

# 1850. Minimum Adjacent Swaps to Reach the Kth Smallest Number

[Leetcode Link](#)

## Problem Explanation

In this problem, we are given a string `num` that represents a large integer, and an integer `k`. The task is to find the minimum number of adjacent digit swaps that needs to be applied to `num` to reach the kth smallest wonderful integer. A wonderful integer is an integer that is a permutation of the digits in `num` and is greater in value than `num`.

## Approach

The solution follows this approach:

1. Generate the `k`th smallest wonderful integer using a permutation algorithm such as `next_permutation`.
2. Count the minimum number of adjacent digit swaps needed to transform the original string into the generated permutation.

Let's walk through an example.

### Example

```
1 num = "5489355142"
2 k = 4
```

1. We will first find the `k`th smallest wonderful integer using `next_permutation`.

- After 1st permutation: "5489355214"
- After 2nd permutation: "5489355241"
- After 3rd permutation: "5489355412"
- After 4th permutation: "5489355421"

So, the 4th smallest wonderful integer is "5489355421".

2. Now, we will count the minimum number of adjacent digit swaps needed.

1. Swap index 7 with index 8: "5489355142" → "5489355412"
2. Swap index 8 with index 9: "5489355412" → "5489355421"

The total number of swaps required is 2.

Now, let's implement the solution in different languages.

## Solution in Python

```
1 from itertools import permutations
2
3 class Solution:
4     def getMinSwaps(self, num: str, k: int) -> int:
5         perm = num
6
7         for _ in range(k):
8             perm = self.next_permutation(perm)
9
10        return self.count_steps(num, perm)
11
12    def count_steps(self, A: str, B: list) -> int:
13        count = 0
14
15        for i in range(len(A)):
16            j = i
17            while A[i] != B[j]:
18                j += 1
19            while i < j:
20                B[j], B[j - 1] = B[j - 1], B[j]
21                j -= 1
22                count += 1
23
24        return count
25
26    def next_permutation(self, s: str) -> list:
27        s = list(s)
28        n = len(s)
29        i = n - 1
30        while i > 0 and s[i - 1] >= s[i]:
31            i -= 1
32
33        j = n - 1
34        while i > 0 and s[i - 1] >= s[j]:
35            j -= 1
36        s[i - 1], s[j] = s[j], s[i - 1]
37        s[i:] = reversed(s[i:])
38
39        return s
```

## Solution in Java

```
1 import java.util.*;
2
3 class Solution {
4     public int getMinSwaps(String num, int k) {
5         List<Character> perm = new ArrayList<>();
6
7         for (char ch : num.toCharArray())
8             perm.add(ch);
9
10        for (int i = 0; i < k; ++i)
11            nextPermutation(perm);
12
13        return countSteps(num, perm);
14    }
15
16    private int countSteps(String A, List<Character> B) {
17        int count = 0;
18
19        for (int i = 0, j = 0; i < A.length(); ++i) {
20            j = i;
21            while (A.charAt(i) != B.get(j))
22                ++j;
23
24            while (i < j) {
25                Collections.swap(B, j, j - 1);
26                --j;
27                ++count;
28            }
29        }
30        return count;
31    }
32
33    private void nextPermutation(List<Character> perm) {
34        int i = perm.size() - 1;
35
36        while (i > 0 && perm.get(i - 1) >= perm.get(i))
37            --i;
38
39        int j = perm.size() - 1;
40        while (i > 0 && perm.get(i - 1) >= perm.get(j))
41            --j;
42
43        Collections.swap(perm, i - 1, j);
44        Collections.reverse(perm.subList(i, perm.size()));
45    }
46 }
```

## Solution in C++

```
1 #include <algorithm>
2 #include <string>
3 #include <vector>
4
5 class Solution {
6 public:
7     int getMinSwaps(std::string num, int k) {
8         std::string perm = num;
9
10        while (k--)
11            std::next_permutation(begin(perm), end(perm));
12
13        return countSteps(num, perm);
14    }
15
16 private:
17     int countSteps(const std::string &A, std::string &B) {
18         int count = 0;
19
20        for (size_t i = 0, j = 0; i < A.length(); ++i) {
21            j = i;
22            while (A[i] != B[j])
23                ++j;
24
25            while (i < j) {
26                std::swap(B[j], B[j - 1]);
27                --j;
28                ++count;
29            }
30        }
31        return count;
32    }
33 };
```

## Solution in C#

```
1 using System;
2 using System.Collections.Generic;
3
4 public class Solution {
5     public int GetMinSwaps(string num, int k) {
6         List<char> perm = new List<char>(num);
7
8         for (int i = 0; i < k; ++i)
9             NextPermutation(perm);
10
11        return CountSteps(num, perm);
12    }
13
14    private int CountSteps(string A, IList<char> B) {
15        int count = 0;
16
17        for (int i = 0, j = 0; i < A.Length; ++i) {
18            j = i;
19            while (A[i] != B[j])
20                ++j;
21
22            while (i < j) {
23                Swap(B, j, j - 1);
24                --j;
25                ++count;
26            }
27        }
28
29        return count;
30    }
31
32    private void NextPermutation(IList<char> perm) {
33        int i = perm.Count - 1;
34
35        while (i > 0 && perm[i - 1] >= perm[i])
36            --i;
37
38        int j = perm.Count - 1;
39        while (i > 0 && perm[i - 1] >= perm[j])
40            --j;
41
42        Swap(perm, i - 1, j);
43        Reverse(perm, i, perm.Count);
44    }
45
46    private void Swap(IList<char> list, int i, int j) {
47        char tmp = list[i];
48        list[i] = list[j];
49        list[j] = tmp;
50    }
51
52    private void Reverse(IList<char> list, int start, int end) {
53        for (int i = start, j = end - 1; i < j; ++i, --j)
54            Swap(list, i, j);
55    }
56 }
57
58 ````# JavaScript Implementation
59
60 ````javascript
61 class Solution {
62     getMinSwaps(num, k) {
63         let perm = num.split('');
64
65         for (let i = 0; i < k; i++) {
66             this.nextPermutation(perm);
67         }
68
69         return this.countSteps(num, perm);
70     }
71
72     countSteps(A, B) {
73         let count = 0;
74
75         for (let i = 0; i < A.length; i++) {
76             let j = i;
77             while (A[i] !== B[j]) {
78                 j++;
79             }
80
81             while (i < j) {
82                 [B[j], B[j - 1]] = [B[j - 1], B[j]];
83                 j--;
84                 count++;
85             }
86
87             return count;
88         }
89
90         nextPermutation(perm) {
91             let i = perm.length - 1;
92
93             while (i > 0 && perm[i - 1] >= perm[i]) {
94                 i--;
95             }
96
97             let j = perm.length - 1;
98             while (i > 0 && perm[i - 1] >= perm[j]) {
99                 j--;
100             }
101
102             [perm[i - 1], perm[j]] = [perm[j], perm[i - 1]];
103             perm.splice(i, perm.length - i, ...perm.slice(i).reverse());
104         }
105     }
106
107     let sol = new Solution();
108     let num = "5489355142";
109     let k = 4;
110     console.log(sol.getMinSwaps(num, k)); // Output: 2
```

The JavaScript implementation follows a similar approach to other languages. We first convert the input string `num` into an array of characters and then apply the `nextPermutation` function `k` times. After that, we count the number of swaps required using the `countSteps` function. Finally, we return the swap count.



Level Up Your  
Algo Skills

Get Premium

Got a question? [Ask the Teaching Assistant](#) anything you don't understand.