# Binary Search – Guess Number

There is another type of binary search, which is not that obvious, you try to get K containers for an array, each container will contain a sum of a subarray (continuously). Your job is to calculate the size of the container.

From the first sight, this problem looks like a DP problem, it is correct. But using DP will normally get a O(N^2) time complexity, but using binary search will give you O(NlogN) time complexity.

The idea is that guess the container size from 0 to sum of the whole array, using binary search, and use each container size to check if the whole array can be packed into K container, if yes, try a smaller size container, if no, try a big one.

## 1011. Capacity To Ship Packages Within D Days

Medium

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

### 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. 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;

while (first < last)

{

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

int d = 1;

int sum = 0;

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

{

sum += weights[i];

if (sum > mid)

{

sum = weights[i];

d++;

// a single cargo may bust

if (sum > mid) d = D + 1;

if (d > D) break;

}

}

// ship too small

if (d > D)

{

first = mid + 1;

}

else

{

last = mid;

}

}

return first;

}

## 875. Koko Eating Bananas

Medium

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 <= piles.length <= 10^4
* piles.length <= H <= 10^9
* 1 <= piles[i] <= 10^9

### Analysis:

You guess the eating speed from 1 to maximum element in the array.

/// <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;

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

{

last = middle;

}

}

return first;

}

## 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];

while (low < high)

{

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

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

{

low = mid + 1;

}

else

{

high = mid;

}

}

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.

## 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 #592. Fraction Addition and Subtraction

/// </summary>

long long LeetCodeMath::gcd(long long a, long long b)

{

a = abs(a);

b = abs(b);

if (a < b) swap(a, b);

if (b == 0) return a;

while (a % b != 0)

{

a = a % b;

swap(a, b);

}

return b;

}

/// <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 LeetCodeMath::nthMagicalNumber(int N, int A, int B)

{

int C = A \* B / (int)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) / 2;

unsigned long long n = middle / A + middle / B - middle / C;

if (n < 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

/// 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.

/// </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];

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) high = mid;

else low = mid + 1;

}

return low;

}

## 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;

}