

Bitonic Array Maximum (easy)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity

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:

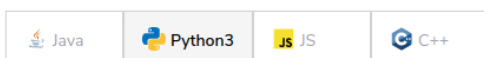
```
Input: [1, 3, 8, 12]
Output: 12
```

Example 4:

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

Try it yourself

Try solving this question here:



```
1 def find_max_in_bitonic_array(arr):
2     # TODO: Write your code here
3     return -1
4
5
6 def main():
7     print(find_max_in_bitonic_array([1, 3, 8, 12, 4, 2]))
8     print(find_max_in_bitonic_array([3, 8, 3, 1]))
9     print(find_max_in_bitonic_array([1, 3, 8, 12]))
10    print(find_max_in_bitonic_array([10, 9, 8]))
11
12
13 main()
14
```

Run

Save

Reset



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:

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 `arr[middle] < arr[middle + 1]`, we are in the first (ascending) part of the bitonic array. Therefore, the required number will be after `middle`. This means we will be doing: `start = middle + 1`.

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:

Java

Python3

C++

JS

```

1 def find_max_in_bitonic_array(arr):
2     start, end = 0, len(arr) - 1
3     while start < end:
4         mid = start + (end - start) // 2
5         if arr[mid] > arr[mid + 1]:
6             end = mid
7         else:
8             start = mid + 1
9
10    # at the end of the while loop, 'start == end'
11    return arr[start]
12
13
14 def main():
15     print(find_max_in_bitonic_array([1, 3, 8, 12, 4, 2]))
16     print(find_max_in_bitonic_array([3, 8, 3, 1]))
17     print(find_max_in_bitonic_array([1, 3, 8, 12]))
18     print(find_max_in_bitonic_array([10, 9, 8]))
19
20
21 main()
22

```

Run

Save

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 $O(\log N)$ where 'N' is the total elements in the given array.

Space complexity

The algorithm runs in constant space $O(1)$.

[← Back](#)
[Next →](#)

 Completed

 Report an Issue  Ask a Question