Cracking Coding Interviews

Next Permutation

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Leetcode 31 - Next Permutation

- Given an array of numbers (which may include duplicates), return the 'next_permutation' of the current array.
  - All permutations of set (1, 2, 3) are (1, 2, 3), (1, 3, 2), (2, 1, 3), **(2, 3, 1)**, **(3, 1, 2)**, and (3, 2, 1)
    - next_permutation((2, 3, 1)) = (3, 1, 2)
  - If there is no next permutation, return the numbers sorted
- Input ⇒ Output
  - [10, 20, 30] ⇒ [10, 30, 20]
  - [1, 1, 5] ⇒ [1, 5, 1]
- Signature [in-place modification]
  - C++: void nextPermutation(vector<int>& nums)
  - Python: def nextPermutation(self, nums: List[int]) -> None
  - Java: public void nextPermutation(int[] nums)

Your turn

- Ask the right questions, if any, and state your assumptions
- Develop some test cases

Questions & test cases

- The problem seems clear
- Several good cases are given already!
  - [50, 40, 10] ⇒ [10, 40, 50]

Controlling your emotions

- With such a problem, you might initially feel there's no way to solve it!
  - Be positive!
  - Be calm and try hard
  - A good thinker can make a little progress
- Do you best to concentrate on the problem, and analyze it
- After making a reasonable attempt, don't hesitate to ask for hints

Your turn

- Can we approach it with brute-force? If so, how?
  - For simplicity. Assume there are no duplicates.
- We can generate every single n! permutation, then find our current permutation, returning the following or 'next' permutation from the current one.
  - Practically, this is very slow
  - Can you improve on this?
  - Tip: permutations are generated in a very systematic way.
  - This should encourage you to deeply analyze examples

Cases analysis: n=3

- Let’s list all permutations for n = 3
  - 1 2 3
  - 1 3 2
  - 2 1 3
  - 2 3 1
  - 3 1 2
  - 3 2 1
- Your turn?
  - How is each item generated? What is the relation of sequence with the next one?
- Observations
  - The first row is increasing sequence and last is decreasing
  - 3 blocks: 1ab, 2gh, 3ij
  - Generation: try 1, then all permutations of {2, 3}. Then try 2 + {1, 3}, then 3 + {1, 2}

Cases analysis: n=4

- **1 2 3 4**
- 1 2 4 3
- 1 3 2 4
- 1 3 4 2
- 1 4 2 3
- **1 4 3 2**
- **2 1 3 4**
- 2 1 4 3
- 2 3 1 4
- 2 3 4 1
- 2 4 1 3
- **2 4 3 1**
- **3 1 2 4**
- 3 1 4 2
- 3 2 1 4
- 3 2 4 1
- 3 4 1 2
- **3 4 2 1**
- **4 1 2 3**
- 4 1 3 2
- 4 2 1 3
- 4 2 3 1
- 4 3 1 2
- 4 3 2 1

- Each major group changes from start -> end as follows:
  - {1 2 3 4} ⇒ {1 4 3 2}
  - {2 1 3 4} ⇒ {2 4 3 1}
  - {3 1 2 4} ⇒ {3 4 2 1}
  - {4 1 2 3} ⇒ {4 3 2 1}
- Make all possible observations!
  - Given a permutation: What is its next permutation? Why?
  - Compare any 2 consecutive permutations? What is the fixed part? What is the changing part? Why?

Cases analysis: n=4

- 1 **2 3 4**
- 1 2 4 3
- 1 3 2 4
- 1 3 4 2
- 1 4 2 3
- 1 **4 3 2**
- 2 1 3 4
- 2 1 4 3
- 2 3 1 4
- 2 3 4 1
- 2 4 1 3
- 2 4 3 1
- 3 1 2 4
- 3 1 4 2
- 3 2 1 4
- 3 2 4 1
- 3 4 1 2
- 3 4 2 1
- 4 1 2 3
- 4 1 3 2
- 4 2 1 3
- 4 2 3 1
- 4 3 1 2
- 4 3 2 1

- 4 groups
  - 1 {2, 3, 4}, 2 {1, 3, 4}, 3 {1, 2, 4}, 4 {1, 2, 3}
  - Each group is **3! = 6**: a permutation of 3 items
  - Each group of 3 items will be 3 groups of 2 items
- Example: first group 1 {2, 3, 4}
  - 3! Permutations of {2, 3, 4} will be listed.
  - Then we reach {4, 3, 2} ⇒ end
  - Next block starts with 2 {1, 3, 4}
  - Then again all permutations of {1, 3, 4} are tried
  - Then we reach {4, 3, 1} ⇒ end
  - Next group starts with 3 {1, 2, 4}
  - Observe the first item was sorted **increasingly** (2, 3, 4) and the last item **decreasingly** (4, 3, 2)

Cases analysis: n=4

- 1 2 3 4
- 1 2 4 3
- 1 3 2 4
- 1 3 4 2
- 1 4 2 3
- 1 4 3 2
- 2 1 3 4
- 2 1 4 3
- 2 3 1 4
- 2 3 4 1
- 2 4 1 3
- 2 4 3 1
- 3 1 2 4
- 3 1 4 2
- 3 2 1 4
- 3 2 4 1
- 3 4 1 2
- 3 4 2 1
- 4 1 2 3
- 4 1 3 2
- 4 2 1 3
- 4 2 3 1
- 4 3 1 2
- 4 3 2 1

- In general for N
  - N groups, each has (N-1)! Items
  - Similarly each group of length N-1
    - N-1 groups, each has (N-2)! Items
    - And so on (recursive thinking)
  - In each group, its (N-1)! Items are listed first, then the next group starts

Cases analysis: n=4

- 1 2 3 4
- 1 2 4 3
- 1 3 2 4
- 1 3 4 2
- 1 4 2 3
- 1 4 3 2
- 2 1 3 4
- 2 1 4 3
- 2 3 1 4
- 2 3 4 1
- 2 4 1 3
- 2 4 3 1
- 3 1 2 4
- 3 1 4 2
- 3 2 1 4
- 3 2 4 1
- 3 4 1 2
- 3 4 2 1
- 4 1 2 3
- 4 1 3 2
- 4 2 1 3
- 4 2 3 1
- 4 3 1 2
- 4 3 2 1

- Let's pick a **sub-group**: e.g. {3, **1, 2, 4**}
  - 3! Permutations of {1, 2, 4} will be listed first before the current listing is finished and next starts: {4, 1, 2, 3}
- What about a **sub-sub group**: e.g. {3, 1, **2, 4**}
  - 2! Permutations of {2, 4} will be listed first before the current listing is finished and next starts: {3, 2, 1, 4}
- *Note: CS students should be aware already with these listed observations so far.*

Cases analysis: n=4

- 1 2 3 4
- 1 2 4 3
- **1 3 2 4**
- **1 3 4 2**
- 1 4 2 3
- 1 4 3 2
- 2 1 3 4
- 2 1 4 3
- 2 3 1 4
- **2 3 4 1**
- **2 4 1 3**
- 2 4 3 1
- 3 1 2 4
- 3 1 4 2
- 3 2 1 4
- 3 2 4 1
- 3 4 1 2
- **3 4 2 1**
- **4 1 2 3**
- 4 1 3 2
- 4 2 1 3
- 4 2 3 1
- 4 3 1 2
- 4 3 2 1

- Compare any 2 consecutive permutations
  - <fixed prefix> <**changing suffix**>
    - 1 3 **2 4** ⇒ 1 3 4 2 ⇒ last 2 numbers changed
    - 2 **3 4 1** ⇒ 2 4 1 3 ⇒ last 3 numbers changed
    - **3 4 2 1** ⇒ 4 1 2 3 ⇒ last 4 numbers changed
- Recall: A sub-group lists its k! Permutations before the next sub-group is listed.
- Each sub-group ends with a **decreasing sequence**
  - Any observations relevant to the **changing suffix**?

Cases analysis: n=4

- 1 2 3 4
- 1 2 4 3
- 1 3 2 4
- 1 3 4 2
- 1 4 2 3
- 1 4 3 2
- 2 1 3 4
- 2 1 4 3
- 2 3 1 4
- 2 3 4 1
- 2 4 1 3
- 2 4 3 1
- **3 1 2 4**
- **3 1 4 2**
- **3 2 1 4**
- **3 2 4 1**
- **3 4 1 2**
- **3 4 2 1**
- 4 1 2 3
- 4 1 3 2
- 4 2 1 3
- 4 2 3 1
- 4 3 1 2
- 4 3 2 1

- Compare any 2 consecutive permutations
  - <fixed prefix> <**changing suffix**>
    - 1 3 **2 4** ⇒ 1 3 4 2 ⇒ last 2 numbers changed
      - {4} is the **longest decreasing suffix**
    - 2 **3 4 1** ⇒ 2 4 1 3 ⇒ last 3 numbers changed
      - {4, 1} is the **longest decreasing suffix**
    - **3 4 2 1** ⇒ 4 1 2 3 ⇒ last 4 numbers changed
      - {4, 2, 1} is the **longest decreasing suffix**
      - Suffix of length 3 vs 4 numbers changed?!
      - 3 was the group element that started listing 3! Items starting from {1, 2, 4} up tp {4, 2, 1}

Cases analysis: n=4

- 1 2 3 4
- 1 2 4 3
- 1 3 2 4
- 1 3 4 2
- 1 4 2 3
- 1 4 3 2
- 2 1 3 4
- 2 1 4 3
- 2 3 1 4
- 2 3 4 1
- 2 4 1 3
- 2 4 3 1
- **3 1 2 4**
- **3 1 4 2**
- **3 2 1 4**
- **3 2 4 1**
- **3 4 1 2**
- **3 4 2 1**
- 4 1 2 3
- 4 1 3 2
- 4 2 1 3
- 4 2 3 1
- 4 3 1 2
- 4 3 2 1

- **3 4 2 1** ⇒ 4 1 2 3 ⇒ last 4 numbers changed
  - The new replacement of 3 is 4. **Why?**
    - 4 {1, 2, 3} where {1, 2, 3} must be sorted
  - 4 = successor(3) in the list {4, 2, 1}
  - Why the successor? Observe here the 4 blocks:
  - 1 {2, 3, 4} then once 3! is listed, the next smallest start after 1 is to start with 2 so we get
  - 2 {1, 3, 4} then once 3! is listed, the next smallest start after 2 is to start with 3 so we get
  - 3 {1, 2, 4} and so on

Cases analysis: n=4

- 1 2 3 4
- 1 2 4 3
- 1 3 2 4
- 1 3 4 2
- 1 4 2 3
- **1 4 3 2**
- 2 1 3 4
- 2 1 4 3
- 2 3 1 4
- 2 3 4 1
- 2 4 1 3
- **2 4 3 1**
- 3 1 2 4
- 3 1 4 2
- 3 2 1 4
- 3 2 4 1
- 3 4 1 2
- 3 4 2 1
- 4 1 2 3
- 4 1 3 2
- 4 2 1 3
- 4 2 3 1
- 4 3 1 2
- 4 3 2 1

- 1 4 3 2
  - {4, 3, 2} is the longest decreasing suffix
  - Successor (1) from {2, 3, 4} is 2
  - Next: {2, 1, 3, 4}
- 2 4 3 1
  - {4, 3, 1} is the same longest decreasing
  - Successor (2) from {1, 3, 4} is 3
  - Next: {3, 1, 2, 4}

Cases analysis: n=4

- 1 2 3 4
- 1 2 4 3
- 1 3 2 4
- 1 3 4 2
- 1 4 2 3
- 1 4 3 2
- **2 1 3 4**
- 2 1 4 3
- 2 3 1 4
- 2 3 4 1
- 2 4 1 3
- 2 4 3 1
- 3 1 2 4
- **3 1 4 2**
- 3 2 1 4
- 3 2 4 1
- 3 4 1 2
- 3 4 2 1
- 4 1 2 3
- 4 1 3 2
- 4 2 1 3
- 4 2 3 1
- 4 3 1 2
- 4 3 2 1

- 3 1 4 2
  - {4 2} longest decreasing suffix
  - Successor (1) from {2, 4} is 2
  - Next: {3, 2, 1, 4}
- 2 1 3 4
  - {4} longest decreasing suffix
  - Successor (3) from {4} is 4
  - Next: {2 1 4 3}

Overall algorithm

- Find the longest decreasing **suffix subarray**
  - E.g. for 5 6 2 **4 3 1** it will be {4, 3, 1}
  - The element before this suffix is the parent value to be changed (2 here)
- Find the successor(2) in the suffix
  - successor(2) in {4, 3, 1} is 3
- Prepare the new sequence
  - Replace 2 with 3
  - The rest of the values sorted increasingly: {1,2,4}
  - So, the new **sub-permutation** is 3 1 2 4
  - Overall, the new permutation is: **5 6 3 1 2 4**
- Try to code in O(n)

Implementation

- To get O(n) solution, consider the following
- Assume the permutation is 5 6 2 **4 3 1**
  - Find the longest decreasing suffix: {4, 3, 1}
  - Suffix(2) = 3
  - Just swap 2 and 3 to get: 5 6 3 **4 2 1**
    - Must remain decreasing as we are replacing with the successor
  - Reverse {4, 2, 1} now we get 5 6 2 1 2 4
  - We are done in O(n) steps

Code

```
void nextPermutation(vector<int>& nums) {
    int sz = nums.size(), i = sz - 2;
    // Find the longest non-increasing suffix subarray
    while (i >= 0 && nums[i] >= nums[i + 1])
        i--;
    if (i >= 0) {   // if not whole array
        // Get next greater element for nums[i] in the suffix
        // To handle duplicates: get the most right
        int j = sz - 1;
        while (nums[j] <= nums[i])
            j--;
        swap(nums[i], nums[j]);
    }
    // Reverse the suffix [i+1, sz-1]
    reverse(nums.begin() + i + 1, nums.end());
}
```

Implementation: Handling duplicates

- Assume the list is : 0, 1, **4, 3, 3, 2, 2, 2, 1, 1**
  - Longest non-increasing: 4, 3, 3, 2, 2, 2, 1, 1
  - Successor(1) = 2 but we have 3 positions of value 2
    - We don’t care as long as the final list is sorted
  - To avoid sorting; if we selected the most right 2, we won’t sort
    - 0, **1**, 4, 3, 3, 2, 2, **2**, 1, 1
    - Swap
    - 0, **2**, 4, 3, 3, 2, 2, **1**, 1, 1
    - Reverse
    - 0, 2, **1, 1, 1, 2, 2, 3, 3, 4** [the next sub-permutation is sorted and ready]
- We already handle duplicates in the previous code. Verify.

So

- This is not an easy problem. You need good understanding for permutations
- You need to find good opervations
  - You might get trapped by misleading or incorrect observations
- I don’t ask someone such a question, but some interviewers do :(
  - Why a bad problem?
    - Limited in approaches
    - Solution depends only on cases analysis
    - Specific observations

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."