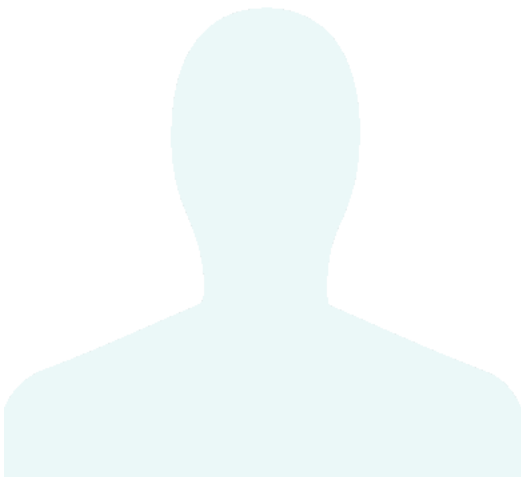


[Coderust 2.0: Faster Coding Interview Preparation using Interactive Visualizations](#)Search

- Arrays
 - [Binary Search](#)
Preview
 - [Find Maximum in Sliding Window](#)
 - [Search rotated array](#)
 - [Find smallest common number](#)
 - [Rotate Array](#)
 - [Find low/high index](#)
 - [Move zeros to left](#)
 - [Find maximum single sell profit](#)
 - [Implement Quicksort](#)
 - [Merge Overlapping Intervals](#)
 - [Sum of Two Values](#)
Preview
- Linked List
 - [Reverse a singly linked list](#)
Preview
 - [Remove Duplicates from a Linked List](#)
 - [Delete node with a given key](#)
 - [Insertion Sort of a Linked List](#)
 - [Intersection Point of Two Lists](#)
Preview
 - [Nth from last node](#)
 - [Swap Nth Node with Head](#)
 - [Merge Two Sorted Linked Lists](#)
 - [Merge Sort](#)
 - [Reverse even nodes](#)
 - [Rotate a Linked List](#)
 - [Reverse k Elements](#)
 - [Add Two Integers](#)
 - [Copy linked list with arbitrary pointer](#)
- Math & Stats
 - [Find kth permutation](#)
Preview
 - [Integer Division](#)
 - [Pythagorean Triplets](#)
 - [All Sum Combinations](#)
 - [Find Missing Number](#)
 - [Permute String](#)
 - [All Subsets](#)
 - [Is Number Valid?](#)
 - [Power of a Number](#)
 - [Calculate Square Root](#)
- String
 - [Reverse words in a sentence](#)
Preview
 - [Remove Duplicates](#)
 - [Remove white spaces](#)
 - [String Segmentation](#)
 - [XML to Tree](#)
 - [Find all palindrome substrings](#)
 - [Regular Expression](#)
- Trees
 - [Check if two binary trees are identical](#)
Preview
 - [Write an Inorder Iterator for a Binary Tree](#)
 - [Iterative Inorder Traversal](#)
 - [Inorder Successor BST](#)
 - [Level Order Traversal of Binary Tree](#)
 - [Is Binary Search Tree?](#)
 - [Convert Binary Tree To Doubly Linked List](#)
 - [Print Tree Perimeter](#)
 - [Connect Same Level Siblings](#)

- [Serialize/Deserialize Binary Tree](#)
- [Connect All Siblings](#)
- [Inorder Successor BST with parent pointers](#)
- [Nth Highest in BST](#)
- [Mirror binary tree nodes](#)
- [Delete zero sum sub-trees](#)
- [N-ary Tree to Binary Tree](#)
- Stacks and Queues
 - [Stack using Queues](#)
Preview
 - [Queue using Stacks](#)
 - [Expression Evaluation](#)
- Graphs
 - [Clone a Directed Graph](#)
Preview
 - [Minimum Spanning Tree](#)
 - [Word Chaining](#)
- Back Tracking
 - [Boggle](#)
Preview
 - [All Possible Parentheses](#)
 - [Solve N-Queens problem](#)
 - [Find K-sum subsets](#)
- Dynamic Programming
 - [Fibonacci numbers](#)
Preview
 - [Largest Sum Subarray](#)
 - [MaxSum Subsequence - Nonadjacent Elements](#)
 - [Game Scoring: Find number of ways a player can score 'n' runs](#)
 - [Coin Changing Problem](#)
 - [Levenshtein Distance](#)
- Miscellaneous
 - [Sum of Three Values](#)
Preview
 - [Make Columns and Rows Zeros](#)
 - [Search in a Matrix](#)
 - [Implement LRU Cache](#)
 - [Host Endianness](#)
 - [Closest Meeting Point](#)
- Appendix
 - [Testimonials](#)
 - [Frequently Asked Questions](#)
 - [Other Resources](#)

[LearnTeach](#)

- [My Profile](#)

- [View](#)
- [Edit](#)
- Logout



Find kth permutation

Given a set of n elements find their k th permutation.

Description

Given a set of n elements find their k^{th} permutation. Consider the following set of elements:

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

All permutations of the above elements are (with ordering):

1st	123
2nd	132
3rd	213
4th	231
5th	312
6th	321

Here, we need to find the k^{th} permutation.

Hints

- Recursion
- Factorial

Solution

Runtime Complexity

Linear, $O(n)$.

Memory Complexity

Linear, $O(n)$.

Recursive solution will consume memory on the stack.

Let's discuss few basics first. We know that $n!$ is the number of permutations of a set of size n . Another obvious and important concept is that if we choose an element for first position then the total permutations of remaining elements are $(n-1)!$. For example if we are given elements $\{1, 2, 3, 4\}$ and we pick 1 as our first element then for remaining elements we have the following permutations:

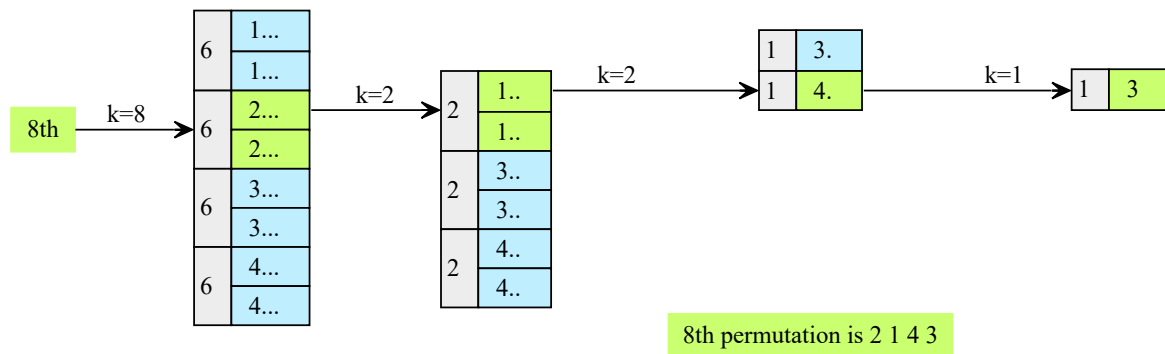
1	→	234
		243
		324
		342
		423
		432

which is equal to 6 i.e. $(n-1)!$. This number is same if we pick another element for the first slot.

1	→	234	6
		...	
		...	
2	→	134	6
		...	
		...	
3	→	124	6
		...	
		...	
4	→	123	6
		...	
		...	

A naive way of finding k^{th} permutation will be to find all permutations and then return the k^{th} permutation or maintain a running count of permutations seen so far and return once k^{th} permutation is reached.

We can do better than this if we closely look at the diagram above. If we are given k and we somehow guess which block it's going to lie in that will help us find at least the first element. Similarly, within that block if we can identify a sub-block where k resides, it will help us find the second element. We can do this recursively until we run out of options. Here is a visual representation of this approach if $k = 8$:



Here goes the algorithm we will follow:

9

1

If input vector is empty return result vector

2

3

block_size = (n-1)! ['n' is the size of vector]

4

Figure out which block k will lie in and select the first element of that block

5

(this can be done by doing $(k-1)/\text{block_size}$)

6

7

Append selected element to result vector and remove it from original input vector

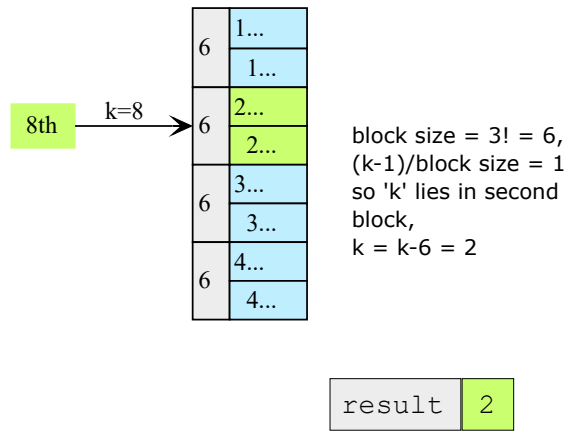
8

9

Deduce from k the blocks that are skipped i.e $k = k - \text{selected} * \text{block_size}$ and goto step 1

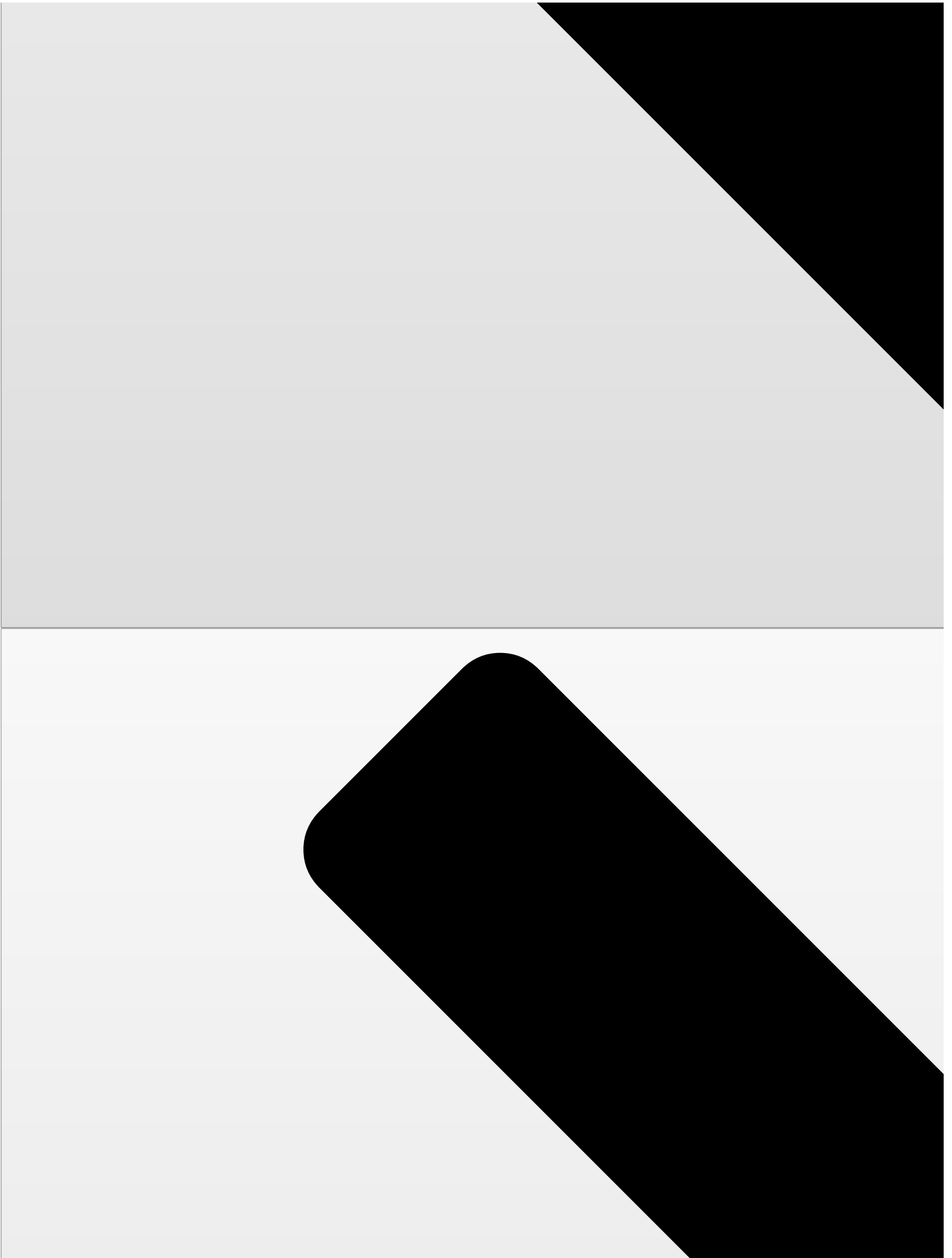
As you can notice the runtime complexity of this algorithm is linear (proportional to the size of the input vector) and memory is linear too because of the recursive calls. If we implement this algorithm non-recursively it will use constant memory.

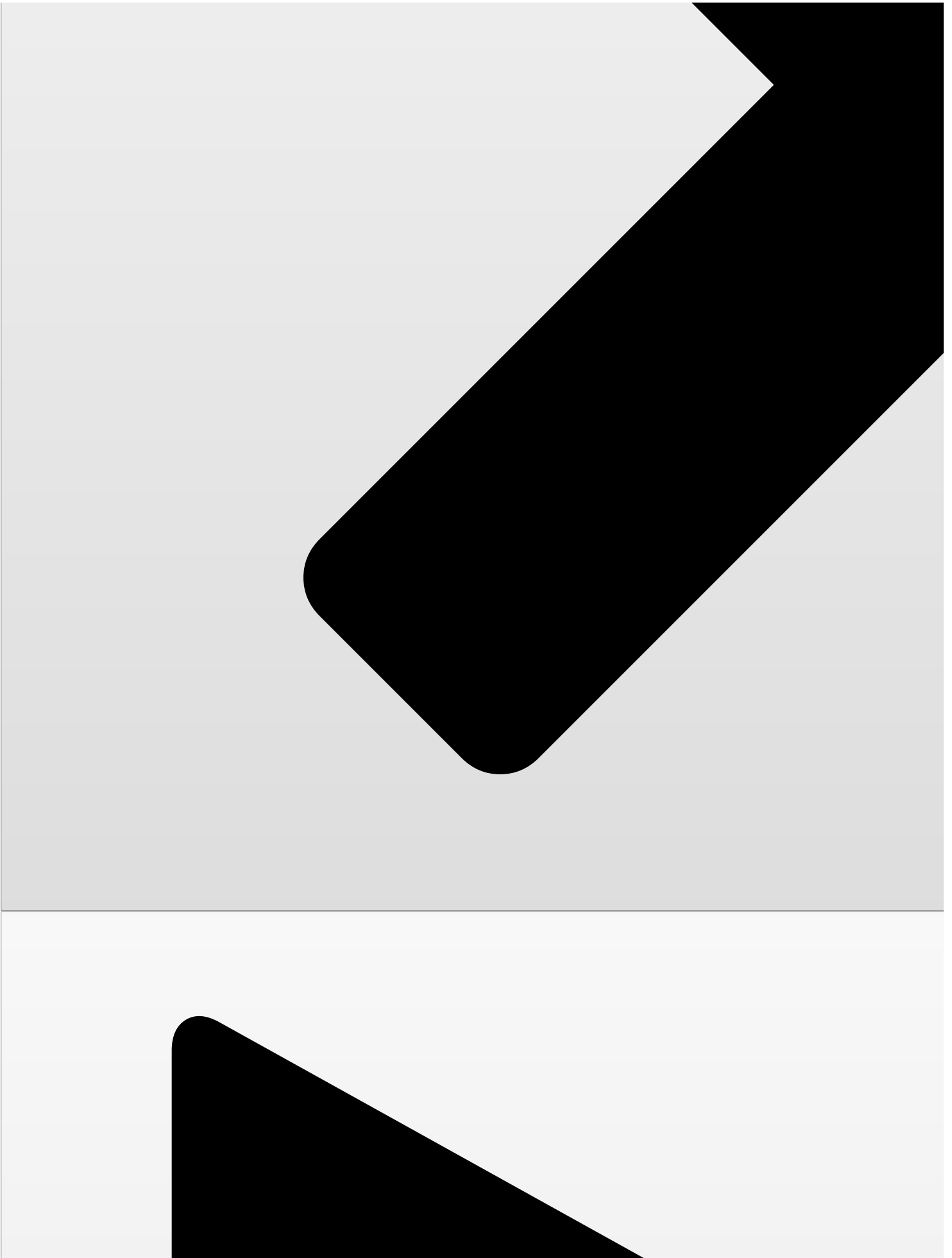
Let's understand this example with k=8 step by step below.

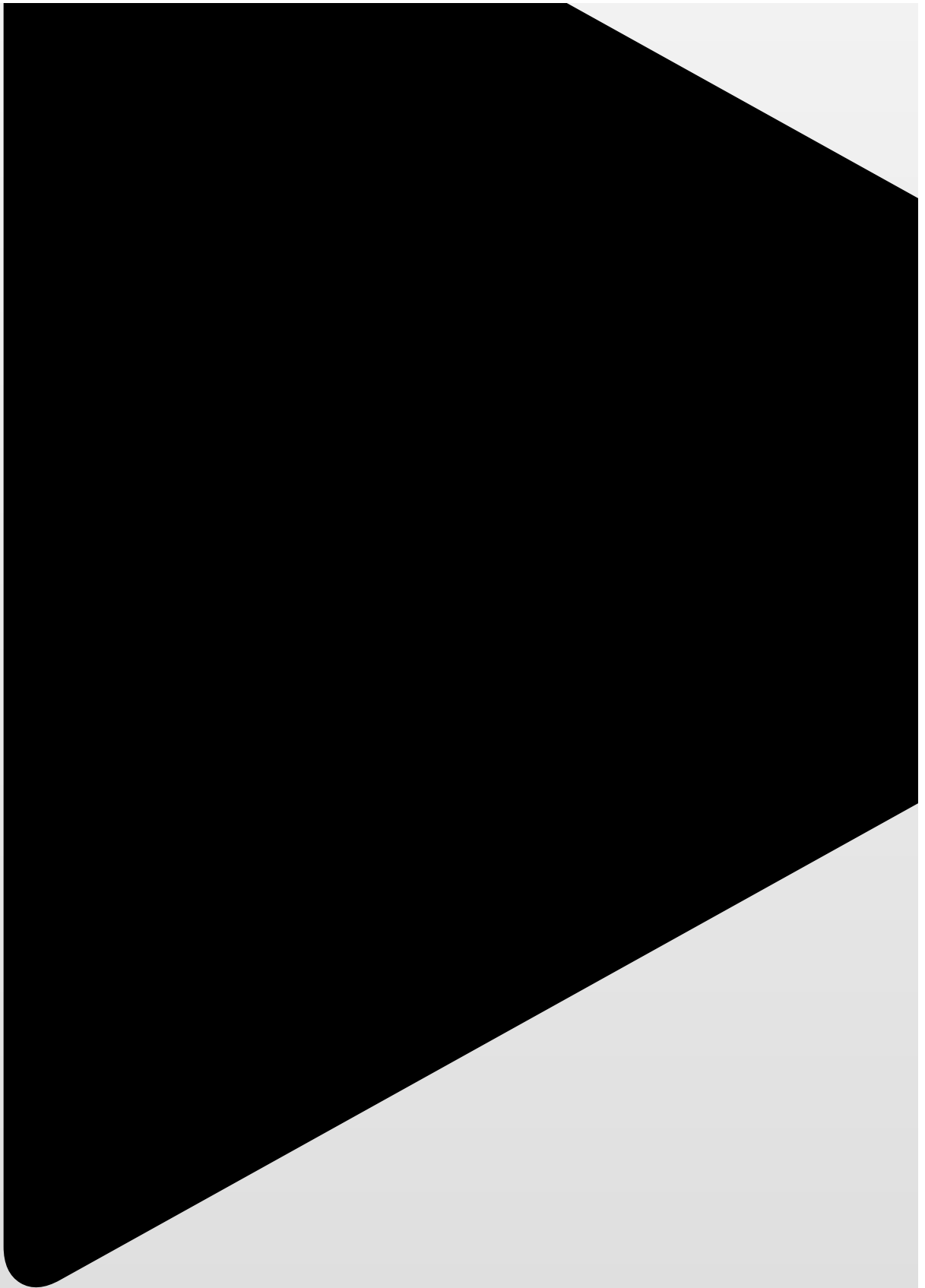


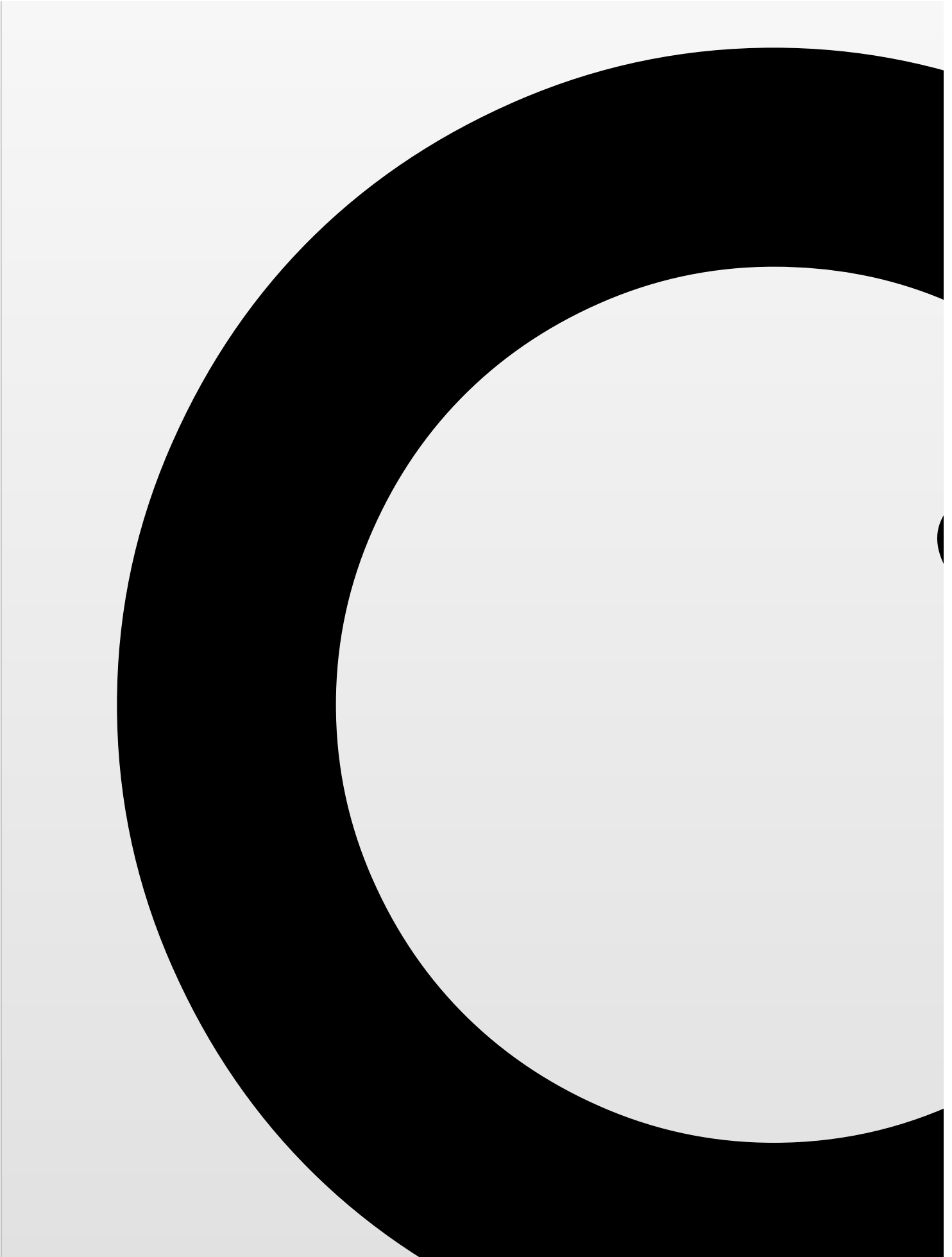
input = {1, 2, 3, 4}, k = 8, n = 4

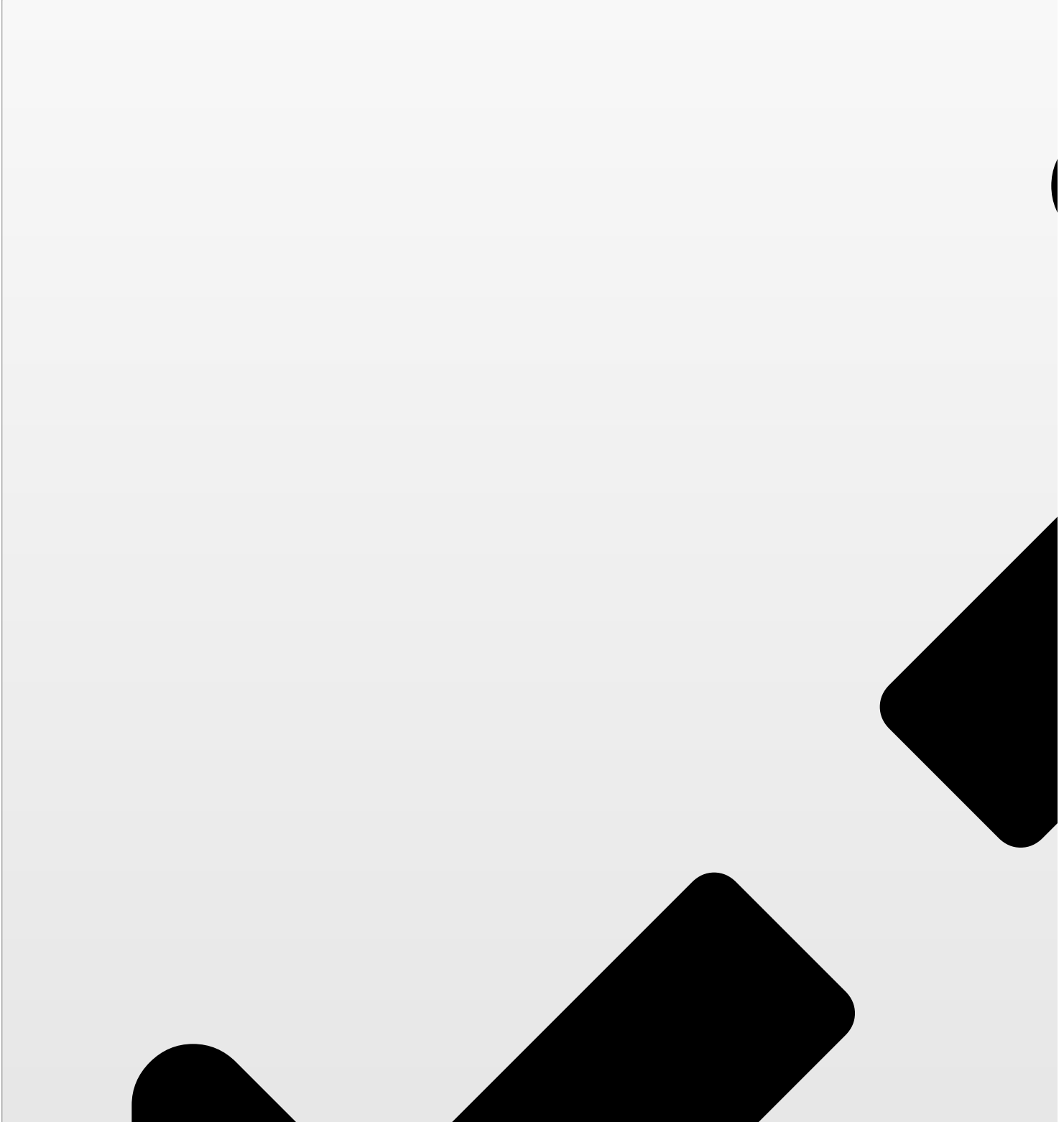
1 of 4













- [C++](#)
- [Java](#)
- [Python](#)
- [JS](#)
- [Ruby](#)

24

1

```
int factorial(int n) {
```

2

```
    if (n == 0 || n == 1) return 1;
```

3

```
    return n * factorial(n -1 );
```

4

```
}
```

5

6

```
void find_kth_permutation(
```

7

```
    vector<char>& v,
```

8

```
    int k,
```

9

```
    string& result) {
```

```
10
    if (v.empty()) {
11
        return;
12
    }
13

14
    int n = (int)(v.size());
15

    // count is number of permutations starting with each digit
16
    int count = factorial(n - 1);
17

    int selected = (k - 1) / count;
18

19
    result += v[selected];
20
    v.erase(v.begin() + selected);
21

22
    k = k - (count * selected);
23

    find_kth_permutation(v, k, result);
24
}
```

[Mark as completed](#)[← Previous](#) [Copy linked list with arbitrary pointer](#) [Next → Integer Division](#)[Send feedback or ask a question](#)



10 recommendations

- [Home](#)
- [Featured](#)
- [Team](#)
- [Blog](#)
- [FAQ](#)
- [Terms of Service](#)
- [Contact Us](#)