Bitonic Array Maximum (easy)

We'll cover the following

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:

Java Python3 JS C++

```python
def find_max_in_bitonic_array(arr):
  # TODO: Write your code here
  return -1


def main():
  print(find_max_in_bitonic_array([1, 3, 8, 12, 4, 2]))
  print(find_max_in_bitonic_array([3, 8, 3, 1]))
  print(find_max_in_bitonic_array([1, 3, 8, 12]))
  print(find_max_in_bitonic_array([10, 9, 8]))


main()

```

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

```python
def find_max_in_bitonic_array(arr):
  start, end = 0, len(arr) - 1
  while start < end:
    mid = start + (end - start) // 2
    if arr[mid] > arr[mid + 1]:
      end = mid
    else:
      start = mid + 1

  # at the end of the while loop, 'start == end'
  return arr[start]


def main():
  print(find_max_in_bitonic_array([1, 3, 8, 12, 4, 2]))
  print(find_max_in_bitonic_array([3, 8, 3, 1]))
  print(find_max_in_bitonic_array([1, 3, 8, 12]))
  print(find_max_in_bitonic_array([10, 9, 8]))


main()
```

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

Space complexity

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

← Back Next →