// Output: 3

/// Explanation:

/// 1st day: 1

/// 2nd day: 2

/// 3rd day: 3

/// 4th day: 1, 1

///

/// Note:

///

/// 1. 1 <= D <= weights.length <= 50000

/// 2. 1 <= weights[i] <= 500

/// </summary>

int LeetCodeBinarySearch::shipWithinDays(vector<int>& weights, int D)

{

int sum = 0;

for (size\_t i = 0; i < weights.size(); i++) sum += weights[i];

int first = 1;

int last = sum;

int result = 0;

while (first <= last)

{

int mid = first + (last - first) / 2;

int d = 1;

int sum = 0;

for (size\_t i = 0; i < weights.size(); i++)

{

// a single cargo may burst

if (weights[i] > mid)

{

d = D + 1;

break;

}

else if (sum + weights[i] > mid)

{

sum = 0;

d++;

}

if (d > D) break;

sum += weights[i];

}

// ship too small

if (d > D)

{

first = mid + 1;

}

else

{

result = mid;

last = mid - 1;

}

}

return result;

}

## 1283. Find the Smallest Divisor Given a Threshold

Medium

Given an array of integers nums and an integer threshold, we will choose a positive integer divisor, divide all the array by it, and sum the division's result. Find the **smallest** divisor such that the result mentioned above is less than or equal to threshold.

Each result of the division is rounded to the nearest integer greater than or equal to that element. (For example: 7/3 = 3 and 10/2 = 5).

It is guaranteed that there will be an answer.

**Example 1:**

**Input:** nums = [1,2,5,9], threshold = 6

**Output:** 5

**Explanation:** We can get a sum to 17 (1+2+5+9) if the divisor is 1.

If the divisor is 4 we can get a sum of 7 (1+1+2+3) and if the divisor is 5 the sum will be 5 (1+1+1+2).

**Example 2:**

**Input:** nums = [44,22,33,11,1], threshold = 5

**Output:** 44

**Example 3:**

**Input:** nums = [21212,10101,12121], threshold = 1000000

**Output:** 1

**Example 4:**

**Input:** nums = [2,3,5,7,11], threshold = 11

**Output:** 3

**Constraints:**

* 1 <= nums.length <= 5 \* 104
* 1 <= nums[i] <= 106
* nums.length <= threshold <= 106

### Analysis:

Binary search divisors for what are within the threshold.

/// <summary>

/// Leet code #1283. Find the Smallest Divisor Given a Threshold

///

/// Medium

///

/// Given an array of integers nums and an integer threshold, we will

/// choose a positive integer divisor and divide all the array by it

/// and sum the result of the division. Find the smallest divisor such

/// that the result mentioned above is less than or equal to threshold.

///

/// Each result of division is rounded to the nearest integer greater

/// than or equal to that element. (For example: 7/3 = 3 and 10/2 = 5).

///

/// It is guaranteed that there will be an answer.

///

///

/// Example 1:

/// Input: nums = [1,2,5,9], threshold = 6

/// Output: 5

///

/// Explanation: We can get a sum to 17 (1+2+5+9) if the divisor is 1.

/// If the divisor is 4 we can get a sum to 7 (1+1+2+3) and if the

/// divisor is 5 the sum will be 5 (1+1+1+2).

///

/// Example 2:

/// Input: nums = [2,3,5,7,11], threshold = 11

/// Output: 3

///

/// Example 3:

/// Input: nums = [19], threshold = 5

/// Output: 4

///

/// Constraints:

/// 1. 1 <= nums.length <= 5 \* 10^4

/// 2. 1 <= nums[i] <= 10^6

/// 3. nums.length <= threshold <= 10^6

/// </summary>

int LeetCodeBinarySearch::smallestDivisor(vector<int>& nums, int threshold)

{

int first = 1;

int last = 1000000;

int result = 0;

while (first <= last)

{

int middle = first + (last - first) / 2;

int sum = 0;

for (size\_t i = 0; i < nums.size(); i++)

{

sum += nums[i] / middle;

if (nums[i] % middle != 0) sum++;

}

if (sum > threshold)

{

first = middle + 1;

}

else

{

result = middle;

last = middle - 1;

}

}

return result;

}

## 1300. Sum of Mutated Array Closest to Target

Medium

Given an integer array arr and a target value target, return the integer value such that when we change all the integers larger than value in the given array to be equal to value, the sum of the array gets as close as possible (in absolute difference) to target.

In case of a tie, return the minimum such integer.

Notice that the answer is not neccesarilly a number from arr.

**Example 1:**

**Input:** arr = [4,9,3], target = 10

**Output:** 3

**Explanation:** When using 3 arr converts to [3, 3, 3] which sums 9 and that's the optimal answer.

**Example 2:**

**Input:** arr = [2,3,5], target = 10

**Output:** 5

**Example 3:**

**Input:** arr = [60864,25176,27249,21296,20204], target = 56803

**Output:** 11361

**Constraints:**

* 1 <= arr.length <= 104
* 1 <= arr[i], target <= 105

### Analysis:

Guess a number and if sum is more than target, decrease the guess, otherwise increase the guess. The candidate for the answer can come from both sides.

/// <summary>

/// Leetcode #1300. Sum of Mutated Array Closest to Target

///

/// Medium

///

/// Given an integer array arr and a target value target, return the

/// integer value such that when we change all the integers larger than

/// value in the given array to be equal to value, the sum of the array

/// gets as close as possible (in absolute difference) to target.

///

/// In case of a tie, return the minimum such integer.

///

/// Notice that the answer is not neccesarilly a number from arr.

///

/// Example 1:

///

/// Input: arr = [4,9,3], target = 10

/// Output: 3

/// Explanation: When using 3 arr converts to [3, 3, 3] which sums 9

/// and that's the optimal answer.

/// Example 2:

///

/// Input: arr = [2,3,5], target = 10

/// Output: 5

/// Example 3:

///

/// Input: arr = [60864,25176,27249,21296,20204], target = 56803

/// Output: 11361

///

/// Constraints:

/// 1. 1 <= arr.length <= 10^4

/// 2. 1 <= arr[i], target <= 10^5

/// </summary>

int LeetCodeBinarySearch::findBestValue(vector<int>& arr, int target)

{

int first = 0;

int last = 100000;

int result = 0;

int diff = 100000;

while (first <= last)

{

int middle = first + (last - first) / 2;

int sum = 0;

for (size\_t i = 0; i < arr.size(); i++)

{

if (arr[i] > middle) sum += middle;

else sum += arr[i];

}

if (abs(sum - target) < diff || (abs(sum - target) == diff) && (middle < result))

{

result = middle;

diff = abs(sum - target);

}

if (sum < target)

{

first = middle + 1;

}

else

{

last = middle - 1;

}

}

return result;

}

## 1482. Minimum Number of Days to Make m Bouquets

Medium

You are given an integer array bloomDay, an integer m and an integer k.

You want to make m bouquets. To make a bouquet, you need to use k **adjacent flowers** from the garden.

The garden consists of n flowers, the ith flower will bloom in the bloomDay[i] and then can be used in **exactly one** bouquet.

Return *the minimum number of days you need to wait to be able to make*m*bouquets from the garden*. If it is impossible to make m bouquets return -1.

**Example 1:**

**Input:** bloomDay = [1,10,3,10,2], m = 3, k = 1

**Output:** 3

**Explanation:** Let us see what happened in the first three days. x means flower bloomed and \_ means flower did not bloom in the garden.

We need 3 bouquets each should contain 1 flower.

After day 1: [x, \_, \_, \_, \_] // we can only make one bouquet.

After day 2: [x, \_, \_, \_, x] // we can only make two bouquets.

After day 3: [x, \_, x, \_, x] // we can make 3 bouquets. The answer is 3.

**Example 2:**

**Input:** bloomDay = [1,10,3,10,2], m = 3, k = 2

**Output:** -1

**Explanation:** We need 3 bouquets each has 2 flowers, that means we need 6 flowers. We only have 5 flowers so it is impossible to get the needed bouquets and we return -1.

**Example 3:**

**Input:** bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3

**Output:** 12

**Explanation:** We need 2 bouquets each should have 3 flowers.

Here is the garden after the 7 and 12 days:

After day 7: [x, x, x, x, \_, x, x]

We can make one bouquet of the first three flowers that bloomed. We cannot make another bouquet from the last three flowers that bloomed because they are not adjacent.

After day 12: [x, x, x, x, x, x, x]

It is obvious that we can make two bouquets in different ways.

**Constraints:**

* bloomDay.length == n
* 1 <= n <= 105
* 1 <= bloomDay[i] <= 109
* 1 <= m <= 106
* 1 <= k <= n

### Analysis:

Guess a day and see if you can get m adjacent flowers.

/// <summary>

/// Leet code #1482. Minimum Number of Days to Make m Bouquets

///

/// Medium

///

/// Given an integer array bloomDay, an integer m and an integer k.

/// We need to make m bouquets. To make a bouquet, you need to use

/// k adjacent flowers from the garden.

///

/// The garden consists of n flowers, the ith flower will bloom in the

/// bloomDay[i] and then can be used in exactly one bouquet.

///

/// Return the minimum number of days you need to wait to be able to

/// make m bouquets from the garden. If it is impossible to make m

/// bouquets return -1.

///

/// Example 1:

/// Input: bloomDay = [1,10,3,10,2], m = 3, k = 1

/// Output: 3

/// Explanation: Let's see what happened in the first three days.

/// x means flower bloomed and \_ means flower didn't bloom in the garden.

/// We need 3 bouquets each should contain 1 flower.

/// After day 1: [x, \_, \_, \_, \_] // we can only make one bouquet.

/// After day 2: [x, \_, \_, \_, x] // we can only make two bouquets.

/// After day 3: [x, \_, x, \_, x] // we can make 3 bouquets. The answer

/// is 3.

///

/// Example 2:

/// Input: bloomDay = [1,10,3,10,2], m = 3, k = 2

/// Output: -1

/// Explanation: We need 3 bouquets each has 2 flowers, that means we

/// need 6 flowers. We only have 5 flowers so it is impossible to get

/// the needed bouquets and we return -1.

///

/// Example 3:

/// Input: bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3

/// Output: 12

/// Explanation: We need 2 bouquets each should have 3 flowers.

/// Here's the garden after the 7 and 12 days:

/// After day 7: [x, x, x, x, \_, x, x]

/// We can make one bouquet of the first three flowers that bloomed.

/// We cannot make another bouquet from the last three flowers that

/// bloomed because they are not adjacent.

/// After day 12: [x, x, x, x, x, x, x]

/// It is obvious that we can make two bouquets in different ways.

///

/// Example 4:

/// Input: bloomDay = [1000000000,1000000000], m = 1, k = 1

/// Output: 1000000000

/// Explanation: You need to wait 1000000000 days to have a flower

/// ready for a bouquet.

///

/// Example 5:

///

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

/// Output: 9

///

/// Constraints:

/// 1. bloomDay.length == n

/// 2. 1 <= n <= 10^5

/// 3. 1 <= bloomDay[i] <= 10^9

/// 4. 1 <= m <= 10^6

/// 5. 1 <= k <= n

/// </summary>

int LeetCodeBinarySearch::minDays(vector<int>& bloomDay, int m, int k)

{

int first = 0;

int last = 1000000000;

int result = -1;

while (first <= last)

{

int middle = first + (last - first) / 2;

int k\_count = 0;

int m\_count = 0;

for (size\_t i = 0; i < bloomDay.size(); i++)

{

if (bloomDay[i] <= middle)

{

k\_count++;

}

else

{

k\_count = 0;

}

if (k\_count == k)

{

m\_count++;

k\_count = 0;

}

}

if (m\_count >= m)

{

result = middle;

last = middle - 1;

}

else

{

first = middle + 1;

}

}

return result;

}

## 1552. Magnetic Force Between Two Balls

Medium

In the universe Earth C-137, Rick discovered a special form of magnetic force between two balls if they are put in his new invented basket. Rick has n empty baskets, the ith basket is at position[i], Morty has m balls and needs to distribute the balls into the baskets such that the **minimum magnetic force** between any two balls is **maximum**.

Rick stated that magnetic force between two different balls at positions x and y is |x - y|.

Given the integer array position and the integer m. Return *the required force*.

**Example 1:**

Diagram

Description automatically generated

**Input:** position = [1,2,3,4,7], m = 3

**Output:** 3

**Explanation:** Distributing the 3 balls into baskets 1, 4 and 7 will make the magnetic force between ball pairs [3, 3, 6]. The minimum magnetic force is 3. We cannot achieve a larger minimum magnetic force than 3.

**Example 2:**

**Input:** position = [5,4,3,2,1,1000000000], m = 2

**Output:** 999999999

**Explanation:** We can use baskets 1 and 1000000000.

**Constraints:**

* n == position.length
* 2 <= n <= 105
* 1 <= position[i] <= 109
* All integers in position are **distinct**.
* 2 <= m <= position.length

### Analysis:

Guess a minimum distance between balls, and see if you can distribute m balls, if yes, keep the answer and increase the distance and try again. Use binary search to close the range.

/// <summary>

/// Leet code #1552. Magnetic Force Between Two Balls

///

/// Medium

///

/// In universe Earth C-137, Rick discovered a special form of magnetic

/// force between two balls if they are put in his new invented basket.

/// Rick has n empty baskets, the ith basket is at position[i], Morty

/// has m balls and needs to distribute the balls into the baskets such

/// that the minimum magnetic force between any two balls is maximum.

///

/// Rick stated that magnetic force between two different balls at

/// positions x and y is |x - y|.

///

/// Given the integer array position and the integer m. Return the

/// required force.

///

/// Example 1:

/// Input: position = [1,2,3,4,7], m = 3

/// Output: 3

/// Explanation: Distributing the 3 balls into baskets 1, 4 and 7 will

/// make the magnetic force between ball pairs [3, 3, 6]. The minimum

/// magnetic force is 3. We cannot achieve a larger minimum magnetic

/// force than 3.

///

/// Example 2:

/// Input: position = [5,4,3,2,1,1000000000], m = 2

/// Output: 999999999

/// Explanation: We can use baskets 1 and 1000000000.

///

/// Constraints:

/// 1. n == position.length

/// 2. 2 <= n <= 10^5

/// 3. 1 <= position[i] <= 10^9

/// 4. All integers in position are distinct.

/// 5. 2 <= m <= position.length

/// </summary>

int LeetCodeBinarySearch::maxDistance(vector<int>& position, int m)

{

std::sort(position.begin(), position.end());

int first = 1;

int last = position.back() - position[0];

int result = last;

while (first <= last)

{

int middle = first + (last - first) / 2;

int prev = position[0];

int curr = 1;

for (int j = 1; j < m; j++)

{

while (curr < (int)position.size() && position[curr] < prev + middle) curr++;

if (curr == position.size()) break;

prev = position[curr];

}

if (curr == position.size())

{

last = middle - 1;

}

else

{

result = middle;

first = middle + 1;

}

}

return result;

}

## 1760. Minimum Limit of Balls in a Bag

Medium

You are given an integer array nums where the ith bag contains nums[i] balls. You are also given an integer maxOperations.

You can perform the following operation at most maxOperations times:

* Take any bag of balls and divide it into two new bags with a **positive**number of balls.
  + For example, a bag of 5 balls can become two new bags of 1 and 4 balls, or two new bags of 2 and 3 balls.

Your penalty is the **maximum** number of balls in a bag. You want to **minimize** your penalty after the operations.

Return *the minimum possible penalty after performing the operations*.

**Example 1:**

**Input:** nums = [9], maxOperations = 2

**Output:** 3

**Explanation:**

- Divide the bag with 9 balls into two bags of sizes 6 and 3. [**9**] -> [6,3].

- Divide the bag with 6 balls into two bags of sizes 3 and 3. [**6**,3] -> [3,3,3].

The bag with the most number of balls has 3 balls, so your penalty is 3 and you should return 3.

**Example 2:**

**Input:** nums = [2,4,8,2], maxOperations = 4

**Output:** 2

**Explanation:**

- Divide the bag with 8 balls into two bags of sizes 4 and 4. [2,4,**8**,2] -> [2,4,4,4,2].

- Divide the bag with 4 balls into two bags of sizes 2 and 2. [2,**4**,4,4,2] -> [2,2,2,4,4,2].

- Divide the bag with 4 balls into two bags of sizes 2 and 2. [2,2,2,**4**,4,2] -> [2,2,2,2,2,4,2].

- Divide the bag with 4 balls into two bags of sizes 2 and 2. [2,2,2,2,2,**4**,2] -> [2,2,2,2,2,2,2,2].

The bag with the most number of balls has 2 balls, so your penalty is 2 an you should return 2.

**Example 3:**

**Input:** nums = [7,17], maxOperations = 2

**Output:** 7

**Constraints:**

* 1 <= nums.length <= 105
* 1 <= maxOperations, nums[i] <= 109

### Analysis:

Guess a maximum balls in one bag and see if you can split them in M operations.

/// <summary>

/// Leet code 1760. Minimum Limit of Balls in a Bag

///

/// Medium

///

/// You are given an integer array nums where the ith bag contains

/// nums[i] balls. You are also given an integer maxOperations.

///

/// You can perform the following operation at most maxOperations times:

///

/// Take any bag of balls and divide it into two new bags with a positive

/// number of balls.

/// For example, a bag of 5 balls can become two new bags of 1 and 4

/// balls, or two new bags of 2 and 3 balls.

/// Your penalty is the maximum number of balls in a bag. You want to

/// minimize your penalty after the operations.

///

/// Return the minimum possible penalty after performing the operations.

///

/// Example 1:

/// Input: nums = [9], maxOperations = 2

/// Output: 3

/// Explanation:

/// - Divide the bag with 9 balls into two bags of

/// sizes 6 and 3. [9] -> [6,3].

/// - Divide the bag with 6 balls into two bags of sizes 3 and 3.

/// [6,3] -> [3,3,3].

/// The bag with the most number of balls has 3 balls, so your

/// penalty is 3 and you should return 3.

///

/// Example 2:

/// Input: nums = [2,4,8,2], maxOperations = 4

/// Output: 2

/// Explanation:

/// - Divide the bag with 8 balls into two bags of sizes 4 and 4.

/// [2,4,8,2] -> [2,4,4,4,2].

/// - Divide the bag with 4 balls into two bags of sizes 2 and 2.

/// [2,4,4,4,2] -> [2,2,2,4,4,2].

/// - Divide the bag with 4 balls into two bags of sizes 2 and 2.

/// [2,2,2,4,4,2] -> [2,2,2,2,2,4,2].

/// - Divide the bag with 4 balls into two bags of sizes 2 and 2.

/// [2,2,2,2,2,4,2] -> [2,2,2,2,2,2,2,2].

/// The bag with the most number of balls has 2 balls, so your penalty

/// is 2 an you should return 2.

///

/// Example 3:

/// Input: nums = [7,17], maxOperations = 2

/// Output: 7

///

/// Constraints:

/// 1. 1 <= nums.length <= 10^5

/// 2. 1 <= maxOperations, nums[i] <= 10^9

/// </summary>

int LeetCodeBinarySearch::minimumSize(vector<int>& nums, int maxOperations)

{

int first = 1;

int last = INT\_MIN;

for (size\_t i = 0; i < nums.size(); i++)

{

last = max(last, nums[i]);

}

int result = INT\_MAX;

while (first <= last)

{

int middle = first + (last - first) / 2;

int count = 0;

for (size\_t i = 0; i < nums.size(); i++)

{

if (nums[i] > middle) count += (nums[i] + middle - 1) / middle - 1;

}

if (count <= maxOperations)

{

result = min(result, middle);

last = middle - 1;

}

else

{

first = middle + 1;

}

}

return result;

}

## 1870. Minimum Speed to Arrive on Time

Medium

You are given a floating-point number hour, representing the amount of time you have to reach the office. To commute to the office, you must take n trains in sequential order. You are also given an integer array dist of length n, where dist[i] describes the distance (in kilometers) of the ith train ride.

Each train can only depart at an integer hour, so you may need to wait in between each train ride.

* For example, if the 1st train ride takes 1.5 hours, you must wait for an additional 0.5 hours before you can depart on the 2nd train ride at the 2 hour mark.

Return *the****minimum positive integer****speed****(in kilometers per hour)****that all the trains must travel at for you to reach the office on time, or*-1*if it is impossible to be on time*.

Tests are generated such that the answer will not exceed 107 and hour will have **at most two digits after the decimal point**.

**Example 1:**

**Input:** dist = [1,3,2], hour = 6

**Output:** 1

**Explanation:** At speed 1:

- The first train ride takes 1/1 = 1 hour.

- Since we are already at an integer hour, we depart immediately at the 1 hour mark. The second train takes 3/1 = 3 hours.

- Since we are already at an integer hour, we depart immediately at the 4 hour mark. The third train takes 2/1 = 2 hours.

- You will arrive at exactly the 6 hour mark.

**Example 2:**

**Input:** dist = [1,3,2], hour = 2.7

**Output:** 3

**Explanation:** At speed 3:

- The first train ride takes 1/3 = 0.33333 hours.

- Since we are not at an integer hour, we wait until the 1 hour mark to depart. The second train ride takes 3/3 = 1 hour.

- Since we are already at an integer hour, we depart immediately at the 2 hour mark. The third train takes 2/3 = 0.66667 hours.

- You will arrive at the 2.66667 hour mark.

**Example 3:**

**Input:** dist = [1,3,2], hour = 1.9

**Output:** -1

**Explanation:** It is impossible because the earliest the third train can depart is at the 2 hour mark.

**Constraints:**

* n == dist.length
* 1 <= n <= 105
* 1 <= dist[i] <= 105
* 1 <= hour <= 109
* There will be at most two digits after the decimal point in hour.

### Analysis:

Guess a speed to see if you can arrive on time, use binary search to close the range.

/// <summary>

/// Leet code 1870. Minimum Speed to Arrive on Time

///

/// Medium

///

/// You are given a floating-point number hour, representing the amount

/// of time you have to reach the office. To commute to the office, you

/// must take n trains in sequential order. You are also given an integer

/// array dist of length n, where dist[i] describes the distance

/// (in kilometers) of the ith train ride.

///

/// Each train can only depart at an integer hour, so you may need to

/// wait in between each train ride.

///

/// For example, if the 1st train ride takes 1.5 hours, you must wait for

/// an additional 0.5 hours before you can depart on the 2nd train ride

/// at the 2 hour mark.

/// Return the minimum positive integer speed (in kilometers per hour)

/// that all the trains must travel at for you to reach the office on

/// time, or -1 if it is impossible to be on time.

///

/// Tests are generated such that the answer will not exceed 10^7 and

/// hour will have at most two digits after the decimal point.

///

/// Example 1:

///

/// Input: dist = [1,3,2], hour = 6

/// Output: 1

/// Explanation: At speed 1:

/// - The first train ride takes 1/1 = 1 hour.

/// - Since we are already at an integer hour, we depart immediately at

/// the 1 hour mark. The second train takes 3/1 = 3 hours.

/// - Since we are already at an integer hour, we depart immediately at

/// the 4 hour mark. The third train takes 2/1 = 2 hours.

/// - You will arrive at exactly the 6 hour mark.

///

/// Example 2:

/// Input: dist = [1,3,2], hour = 2.7

/// Output: 3

/// Explanation: At speed 3:

/// - The first train ride takes 1/3 = 0.33333 hours.

/// - Since we are not at an integer hour, we wait until the 1 hour mark

/// to depart. The second train ride takes 3/3 = 1 hour.

/// - Since we are already at an integer hour, we depart immediately at

/// the 2 hour mark. The third train takes 2/3 = 0.66667 hours.

/// - You will arrive at the 2.66667 hour mark.

///

/// Example 3:

/// Input: dist = [1,3,2], hour = 1.9

/// Output: -1

/// Explanation: It is impossible because the earliest the third train

/// can depart is at the 2 hour mark.

///

/// Constraints:

/// 1. n == dist.length

/// 2. 1 <= n <= 10^5

/// 3. 1 <= dist[i] <= 10^5

/// 4. 1 <= hour <= 10^9

/// 5. There will be at most two digits after the decimal point in hour.

/// </summary>

int LeetCodeBinarySearch::minSpeedOnTime(vector<int>& dist, double hour)

{

int low = 1;

int high = 10000000;

int result = -1;

while (low <= high)

{

int speed = low + (high - low) / 2;

double time = 0;

for (size\_t i = 0; i < dist.size(); i++)

{

if (i == dist.size() - 1) time += (double)dist[i] / speed;

else time += (dist[i] + speed - 1) / speed;

if (time > hour) break;

}

if (time <= hour)

{

result = speed;

high = speed - 1;

}

else

{

low = speed + 1;

}

}

return result;

}

## 1891. Cutting Ribbons

Medium

You are given an integer array ribbons, where ribbons[i] represents the length of the ith ribbon, and an integer k. You may cut any of the ribbons into any number of segments of **positive integer** lengths, or perform no cuts at all.

* For example, if you have a ribbon of length 4, you can:
  + Keep the ribbon of length 4,
  + Cut it into one ribbon of length 3 and one ribbon of length 1,
  + Cut it into two ribbons of length 2,
  + Cut it into one ribbon of length 2 and two ribbons of length 1, or
  + Cut it into four ribbons of length 1.

Your goal is to obtain k ribbons of all the **same positive integer length**. You are allowed to throw away any excess ribbon as a result of cutting.

Return *the****maximum****possible positive integer length that you can obtain*k*ribbons of, or*0*if you cannot obtain*k*ribbons of the same length*.

**Example 1:**

**Input:** ribbons = [9,7,5], k = 3

**Output:** 5

**Explanation:**

- Cut the first ribbon to two ribbons, one of length 5 and one of length 4.

- Cut the second ribbon to two ribbons, one of length 5 and one of length 2.

- Keep the third ribbon as it is.

Now you have 3 ribbons of length 5.

**Example 2:**

**Input:** ribbons = [7,5,9], k = 4

**Output:** 4

**Explanation:**

- Cut the first ribbon to two ribbons, one of length 4 and one of length 3.

- Cut the second ribbon to two ribbons, one of length 4 and one of length 1.

- Cut the third ribbon to three ribbons, two of length 4 and one of length 1.

Now you have 4 ribbons of length 4.

**Example 3:**

**Input:** ribbons = [5,7,9], k = 22

**Output:** 0

**Explanation:** You cannot obtain k ribbons of the same positive integer length.

**Constraints:**

* 1 <= ribbons.length <= 105
* 1 <= ribbons[i] <= 105
* 1 <= k <= 109

### Analysis:

Guess the maximum length and see if you can cut to K slices. The minimum length is 1 and maximum length is max ribbon.

/// <summary>

/// Leet Code 1891. Cutting Ribbons

///

/// Medium

///

/// You are given an integer array ribbons, where ribbons[i] represents

/// the length of the ith ribbon, and an integer k. You may cut any of

/// the ribbons into any number of segments of positive integer lengths,

/// or perform no cuts at all.

///

/// For example, if you have a ribbon of length 4, you can:

/// Keep the ribbon of length 4,

/// Cut it into one ribbon of length 3 and one ribbon of length 1,

/// Cut it into two ribbons of length 2,

/// Cut it into one ribbon of length 2 and two ribbons of length 1, or

/// Cut it into four ribbons of length 1.

/// Your goal is to obtain k ribbons of all the same positive integer

/// length. You are allowed to throw away any excess ribbon as a result

/// of cutting.

///

/// Return the maximum possible positive integer length that you can

/// obtain k ribbons of, or 0 if you cannot obtain k ribbons of the

/// same length.

///

/// Example 1:

/// Input: ribbons = [9,7,5], k = 3

/// Output: 5

/// Explanation:

/// - Cut the first ribbon to two ribbons, one of length 5 and one of

/// length 4.

/// - Cut the second ribbon to two ribbons, one of length 5 and one of

/// length 2.

/// - Keep the third ribbon as it is.

/// Now you have 3 ribbons of length 5.

///

/// Example 2:

///

/// Input: ribbons = [7,5,9], k = 4

/// Output: 4

/// Explanation:

/// - Cut the first ribbon to two ribbons, one of length 4 and one of

/// length 3.

/// - Cut the second ribbon to two ribbons, one of length 4 and one of

/// length 1.

/// - Cut the third ribbon to three ribbons, two of length 4 and one of

/// length 1.

/// Now you have 4 ribbons of length 4.

///

/// Example 3:

/// Input: ribbons = [5,7,9], k = 22

/// Output: 0

/// Explanation: You cannot obtain k ribbons of the same positive integer

/// length.

///

/// Constraints:

/// 1. 1 <= ribbons.length <= 10^5

/// 2. 1 <= ribbons[i] <= 10^5

/// 3. 1 <= k <= 10^9

/// </summary>

int LeetCodeBinarySearch::maxLength(vector<int>& ribbons, int k)

{

int last = 0;

for (auto l : ribbons) last = max(last, l);

int first = 1;

int result = 0;

while (first <= last)

{

int middle = first + (last - first) / 2;

int count = 0;

for (size\_t i = 0; i < ribbons.size(); i++)

{

count += ribbons[i] / middle;

}

if (count >= k)

{

result = middle;

first = middle + 1;

}

else

{

last = middle - 1;

}

}

return result;

}

## 2064. Minimized Maximum of Products Distributed to Any Store

Medium

You are given an integer n indicating there are n specialty retail stores. There are m product types of varying amounts, which are given as a **0-indexed** integer array quantities, where quantities[i] represents the number of products of the ith product type.

You need to distribute **all products** to the retail stores following these rules:

* A store can only be given **at most one product type** but can be given **any** amount of it.
* After distribution, each store will have been given some number of products (possibly 0). Let x represent the maximum number of products given to any store. You want x to be as small as possible, i.e., you want to **minimize** the **maximum** number of products that are given to any store.

Return *the minimum possible* x.

**Example 1:**

**Input:** n = 6, quantities = [11,6]

**Output:** 3

**Explanation:** One optimal way is:

- The 11 products of type 0 are distributed to the first four stores in these amounts: 2, 3, 3, 3

- The 6 products of type 1 are distributed to the other two stores in these amounts: 3, 3

The maximum number of products given to any store is max(2, 3, 3, 3, 3, 3) = 3.

**Example 2:**

**Input:** n = 7, quantities = [15,10,10]

**Output:** 5

**Explanation:** One optimal way is:

- The 15 products of type 0 are distributed to the first three stores in these amounts: 5, 5, 5

- The 10 products of type 1 are distributed to the next two stores in these amounts: 5, 5

- The 10 products of type 2 are distributed to the last two stores in these amounts: 5, 5

The maximum number of products given to any store is max(5, 5, 5, 5, 5, 5, 5) = 5.

**Example 3:**

**Input:** n = 1, quantities = [100000]

**Output:** 100000

**Explanation:** The only optimal way is:

- The 100000 products of type 0 are distributed to the only store.

The maximum number of products given to any store is max(100000) = 100000.

**Constraints:**

* m == quantities.length
* 1 <= m <= n <= 105
* 1 <= quantities[i] <= 105

### Analysis:

Guess the maximum product k and (products + k - 1) / k will tell you how many stores you can distribute this product. If total stores are more than n, then it means the number is too small. Only less or equal to n stores can be valid answer.

/// <summary>

/// Leet Code 2064. Minimized Maximum of Products Distributed to Any Store

///

/// Medium

///

/// You are given an integer n indicating there are n specialty retail

/// stores. There are m product types of varying amounts, which are

/// given as a 0-indexed integer array quantities, where quantities[i]

/// represents the number of products of the ith product type.

///

/// You need to distribute all products to the retail stores following

/// these rules:

///

/// A store can only be given at most one product type but can be given

/// any amount of it.

/// After distribution, each store will have been given some number of

/// products (possibly 0). Let x represent the maximum number of products

/// given to any store. You want x to be as small as possible, i.e., you

/// want to minimize the maximum number of products that are given to any

/// store.

/// Return the minimum possible x.

/// Example 1:

/// Input: n = 6, quantities = [11,6]

/// Output: 3

/// Explanation: One optimal way is:

/// - The 11 products of type 0 are distributed to the first four stores

/// in these amounts: 2, 3, 3, 3

/// - The 6 products of type 1 are distributed to the other two stores in

/// these amounts: 3, 3

/// The maximum number of products given to any store is max(2, 3, 3, 3,

/// 3, 3) = 3.

///

/// Example 2:

/// Input: n = 7, quantities = [15,10,10]

/// Output: 5

/// Explanation: One optimal way is:

/// - The 15 products of type 0 are distributed to the first three stores

/// in these amounts: 5, 5, 5

/// - The 10 products of type 1 are distributed to the next two stores in

/// these amounts: 5, 5

/// - The 10 products of type 2 are distributed to the last two stores in

/// these amounts: 5, 5

/// The maximum number of products given to any store is max(5, 5, 5, 5,

/// 5, 5, 5) = 5.

///

/// Example 3:

/// Input: n = 1, quantities = [100000]

/// Output: 100000

/// Explanation: The only optimal way is:

/// - The 100000 products of type 0 are distributed to the only store.

/// The maximum number of products given to any store is max(100000) = 100000.

///

/// Constraints:

/// 1. m == quantities.length

/// 2. 1 <= m <= n <= 10^5

/// 3. 1 <= quantities[i] <= 10^5

/// </summary>

int LeetCodeBinarySearch::minimizedMaximum(int n, vector<int>& quantities)

{

int first = 1;

int last = 0;

int result = 0;

for (auto k : quantities) last = max(last, k);

while (first <= last)

{

int middle = first + (last - first) / 2;

int count = 0;

for (size\_t i = 0; i < quantities.size(); i++)

{

count += (quantities[i] + middle - 1) / middle;

if (count > n) break;

}

if (count <= n)

{

result = middle;

last = middle - 1;

}

else

{

first = middle + 1;

}

}

return result;

}

## 1918. Kth Smallest Subarray Sum

Medium

Given an integer array nums of length n and an integer k, return *the*kth ***smallest subarray sum****.*

A **subarray** is defined as a **non-empty** contiguous sequence of elements in an array. A **subarray sum** is the sum of all elements in the subarray.

**Example 1:**

**Input:** nums = [2,1,3], k = 4

**Output:** 3

**Explanation:** The subarrays of [2,1,3] are:

- [2] with sum 2

- [1] with sum 1

- [3] with sum 3

- [2,1] with sum 3

- [1,3] with sum 4

- [2,1,3] with sum 6

Ordering the sums from smallest to largest gives 1, 2, 3, 3, 4, 6. The 4th smallest is 3.

**Example 2:**

**Input:** nums = [3,3,5,5], k = 7

**Output:** 10

**Explanation:** The subarrays of [3,3,5,5] are:

- [3] with sum 3

- [3] with sum 3

- [5] with sum 5

- [5] with sum 5

- [3,3] with sum 6

- [3,5] with sum 8

- [5,5] with sum 10

- [3,3,5], with sum 11

- [3,5,5] with sum 13

- [3,3,5,5] with sum 16

Ordering the sums from smallest to largest gives 3, 3, 5, 5, 6, 8, 10, 11, 13, 16. The 7th smallest is 10.

**Constraints:**

* n == nums.length
* 1 <= n <= 2 \* 104
* 1 <= nums[i] <= 5 \* 104
* 1 <= k <= n \* (n + 1) / 2

### Analysis:

First, calculate prefix sum for all the array from left to right. Then Guess a number and check on every position, use binary search to see how many X elements fall in right which means you have X subarray end by position i with sum less than or equal to X, sum them up and to see if the total number more than K or less than K. Reduce the scope by binary search.

/// <summary>

/// Leet Code 1918. Kth Smallest Subarray Sum

///

/// Medium

///

/// Given an integer array nums of length n and an integer k, return the

/// kth smallest subarray sum.

///

/// A subarray is defined as a non-empty contiguous sequence of elements

/// in an array. A subarray sum is the sum of all elements in the subarray.

///

/// Example 1:

/// Input: nums = [2,1,3], k = 4

/// Output: 3

/// Explanation: The subarrays of [2,1,3] are:

/// - [2] with sum 2

/// - [1] with sum 1

/// - [3] with sum 3

/// - [2,1] with sum 3

/// - [1,3] with sum 4

/// - [2,1,3] with sum 6

/// Ordering the sums from smallest to largest gives 1, 2, 3, 3, 4, 6.

/// The 4th smallest is 3.

///

/// Example 2:

/// Input: nums = [3,3,5,5], k = 7

/// Output: 10

/// Explanation: The subarrays of [3,3,5,5] are:

/// - [3] with sum 3

/// - [3] with sum 3

/// - [5] with sum 5

/// - [5] with sum 5

/// - [3,3] with sum 6

/// - [3,5] with sum 8

/// - [5,5] with sum 10

/// - [3,3,5], with sum 11

/// - [3,5,5] with sum 13

/// - [3,3,5,5] with sum 16

/// Ordering the sums from smallest to largest gives 3, 3, 5, 5, 6, 8,

/// 10, 11, 13, 16. The 7th smallest is 10.

///

/// Constraints:

/// 1. n == nums.length

/// 2. 1 <= n <= 2 \* 10^4

/// 3. 1 <= nums[i] <= 5 \* 10^4

/// 4. 1 <= k <= n \* (n + 1) / 2

/// </summary>

int LeetCodeBinarySearch::kthSmallestSubarraySum(vector<int>& nums, int k)

{

vector<int> dp(nums.size()+1);

int last = 0;

for (size\_t i = 0; i < nums.size(); i++)

{

dp[i+1] = last + nums[i];

last += nums[i];

}

int first = 1;

int result = 0;

while (first <= last)

{

int middle = first + (last - first) / 2;

int count = 0;

for (size\_t i = 1; i < dp.size(); i++)

{

auto itr = lower\_bound(dp.begin(), dp.begin() + i, dp[i] - middle);

count += dp.begin() + i - itr;

}

if (count >= k)

{

result = middle;

last = middle - 1;

}

else

{

first = middle + 1;

}

}

return result;

}

## 1802. Maximum Value at a Given Index in a Bounded Array

Medium

You are given three positive integers: n, index, and maxSum. You want to construct an array nums (**0-indexed**)that satisfies the following conditions:

* nums.length == n
* nums[i] is a **positive** integer where 0 <= i < n.
* abs(nums[i] - nums[i+1]) <= 1 where 0 <= i < n-1.
* The sum of all the elements of nums does not exceed maxSum.
* nums[index] is **maximized**.

Return nums[index]*of the constructed array*.

Note that abs(x) equals x if x >= 0, and -x otherwise.

**Example 1:**

**Input:** n = 4, index = 2, maxSum = 6

**Output:** 2

**Explanation:** nums = [1,2,**2**,1] is one array that satisfies all the conditions.

There are no arrays that satisfy all the conditions and have nums[2] == 3, so 2 is the maximum nums[2].

**Example 2:**

**Input:** n = 6, index = 1, maxSum = 10

**Output:** 3

**Constraints:**

* 1 <= n <= maxSum <= 109
* 0 <= index < n

### Analysis:

You guess the number of peak, see if you can build the left, right and with sum within limit.

/// <summary>

/// Leet code 1802. Maximum Value at a Given Index in a Bounded Array

///

/// Medium

///

/// You are given three positive integers n, index and maxSum. You want

/// to construct an array nums (0-indexed) that satisfies the following

/// conditions:

///

/// nums.length == n

/// nums[i] is a positive integer where 0 <= i < n.

/// abs(nums[i] - nums[i+1]) <= 1 where 0 <= i < n-1.

/// The sum of all the elements of nums does not exceed maxSum.

/// nums[index] is maximized.

/// Return nums[index] of the constructed array.

/// Note that abs(x) equals x if x >= 0, and -x otherwise.

///

/// Example 1:

/// Input: n = 4, index = 2, maxSum = 6

/// Output: 2

/// Explanation: The arrays [1,1,2,1] and [1,2,2,1] satisfy all the conditions. There

/// are no other valid arrays with a larger value at the given index.

///

/// Example 2:

/// Input: n = 6, index = 1, maxSum = 10

/// Output: 3

///

/// Constraints:

/// 1. 1 <= n <= maxSum <= 10^9

/// 2. 0 <= index < n

/// </summary>

int LeetCodeBinarySearch::maxValue(int n, int index, int maxSum)

{

int first = 1;

int last = maxSum;

int result = 0;

while (first <= last)

{

int mid = first + (last - first) / 2;

int left = max((mid - index), 1);

int right = max(mid - ((n - 1) - index), 1);

int left\_extra = max((index + 1 - mid), 0);

int right\_extra = max((n - index - mid), 0);

long long sum = ((long long)left + (long long)mid) \* ((long long)mid - (long long)left + 1) / 2;

sum += ((long long)right + (long long)mid) \* ((long long)mid - (long long)right + 1) / 2;

sum -= mid;

sum += (long long)left\_extra + (long long)right\_extra;

if (sum > maxSum) last = mid - 1;

else

{

result = max(result, mid);

first = mid + 1;

}

}

return result;

}