Two Pointers Technique

- Pointer = index
  - No relationship to C++ pointers **int **x;**
- It is not a specific algorithm. Just easy idea that might be effective for specific problems
- You probably coded it before, but don't know a name for it
- In 2010, with a Codeforces tag, the name become more popular
- I will utilize this tutorial

Two Pointers Technique

- Technique that uses 2 **constrained** indices (move of one **can** be limited by the another)
  - Typically each pointer iterates on O(N) array positions.
  - Hence overall **increment/decrement** is O(N)
- Applications
  - In sorted arrays, where we want to find some positions
  - Or cumulative array of positive numbers array (sorted)
  - Variable size sliding window, where we search for a window (range) of specific property (max sum)
  - Ad Hoc cases

Sum of 2 numbers problem

- It is one of the best problems to clarify the 2-pointers technique
- Given a **sorted** array A, having N integers. You need to find any **pair(i,j)** having sum as given number **X**.
  - O(N^2): 2 nested loops and compare the sum
  - O(Nlogn): For each array value V, binary search for X-V
  - O(N) using 2-pointers!

Sum of 2 numbers problem

- 2-pointers based on the sortedness of array
  - Let pointer(index) p1 on the first element of array
  - Let p2 on the last element of the array
  - Let Y = the sum of these 2 numbers
  - If Y > X => shift p2 to the left => decrease Y
  - If Y < X => shift p1 to the right => increase Y
  - Keep doing so untill Y == X or no way
  - Then each pointer moves O(N), total O(N)

Sum of 2 numbers problem

- Let A = {2, 4, 5, 7, 8, 20}, X = 11
  - P1 = 0, P2 = 5, Y = 2 + 20 = 22 > 11
  - The only thing we can do is to move **p2 left**
  - P1 = 0, P2 = 4, Y = 2 + 8 = 10 < 11
  - Now we need bigger sum => move **p1 right**
  - P1 = 1, P2 = 4, Y = 4 + 8 = 12 > 11
  - Again, move **p2 left** to decrease sum
  - P1 = 1, P2 = 3, Y = 4 + 7 = 11 == 11 (Found)

Sum of 2 numbers problem

```
#define lli long long

bool f(lli sum) {
    int l = 0, r = n - 1; //two pointers
    while ( l < r ) {
        if ( A[l] + A[r] == sum ) return 1;
        else if ( A[l] + A[r] > sum ) r--;
        else l++;
    }
    return 0;
}
```

Sliding Windows

- A window is a range with start/end indices
  - So by definition, we have a point for its start & end
  - **Fixed size window of length K**
  - In this windows, we have specific range and searching for a range with specific property. Easy to handle
  - **Variable size window**
  - In this windows, the window can be of any size. More tricky