

3.1 Next Permutation

MEDIUM

■ What You Will Learn

- ✓ The 3-step algorithm for next permutation
- ✓ How to find the 'pivot' and the right swap partner
- ✓ Why reversing the suffix gives the next smallest arrangement

■ Intuition First

Imagine numbers like digits in a lock — you want the next bigger combination. Find the rightmost digit that's smaller than something to its right (the 'dip'). Swap it with the smallest digit to its right that's still larger than it. Then sort the suffix to get the smallest possible tail.

■ 3-Step Algorithm

- Step 1: Find rightmost index i where $arr[i] < arr[i+1]$ (find the 'pivot')
- Step 2: Find rightmost index j where $arr[j] > arr[i]$ (find swap partner)
- Step 3: Swap $arr[i]$ and $arr[j]$, then REVERSE the suffix $arr[i+1..end]$
- Edge: if no pivot found (fully descending), just reverse entire array

■ Visual Trace on [1,2,3] → [1,3,2]

Input: [1,2,3]

1	2	3
---	---	---

Step 1: Scan right-to-left for dip: $arr[1]=2 < arr[2]=3 \rightarrow$ pivot at $i=1$

Step 2: Find rightmost $> arr[1]=2$: $j=2$, $arr[2]=3$

Step 3: Swap $arr[1]$ and $arr[2] \rightarrow [1,3,2]$

Reverse suffix from $i+1=2$ to end $\rightarrow [1,3,2]$ (single element, no change)

Next permutation = [1,3,2] ✓

1	3	2
---	---	---

Another example: [1,3,2] $\rightarrow$ [2,1,3]

Input: [1,3,2]

1	3	2
---	---	---

Step 1: $\text{arr}[0]=1 < \text{arr}[1]=3 \rightarrow$ pivot at $i=0$

Step 2: $\text{rightmost} > 1$ in suffix $[3,2]: \text{arr}[2]=2 \rightarrow j=2$

Step 3: Swap $\text{arr}[0]$ and $\text{arr}[2] \rightarrow [2,3,1]$, reverse $[3,1] \rightarrow [2,1,3]$

Next permutation = $[2,1,3]$ ✓

2	1	3
---	---	---

■ Java Code

```
public static void nextPermutation(int[] arr) {
    int n = arr.length;
    int i = n - 2;

    // Step 1: Find rightmost dip (arr[i] < arr[i+1])
    while (i >= 0 && arr[i] >= arr[i + 1]) i--;
    if (i >= 0) {
        // Step 2: Find rightmost element > arr[i]
        int j = n - 1;
        while (arr[j] <= arr[i]) j--;

        // Step 3a: Swap
        int t = arr[i]; arr[i] = arr[j]; arr[j] = t;
    }

    // Step 3b: Reverse suffix from i+1 to end
    int l = i + 1, r = n - 1;
    while (l < r) {
        int t = arr[l]; arr[l] = arr[r]; arr[r] = t;
        l++; r--;
    }
}
```

■■ Complexity

Case	Time	Space	What happens?
All Cases	$O(n)$	$O(1)$	Three passes at most: find pivot, find j, reverse suffix

■■ Common Mistakes

- If $i=-1$ (fully descending like $[3,2,1]$), skip swap — just reverse entire array
- The suffix after swap is always in descending order — reverse gives ascending (smallest)
- Don't sort the suffix — reversing is $O(n)$ vs $O(n \log n)$ for sort

■ Why Reverse and Not Sort?

- After swap, the suffix is always in descending order (guaranteed by how we found the pivot)
- Reversing a descending list gives ascending — $O(n)$
- Sorting would be $O(n \log n)$ — unnecessary here

■ Practice Problems

[Med] Previous permutation (reverse direction)

[Med] Find all permutations of an array

[Hard] Kth permutation sequence in $O(n \log n)$

■ Quick Revision

- Find rightmost dip: $\text{arr}[i] < \text{arr}[i+1]$ scanning right to left
- Find rightmost $> \text{arr}[i]$ in suffix $\rightarrow j$
- Swap $\text{arr}[i]$ and $\text{arr}[j]$
- Reverse suffix from $i+1$ to end
- Fully descending array: just reverse all — wraps around to smallest