G. Jolly Jumpers

Given an array a containing $0 < n \leq 3000$ integers. Check if $|a_i - a_{i+1}|$ takes on all values between 1 and $n-1$.

Solution

- Store (count) all values $|a_i - a_{i+1}|$ $(1 \leq i \leq n-1)$ into a *map* structure. Then, you check for all keys between 1 and $n-1$ if they are non-empty. If there is an empty key, then it means that the sequence is not a jolly jumper.
- This can be achieved by either using std::vector or std::map since $n \leq 3000$.

```
if n <= 1 then
    print JOLLY and return
for i = 1 to n-1 do
    diff = abs(a[i]-a[i+1]) // Note: diff can be greater than n-1!!
    cnt[diff] += 1
for d = 1 to n-1 do
    if cnt[d] <= 0 then
        print NOT JOLLY and return
print JOLLY
```

- Time Complexity: $O(n)$.

H. Good Sequence

A sequence b is a good sequence if: *For each element x in b, x has to occur exactly x times in b.* Given an array a containing N integers, find the minimum number of elements that need to be removed in order to make a a *good* sequence.

Solution

- If an element x occurs less than x times in a, we cannot add more x's in there. Then, the best move is to remove all occurrences of x.
- If an element x occurs y times such that $y \geq x$. Then, we should remove $y - x$ occurrences.
- Let cnt_x be the number of x's in a. If $cnt_x < x$ then $ans := ans + cnt_x$. Otherwise, $ans := ans + (cnt_x - x)$.

```
for x = 1 to 1e9 do // Note: TLE => How can you optimize this?
    if cnt[x] < x then ans += cnt[x]
    else then ans += (cnt[x]-x)
print ans and return
```

- Time Complexity: $O(n)$

I. Two Sets

Divide all the numbers 1, 2, 3, ..., n into two sets of equal sum or state that it is impossible.

Solution

- If the sum of all elements is odd, then it is obviously IMPOSSIBLE. Recall that the sum can be computed using the formula $S_n = \frac{n(n+1)}{2}$
- Suppose we divide the numbers into two sets set_1 and set_2 respectively. The sum of all the elements in set_1 is equal to set_2, which is also equal to $\frac{S_n}{2}$. Then, if we could construct the first set set_1 such that the sum of its elements is equal to $\frac{S_n}{2}$, then the remaining elements in the original sequence can be used to construct the set_2.
- We can start by adding elements one-by-one into set_1, *starting with the largest elements first*, while keeping track of the sum sum_1. Consider an element x in the original set, if $sum_1 + x > \frac{S_n}{2}$, then we add x to set_2. Otherwise, we add x to set_1. If at any point $sum_1 = \frac{S_n}{2}$, we stop. The rest of the elements are added then to set_2.

```
total_sum = n*(n+1)/2
if total_sum is odd then
    print NO and return
half_sum = total_sum / 2
for x = N to 1 do
    if current_sum + x <= half_sum then
        add x to set_1
    else then
        add x to set_2
print answer and return
```

- Time Complexity: $O(n)$

Extra. If you start by adding elements from 1 to n instead of the other way around, you would notice that the solution no longer works. Can you prove why this is the case?

J. Boxes Packing

You are given n boxes where the i-th box has the side length of a_i. Find the minimum number of visible boxes after packing. *Recall that box i can be put into box j iff (1) $a_i < a_j$, and (2) neither box i nor j contains any boxes.*

Solution

- Suppose we consider a box of size x, we can pack all boxes of sizes less than x inside of it, making them a single visible unit.
- The process of packing is as follows:
  1. Start with a box with the biggest size in the batch.
  2. For all boxes strictly smaller than the original box, we take one for each size and pack them into each other.
  3. After the first two steps, we have constructed ONE visible unit.
  4. Repeat (1) and (2) until we have no free boxes left.

  Since the number of boxes for each size decreases, the process will eventually terminate We can also prove that the number of visible units is minimal, due to the strict order of the boxes we consider in step (2). A helpful data structure to use is a *map* structure. You can simulate this process which will take $O(n^2)$ time.
- To optimize this solution it is helpful to observe that: *if we have several boxes of the same size, it is clear that we cannot pack them inside one bigger box (if it exists) because they will not fit into each other. Therefore, we must separate them into several visible units.*
- Then the answer to the problem is *maximum number of boxes of any size.*

```
// cnt[x] = count of boxes with side length x
ans = -1
for i = 1 to N do
    ans = max(ans, cnt[a[i]])
print ans and return
```