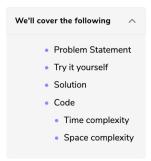
(3)



Bitonic Array Maximum (easy)



Problem Statement

Find the maximum value in a given Bitonic array. An array is considered bitonic if it is monotonically increasing and then monotonically decreasing. Monotonically increasing or decreasing means that for any index i in the array arr[i] != arr[i+1].

Example 1:

```
Input: [1, 3, 8, 12, 4, 2]
Output: 12
Explanation: The maximum number in the input bitonic array is '12'.
```

Example 2:

```
Input: [3, 8, 3, 1]
Output: 8
```

Example 3:

```
Input: [1, 3, 8, 12]
Output: 12
```

Example 4:

```
Input: [10, 9, 8]
Output: 10
```

Try it yourself

Try solving this question here:

Solution

A bitonic array is a sorted array; the only difference is that its first part is sorted in ascending order and the second part is sorted in descending order. We can use a similar approach as discussed in Order-agnostic Binary Search. Since no two consecutive numbers are same (as the array is monotonically increasing or decreasing), whenever we calculate the middle, we can compare the numbers pointed out by the index middle and middle+1 to find if we are in the ascending or the descending part. So:

minare and minare. I come in the are in an accounting or are accounting bare on

- 1. If arr[middle] > arr[middle + 1], we are in the second (descending) part of the bitonic array. Therefore, our required number could either be pointed out by middle or will be before middle. This means we will be doing: end = middle.
- 2. If <code>arr[middle] < arr[middle + 1]</code>, we are in the first (ascending) part of the bitonic array. Therefore, the required number will be after <code>middle</code>. This means we will be doing: <code>start = middle + 1</code>.

We can break when start == end. Due to the two points mentioned above, both start and end will be pointing at the maximum number of the bitonic array.

Code

Here is what our algorithm will look like:

```
🦆 Python3
                             ⊚ C++
👙 Java
      lass MaxInBitonicArray {
        public static int findMax(int[] arr) {
         int start = 0, end = arr.length - 1;
while (start < end) {</pre>
            int mid = start + (end - start) / 2;
            if (arr[mid] > arr[mid + 1]) {
              end = mid;
              start = mid + 1;
          return arr[start];
       public static void main(String[] args) {
          System.out.println(MaxInBitonicArray.findMax(new int[] { 1, 3, 8, 12, 4, 2 }));
         System.out.println(MaxInBitonicArray.findMax(new int[] { 3, 8, 3, 1 }));
System.out.println(MaxInBitonicArray.findMax(new int[] { 1, 3, 8, 12 }));
          System.out.println(MaxInBitonicArray.findMax(new int[] { 10, 9, 8 }));
 Run
                                                                                                                Reset []
```

Time complexity

Since we are reducing the search range by half at every step, this means that the time complexity of our algorithm will be O(logN) where 'N' is the total elements in the given array.

Space complexity

The algorithm runs in constant space O(1).

