Problem Solving: Binary Search on Answer

Relevel
by Unacademy

What does it take?

As we would be concentrating on hands-on advanced problem-solving in this session, please revise the concepts covered in the last 2 sessions:

- Linear Search
- Binary Search

List of Problems Involved

- Pair of Topics
- Sqrt using Binary Search
- Number of sextuplets
- Splitting an array
- Subset sum between A and B
- Not a Triangle

Pair of Topics

The aim is to discover the number of pairs of indices {i, j} in two integer arrays A[] and B[] of equal size such that A[i] + A[j] > B[i] + B[j] and i < j.

The basic technique is to verify if A[i] + A[j] > B[i] + B[j] for all possible pairs of {i, j} in the specified arrays. The concept of nested loops can be used to do this.

Pair of Topics

The essential takeaway from the problem is that the stated condition may also be visualised as (ai-bi) + (aj-bj)> 0, which allows us to create a second array to represent the difference between the two arrays. Let's call this array D. As a result, the task boils down to finding combinations where Di+Dj>0. Now we can sort the D array, and for each associated element Di, we'll find the number of suitable pairs Di may form and save that number in a count variable. We may use the binary search to discover the upper bound of -Di for each element Di to find the number of suitable pairs it can make. All elements present after -Di will also make sense because the array is sorted. Because the array is sorted, all elements after -Di will make good pairs for Di. As a result, if the upper bound of -Di is x and the entire size of the array is n, the total number of pairs corresponding to Di is n-x. This method takes O(NlogN) time to complete.

- The given condition in the question can be rewritten as:

A[i] + A[j] > B[i] + B[j]
A[i] + A[j] - B[i] - B[j] > 0
(A[i] - B[i]) + (A[j] - B[j]) > 0

- Create a new array, D, to hold the difference between the elements in both arrays at the relevant index, i.e.

D[i] = A[i] - B[i]

- Sort the difference array D so that each element i is smaller than the ones to its right to ensure that the constraint i < j is satisfied.

- If the value in the difference array D is negative at index i all we have to do is identify the nearest place 'j' where the value is just more than -D[i], such that the value sums to greater than 0.

- Because the array is sorted, the Binary Search can be used to identify such index 'j'.

Solution: https://jsfiddle.net/kg054y7d/1/

Sqrt using Binary Search

Given a positive number n and precision p, find the square root of the number upto decimal places using binary search.

Examples:

Input : number = 50, precision = 3
Output : 7.071

Input : number = 10, precision = 4
Output : 3.1622

Sqrt using Binary Search

Approach

1. Because the square root of an integer is in the range 0 <= squareRoot <= number, set start and end to 0 and number, respectively.

2. Compare the given number to the square of the mid integer. The square root is found if it equals the number. Otherwise, depending on the situation, seek the same in the left or right side.

3. Begin computing the fractional portion once the integral part has been found.

4. Set the increment variable to 0.1 and compute the fractional component up to P places iteratively. Each iteration reduces the increment to 1/10th of its previous value.

5. Finally, return the computed result.

Solution: https://jsfiddle.net/xcu3vo8f/

Number of sextuplets (or six values) that satisfy an equation

Reference - https://www.spoj.com/problems/ABCDEF/
Sextuplets - combination of six values
Problem – We need to find number of sextuplets which satisfies the given equation -

$$\frac{a * b + c}{d} - e = f$$

Example -
Input - [1]
Output - 1 (a=b=c=d=e=f=1)

Input - [2,3]
Output - 4

Number of sextuplets (or six values) that satisfy an equation

Intuition – Since mathematical expression is provided in the question, we can reorder the expression. Now we will be having the Left Hand side and Right Hand side in the equation. We can consider different arrays and check if any number is common in both the arrays.

Approach – Let's see each step –

1. Create two arrays LHSArray for LHS side and RHSArray for RHS
2. Verify if any value from RHSValue is present in LHSArray
3. If value is present in LHSArray, increase the final count.

Time Complexity – If there are N numbers in a given arr. Then complexity will be O(N^3 LogN)

Code Link -> https://jsfiddle.net/re5bsqtd/

Splitting an Array

Reference - https://codeforces.com/edu/course/2/lesson/6/3/practice/contest/285083/problem/B

Problem – You will be given an array having N elements and a number K (1<=K<=N). We need to split our array into K segments such that the maximum sum of the segments is the minimum possible.

Example -
Input - [1, 2, 3, 4], K = 3
Output - 4 ({1,2}, {3}, {4} (Maximum sum is 4)

Input - [1, 3, 2, 4, 10, 8, 4, 2, 5, 3], K = 4
Output - 12

Splitting an Array

Intuition – Our task is to find the minimum of possible maximum sum of all segments. This can be done by keeping track of the sum and updating it based on the given condition.

Naive Approach – We can check for all possibilities and update our maximum sum. These possibilities can be found using backtracking which we have already seen in our previous sessions.

Efficient Approach – We can use Binary search to get an optimal solution. Let's see each step -

1. In binary search, minimum sum is 1 and maximum sum will be sum of all elements in array
2. Consider mid as maximum segment sum
3. Keep track of count of segments, till their sum is less than mid value.
4. If count is less than or equal to mid value, then mid value is possible else it will not
5. At last, find the minimum possible mid value which satisfies the condition.

Code Link -> https://jsfiddle.net/p5j4s7h1/

Time Complexity –
If there are N numbers in a given array and Sum is the sum of all elements of the array. Then complexity will be O(N*log(Sum))

Subsets having Sum between A and B

Reference - https://www.spoj.com/problems/SUBSUMS/

Problem – You will be given an array having N elements. We need to find the number of subsets which will have a sum between A and B.

Example -
Input - [1, -2, 3], A = -1, B = 2

Output - 5

Subsets - [0, {1}, {1,-2}, {-2,3}, {1,-2,3}]

Subsets having Sum between A and B

Intuition – Here, our main task is to find segments having sum between A and B. To get an optimal solution, we can use binary search technique here to divide our array and then apply logic to find the solution.
Naive Approach – We can generate all possible subsets and find the subsets having sum between A and B.

Efficient Approach – We can use Binary search to get an optimal solution. Let's see each step -

1. Divide the set into 2 subsets [0....N/2] and [(N/2+1)....(N-1)]
2. Generate all the subsets sum for above 2 sets.
3. Sort one of the sets and binary search the value which will give the sum for the specific value of another set.
4. Sort second set and for each element present in first set, search for the lower bound of A - S2[i] i.e. low and upper bound of B - S2[i] i.e.high
5. Subtract high - low to get the final output.

Code Link -> https://jsfiddle.net/qh52sxp0/

Time Complexity – If there are N numbers in a given array. Then complexity will be O(2^(N/2))

Not a Triangle

Reference - https://www.spoj.com/problems/NOTATRI/

Problem – You will be given a list of numbers (1 to N). Each number is considered as one stick. The ith stick will be having specific length. We need to play a game in this problem. We need to pick 3 sticks randomly, and if our opponent is able to form a triangle using those 3 sticks, he will win. Our task is to find the number of ways to select 3 sticks such that it is not possible for our opponent to form a triangle using them.

Example -

Input - [4,2,10]

Output - 1

Triplets - {4,2,10} -> 4+2 < 10(Triangle violating property)

Not a Triangle

Intuition – Here, we need to check if there is any triplet which is not creating any triangle. We can use the triangle property which says that the sum of two sides should be greater than the third side. We will check its opposite i.e. we will find triplets in which the sum of 2 numbers is less than the third one.
Naive Approach – Iterate 3 loops and verify that if there is a triplet such that the sum of two numbers is less than the third number.

Efficient Approach – We can use Binary search to get an optimal solution. Let's see each step -

1. Sort the given input array
2. Create a nested loop having first iteration will be from array (0 to n-1) and second iteration will be j (from i+1 to n-1)
3. Take the sum of both elements in a nested loop and check if there any number exists which is greater than sum. If yes, increase the final count.

Code Link -> https://jsfiddle.net/02cdtovr/

Time Complexity – If there are N numbers in a given array. Then complexity will be O(N^2*log N)

Space Complexity – If there are N numbers in a given array. Then complexity will be O(1)

Practice Questions

1. Write a program to find the length of the smallest substring in String S having a maximum number of distinct characters.

2. A string is given having either 0 or 1. Write a program to find the length of the largest substring in String S having more numbers of 0 than 1.

Upcoming Teaser

- Binary Search with minimax problem:
- Aggressive cows
- Book allocation
- Painter partition
- Median of sorted arrays
- Few more problems if time persists

Thank You!