Leetcode 456. 132 Pattern

Priyam Mehta @priyam6f4

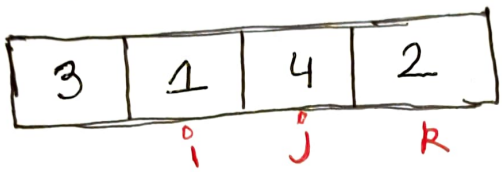
Problem : Given an integer array `nums` a 132 pattern is a "subsequence of three integers nums[i], nums[j], nums[k] such that, i < j < k and nums[i] < nums[k] < nums[j].

Return true if its a 132 pattern else return false.

Solution : The twist here is, the actual value of the k pointer (nums[k]) is greater than nums[i] but less than nums[j].

- But, the i and j pointers should be less than k, but in the given problem (nums[k]) is in between n[i] and n[j].

So technically the pointer will be at the highest position but its value will be greater than i and less than j.



Here as we can see there exists a 132 pattern where nums[k] > nums[i] and nums[k] < nums[j].

- nums[k] to be greater than nums[i].
- nums[k] to be ~~greater~~ lesser than nums[j].
- nums[k] to be greater than nums[i].

nums[i] < nums[k] < nums[j]

So if there are three pointers i, j and k such that in the order

i j k

for the nums[k] we need to check if any larger value is present? If it is, then it is nums[j]. And we also need to check that whether any smaller value than both nums[j] and nums[k] is there.

Solution : Monotonic Decreasing Stack Data Structure

- We maintain a stack which stores the values in a "monotonically decreasing" manner.
- There are only decreasing values in the stack, and the mast min. at every iteration is stored as a pair.
- This means that, the moment a larger value is found than the previous two values, that larger value is not popped and the stack operation proceeds further.

Algorithm :

Priyam Mehta @priyam6414

(1) Make a Pair class that will store a Pair of integers in the stack.

```
class Pair {
int num;
int min;
Pair (int num, int min) {
  this.num = num;
  this.min = min;
}
}
```

Method to create a pair in Java.

(2) While iterating if we find any element larger than all of the elements, we start popping out other elements and the minimum element is already stored.

• Note : The stack is designed to be a "Monotone Decreasing" stack.

So for the case

3	1	4	2

for the data, we have the

1) Maximum of all elements
2) Minimum of all elements

Now we need to find the element between the minimum and the maximum.

3	1	4	2

The Stack is doing the work of storing the most minimum element and the largest element is at the top.

(3) Now, check if the current iterating element nums[i] is less than the peek() (largest element) and greater than the stack.peek().min.

(4) Default Operation : Keep pushing nums[i] and min