Two Pointers

https://usaco.guide/silver/2P

CP Initiative
joincpi.org

Two Pointers

- Iterating two monotonic pointers across an array to search for a pair of indices satisfying some condition in linear time.
- Can be used to track intervals/subarrays.
- Can be used on a sorted array.

Example Problem 1

- You are given an array of n integers, and your task is to find two values (at distinct positions) whose sum is x.
    - Inputs
        - The first input line has two integers n and x
        - The second line has n integers a1,a2,...,an: the array values.
    - Output
        - Print two integers: the positive of the values. If these are several solutions, you may print any of them. If there are no solutions, print IMPOSSIBLE.
    - Constraints
        - 1≤n≤2⋅105
        - 1≤x,ai≤109
    - Example
        - Input
            - 4 8
            - 2 7 51
        - Output
            - 2 4

CP Initiative
joincpi.org

Example Solution

- First, let’s sort the array.
    - Consider two pointers at indices `i` and `j` where `j > i`.
        - When the right pointer moves leftwards, the sum of values decreases.
        - When the left pointer moves rightwards, the sum of values increases.

    - We can move the pointers to achieve `X`.
        - Start the right pointer at the highest number and the left pointer at the lowest number in the array.
        - If `sum < X`, move the left pointer (to the right).
        - Otherwise, move the right pointer (to the left).

CP Initiative
joincpi.org

Solution Code (C++)

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, target;
    cin >> n >> target;

    vector<pair<int, int>> values;
    // append the element and its index
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        values.push_back({x, i + 1});
    }

    sort(values.begin(), values.end());
```

CP Initiative
joincpi.org

Solution Code (C++)

```cpp
    int left = 0;
    int right = n - 1;
    while (left < right) {
        // adjust left and right pointers
        if (values[left].first + values[right].first > target) {
            right--;
        } else if (values[left].first + values[right].first < target) {
            left++;
        } else if (values[left].first + values[right].first == target) {
            cout << values[left].second << " " << values[right].second << endl;
            return 0;
        }
    }

    // print IMPOSSIBLE if we haven't found a pair
    cout << "IMPOSSIBLE" << endl;
}
```

CP Initiative
joincpi.org

Example Problem 2

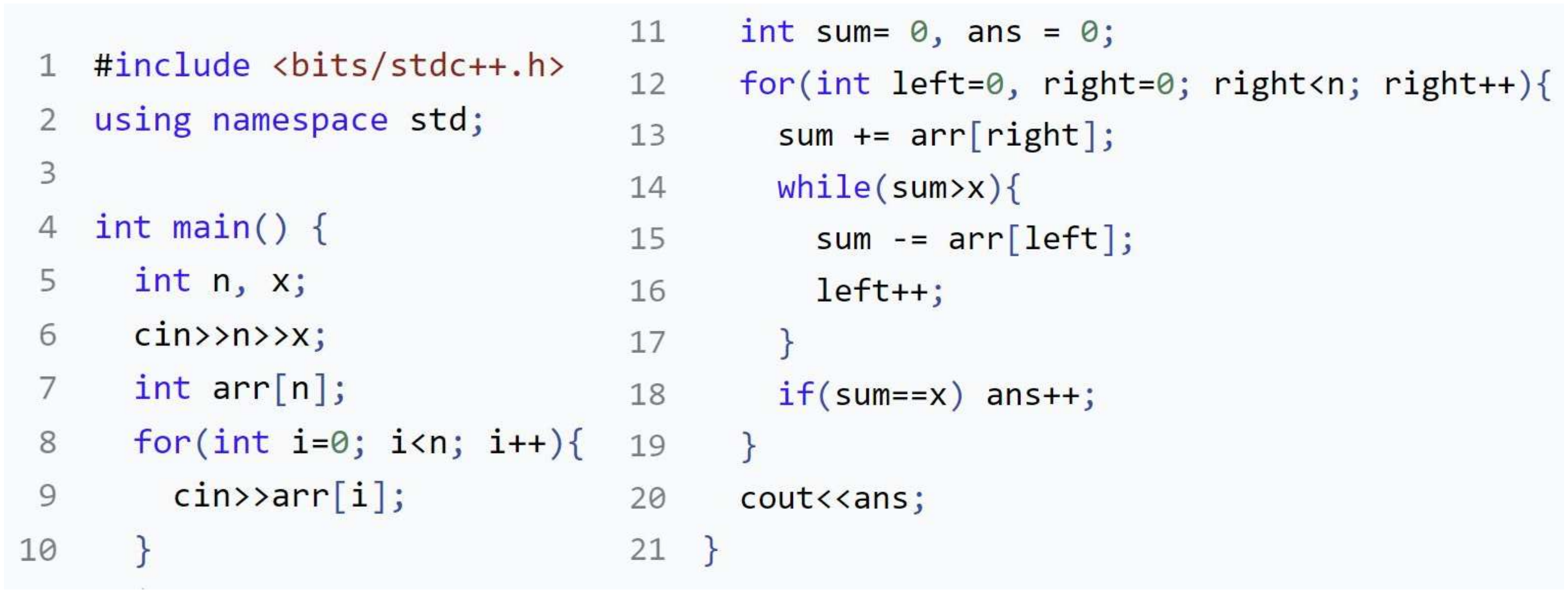
- Given an array of n positive integers, your task is to count the number of subarrays having sum x.
    - Input
        - The first input line has two integers n and x: the size of the array and the target sum x.
        - The next line has n integers a1,a2,…,an: the contents of the array.
    - Output
        - Print one integer: the required number of subarrays.
    - Constraints
        - 1≤n≤2⋅105
        - 1≤x,ai≤109
    - Example
        - Input:
            - 5 7
            - 2 4 1 2 7
        - Output:
            - 3

CP Initiative
joincpi.org

Subarray Sums I Solution Sketch

- The two pointers will both start at index 0.
- Keep two pointers for the start and end of the current subarray
- We maintain the sum of the current subarray
- If the sum of the current subarray is too small, increment the right pointer
    - Since all the elements are positive, the sum will always increase
- If the sum is too large, we increment the left pointer
- If the sum matches the target, we can increment both pointers

Solution Code (C++)



```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
  int n, x;
  cin>>n>>x;
  int arr[n];
  for(int i=0; i<n; i++){
    cin>>arr[i];
  }
  int sum= 0, ans = 0;
  for(int left=0, right=0; right<n; right++){
    sum += arr[right];
    while(sum>x){
      sum -= arr[left];
      left++;
    }
    if(sum==x) ans++;
  }
  cout<<ans;
}
```

CP Initiative
joincpi.org

Two Pointers Challenge Problem

- A Huge Tower
  - The ancient Babylonians decided to build a huge tower. The tower consists of N cubic building blocks that are stacked one onto another. The Babylonians gathered many building blocks of various sizes from all over the country. From their last unsuccessful attempt they have learned that if they put a large block directly onto a much smaller block, the tower will fall.
- Task specification
  - Each two building blocks are different, even if they have the same size. For each building block you are given its side length. You are also given an integer D with the following meaning: you are not allowed to put block A directly onto block B if the side length of A is strictly larger than D plus the side length of B.
  - Compute the number of different ways in which it is possible to build the tower using all the building blocks. Since this number can be very large, output the result modulo 109 + 9.
- Input specification
  - The first line of the input contains two positive integers N and D: the number of building blocks and the tolerance respectively.
  - The second line contains N space-separated integers; each represents the size of one building block.
- Constraints
  - All numbers in the input files are positive integers not exceeding 109. N is always at least 2.
  - In test cases worth 70 points N will be at most 70.
  - Out of those, in test cases worth 45 points, N will be at most 20.
  - Out of those, in test cases worth 10 points, N will be at most 10.
  - For some of the test cases the total number of valid towers will not exceed . These test cases are worth
  - For the last six test cases (worth 30 points) the value of N is larger than 70. No upper bound on N is given for these test cases.
  - 30 points in total.

CP Initiative
joincpi.org

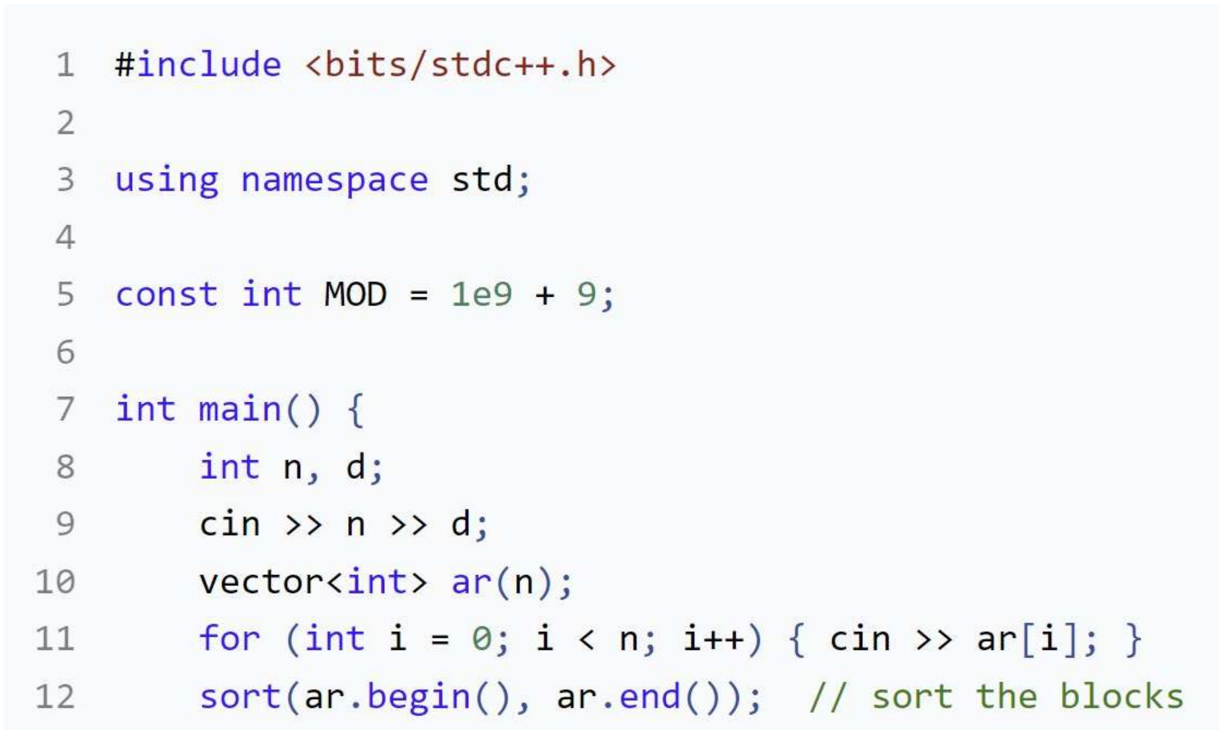
Two Pointers Challenge Problem (Cont.)

- Output specification
  - Output a single line containing a single integer: the number of towers that can be built, modulo $10^9 + 9$
- Examples
  - Input:
    - 4 1
    - 1 2 3 100
  - Output:
    - 4
  - We can arrange the first three blocks in any order, except for 2,1,3 or 1,3,2. The last block has to be at the bottom.
  - Input:
    - 6 9
    - 10 20 20 10 10 20
  - Output:
    - 36
  - We are not allowed to put a cube of size 20 onto a cube of size 10. There are six ways to order the cubes of size 10, and six ways to order the cubes of size 20.

A Huge Tower Solution Sketch

- Let $a_1, a_2, \ldots, a_n$ be the block sizes, sorted from least to greatest
- Let b_i be the number of indices j such that j<i and $a_i \leq a_j + d$
- Let ans_i be the answer if the tower consisted of only the blocks $a_1, a_2, \ldots, a_i$. Then, ans_n will be the final answer
- For *any* tower consisting of the first i−1 blocks, there will always be b_i+1 number of ways to insert the block of size a_i into the tower
    - The block of size aia_iai can always be inserted at the bottom of the tower, because there is no block larger than a_i in the tower
    - The block with size a_i can be inserted at the top of some block x with size a_x if and only if $a_i \leq a_j + d$.
- Thus, we see that $ans_i = ans_{i-1} * (b_i+1)$, because there are b_{i+1} valid ways to insert block a_i into any of the ans_{i-1} valid towers.

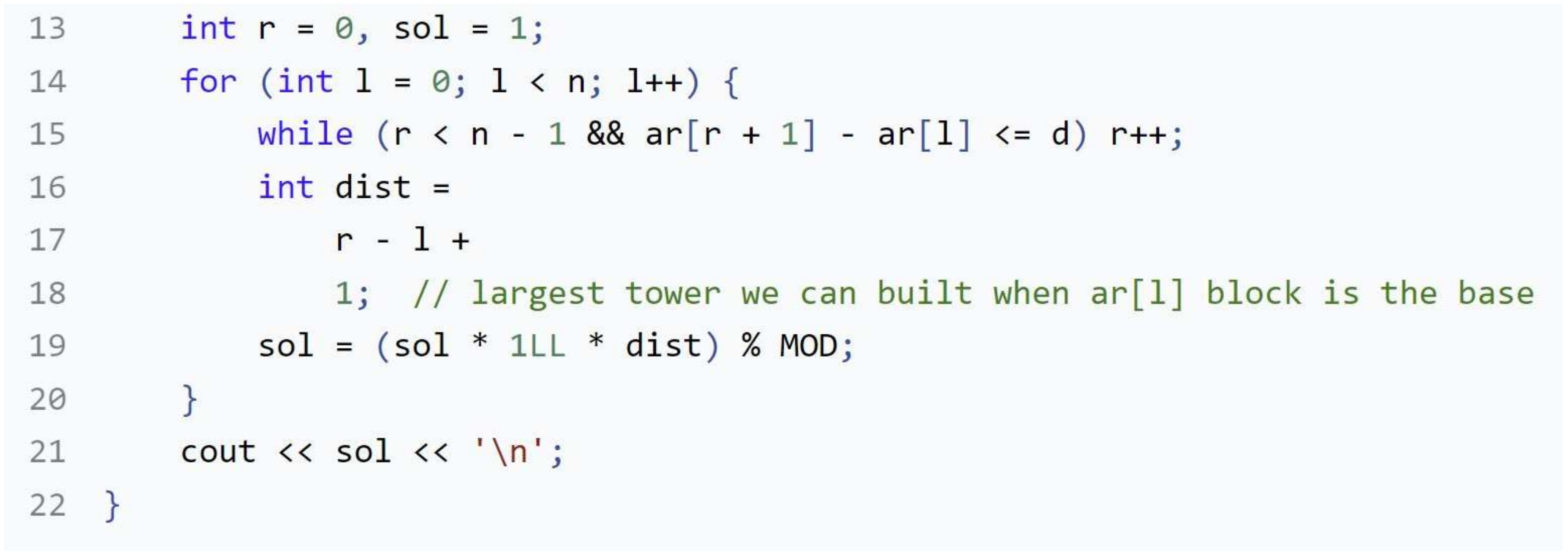
Two Pointers Challenge Problem Solution



```cpp
#include <bits/stdc++.h>

using namespace std;

const int MOD = 1e9 + 9;

int main() {
    int n, d;
    cin >> n >> d;
    vector<int> ar(n);
    for (int i = 0; i < n; i++) { cin >> ar[i]; }
    sort(ar.begin(), ar.end());  // sort the blocks
```

CP Initiative
joincpi.org

Two Pointers Challenge Problem Solution



```cpp
    int r = 0, sol = 1;
    for (int l = 0; l < n; l++) {
        while (r < n - 1 && ar[r + 1] - ar[l] <= d) r++;
        int dist =
            r - l +
            1;  // largest tower we can built when ar[l] block is the base
        sol = (sol * 1LL * dist) % MOD;
    }
    cout << sol << '\n';
}
```

CP Initiative
joincpi.org