COSC 2436: Sorting Practice

J. Eoin Donovan - 2023

Sorting methods we'll cover

- **Merge Sort**
- **Quick Sort**
- **Shell Sort**
- **Heap Sort**
- **Bucket Sort (Radix)**

Code for Sorting Methods

https://replit.com/@EoinDonovan/Sorting

Merge Sort

- **Divide: The unsorted array is divided into two equal halves**
- **Conquer: Each half is recursively sorted until it becomes a single element or a sorted sub-array**
- **Merge: The sorted halves are merged back together into a single sorted array**
- **Best Case Time Complexity: O(nlogn)**
- **Worst Case Time Complexity: O(nlogn)**

Merge Sort

Sort the array below using merge sort:

```
{5, 99, 8, 12, 45, 23, 22, 10}
```

Start: {5, 99, 8, 12, 45, 23, 22, 10}

{5, 99, 8, 12} {45, 23, 22, 10}

{5, 99} {8, 12} {45, 23} {22, 10}

{5} {99} {8} {12} {45} {23} {22} {10}

{5, 99} {8, 12} {23, 45} {10, 22}

{5, 8, 12, 99} {10, 22, 23, 45}

End: {5, 8, 10, 12, 22, 23, 45, 99}

Quick Sort

- **Partitioning: Select a pivot element from the array and rearrange the elements, placing those less than the pivot on the left and those greater on the right**
- **Recursion: Recursively apply the partitioning process to the sub-arrays on the left and right of the pivot until the entire array is sorted**
- **Combining: Combine the sorted sub-arrays to form the final sorted array**
- **Best Case Time Complexity: O(nlogn)**
- **Worst Case Time Complexity: $O(n^2)$**

Quick Sort

Sort the array below using QuickSort. Use the middle element as the pivot.

```
{5, 1, 9, 3, 7, 6, 2}
```

Quick Sort

{5, 1, 9, 3, 7, 6, 2}

Pivot = 3:

{5, 1, 9, 3, 7, 6, 2}
{2, 1, 9, 3, 7, 6, 5}
{2, 1, 9, 3, 7, 6, 5}
{2, 1, 9, 3, 7, 6, 5}
{2, 1, 9, 3, 7, 6, 5}
{2, 1, 3, 9, 7, 6, 5}
{2, 1, 3, 9, 7, 6, 5}

Quick Sort

`{2, 1, 3, 9, 7, 6, 5}`

Partition `{2, 1, 3}` and `{9, 7, 6, 5}` separately.

Quick Sort

{2, 1, 3}

Pivot = 1:

{2, 1, 3}
{2, 1, 3}
{1, 2, 3}
{1, 2, 3}

Quick Sort

{9, 7, 6, 5}

Pivot = 7:

{9, 7, 6, 5}
{5, 7, 6, 9}
{5, 7, 6, 9}
{5, 6, 7, 9}
{5, 6, 7, 9}

Quick Sort

```
Sorted Array:
                    {1, 2, 3, 5, 6, 7, 9}
```

Shell Sort

- Shell sort repeatedly sorts subarrays with a decreasing gap between elements, gradually moving smaller elements towards the beginning and reducing the overall number of comparisons and swaps.
- Best Case Time Complexity: O(nlogn)
- Worst Case Time Complexity: $O(n(\log n)^2)$

Shell Sort

Sort the array below using shell sort.

```
{8, 4, 1, 6, 3, 7, 2, 5}
```

Shell Sort

```
Start: {8, 4, 1, 6, 3, 7, 2, 5}

Gap = 4
{8, 4, 1, 6, 3, 7, 2, 5}
{3, 4, 1, 5, 8, 7, 2, 6}

Gap = 2
{3, 4, 1, 5, 8, 7, 2, 6}
{1, 4, 2, 5, 3, 6, 8, 7}

Gap = 1
{1, 4, 2, 5, 3, 6, 8, 7}
{1, 2, 3, 4, 5, 6, 7, 8}

End: {1, 2, 3, 4, 5, 6, 7, 8}
```

Heap Sort

- Heap sort is a comparison-based sorting algorithm that builds a max-heap (or min-heap for desecnding order), swaps the root element with the last element, and then maintains the heap property while reducing the size of the heap, iteratively moving the largest (or smallest for descending) elements to the sorted portion of the array.
- Best Case Time Complexity: O(nlogn)
- Worst Case Time Complexity: O(nlogn)

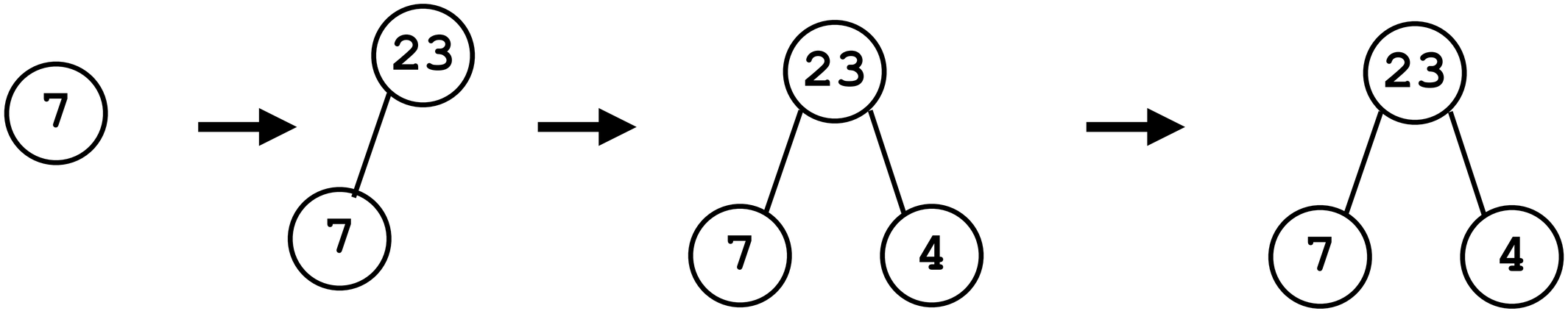
HeapSort

Sort the array below in ascending order using heapsort:

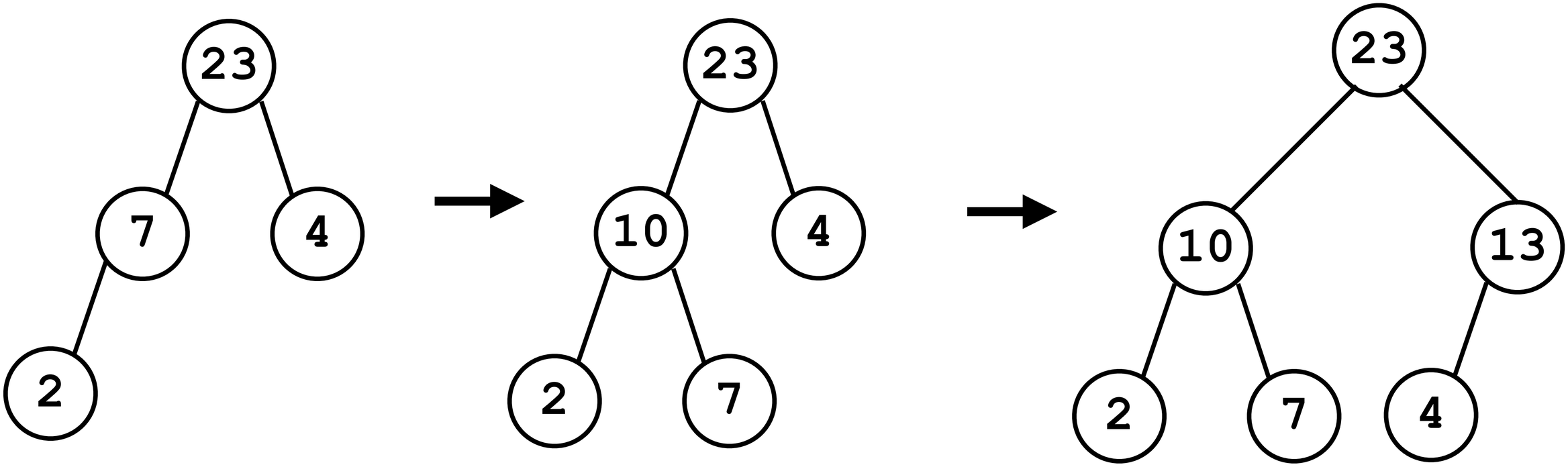
{7, 23, 4, 2, 10, 13, 14}

Build Max-Heap

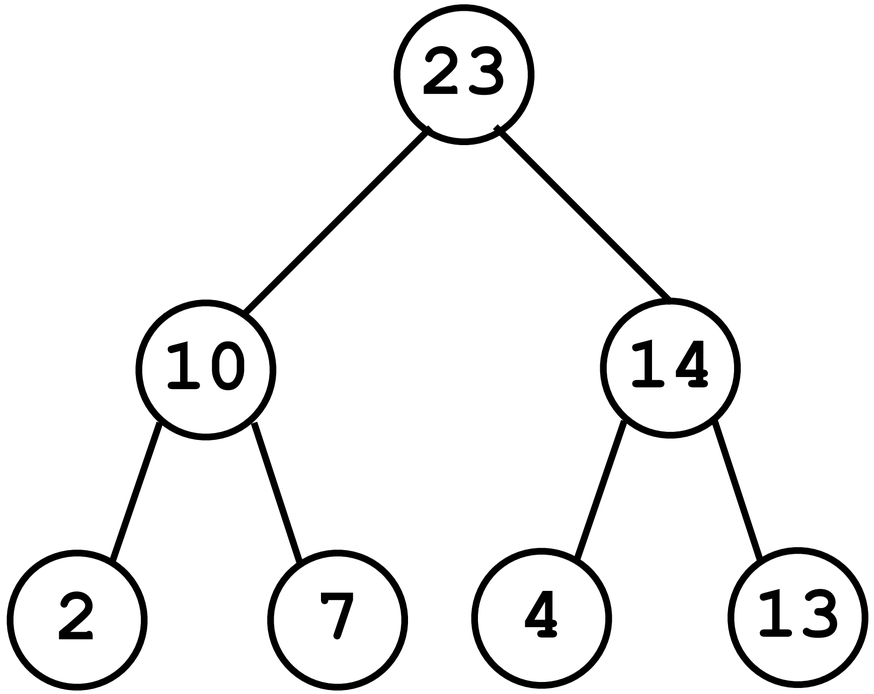
{7, 23, 4, 2, 10, 13, 14}

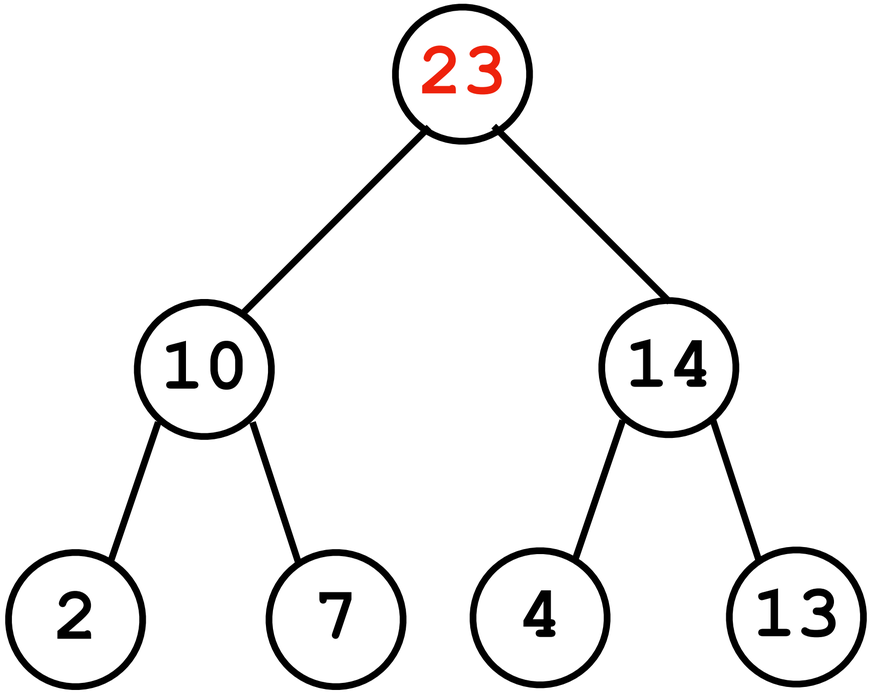


{7, 23, 4, 2, 10, 13, 14}

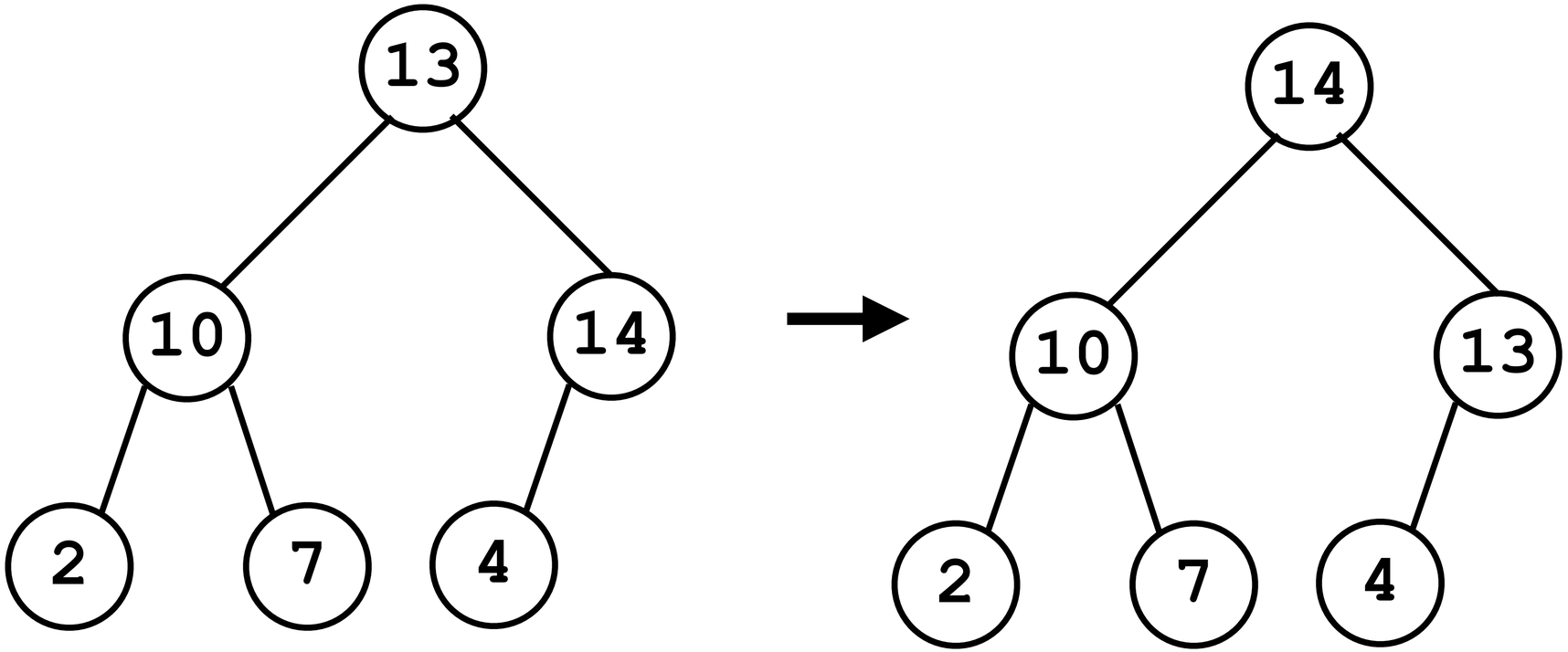


{7, 23, 4, 2, 10, 13, 14}

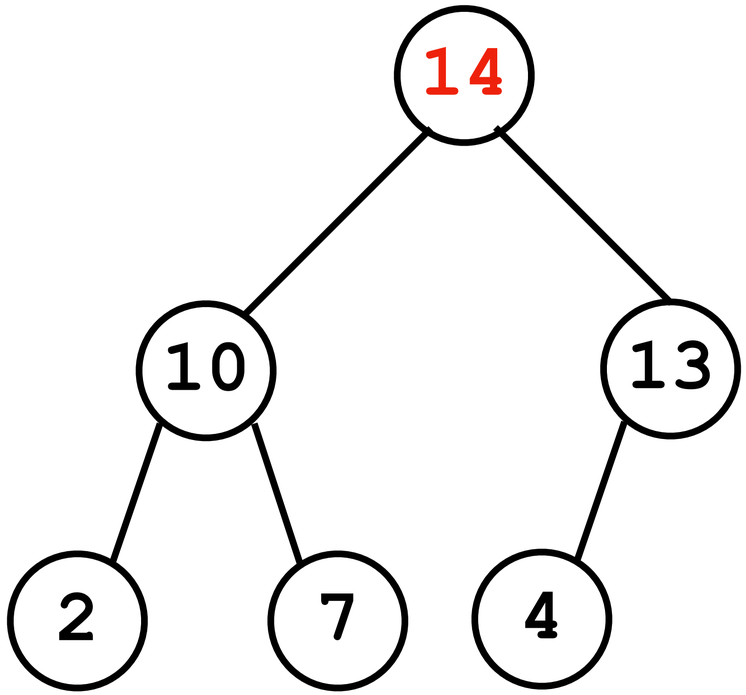




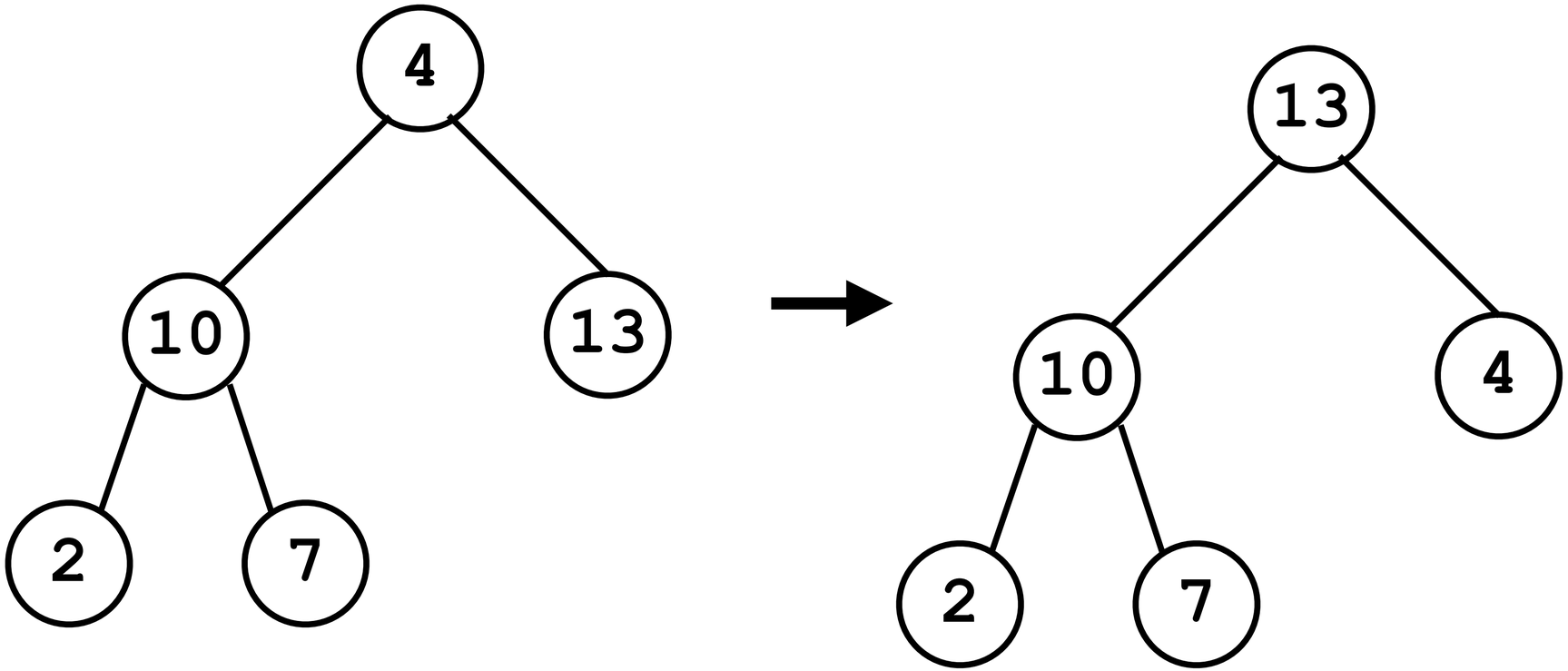
{23, 10, 14, 2, 7, 4, 13}



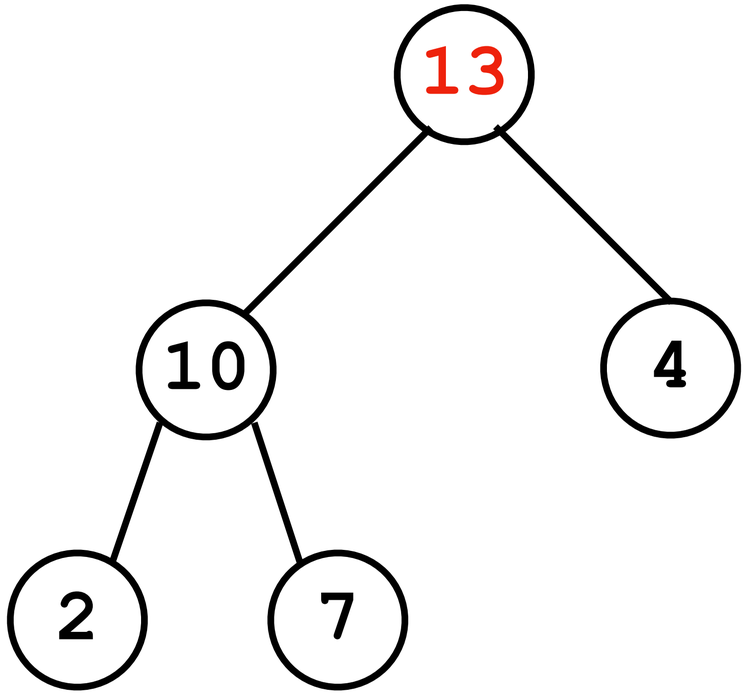
{13 10 14 2 7 4 23} => {14 10 13 2 7 4 23}



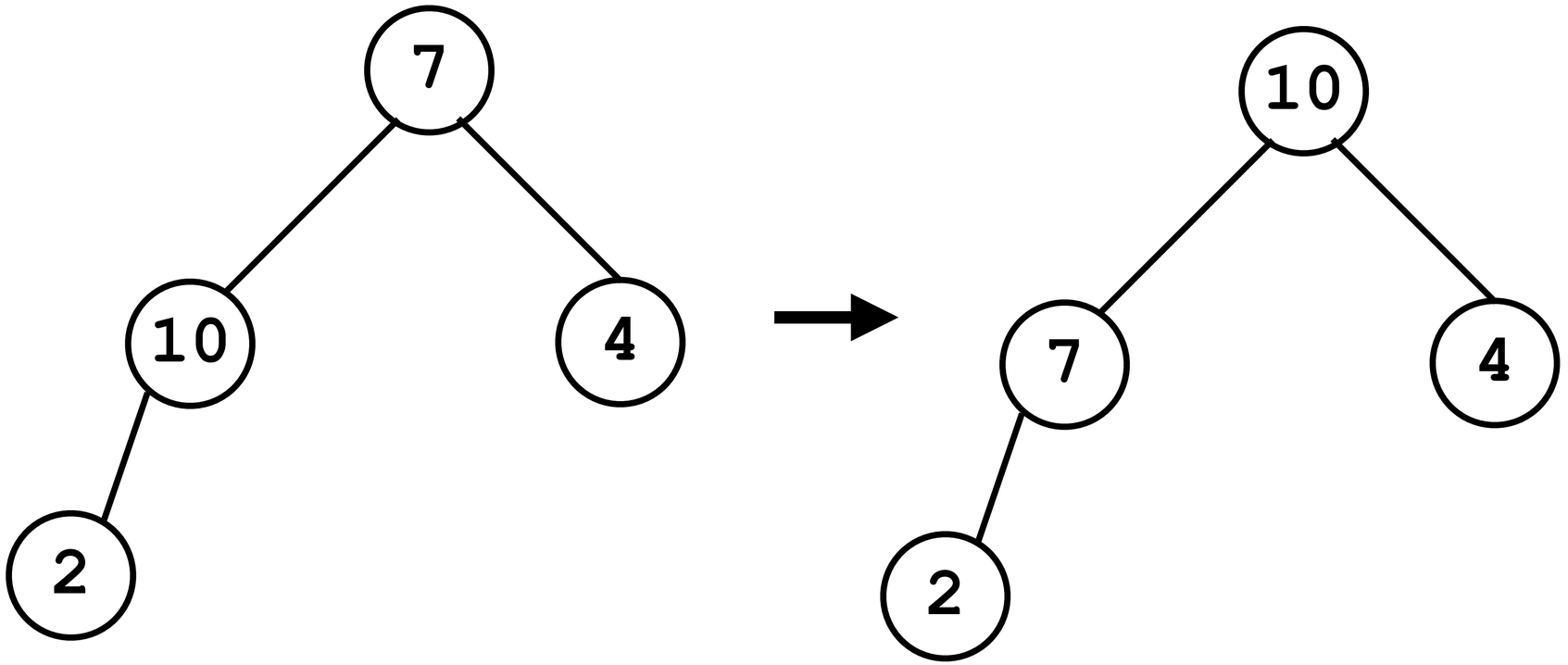
{14 10 13 2 7 4 23}



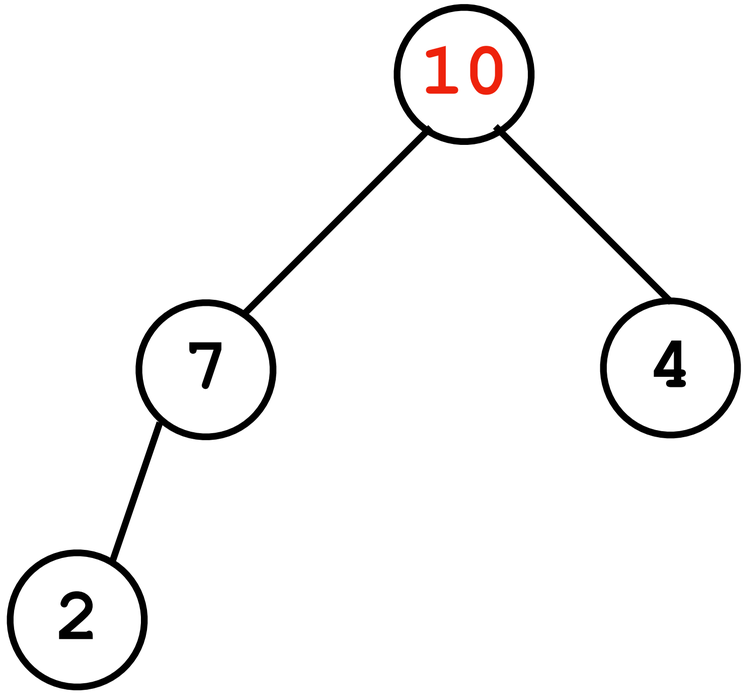
{4 10 13 2 7 14 23} => {13 10 4 2 7 14 23}



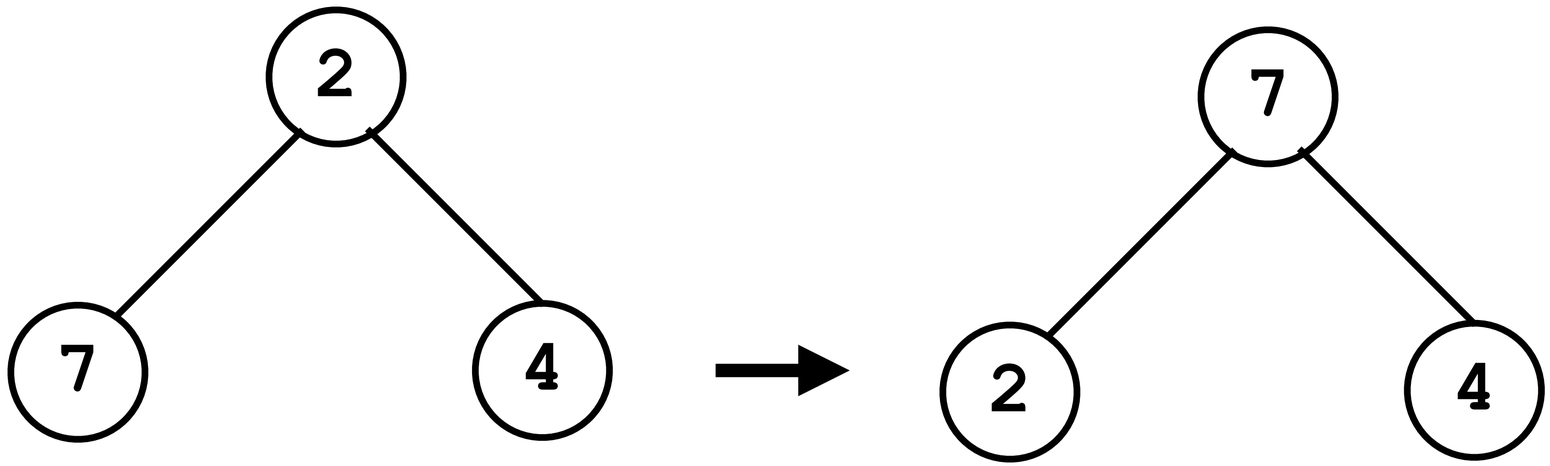
{13 10 4 2 7 14 23}



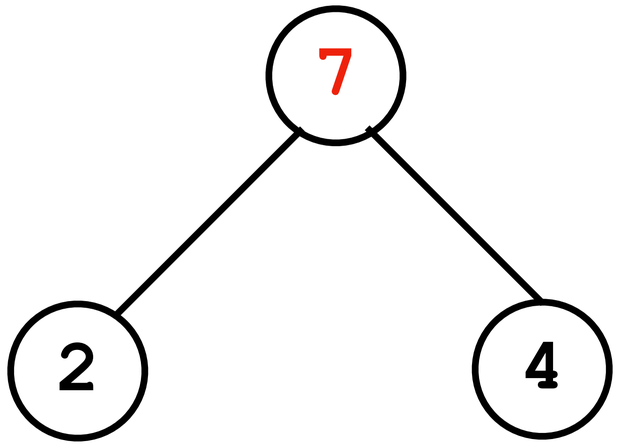
{7 10 4 2 13 14 23} => {10 7 4 2 13 14 23}



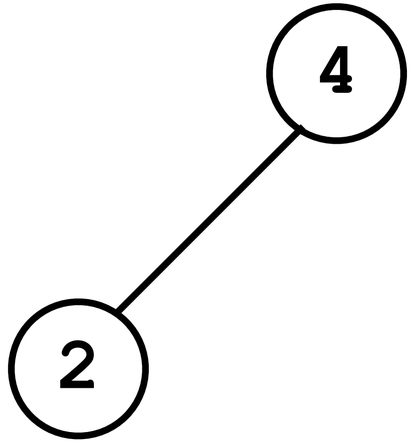
{10 7 4 2 13 14 23}



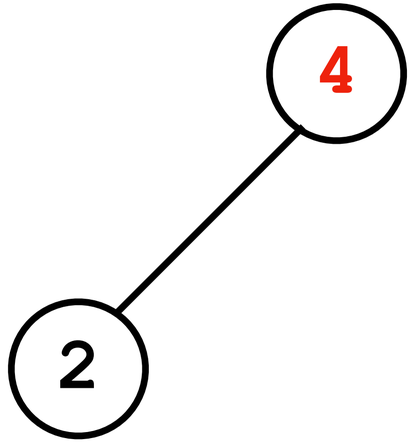
{2 7 4 10 13 14 23} => {7 4 2 10 13 14 23}



{7 2 4 10 13 14 23}



{4 2 7 10 13 14 23}



{2 4 7 10 13 14 23}

{2 4 7 10 13 14 23}

{2 4 7 10 13 14 23}

{2 4 7 10 13 14 23}

Bucket Sort (Radix Sort)

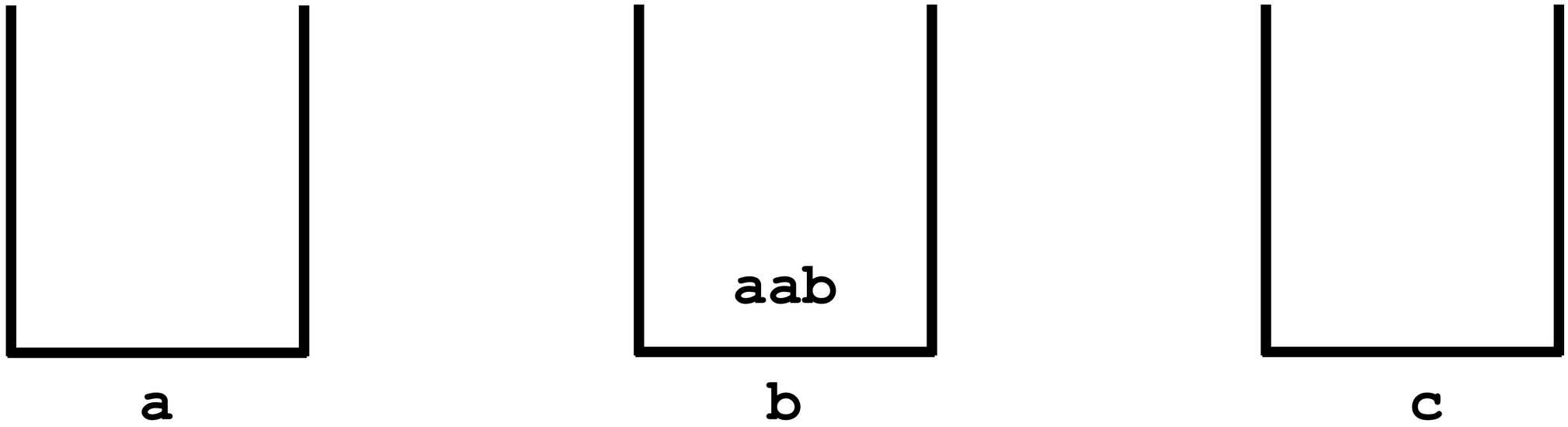
- **BucketSort sorts an array by starting from the least-significant digit and moving to the most-significant digit. At each digit it, it places the element into a bucket and then empties the bucket back into the array.**
- **Best Case Time Complexity: O(nk)**
- **Worst Case Time Complexity: O(nk)**

Bucket Sort

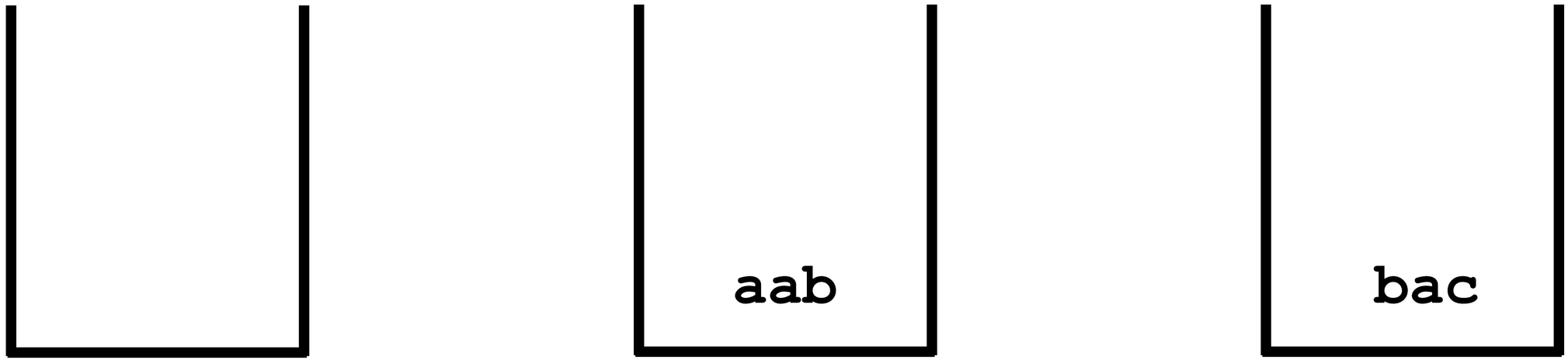
Sort the following array using bucket sort.

`{aab, bac, cba, bca, abc, cca}`

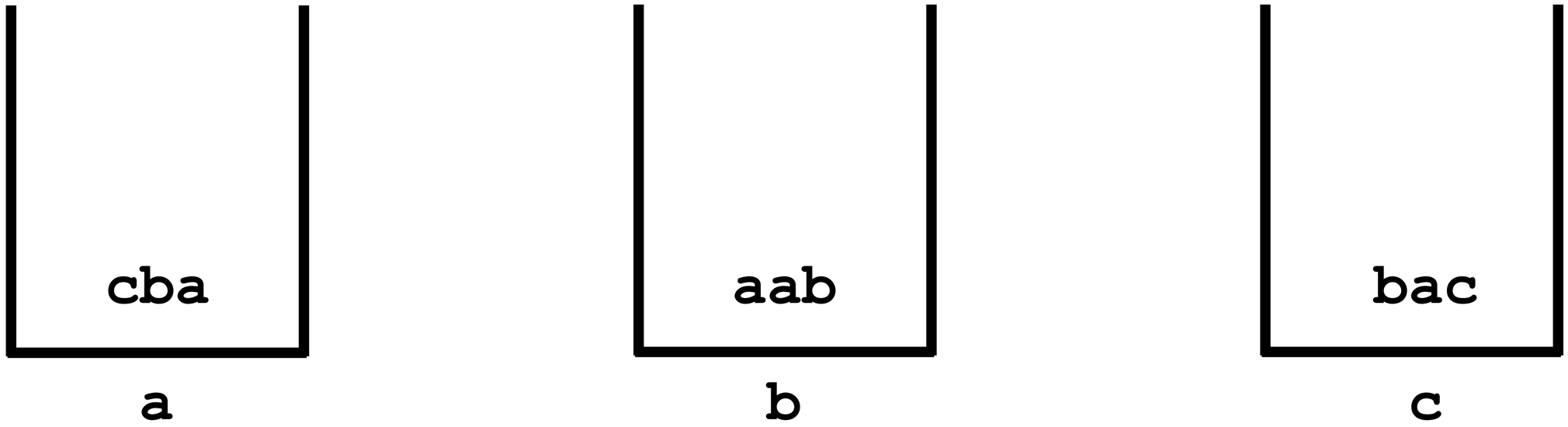
{aab, bac, cba, bca, abc, cca}



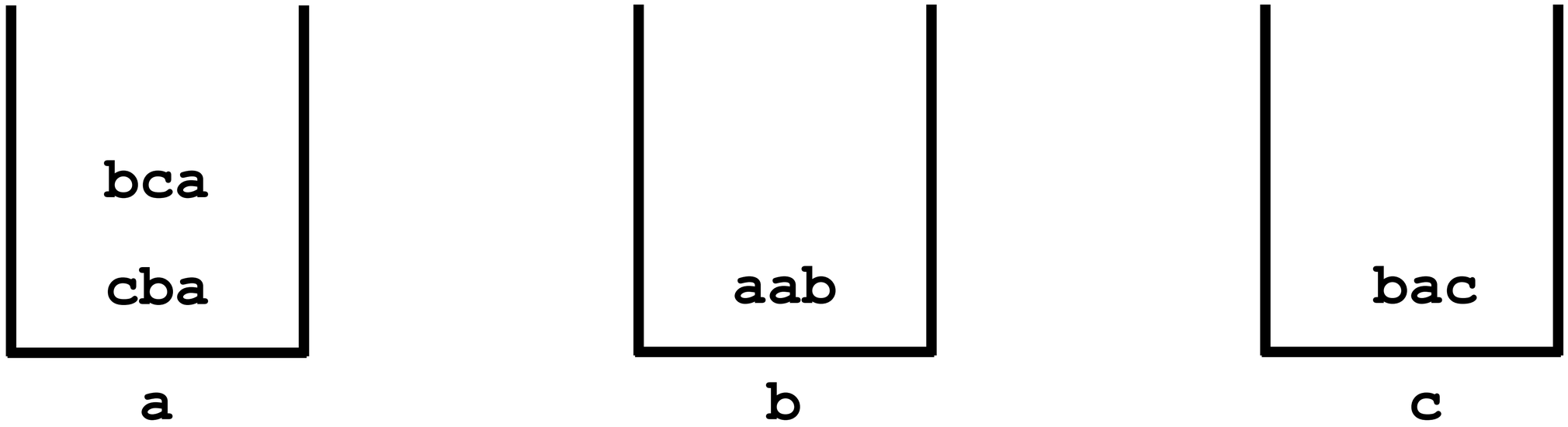
{aab, bac, cba, bca, abc, cca}



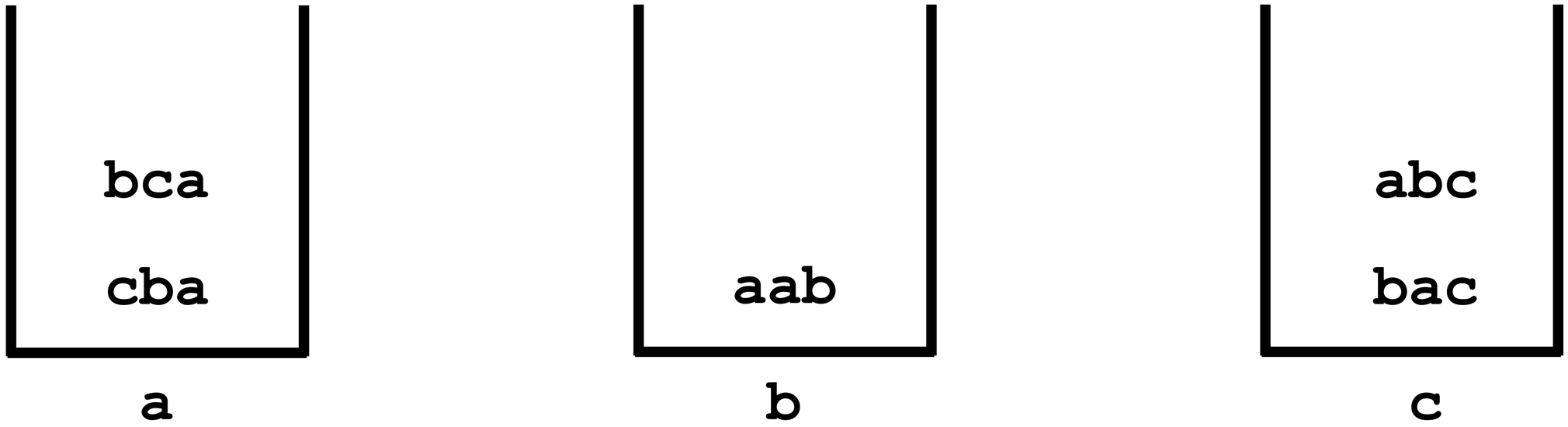
{aab, bac, cba, bca, abc, cca}



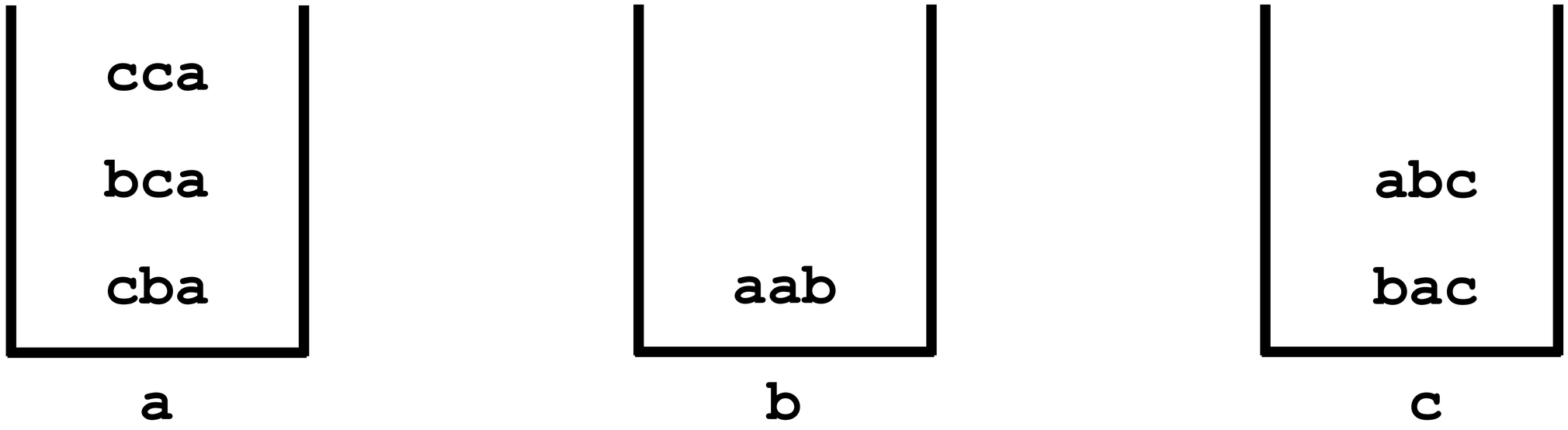
{aab, bac, cba, bca, abc, cca}



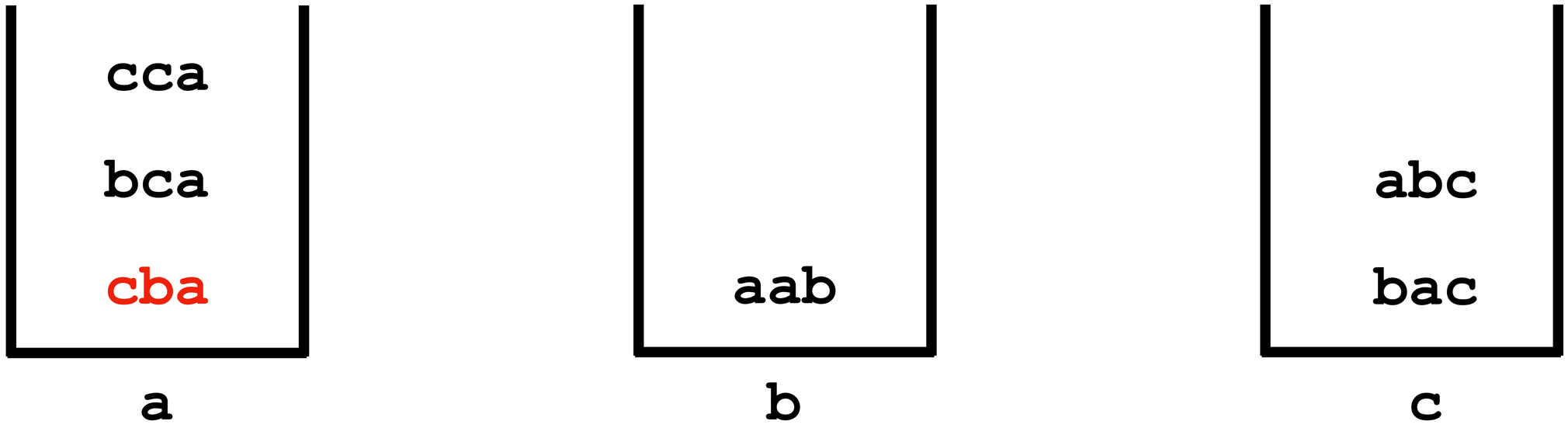
{aab, bac, cba, bca, abc, cca}



{aab, bac, cba, bca, abc, cca}

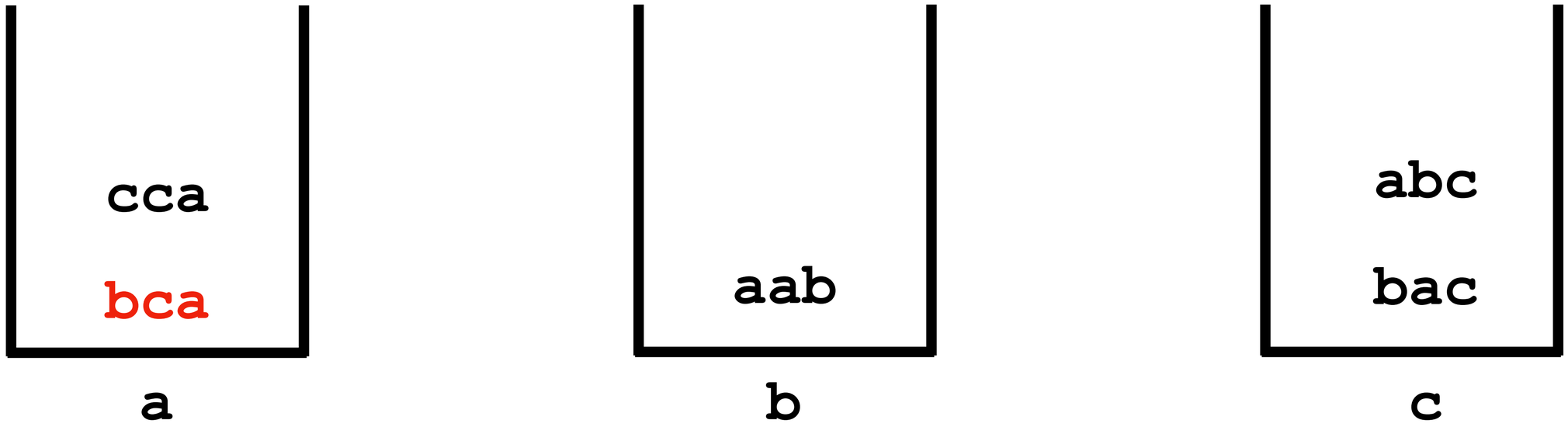


{aab, bac, cba, bca, abc, cca}



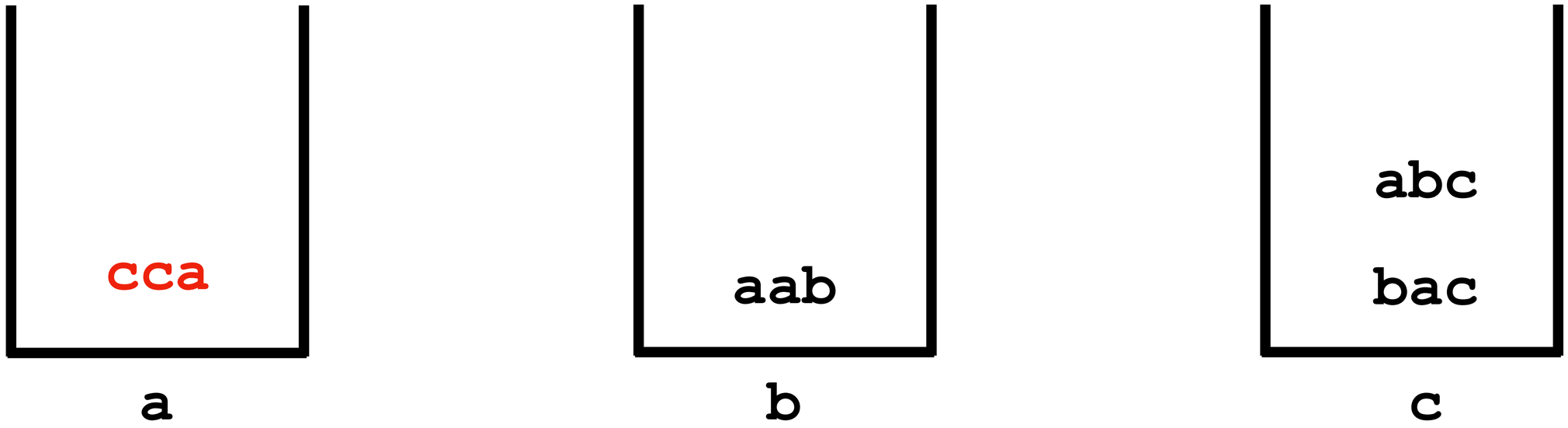
{cba

{aab, bac, cba, bca, abc, cca}



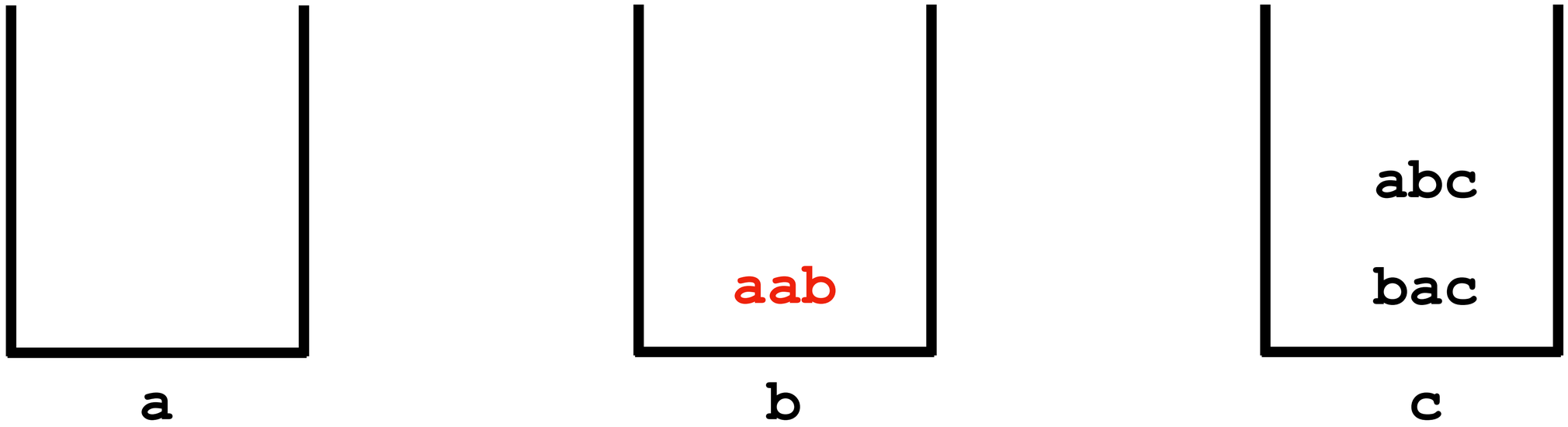
{cba, bca

{aab, bac, cba, bca, abc, cca}



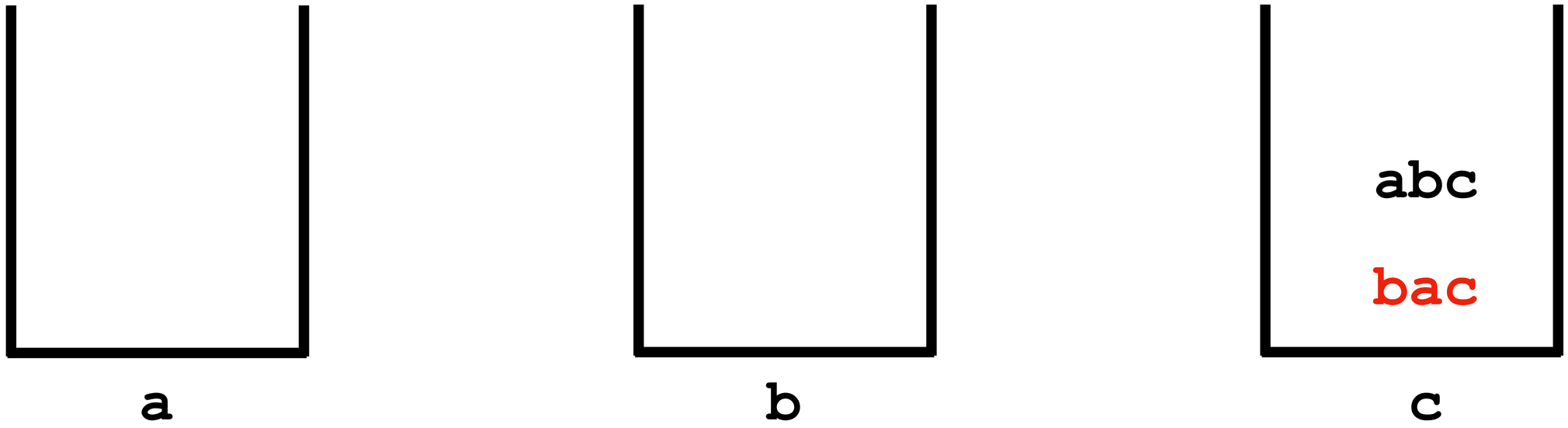
{cba, bca, cca

{aab, bac, cba, bca, abc, cca}



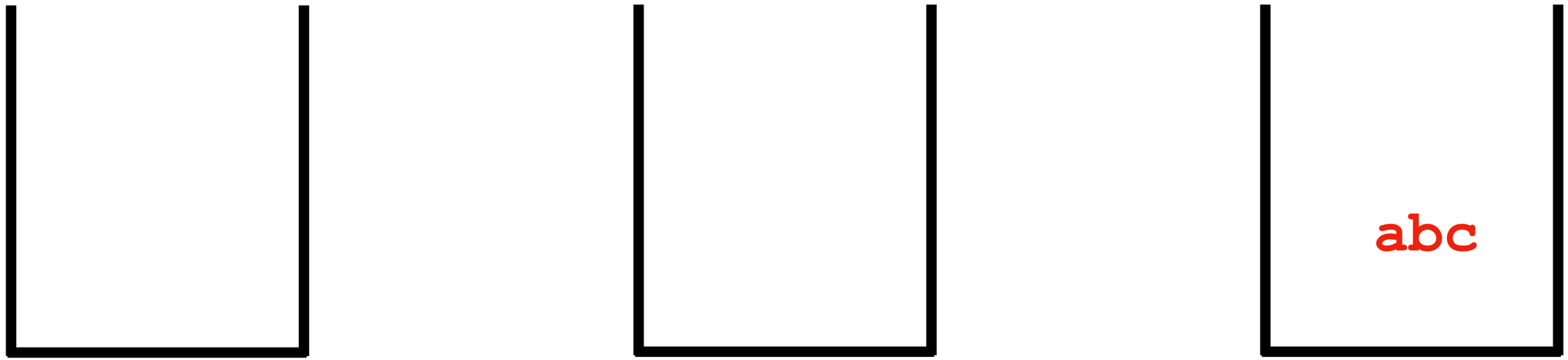
{cba, bca, cca, aab

{aab, bac, cba, bca, abc, cca}



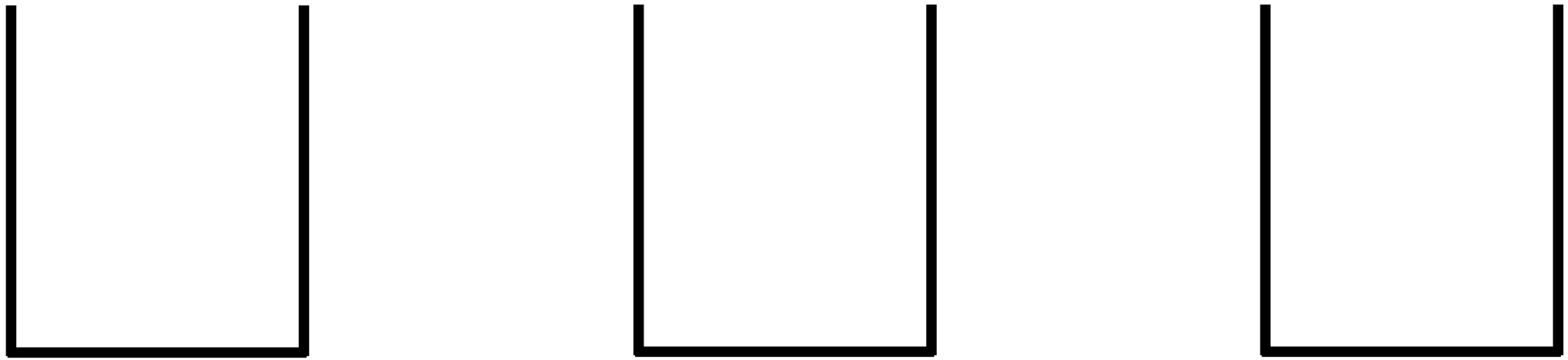
{cba, bca, cca, aab, bac

{aab, bac, cba, bca, abc, cca}



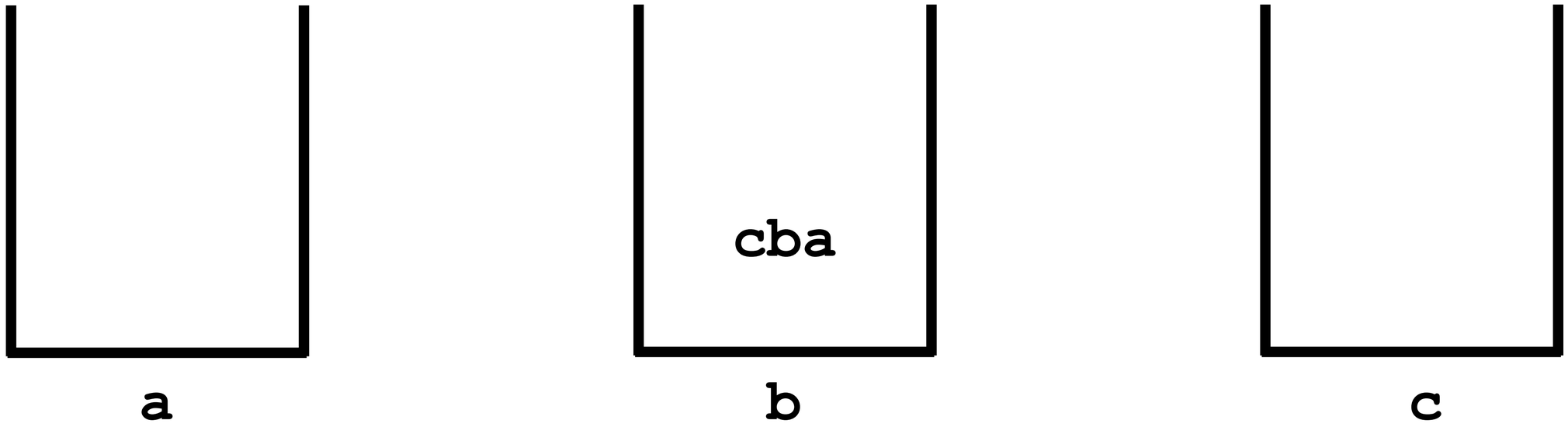
{cba, bca, cca, aab, bac, abc

{aab, bac, cba, bca, abc, cca}

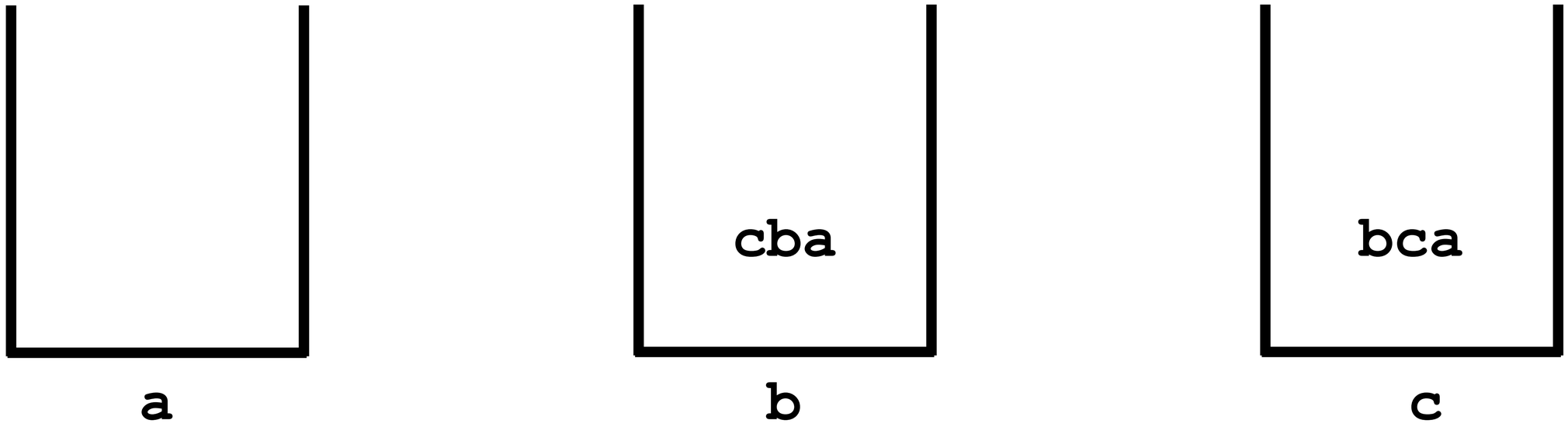


{cba, bca, cca, aab, bac, abc}

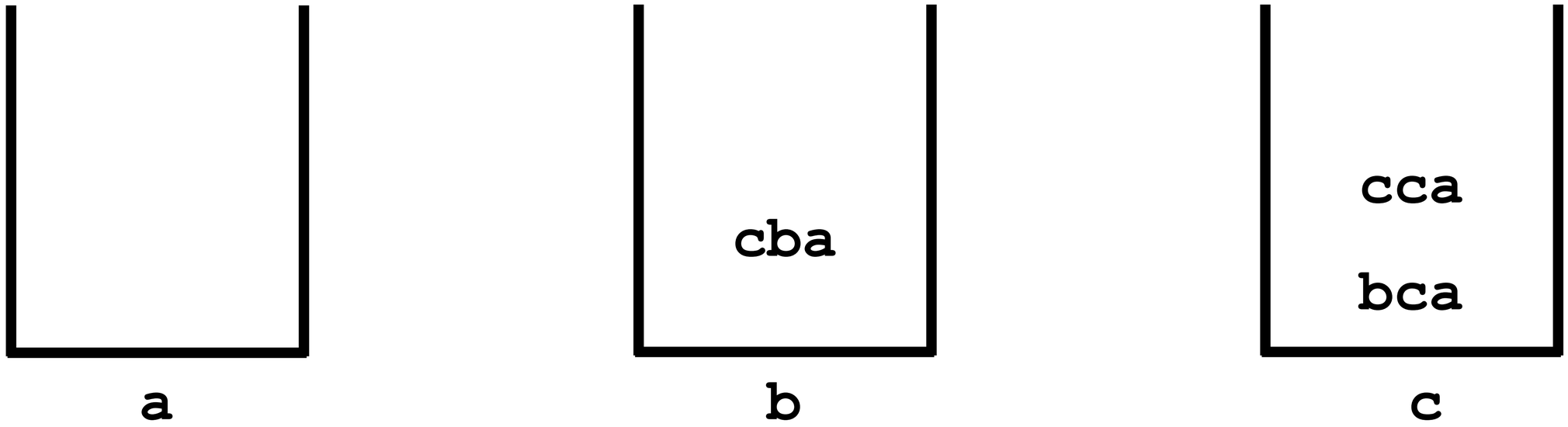
{cba, bca, cca, aab, bac, abc}



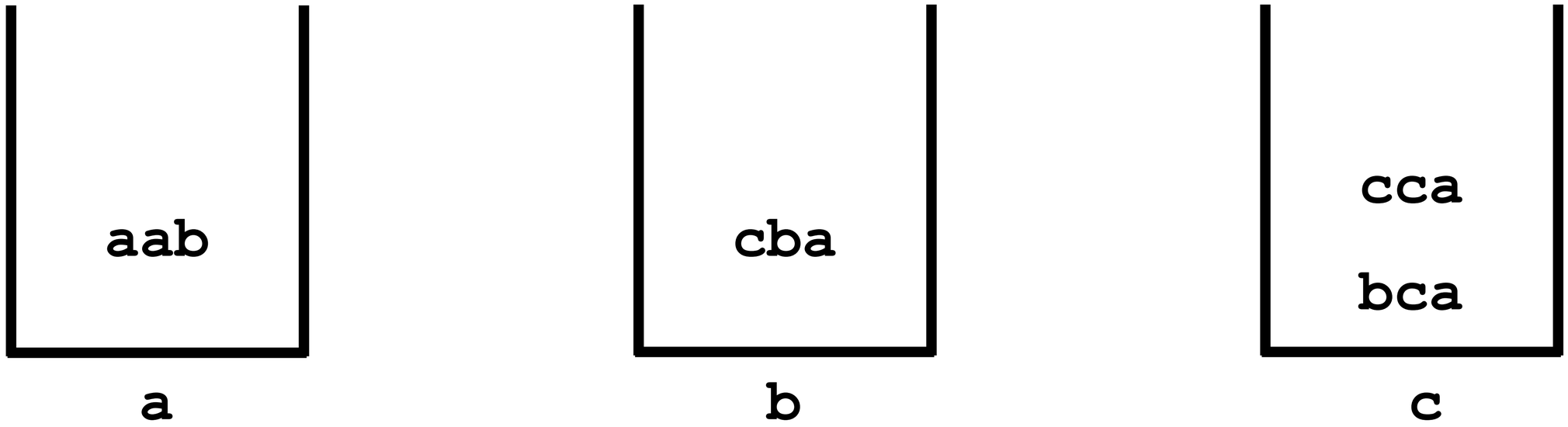
{cba, bca, cca, aab, bac, abc}



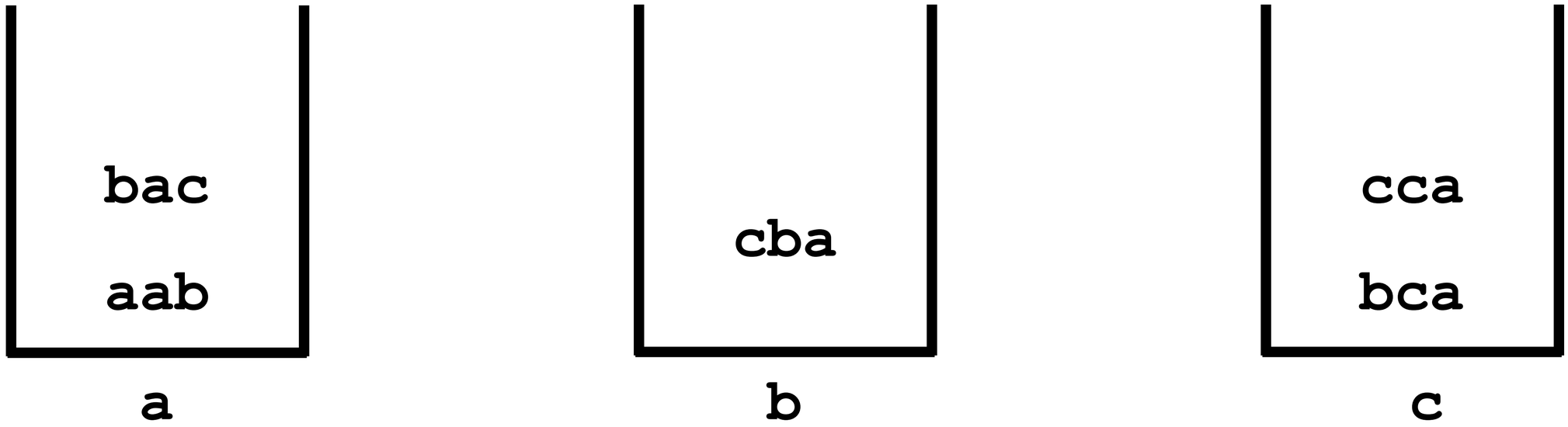
{cba, bca, cca, aab, bac, abc}



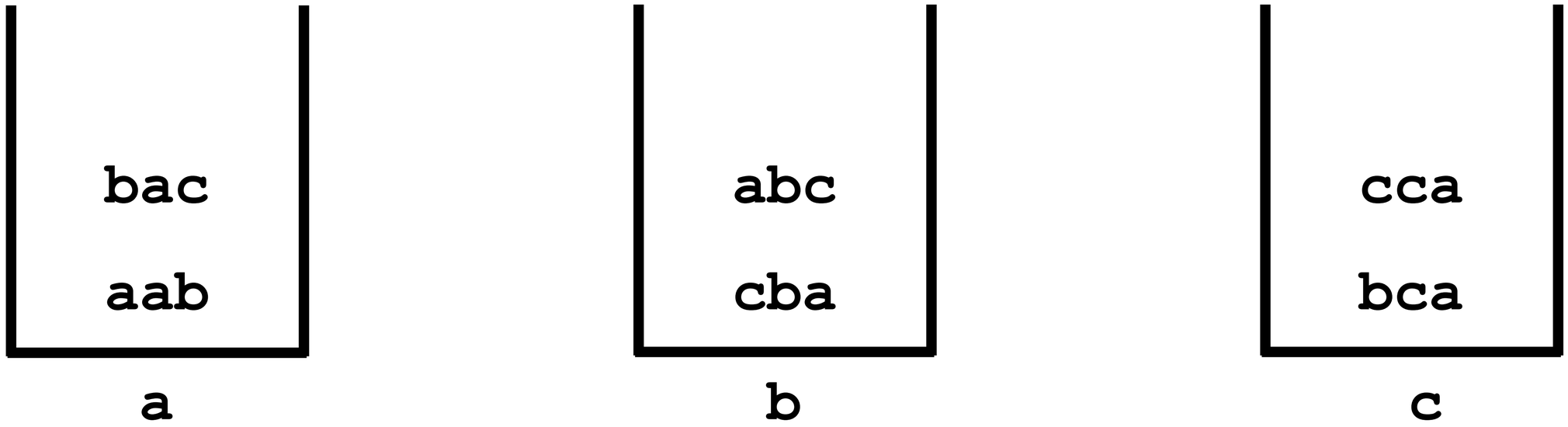
{cba, bca, cca, aab, bac, abc}



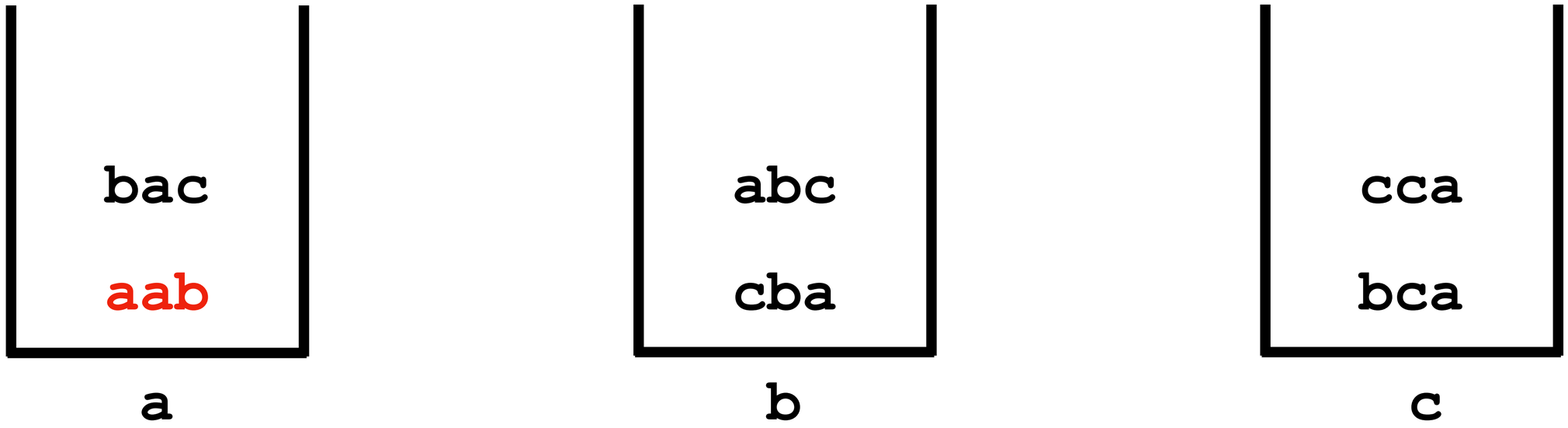
{cba, bca, cca, aab, bac, abc}



{cba, bca, cca, aab, bac, abc}

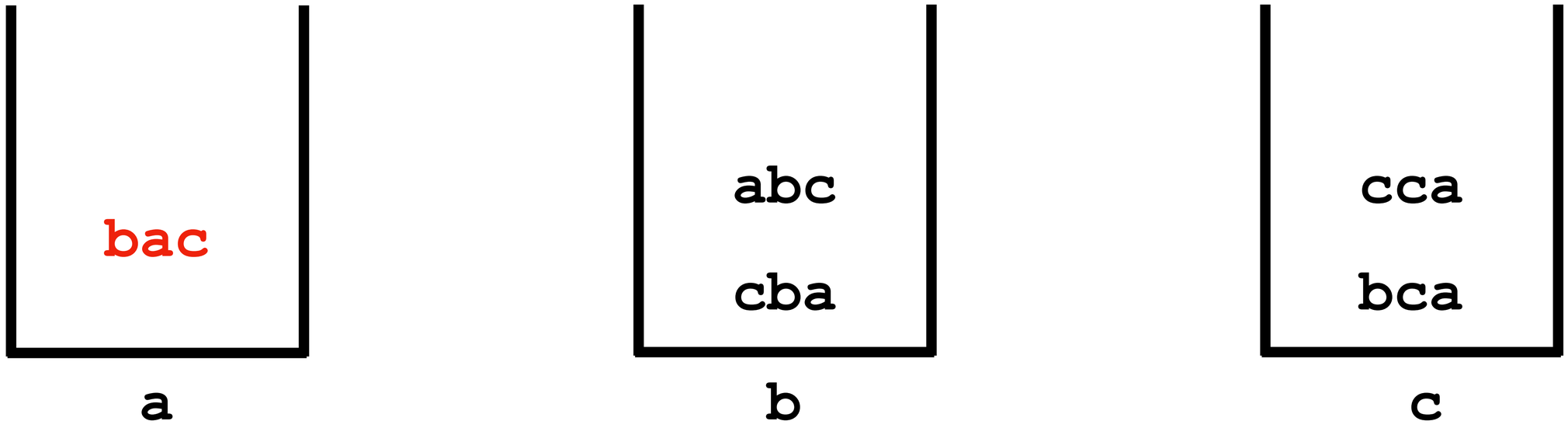


{cba, bca, cca, aab, bac, abc}



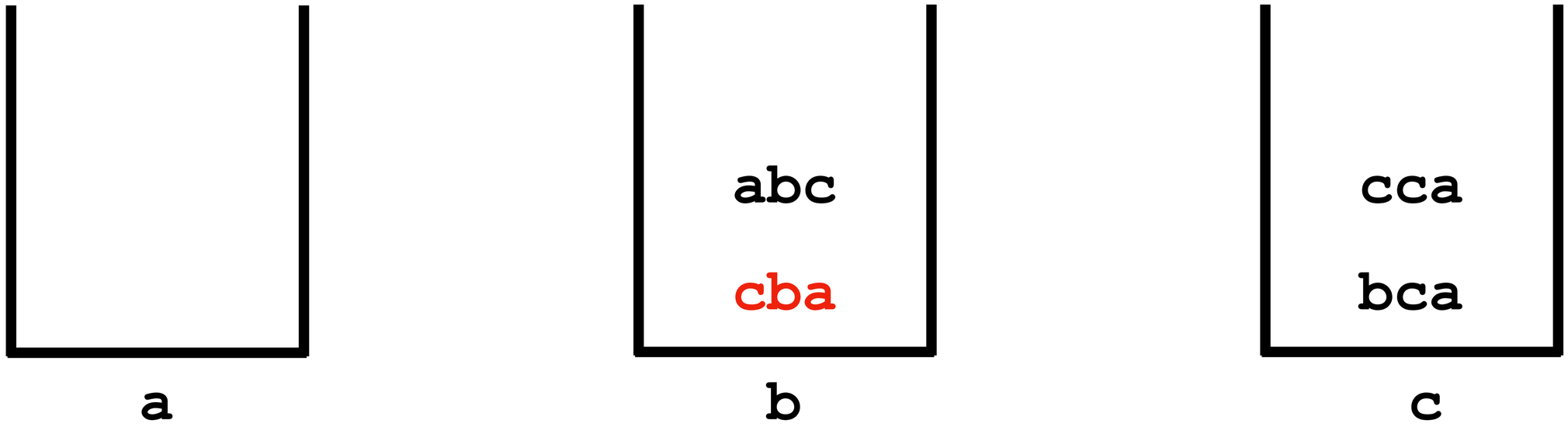
{aab

{cba, bca, cca, aab, bac, abc}



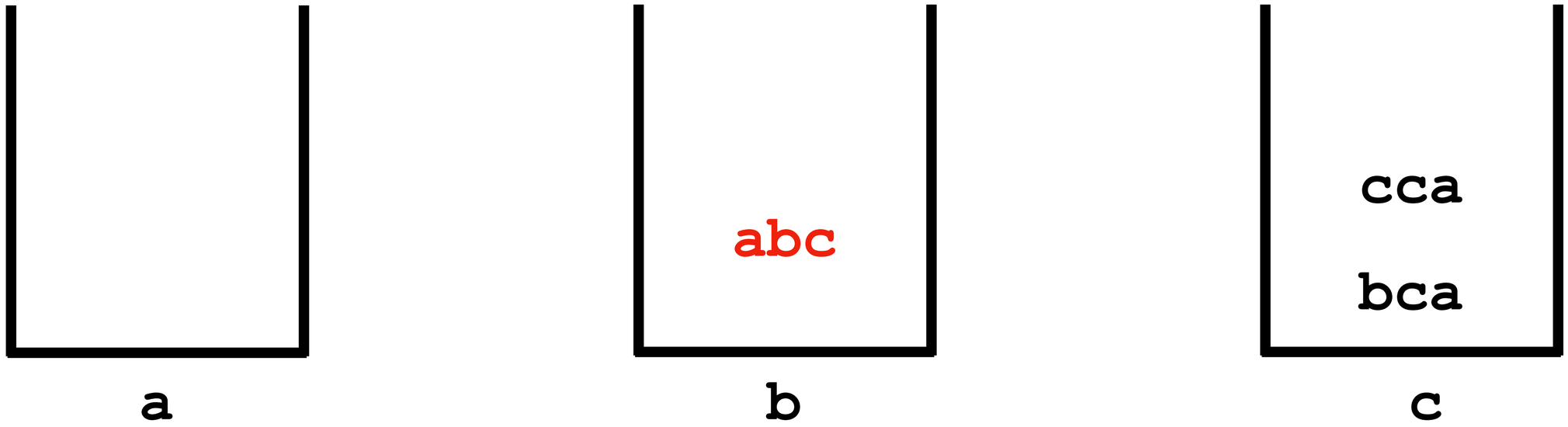
{aab, bac

{cba, bca, cca, aab, bac, abc}



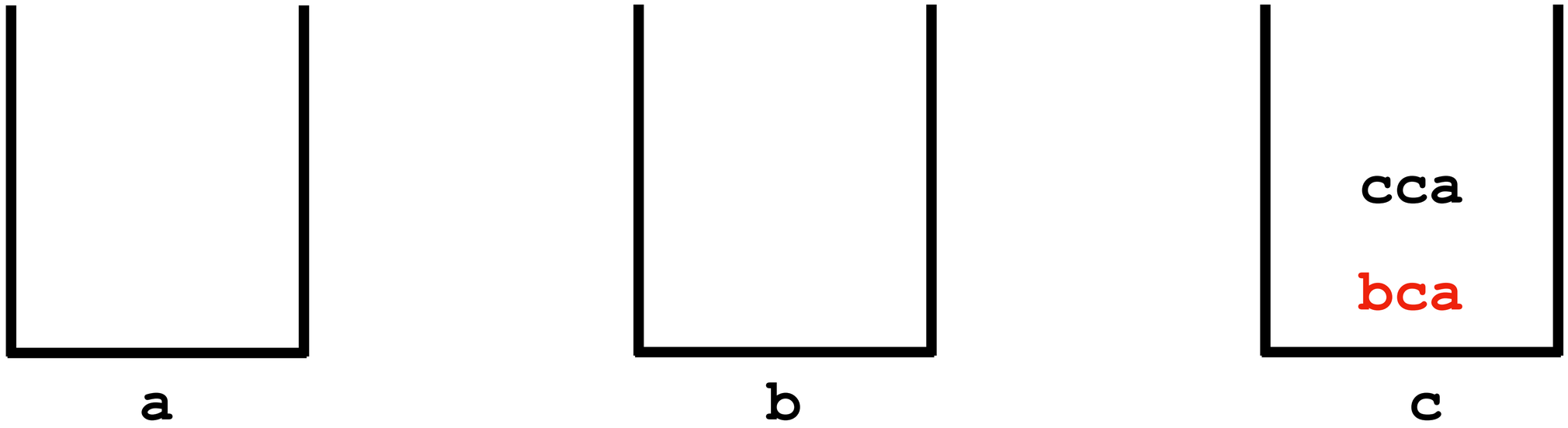
{aab, bac, cba

{cba, bca, cca, aab, bac, abc}



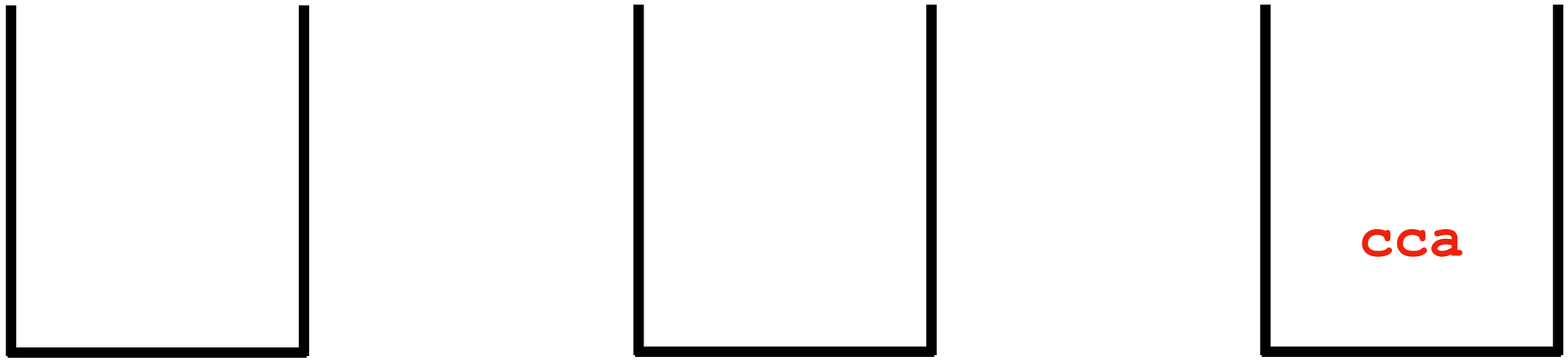
{aab, bac, cba, abc

{cba, bca, cca, aab, bac, abc}



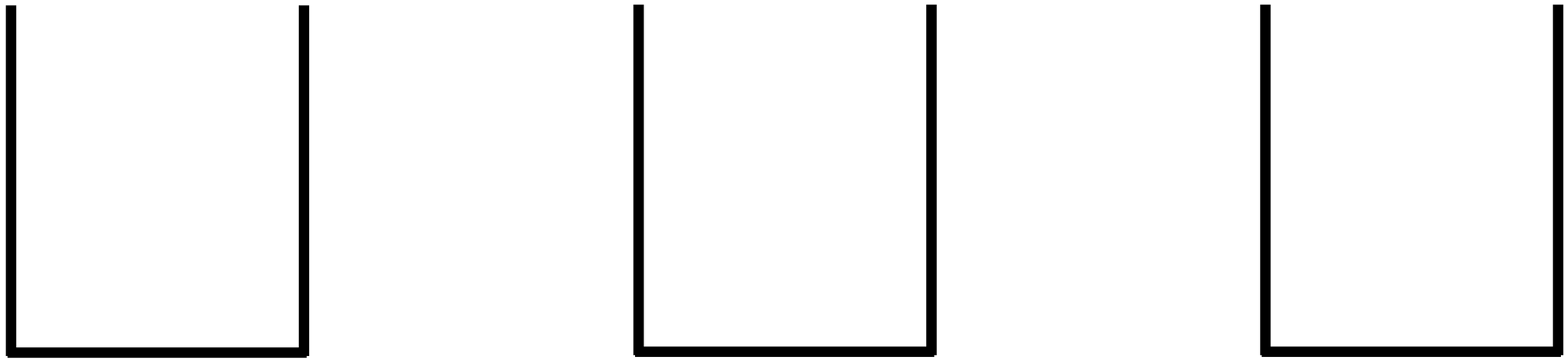
{aab, bac, cba, abc, bca

{cba, bca, cca, aab, bac, abc}



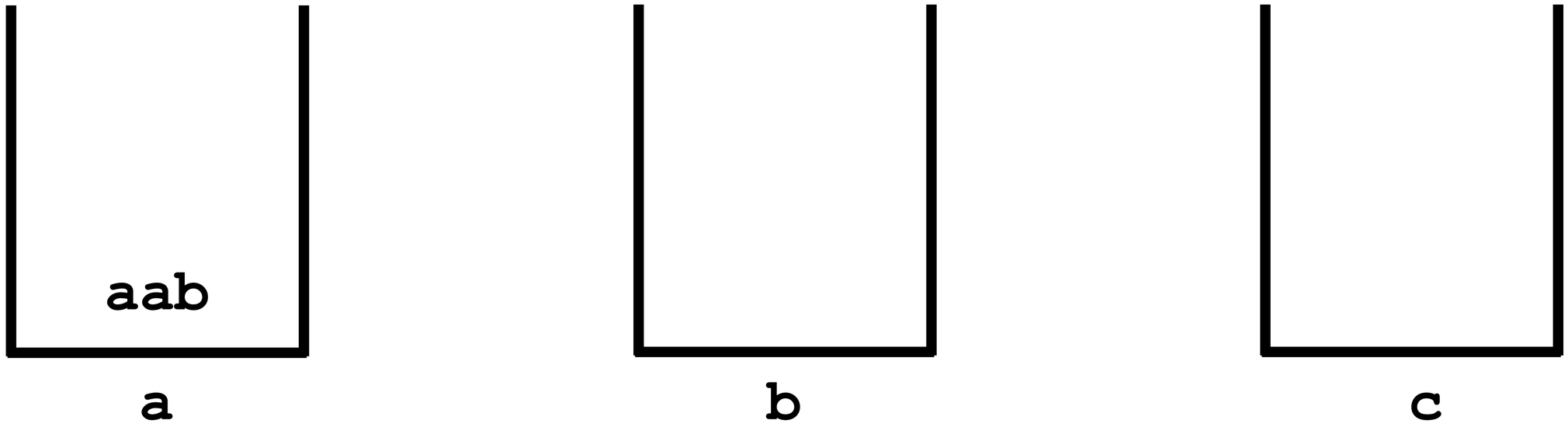
{aab, bac, cba, abc, bca, cca

{cba, bca, cca, aab, bac, abc}

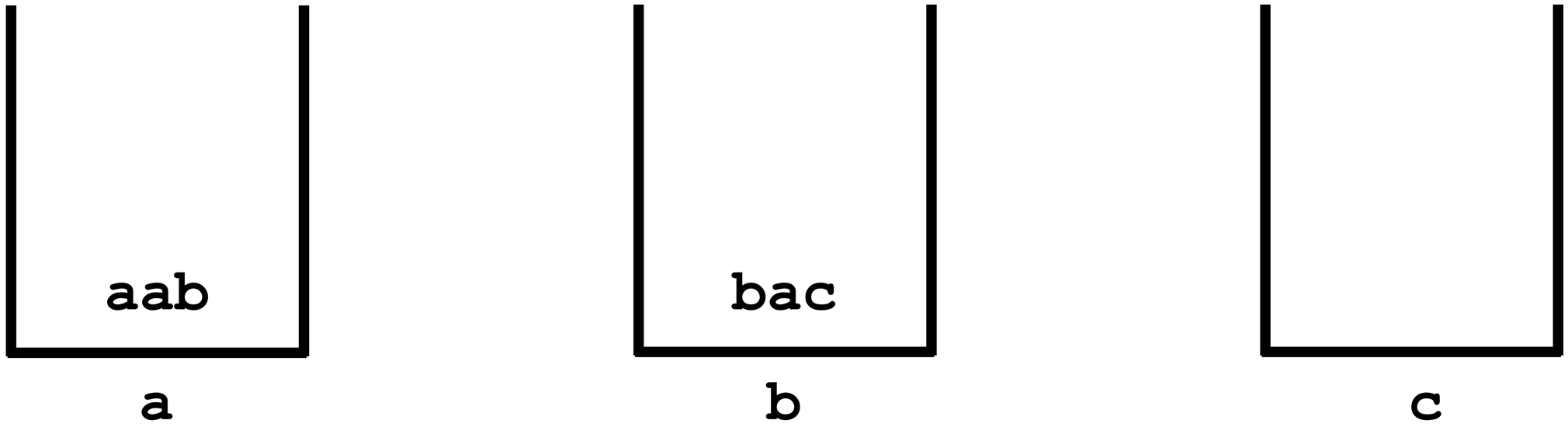


{aab, bac, cba, abc, bca, cca}

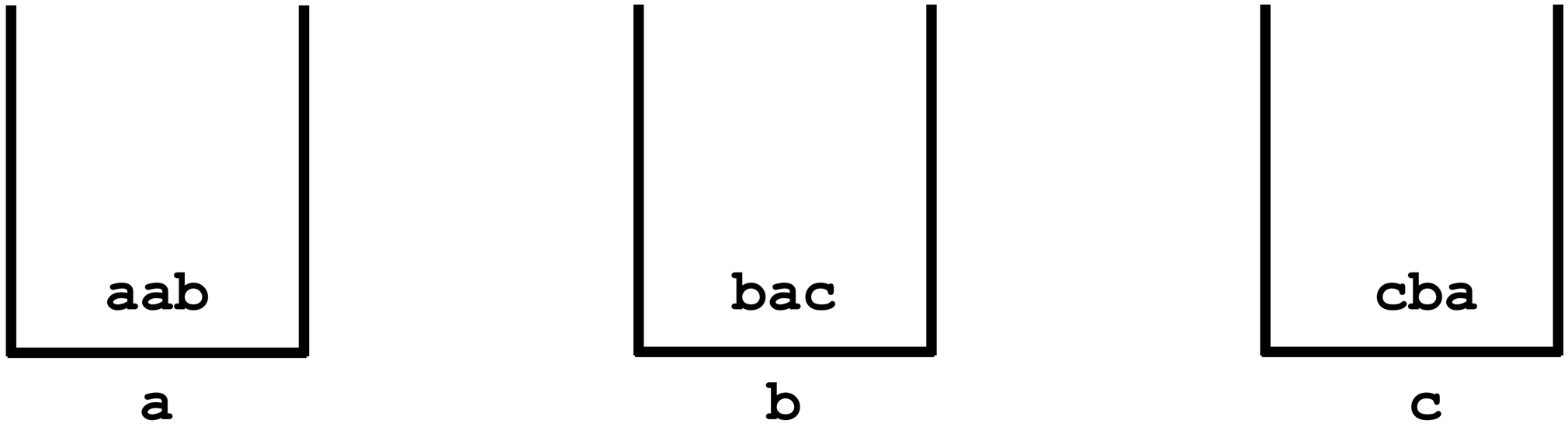
{aab, bac, cba, abc, bca, cca}



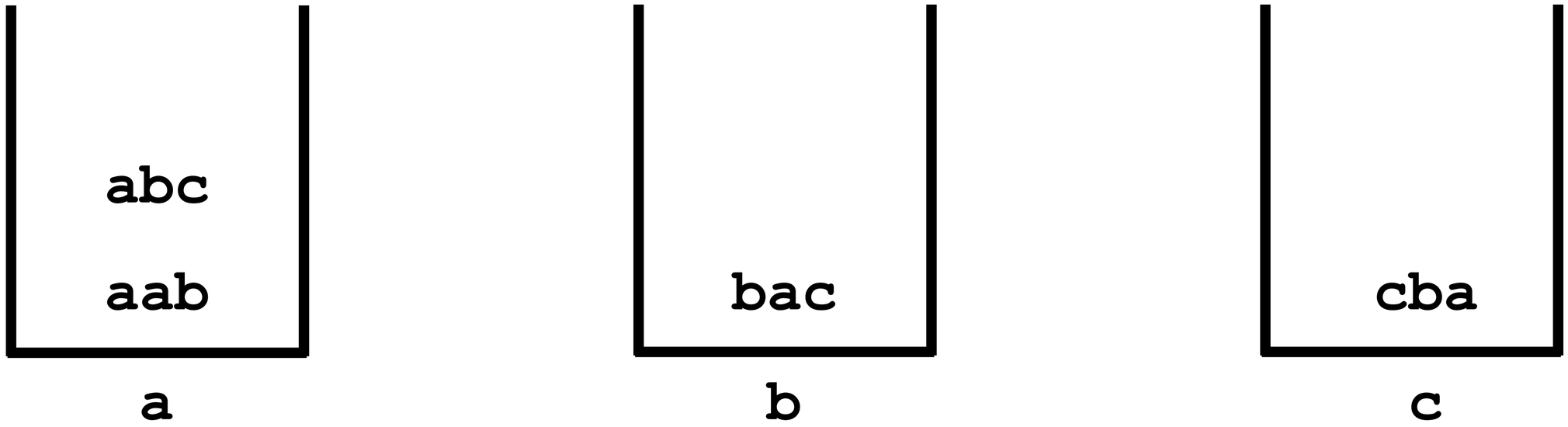
{aab, bac, cba, abc, bca, cca}



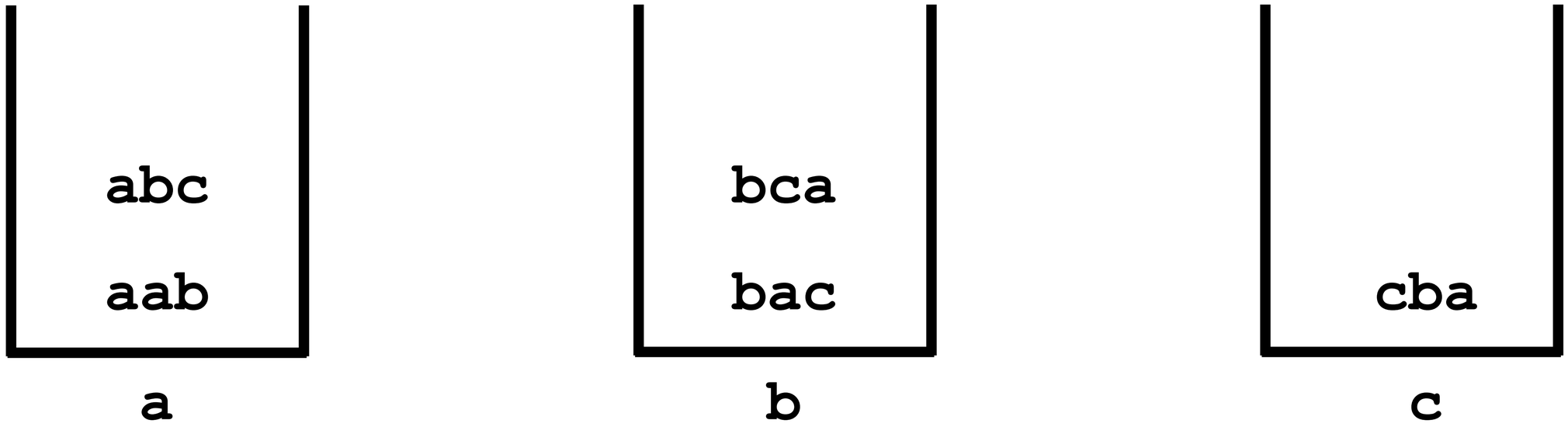
{aab, bac, cba, abc, bca, cca}



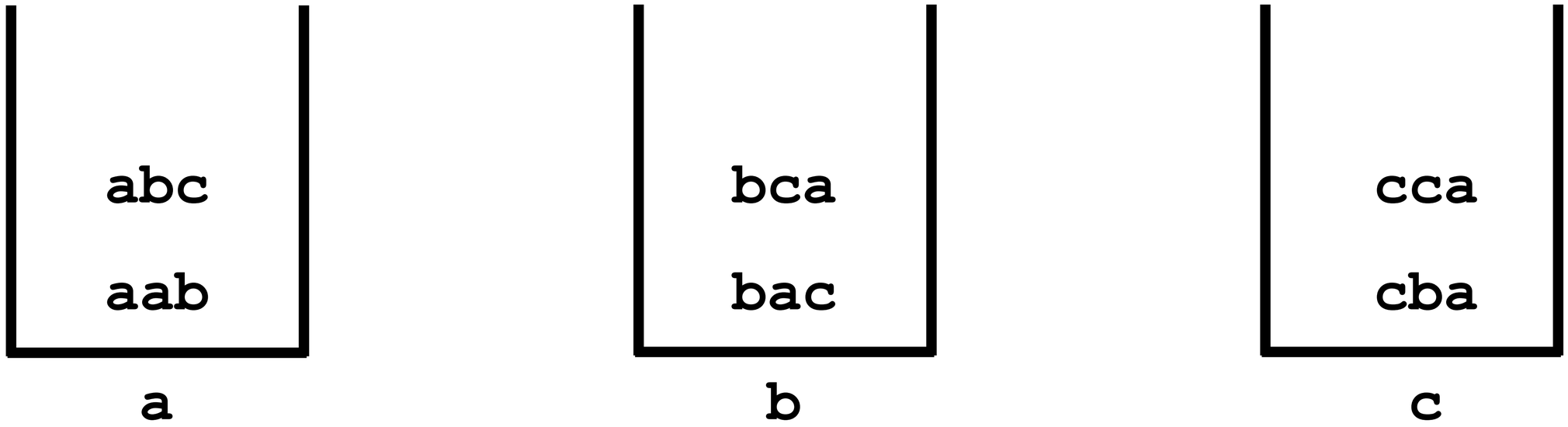
{aab, bac, cba, abc, bca, cca}



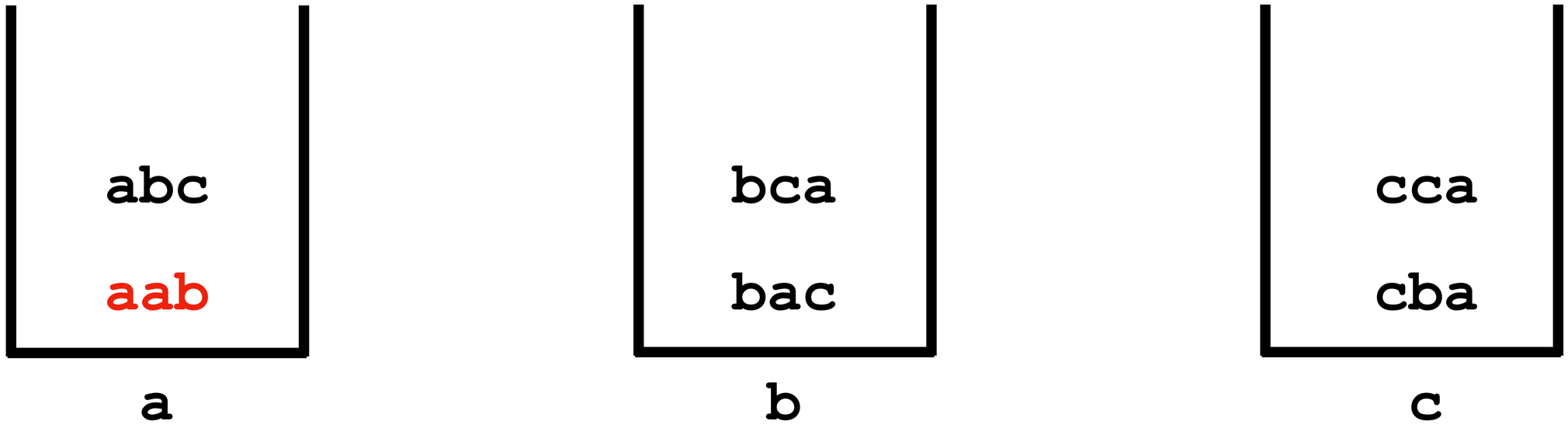
{aab, bac, cba, abc, bca, cca}



{aab, bac, cba, abc, bca, cca}

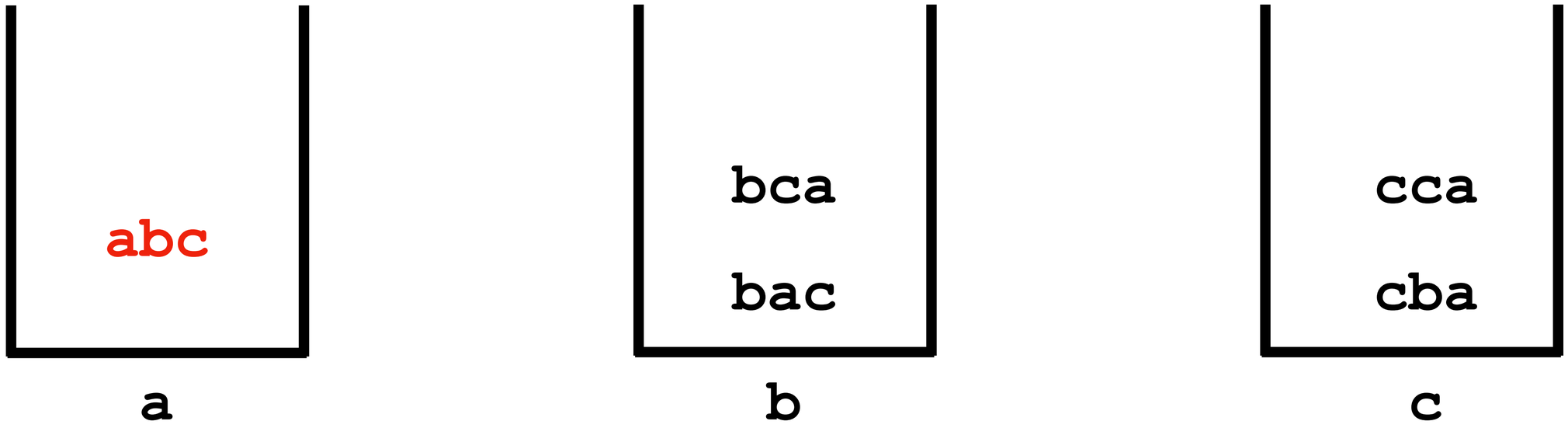


{aab, bac, cba, abc, bca, cca}



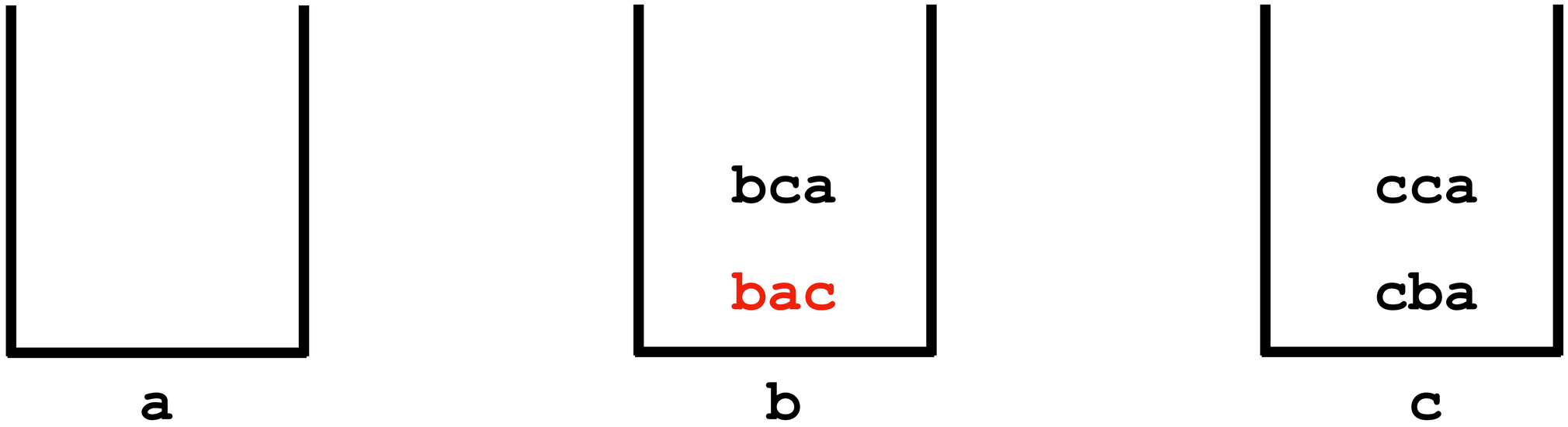
{aab

{aab, bac, cba, abc, bca, cca}



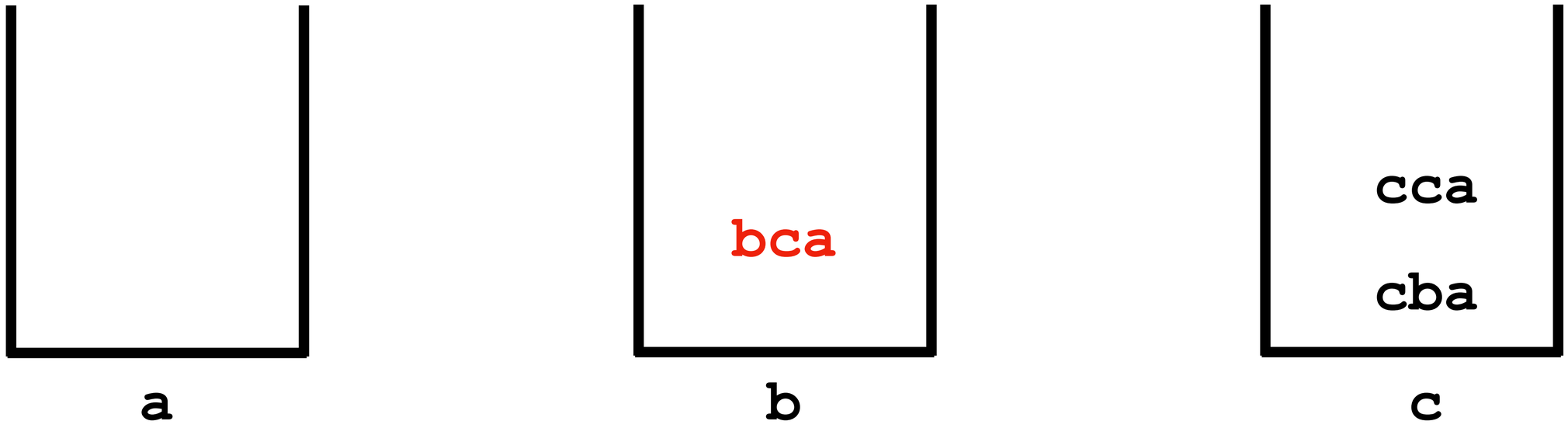
{aab, abc

{aab, bac, cba, abc, bca, cca}



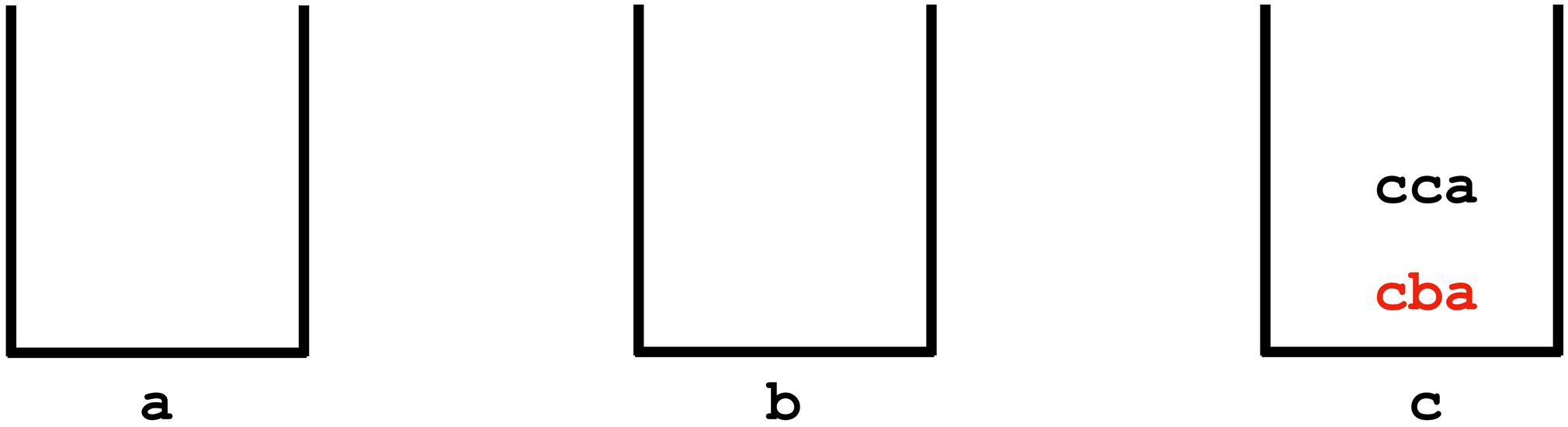
{aab, abc, bac

{aab, bac, cba, abc, bca, cca}



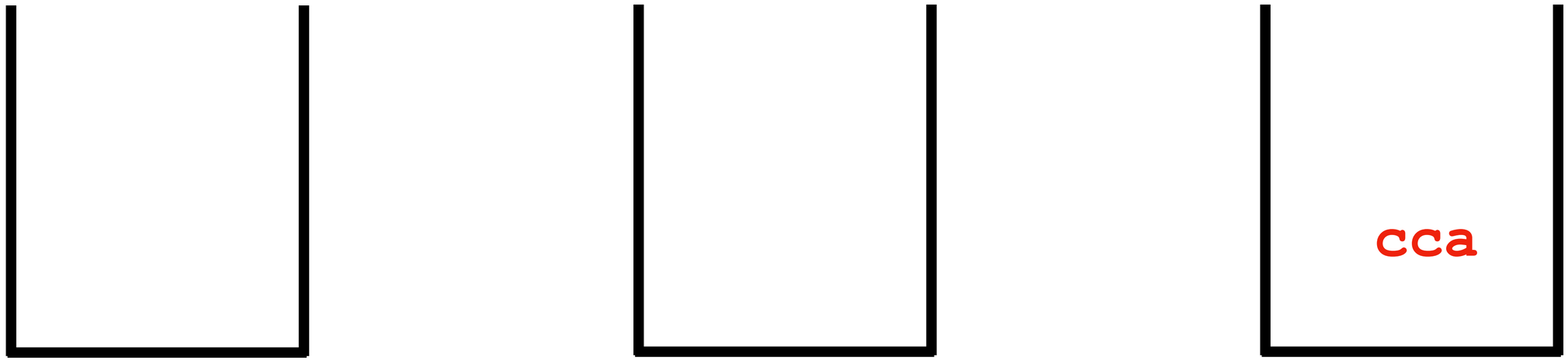
{aab, abc, bac, bca

{aab, bac, cba, abc, bca, cca}



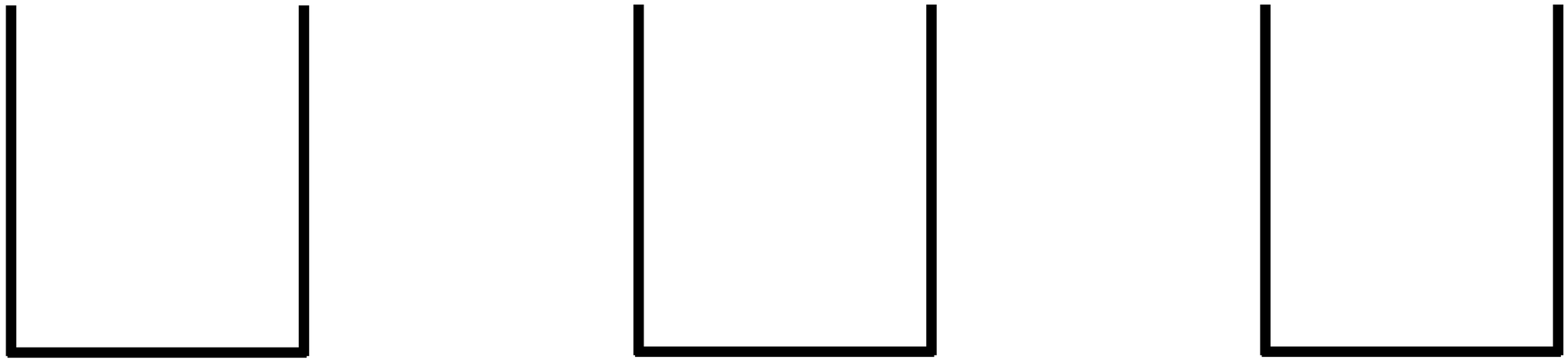
{aab, abc, bac, bca, cba

{aab, bac, cba, abc, bca, cca}



{aab, abc, bac, bca, cba, cca

{aab, bac, cba, abc, bca, cca}



{aab, abc, bac, bca, cba, cca}

{aab, abc, bac, bca, cba, cca}