Ceiling of a Number (medium)

We'll cover the following

Problem Statement

Given an array of numbers sorted in an ascending order, find the ceiling of a given number 'key'. The ceiling of the 'key' will be the smallest element in the given array greater than or equal to the 'key'.

Write a function to return the index of the ceiling of the 'key'. If there isn't any ceiling return -1.

Example 1:

```
Input: [4, 6, 10], key = 6
Output: 1
Explanation: The smallest number greater than or equal to '6' is '6' having index '1'.
```

Example 2:

```
Input: [1, 3, 8, 10, 15], key = 12
Output: 4
Explanation: The smallest number greater than or equal to '12' is '15' having index '4'.
```

Example 3:

```
Input: [4, 6, 10], key = 17
Output: -1
Explanation: There is no number greater than or equal to '17' in the given array.
```

Example 4:

```
Input: [4, 6, 10], key = -1
Output: 0
Explanation: The smallest number greater than or equal to '-1' is '4' having index '0'.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

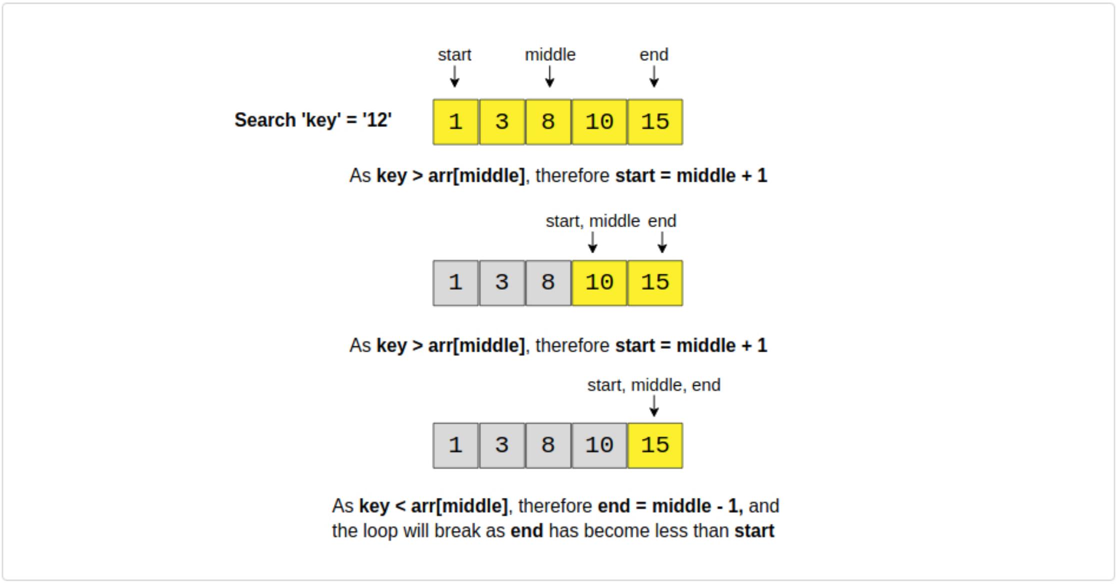
```
def search_ceiling_of_a_number(arr, key):
  # TODO: Write your code here
  return -1


def main():
    print(search_ceiling_of_a_number([4, 6, 10], 6))
    print(search_ceiling_of_a_number([1, 3, 8, 10, 15], 12))
    print(search_ceiling_of_a_number([4, 6, 10], 17))
    print(search_ceiling_of_a_number([4, 6, 10], -1))


main()

```

Run Save Reset

Solution

This problem follows the **Binary Search** pattern. Since Binary Search helps us find a number in a sorted array efficiently, we can use a modified version of the Binary Search to find the ceiling of a number.

We can use a similar approach as discussed in Order-agnostic Binary Search. We will try to search for the 'key' in the given array. If we find the 'key', we return its index as the ceiling. If we can't find the 'key', the next big number will be pointed out by the index `start`. Consider Example-2 mentioned above:



start
middle
end
Search 'key' = '12'
1
3
8
10
15
As **key > arr[middle]**, therefore **start = middle + 1**
start, middle
end
As **key > arr[middle]**, therefore **start = middle + 1**
start, middle, end
As **key < arr[middle]**, therefore **end = middle - 1**, and
the loop will break as **end** has become less than **start**

Since we are always adjusting our range to find the 'key', when we exit the loop, the start of our range will point to the smallest number greater than the 'key' as shown in the above picture.

We can add a check in the beginning to see if the 'key' is bigger than the biggest number in the input array. If so, we can return '-1'.

Code

Here is what our algorithm will look like:

Java Python3 C++ JS

```python
def search_ceiling_of_a_number(arr, key):
  n = len(arr)
  if key > arr[n - 1]:  # if the 'key' is bigger than the biggest element
    return -1

  start, end = 0, n - 1
  while start <= end:
    mid = start + (end - start) // 2
    if key < arr[mid]:
      end = mid - 1
    elif key > arr[mid]:
      start = mid + 1
    else:  # found the key
      return mid
  # since the loop is running until 'start <= end', so at the end of the while loop, 'start == end+1'
  # we are not able to find the element in the given array, so the next big number will be arr[start]
  return start


def main():
  print(search_ceiling_of_a_number([4, 6, 10], 6))
  print(search_ceiling_of_a_number([1, 3, 8, 10, 15], 12))
  print(search_ceiling_of_a_number([4, 6, 10], 17))
  print(search_ceiling_of_a_number([4, 6, 10], -1))


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 $O(1)$.

Similar Problems

Problem 1

Given an array of numbers sorted in ascending order, find the floor of a given number 'key'. The floor of the 'key' will be the biggest element in the given array smaller than or equal to the 'key'

Write a function to return the index of the floor of the 'key'. If there isn't a floor, return -1.

Example 1:

```
Input: [4, 6, 10], key = 6
Output: 1
Explanation: The biggest number smaller than or equal to '6' is '6' having index '1'.
```

Example 2:

```
Input: [1, 3, 8, 10, 15], key = 12
Output: 3
Explanation: The biggest number smaller than or equal to '12' is '10' having index '3'.
```

Example 3:

```
Input: [4, 6, 10], key = 17
Output: 2
Explanation: The biggest number smaller than or equal to '17' is '10' having index '2'.
```

Example 4:

```
Input: [4, 6, 10], key = -1
Output: -1
Explanation: There is no number smaller than or equal to '-1' in the given array.
```

Code

The code is quite similar to the above solution; only the highlighted lines have changed:

Java Python3 C++ JS

```python
def search_floor_of_a_number(arr, key):
  if key < arr[0]:  # if the 'key' is smaller than the smallest element
    return -1

  start, end = 0, len(arr) - 1
  while start <= end:
    mid = start + (end - start) // 2
    if key < arr[mid]:
      end = mid - 1
    elif key > arr[mid]:
      start = mid + 1
    else:  # found the key
      return mid

  # since the loop is running until 'start <= end', so at the end of the while loop, 'start == end+1'
  # we are not able to find the element in the given array, so the next smaller number will be arr[end]
  return end


def main():
  print(search_floor_of_a_number([4, 6, 10], 6))
  print(search_floor_of_a_number([1, 3, 8, 10, 15], 12))
  print(search_floor_of_a_number([4, 6, 10], 17))
  print(search_floor_of_a_number([4, 6, 10], -1))


main()
```

Run Save Reset

← Back

Order-agnostic Binary Search (easy)

Next →

Next Letter (medium)

✓ Completed

Report an Issue Ask a Question