Binary Search Problems - Medium - Hard

Created	@July 20, 2022 8:00 PM
Class	
Type	
Materials	
Reviewed	☐

Problem 1:

https://codeforces.com/contest/1324/problem/D

Given two arrays each of length n (n ≤ 2*10^5). Both the arrays represent the value of degree of interesting topic that can be taught in some class. First array (A) represents how much the topic is interesting for the teacher and the second array (B) represents how much the topic is interesting for the students. Ai represents how much the ith topic is interesting for teacher, and Bi represents how much the ith topic is interesting for students.

We call a pair of topic to be **good** pair, if (Ai + Aj) > (Bi + Bj) for (i < j) i.e. the pair of topic is more interesting for the teacher.

We need to count all possible good pairs of topics.

Ex:

A = [4,8,2,6,2] // teacher → i = 1, j = 2 → Ai = 8 Aj = 2 → Ai+Aj = 10

B = [4,5,4,1,3] // students → Bi = 5, Bj = 4 → Bi + Bj = 9 → 10 > 9

Ans: 7

All the good pairs are (0,1) (0,3) (1,2) (1,3) (1,4) (3,4) (2,3)

Hint 1: The problem can be solved with merge sort as well as binary search. And there are few advanced methods as well (we will not discuss)

Hint 2: Can we simplify the equation ?

Hint 3: You might have solved the problem of counting inversions in an array. Can we use it here ?

Solution:

We can easily solve this problem using Binary Search.

Let's try to simplify the equation

```
Ai + Aj > Bi + Bj
Ai - Bi > Bj - Aj
Ai - Bi > -(Aj - Bj)
Assume Ai - Bi as Ci
Ci > -Cj
```

From the above simplification we can say, we need to find all j's for a given i such that j > i and

`Ci > -Cj`

Let's say we can create a new array C, such that Ci = Ai - Bi, and then sort the array C in ascending order. Because sorting the array will help us to apply binary search. But why do we want to apply binary search ?

We need to find all indexes j such that `j > i` and `-Ci < Cj` (we converted the inequality by multiplying -1 on both sides), so for any value Ci, we need to count the number of values which are greater than negative of Ci.

We know how to get the first value greater than a given value in a sorter array → `upper_bound` Upper bound gives us the first index greater than a value x.

So in the array we need to find all values greater than -Ci, after the index i. We can use upper bound to get the first value greater than -Ci, because if we get this first value then all the values to the right of it are also going to be greater as the array is sorted.

Example:

A = [4,8,2,6,2]

B = [4,5,4,1,3]

C = [0,3,-2,5,-1] // Ci = Ai - Bi

now sort the array C in ascending order

C = [-2,-1,0,3,5]

now for i = 0, we need to find all `j > i` such that `-Ci < Cj`

C0 = -2 → -C0 = 2, so from index `1 to C.length - 1` we need to find all values greater than 2

Upper bound of 2 in C is index 3, so everything starting from index 3 to the last index will all be greater than -C0. For i = 0, we got `n-upper_bound(-C0)` → `n-upper_bound(2)` → `n-3` → `5-3` → 2

So for i = 0, we got 2 pairs.

Now for i = 1, we need to find all `j > i` such that `-Ci < Cj`

C1 = -1 → -C1 = 1, so from index `2 to C.length - 1` we need to find all values greater than 1

Upper bound of 1 in C is index 3, so again we got 2 more pairs.

For i = 2, we need to find all `j > i` such that `-Ci < Cj`

C2 = 0, -C2 = 0 , upper bound of 0 from index 3 to C.length - 1 is 3, and again we got 2 more pairs.

for i = 3, C3 = 3, -C3 = -3, so from index `4 to C.length - 1`

we need to find all values greater than -3 , we have only 1 value, so one more pair.

That's how we got 7 pairs.

What about the time complexity ?

`O(nlogn)` why ? Because for all the values almost we are doing binary search on the remaining array. Total N values, and binary search of logn on each values leads to nlogn.

Space: `O(n)` → due to array C.

```
function upperBound(arr, x, st) {
  let start = st, end = arr.length - 1;
  let ans = arr.length;
  while(start <= end) {
    let mid = Math.floor(start + (end - start) / 2);
    if(arr[mid] <= x) {
      // discard left half
      start = mid + 1;
    } else {
      // element at mid can be a potential answer
      ans = mid;
      end = mid - 1; // go and find something even better on left side
    }
  }
  return ans;
}

let a = [4,8,2,6,2]
let b = [4,5,4,1,3]
let c = []
for(let i = 0; i < a.length; i++) {
  c.push(a[i] - b[i]);
}
c.sort((a, b) => a-b);
let ans = 0;
for(let i = 0; i < c.length - 1; i++) {
  let ub = upperBound(c, -c[i], i+1);
  ans += c.length - ub;
}
console.log(ans);
```

Problem 2:

SPOJ.com - Problem NOTATRI

You have N (3 ≤ N ≤ 2,000) wooden sticks, which are labeled from 1 to N. The i-th stick has a length of L i (1 ≤ L i ≤ 1,000,000). Your friend has challenged you to a simple game: you will pick three

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

Given an array of integers of length n (n ≤ 2000) we need to find the count of triplets such that the triplets cannot form a triangle.

Example: [5,2,9,6]

Ans: 2 → (5,2,9) (2,9,6) these two triplets will not form a triangle.

Pre requisite: In order to solve the problem you need to have basic knowledge about the property of a triangle that is what combination of sides form a valid triangle. If you have three sides as a, b, c

```
1.a + b > c
2.a + c > b
3.b + c > a
```

All of the above three conditions should be true.

Hint 1: It is almost similar to the previous problem

Hint 2: A triangle will not be formed if any one of the 3 condition fails, i.e. for any pair a, b if we have a `c` such that a + b < c then this triplet will not form a triangle

Solution:

Brute Force: In the brute force approach we can try to form all possible triplets and then check what all triplets don't follow the property of a triangle.

This approach will require `O(n^3)`

```
for(let i = 0; i < n; i++)
  for(let j = i+1; j < n; j++)
    for(let k = j+1; k < n; k++)
      // console.log(a[i], a[j], a[k]) -> filter first and then print
```

How can we optimise ?

In order to find a triplet that doesn't form a triangle, we can try to find all the triplets (a,b,c) that follows the given condition i.e. `a+b < c`

If we somehow generate all possible `a,b` then we can try to find all the `c` such that `a+b < c`

To generate all possible `a,b` we can try to get all possible pairs in O(n^2) time. Now for a given pair, if we need to find all the values greater than it's sum, then if we have the access to the upper_bound then all the values from upper_bound are going to be greater than it.

[5,2,9,6,11] → [2,5,6,9,11]

let's say a = 2, b = 5, a+b = 7 → upper_bound will be index 3 → from here we got 2 triplet

let's say a = 2, b = 6, a+b = 8 → upper_bound will be index 3 → from here we got 2 triplet

So in the prev problem we were getting upper_bound of -Ci, instead of this we have to get upper bound of a+b in this question where a,b are all possible pairs.

Time Complexity: `O(n*n*logn)` → n^2 due to generation of all possible pairs and then for each pair logn for the upper bound.

Space: `O(1)`

Problem 3:

https://www.spoj.com/problems/ABCDEF/

https://www.spoj.com/problems/ABCDEF/

You are given an array of length n (n ≤ 100), we need to count all possible sextuples which follow the given equation

```
((a*b+c)/d ) - e = f // d != 0
```

Note: In a sextuple for this given question numbers can be repeated.

Example: [2,3] → one of the sextuples can be (a=2,b=3,c=2,d=2,e=2,f=2)

Ans: 4 , total 4 different sextuples will be there

Solution:

Brute force: In the brute force we can try to find all possible sextuples and then filter out the required one.

Time Complexity : `O(n^6)` This solution will give TLE

How can we optimise ?

```
// lets simplify the equation
((a*b+c)/d ) - e = f
((a*b+c)/d ) = f + e
(a*b+c) = (f + e)*d // d!=0
```

What is this simplified form denoting ?

The meaning is if we are able to find two triplets (a,b,c) and (d,e,f) such that the follow the above equation or we can say, if we put them in the above equation the values are equal then, these two triplets can be combined to form a sextuple.

Ex: (2,3,2) (2,2,2) → (2,3,2,2,2,2)

2*3 + 2 = 8

(2+2)*2 = 8

And, n can be as big as 100, so if we want to generate all possible triplets complexity will be `O(n^3)` so generating triplets will be within the bounds.

Home work : Now knowing that triplets needs to be compared can we somehow use binary search to solve the question ?