Next Permutation

Difficulty Level : Medium • Last Updated : 16 Feb, 2023

Read Discuss Courses Practice Video

Given an array **arr[]** of size **N**, the task is to print the lexicographically next greater permutation of the given array. If there does not exist any greater permutation, then print the lexicographically smallest permutation of the given array.

Examples:

Input: *N = 6, arr = {1, 2, 3, 6, 5, 4}*
Output: *{1, 2, 4, 3, 5, 6}*
Explanation: *The next permutation of the given array is {1, 2, 4, 3, 5, 6}.*

Input: *N = 3, arr = {3, 2, 1}*
Output: *{1, 2, 3}*
Explanation: *As arr[] is the last permutation.*
So, the next permutation is the lowest one.

Let's first understand what is lexicographical order in the above-given program.

We have to check that the order of the array sequence is greater than the previous array sequence. The output will be just larger sequence of the array.

Brute Force Approach :

- Find all possible permutations of the given array.
- Print the Next permutation right after the er given input sequence.

Time Complexity: O(N * N!), N represents the number of elements present in the input sequence. representsent all possible permutation. Therefore, It takes the time complexity O(N*N!).
Auxiliary Space: O(N), for storing the permutation in some data structure.

sing C++ in-build function:

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to find the next permutation
void nextPermutation(vector<int>& arr)
{
    next_permutation(arr.begin(),arr.end());
}

// Driver code
int main()
{

    // Given input array
    vector<int> arr = { 1, 2, 3, 6, 5, 4 };

    // Function call
    nextPermutation(arr);

    // Printing the answer
    for (auto i : arr) {
        cout << i << " ";
    }

    return 0;
}
```

Output

```
1 2 4 3 5 6
```

Next Permutation in linear time complexity:

Illustration:

Let's try some examples to see if we can recognize some patterns.

[3, 1, 3] = next greater number is 331
[5, 1, 3] = next greater number is 531
[1, 2, 3] = next greater number is 132
[1, 3, 5, 4] = next greater number is 1435
[3, 2, 1] = we can't form a number greater than the current number from all the possible permutations

So, it is clear that to get the next permutation we will have to change the number in a position which is as right as possible. Each permutation (except the very first) has a increasing suffix. Now if we change the pattern from the pivot point (where the increasing suffix breaks) to its next possible lexicographic representation we will get the next greater permutation.

Observation of Next permutation:

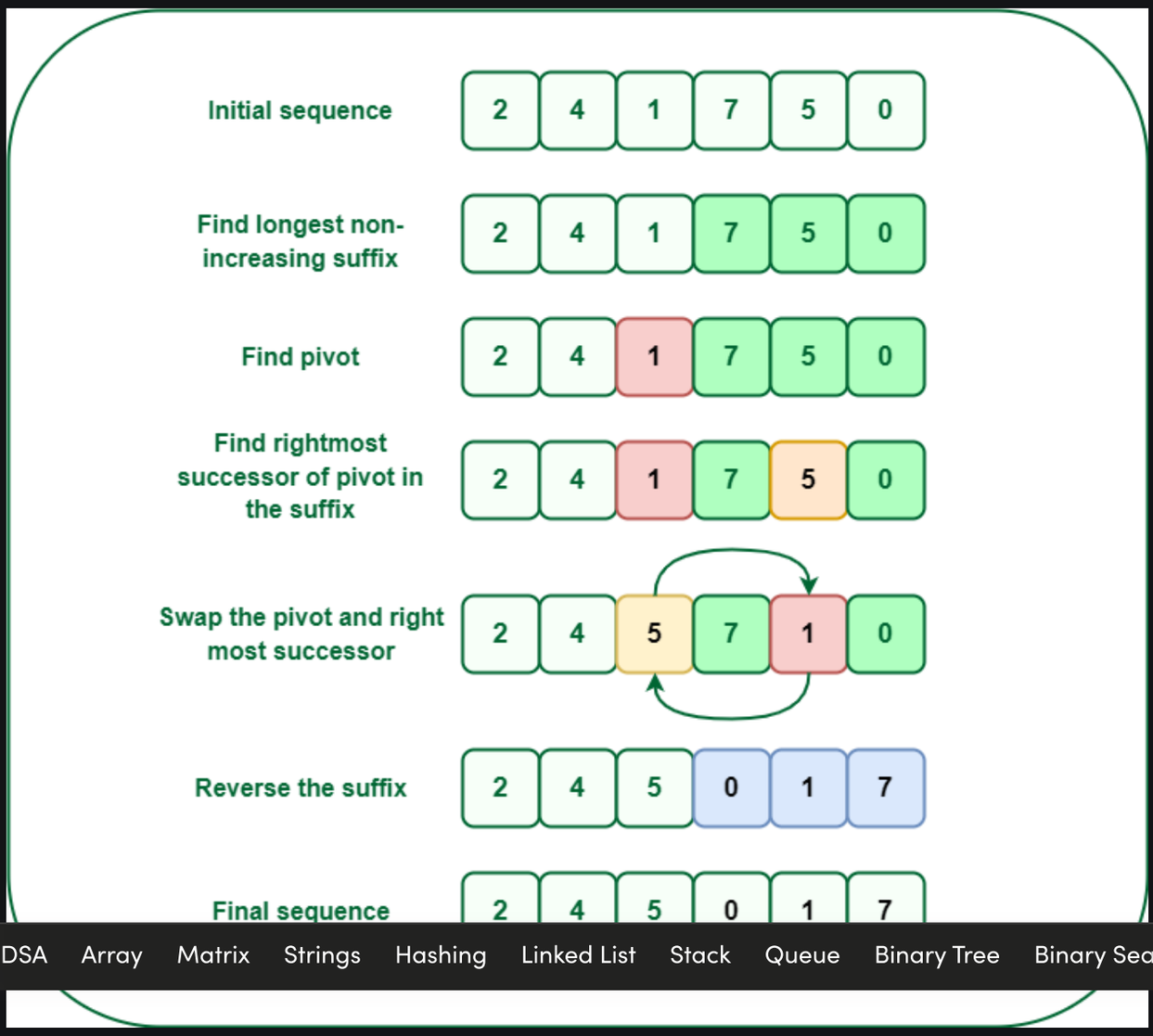

Illustration of next_permutation

Follow the steps below to implement the above observation:

- Iterate over the given array from end and find the first index **(pivot)** which doesn't follow property of non-increasing suffix, **(i.e, arr[i] < arr[i + 1]).**
- Check if **pivot** index does not exist
  - This means that the given sequence in the array is the largest as possible. So, swap the complete array.
- Otherwise, Iterate the array from the end and find for the **successor** of **pivot** in suffix.
- Swap the **pivot** and **successor**
- Minimize the suffix part by reversing the array from **pivot + 1** till **N**.

Below is the implementation of the above approach:

C++

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the next permutation
```

```
    // Find for the pivot element.
    // A pivot is the first element from
    // end of sequenc ewhich doesn't follow
    // property of non-increasing suffix
    for (i = n - 2; i >= 0; i--) {
        if (arr[i] < arr[i + 1]) {
            break;
        }
    }

    // Check if pivot is not found
    if (i < 0) {
        reverse(arr.begin(), arr.end());
    }

    // if pivot is found
    else {

        // Find for the successor of pivot in suffix
        for (j = n - 1; j > i; j--) {
            if (arr[j] > arr[i]) {
                break;
            }
        }

        // Swap the pivot and successor
        swap(arr[i], arr[j]);

        // Minimise the suffix part
        reverse(arr.begin() + i + 1, arr.end());
    }
}

// Driver code
int main()
{

    // Given input array
    vector<int> arr = { 1, 2, 3, 6, 5, 4 };

    // Function call
    nextPermutation(arr);

    // Printing the answer
    for (auto i : arr) {
        cout << i << " ";
    }

    return 0;
}
```

Java

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {
```

```
  int n = arr.length, i, j;

  // Find for the pivot element.
  // A pivot is the first element from
  // end of sequencewhich doesn't follow
  // property of non-increasing suffix
  for (i = n - 2; i >= 0; i--) {
    if (arr[i] < arr[i + 1]) {
      break;
    }
  }

  // Check if pivot is not found
  if (i < 0) {
    reverse(arr, 0, arr.length - 1);
  }

  // if pivot is found
  else {

    // Find for the successor of pivot in suffix
    for (j = n - 1; j > i; j--) {
      if (arr[j] > arr[i]) {
        break;
      }
    }

    // Swap the pivot and successor
    swap(arr, i, j);

    // Minimise the suffix part
    reverse(arr, i + 1, arr.length - 1);
  }
}

static void reverse(int[] arr, int start, int end)
{
  while (start < end) {
    swap(arr, start, end);
    start++;
    end--;
  }
}

static void swap(int[] arr, int i, int j)
{
  int temp = arr[i];
  arr[i] = arr[j];
  arr[j] = temp;
}

public static void main(String[] args)
{

  // Given input array
  int[] arr = new int[] { 1, 2, 3, 6, 5, 4 };

  // Function call
  nextPermutation(arr);

  // Printing the answer
  for (int i : arr) {
```

```
}

// This code is contributed by aadityaburujwale.
```

Python3

```
# Python code to implement the above approach
def swapPositions(list, pos1, pos2):
    list[pos1], list[pos2] = list[pos2], list[pos1]
    return list

# Function to find the next permutation
def nextPermutation(arr):
    n = len(arr)
    i = 0
    j = 0

    # Find for the pivot element.
    # A pivot is the first element from
    # end of sequencewhich doesn't follow
    # property of non-increasing suffix
    for i in range(n-2, -1, -1):
        if (arr[i] < arr[i + 1]):
            break

    # Check if pivot is not found
    if (i < 0):
        arr.reverse()

    # if pivot is found
    else:
        # Find for the successor of pivot in suffix
        for j in range(n-1, i, -1):
            if (arr[j] > arr[i]):
                break

        # Swap the pivot and successor
        swapPositions(arr, i, j)

        # Minimise the suffix part
        # initializing range
        strt, end = i+1, len(arr)

        # Third arg. of split with -1 performs reverse
        arr[strt:end] = arr[strt:end][::-1]

# Driver code
if __name__ == "__main__":
    arr = [1, 2, 3, 6, 5, 4]

    # Function call
    nextPermutation(arr)

    # Printing the answer
    for i in arr:
        print(i, end=" ")

# This code is contributed by Rohit Pradhan
```

Start Your Coding Journey Now!

```
// Include namespace system
using System;

public class GFG
{

  // Function to find the next permutation
  public static void nextPermutation(int[] arr)
  {
    var n = arr.Length;
    int i;
    int j;

    // Find for the pivot element.
    // A pivot is the first element from
    // end of sequencewhich doesn't follow
    // property of non-increasing suffix
    for (i = n - 2; i >= 0; i--)
    {
      if (arr[i] < arr[i + 1])
      {
        break;
      }
    }

    // Check if pivot is not found
    if (i < 0)
    {
      GFG.reverse(arr, 0, arr.Length - 1);
    }
    else
    {

      // Find for the successor of pivot in suffix
      for (j = n - 1; j > i; j--)
      {
        if (arr[j] > arr[i])
        {
          break;
        }
      }

      // Swap the pivot and successor
      GFG.swap(arr, i, j);

      // Minimise the suffix part
      GFG.reverse(arr, i + 1, arr.Length - 1);
    }
  }
  public static void reverse(int[] arr, int start, int end)
  {
    while (start < end)
    {
      GFG.swap(arr, start, end);
      start++;
      end--;
    }
  }
  public static void swap(int[] arr, int i, int j)
  {
    var temp = arr[i];
```

```
  public static void Main(String[] args)
  {

    // Given input array
    int[] arr = new int[]{1, 2, 3, 6, 5, 4};

    // Function call
    GFG.nextPermutation(arr);

    // Printing the answer
    foreach (int i in arr)
    {
      Console.Write(i.ToString() + " ");
    }
  }
}

// This code is contributed by aadityaburujwale.
```

Javascript

```
// javascript code implementation

// Function to find the next permutation
function nextPermutation(arr)
{
    let n = arr.length, i, j;

    // Find for the pivot element.
    // A pivot is the first element from
    // end of sequencewhich doesn't follow
    // property of non-increasing suffix
    for (i = n - 2; i >= 0; i--) {
        if (arr[i] < arr[i + 1]) {
            break;
        }
    }

    // Check if pivot is not found
    if (i < 0) {
        arr.reverse();
    }

    // if pivot is found
    else {

        // Find for the successor of pivot in suffix
        for (j = n - 1; j > i; j--) {
            if (arr[j] > arr[i]) {
                break;
            }
        }

        // Swap the pivot and successor
        let temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;

        // Minimise the suffix part
        let arr1 = arr.slice(i+1, n);
```

```
}

// Driver code

    // Given input array
    let arr = [ 1, 2, 3, 6, 5, 4 ];

    // Function call
    nextPermutation(arr);

    // Printing the answer
    for (let i = 0; i < arr.length; i++) {
        console.log(arr[i]);
    }

// this code is contributed by ksam24000
```

Output

```
1 2 4 3 5 6
```

Time Complexity: O(N), where N is the size of the given array.

Auxiliary Space: O(1)

Previous Next

Related Articles

1. Check if permutation of one string can break permutation of another
2. Minimum number of given operations required to convert a permutation into an identity permutation
3. Minimum number of adjacent swaps required to convert a permutation to another permutation by given condition
4. Find next Smaller of next Greater in an array

6. Generate all permutation of a set in Python
7. Permutation Coefficient
8. Permutation and Combination in Python
9. Find a permutation that causes worst case of Merge Sort
10. How to find Lexicographically previous permutation?

Article Contributed By :

hkdass001
@hkdass001

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : mitalibhola94, aadityapburujwale, rohit768, ksam24000, rohitvish2110, shreoshi63

Article Tags : Picked, two-pointer-algorithm, Arrays, DSA

Practice Tags : Arrays, two-pointer-algorithm

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org