

## Grokking the Coding Interview: Patterns for Coding Questions

65% completed



- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

### Pattern: Modified Binary Search

- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)**
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

### Pattern: Bitwise XOR

- Introduction
- Single Number (easy)
- Two Single Numbers (medium)
- Complement of Base 10 Number (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

### Pattern: Top 'K' Elements

- Introduction
- Top 'K' Numbers (easy)
- Kth Smallest Number (easy)
- 'K' Closest Points to the Origin (easy)
- Connect Ropes (easy)
- Top 'K' Frequent Numbers (medium)
- Frequency Sort (medium)
- Kth Largest Number in a Stream (medium)
- 'K' Closest Numbers (medium)

## Next Letter (medium)

### We'll cover the following

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

### Problem Statement

Given an array of lowercase letters sorted in ascending order, find the **smallest letter** in the given array **greater than a given 'key'**.

Assume the given array is a **circular list**, which means that the last letter is assumed to be connected with the first letter. This also means that the smallest letter in the given array is greater than the last letter of the array and is also the first letter of the array.

Write a function to return the next letter of the given 'key'.

#### Example 1:

```
Input: ['a', 'c', 'f', 'h'], key = 'f'
Output: 'h'
Explanation: The smallest letter greater than 'f' is 'h' in the given array.
```

#### Example 2:

```
Input: ['a', 'c', 'f', 'h'], key = 'b'
Output: 'c'
Explanation: The smallest letter greater than 'b' is 'c'.
```

#### Example 3:

```
Input: ['a', 'c', 'f', 'h'], key = 'm'
Output: 'a'
Explanation: As the array is assumed to be circular, the smallest letter greater than 'm' is 'a'.
```

#### Example 4:

```
Input: ['a', 'c', 'f', 'h'], key = 'h'
Output: 'a'
Explanation: As the array is assumed to be circular, the smallest letter greater than 'h' is 'a'.
```

### Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 const search_next_letter = function(letters, key) {
2   // TODO: Write your code here
3   return letters[0];
4 };
5
6
7 console.log(search_next_letter(['a', 'c', 'f', 'h'], 'f'))
8 console.log(search_next_letter(['a', 'c', 'f', 'h'], 'b'))
9 console.log(search_next_letter(['a', 'c', 'f', 'h'], 'm'))
10
```

RUNSAVERESET

Output1.657s

a  
a  
a

Close

### Solution

The problem follows the **Binary Search** pattern. Since **Binary Search** helps us find an element in a sorted array efficiently we can use a modified version of it to find the next letter.

Maximum Distinct Elements (medium)

Sum of Elements (medium)

Rearrange String (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Problem Challenge 3

Solution Review: Problem Challenge 3

Pattern: K-way merge

Introduction

Merge K Sorted Lists (medium)

Kth Smallest Number in M Sorted Lists (Medium)

Kth Smallest Number in a Sorted Matrix (Hard)

Smallest Number Range (Hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Pattern : 0/1 Knapsack (Dynamic Programming)

Introduction

0/1 Knapsack (medium)

Equal Subset Sum Partition (medium)

Subset Sum (medium)

Minimum Subset Sum Difference (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Pattern: Topological Sort (Graph)

Introduction

Topological Sort (medium)

Tasks Scheduling (medium)

Tasks Scheduling Order (medium)

All Tasks Scheduling Orders (hard)

Alien Dictionary (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Miscellaneous

Kth Smallest Number (hard)

Conclusions

Where to Go from Here

Mark Course as Completed

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

key similarly, we can use a modified version of it to find the next letter.

We can use a similar approach as discussed in [Ceiling of a Number](#). There are a couple of differences though:

1. The array is considered circular, which means if the 'key' is bigger than the last letter of the array or if it is smaller than the first letter of the array, the key's next letter will be the first letter of the array.
2. The other difference is that we have to find the next biggest letter which can't be equal to the 'key'. This means that we will ignore the case where `key == arr[middle]`. To handle this case, we can update our start range to `start = middle + 1`.

In the end, instead of returning the element pointed out by `start`, we have to return the letter pointed out by `start % array_length`. This is needed because of point 2 discussed above. Imagine that the last letter of the array is equal to the 'key'. In that case, we have to return the first letter of the input array.

### Code #

Here is what our algorithm will look like; the most important changes are in the highlighted lines:

JavaPython3C++JS JS

```
1 function search_next_letter(letters, key) {
2   const n = letters.length;
3   if (key < letters[0] || key > letters[n - 1]) {
4     return letters[0];
5   }
6
7   let start = 0;
8   let end = n - 1;
9   while (start <= end) {
10    mid = Math.floor(start + (end - start) / 2);
11    if (key < letters[mid]) {
12      end = mid - 1;
13    } else { // key >= letters[mid]:
14      start = mid + 1;
15    }
16  }
17  // since the loop is running until 'start <= end', so at the end of the while loop, 'start === end+1'
18  return letters[start % n];
19 }
20
21 console.log(search_next_letter(['a', 'c', 'f', 'h'], 'f'));
22 console.log(search_next_letter(['a', 'c', 'f', 'h'], 'b'));
23 console.log(search_next_letter(['a', 'c', 'f', 'h'], 'm'));
```

RUNSAVERESETClose

Output2.375s

h  
c  
a

### 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  $O(1)$ .

Interviewing soon? We've partnered with [Hired](#) so that companies apply to you instead of you applying to them. [See how](#)

Back

Next

Ceiling of a Number (medium)

Number Range (medium)

Mark as Completed

Report an Issue

Ask a Question