

Grokking the Coding Interview: Patterns for Coding Questions

- 60% completed 
- Search Course
-  Subsets (easy)
 -  Subsets With Duplicates (easy)
 -  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

String Permutations by changing case (medium)

We'll cover the following ^

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

Problem Statement

Given a string, find all of its permutations preserving the character sequence but changing case.

Example 1:

```
Input: "ad52"
Output: "ad52", "Ad52", "aD52", "AD52"
```

Example 2:

```
Input: "ab7c"
Output: "ab7c", "Ab7c", "aB7c", "AB7c", "ab7C", "Ab7C", "aB7C", "AB7C"
```

Try it yourself

Try solving this question here:

Java
Python3
JS
C++

```
1 const find_letter_case_string_permutations = function(str) {
2   permutations = [];
3   // TODO: Write your code here
4   return permutations;
5 };
6
7
8 console.log(`String permutations are: ${find_letter_case_string_permutations("ad52")}`)
9 console.log(`String permutations are: ${find_letter_case_string_permutations("ab7c")}`)
10
```

RUN
SAVE
RESET

Solution

This problem follows the [Subsets](#) pattern and can be mapped to [Permutations](#).

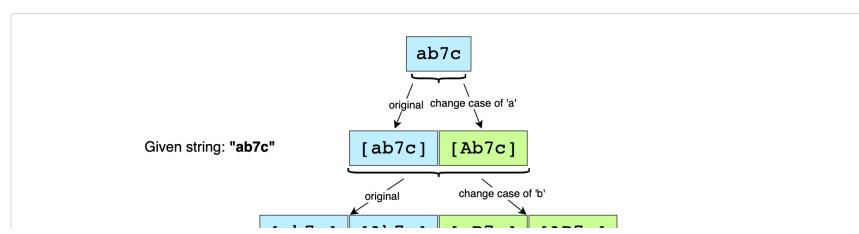
Let's take Example-2 mentioned above to generate all the permutations. Following a [BFS](#) approach, we will consider one character at a time. Since we need to preserve the character sequence, we can start with the actual string and process each character (i.e., make it upper-case or lower-case) one by one:

1. Starting with the actual string: `"ab7c"`
2. Processing the first character ('a'), we will get two permutations: `"ab7c"`, `"Ab7c"`
3. Processing the second character ('b'), we will get four permutations: `"ab7c"`, `"Ab7c"`, `"aB7c"`, `"AB7c"`
4. Since the third character is a digit, we can skip it.
5. Processing the fourth character ('c'), we will get a total of eight permutations: `"ab7c"`, `"Ab7c"`, `"aB7c"`, `"AB7c"`, `"ab7C"`, `"Ab7C"`, `"aB7C"`, `"AB7C"`

Let's analyze the permutations in the 3rd and the 5th step. How can we generate the permutations in the 5th step from the permutations in the 3rd step?

If we look closely, we will realize that in the 5th step, when we processed the new character ('c'), we took all the permutations of the previous step (3rd) and changed the case of the letter ('c') in them to create four new permutations.

Here is the visual representation of this algorithm:



- (medium)
- 'K' Closest Numbers (medium)
- 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)

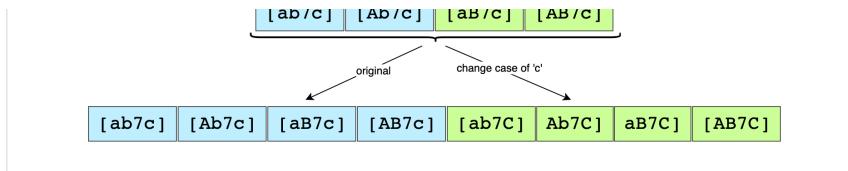
- Introduction
- 0/1 Knapsack (medium)
- Equal Subset Sum Partition (medium)
- Subset Sum (medium)
- Minimum Subset Sum Difference (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

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)



Code

Here is what our algorithm will look like:

Java	Python3	C++	JS
------	---------	-----	----

```

1  function find_letter_case_string_permutations(str) {
2    permutations = [];
3    permutations.push(str);
4    // process every character of the string one by one
5    for (i = 0; i < str.length; i++) {
6      if (isNaN(parseInt(str[i], 10))) { // only process characters, skip digits
7        // we will take all existing permutations and change the letter case appropriately
8        const n = permutations.length;
9        for (j = 0; j < n; j++) {
10          const chs = permutations[j].split('');
11          // if the current character is in upper case, change it to lower case or vice versa
12          if (chs[i] === chs[i].toLowerCase()) {
13            chs[i] = chs[i].toUpperCase();
14          } else {
15            chs[i] = chs[i].toLowerCase();
16          }
17          permutations.push(chs.join(''));
18        }
19      }
20    }
21    return permutations;
22  }
23
24
25
26 console.log(`String permutations are: ${find_letter_case_string_permutations('ad52')}`);
27 console.log(`String permutations are: ${find_letter_case_string_permutations('ab7c')}`);

```

RUN
SAVE
RESET
Close

Output 2.351s

```
String permutations are: ad52,Ad52,ad52,AD52
String permutations are: ab7c,Ab7c,aB7c,AB7c,ab7C,aB7C,AB7C
```

Time complexity

Since we can have 2^N permutations at the most and while processing each permutation we convert it into a character array, the overall time complexity of the algorithm will be $O(N * 2^N)$.

Space complexity

All the additional space used by our algorithm is for the output list. Since we can have a total of $O(2^N)$ permutations, the space complexity of our algorithm is $O(N * 2^N)$.

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

[← Back](#)

[Next →](#)

Permutations (medium)

Balanced Parentheses (hard)

[Mark as Completed](#)

[Report an Issue](#) [Ask a Question](#)