Find Smallest Letter Greater Than Target

Given a list of sorted characters `letters` containing only lowercase letters, and given a target letter `target`, find the smallest element in the list that is larger than the given target.

Letters also wrap around. For example, if the target is `target = 'z'` and `letters = ['a', 'b']`, the answer is `'a'`.

Examples:

```
Input:
letters = ["c", "f", "j"]
target = "a"
Output: "c"

Input:
letters = ["c", "f", "j"]
target = "c"
Output: "f"

Input:
letters = ["c", "f", "j"]
target = "d"
Output: "f"

Input:
letters = ["c", "f", "j"]
target = "g"
Output: "j"

Input:
letters = ["c", "f", "j"]
target = "j"
Output: "c"

Input:
letters = ["c", "f", "j"]
target = "k"
Output: "c"
```

Note:

1. `letters` has a length in range `[2, 10000]`.
2. `letters` consists of lowercase letters, and contains at least 2 unique letters.
3. `target` is a lowercase letter.

Solution 1

Binary search for the character which comes immediately after character target in the alphabets, or if the target is greater than or equal to the last character in the input list, then search for the first character in the list.

```
class Solution {
    public char nextGreatestLetter(char[] a, char x) {
        int n = a.length;

        if (x >= a[n - 1])   x = a[0];
        else    x++;

        int lo = 0, hi = n - 1;
        while (lo < hi) {
            int mid = lo + (hi - lo) / 2;
            if (a[mid] == x)    return a[mid];
            if (a[mid] < x)     lo = mid + 1;
            else    hi = mid;
        }
        return a[hi];
    }
}
```

EDIT: Thanks to @xpfxzxc for a further optimized solution:

```
class Solution {
    public char nextGreatestLetter(char[] a, char x) {
        int n = a.length;

        //hi starts at 'n' rather than the usual 'n - 1'.
        //It is because the terminal condition is 'lo < hi' and if hi starts from 'n - 1',
        //we can never consider value at index 'n - 1'
        int lo = 0, hi = n;

        //Terminal condition is 'lo < hi', to avoid infinite loop when target is smaller than the first element
        while (lo < hi) {
            int mid = lo + (hi - lo) / 2;
            if (a[mid] > x)     hi = mid;
            else    lo = mid + 1;                //a[mid] <= x
        }

        //Because lo can end up pointing to index 'n', in which case we return the first element
        return a[lo % n];
    }
}
```

written by nrl original link here

Solution 2

Since the input is sorted, you can use binary search, or simply linear scan and it will still be accepted.

- *Yangshun*

```
class Solution(object):
    def nextGreatestLetter(self, letters, target):
        for letter in letters:
            if letter > target:
                return letter
        return letters[0] # If not found
```

Using `bisect`:

```
class Solution(object):
    def nextGreatestLetter(self, letters, target):
        pos = bisect.bisect_right(letters, target)
        return letters[0] if pos == len(letters) else letters[pos]
```

written by yangshun original link here

Solution 3

bisect exactly gives you the rightmost position at which to insert target to maintain sorted order.
If position for target is position after last element, you need to return the first element, which is achieved using mod.

```
    def nextGreatestLetter(self, letters, target):
        return letters[bisect.bisect_right(letters, target)%len(letters)]
```

written by candy.tgz original link here