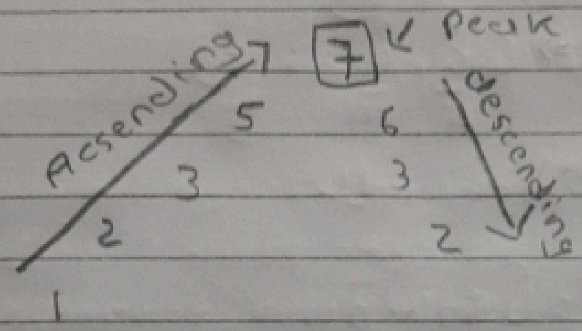
6] Peak in Mountain Array

arr = [1, 2, 3, 5, 7, 6, 3, 2]



- we have to find Peak(max) in mountain array.

```
 ↓s                 ↓m   *              ↓e
 0    1    2    3    4    5    6    7
[1,   2,   3,   5,   7,   6,   3,   2]
                m   Peak
                     ↓
```

* => case => 1

```
if (arr[mid] > arr[mid+1]) {
    // you are in descending part of array
    // it may be potential ans. but look
    further more in left side by
                                  s[ m]   e
                                     ↑e ←
        end = mid;
}
```

* => case => 2

```
if (arr[mid] < arr[mid+1]) { => 5 < 7 => yes
    // you are in ascending part of array
    // search in right side by
    start = mid + 1;              s  m|  e|
                                  ⟶ s    ↑e
}
```

=) When loop is break let's see ?

```
       0  1  2  3  4  5  6  7
    [  1, 2, 3, 5, 6, 4, 3, 2 ]
       s        m           e
```

(case 2) mid < mid+1
↓
s = mid+1 => [6, 4, 3, 2] (indices 4, 5, 6, 7; s at 6, m at 4, e at 2)

(case 1)
now, mid > mid+1
↓
e = mid => [6, 4] (indices 4, 5; m, s at 6, e at 4)

⇓

[6] (index 4; s and e) =) In the end, start & end both point to target element because of case 1 & case 2.

loop will be run
while (start < end)

- So our ans is start or end which is peak in array.

*) Let's code

```
peakInMountain(int[] arr){

    int start = 0;
    int end = l;

    while(start < end){
        int mid = start + (end-start)/2;
```

```
        if (arr[mid] > arr[mid+1]){
            end = mid;
        }
        else {
            start = mid+1;
        }

    }

    return start; // this is answer we can
                  return start or end
}                 because both are pointing
                  to peak element after
                  loop is break.
```