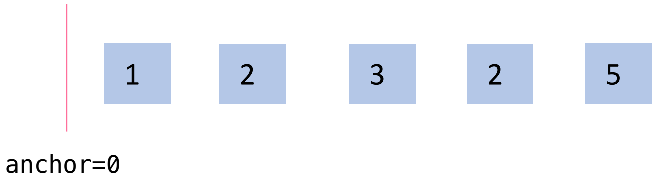
Sliding window:

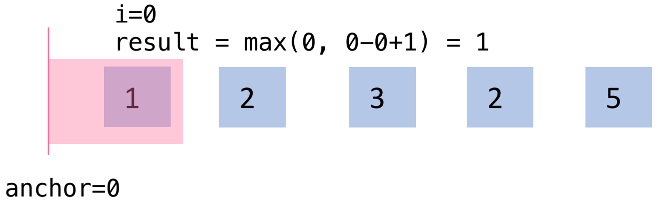
Let’s say you are given the problem: for a given array *L* find the longest increasing subsequence, defined as a contiguous range made up of only increasing numbers. If *L* is [1, 2, 3, 2, 5], the increasing subsequences of this array are [1, 2, 3] and [2, 5]. Of those, [1, 2, 3] is longer, so we return len([1,2,3]) => 3.

We instantiate our left-hand boundary, “anchor”, and our result, “result”, as 0. In our index-based loop, we check for if i>0 (as to prevent comparing the first and last indices) and if we are *not* still in an increasing subsequence. That’s defined by the value at the previous index of *L* being >= the value at our current index of *L*. If this is true, our left-hand boundary *anchor* must be dragged across to our current *i* as to reset the size of our window to *0*.

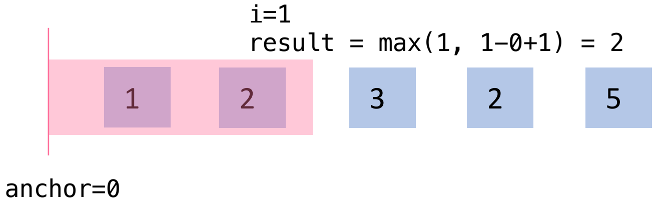
Let’s follow along. We start with *anchor* being 0,



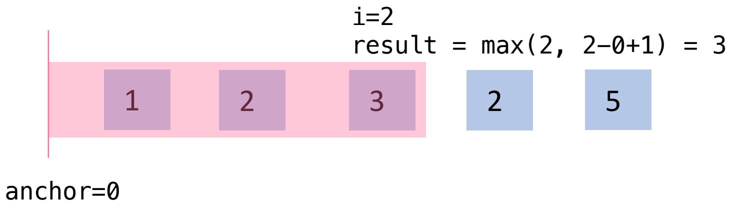
As we iterate over the indices of *L*, so long as *i*>0 we check to see if val. @ index *i*-1 >= index @ our current val.. Let’s see how our window (in pink) changes:



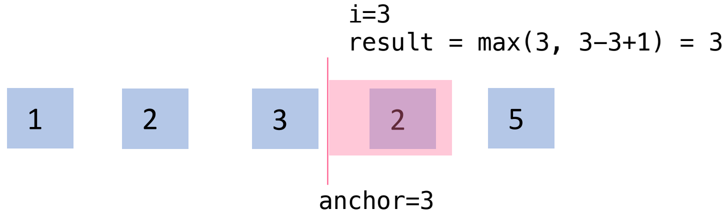
Now *result* has been set to 1, since our window covers 1 number. When *i* =1, we evaluate the statement `L[i-1] >= L[i]` as `L[1-1] >= L[1]` , which equals `1 >= 2` . This is false, indicating our subsequence is still increasing. That means our pink window will get 1 larger:



This continues for *i*=2, as `L[i-1] >= L[i]` --> `L[2-1] >= L[2]` --> `L[1] >= L[2]--> 2 >= 3` . The statement is False, meaning our anchor does *not* get moved, and our window keeps increasing.

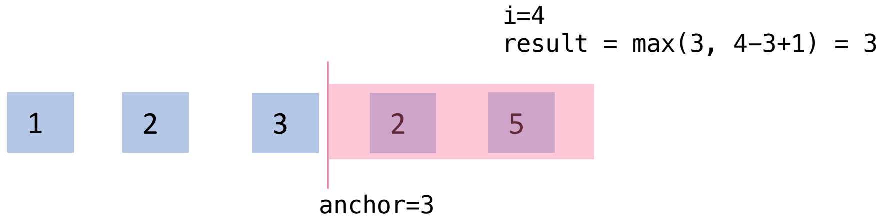


When we hit *i*=3, let's see what happens:



Since `L[i-1] >= L[i]` --> `L[3-1] >= L[3]` --> `L[2] >= L[3]--> 3 >= 2` evaluates to *True*, our previous val is bigger than our current, we drag our anchor our to *i*, which is 3, effectively resetting the size of our window to 1, only encompassing the current val.

Finally, we go over to the last index:



The code for this is shown below:

```
def longest_sub(L) -> int:

        anchor, result =0,0
        for i in range(len(L)):
                if i>0 and L[i-1] >= L[i]: anchor = i
                result = max(result, i - anchor + 1)
        return result
```