# Find Smallest Letter Greater than target

Monday, December 29, 2025 11:20 PM

## 1. The Goal

We need to find the smallest character in a sorted list that is strictly larger than the target.

- Input : Sorted list ['c','f','j'], Target 'a'
- Constraints : If no letter is larger than the target, return the very first letter.

## 2. The Approach : Modified Binary Search

Since the i/p letters is sorted, we should immediately think of Binary Search (O(logn)) instead of scanning every element (O(n)).

### The "Wrap Around" Trick.

Before we even start searching, we handle the edge case:

- Problem : what if target is 'z' and our list only goes upto 'j'?
- Solution : We initialize our result variable to letters[0].
  - → If we find a valid answer during the search, we update result.
  - → If the search finishes and we found nothing bigger, result remains letters[0], automatically solving the wrap-around requirement.

## 3. The Logic

We are looking for the upper Bound (first value > Target)

The Decision at mid : When we look at letters[mid], we have two possibilities :

CASE A : letters[mid] > target (Potential Ans)

- Observation : This letter is valid! It is bigger than the target
- Action 1 : Store it in result (it might be the best answer we find).
- Action 2 : Move Left (end = mid−1)
- Why Left? We want the smaller greater letter. Even though mid is valid, there might be an even smaller valid letter to its left.

CASE B : letters [mid] <= target (Not Valid)

- Observation : This letter is either too small or exactly equal to the target. We need strictly greater.
- Action : Move Right (Start = mid+1)
- Why Right? All numbers to the left are even smaller. We must look to the right for bigger numbers.

## 4. Dry Run Example

List : ['c','f','j'] | Target : 'a'

- Default result 'c'

Iteration 1 :

- mid point to 'f'
- Is 'f' > 'a' ? Yes
- Update : result = 'f'
- Move : end moves left

Iteration 2 :

- mid point to 'c'
- Is 'c' > 'a' ? Yes
- Update : result = 'c'
- move : end moves left (past sort)

Loop Ends : Return 'c'

TIME COMPLEXITY : O(log n) − very efficient

SPACE COMPLEXITY : O(1) − No extra lists used