1. Given an encoded string, return its decoded string.

   The encoding rule is: `k[encoded_string]`, where the `encoded_string` inside the square brackets is being repeated exactly `k` times. Note that `k` is guaranteed to be a positive integer. You may assume that the input string is always valid; No extra white spaces, square brackets are well-formed, etc.

   Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, `k`. For example, there won't be input like `3a` or `2[4]`.

   Example 1:

   ```
   Input: s = "3[a]2[bc]"

   Output: "aaabcbc"
   ```

   Example 2:

   ```
   Input: s = "3[a2[c]]"

   Output: "accaccacc"
   ```

   Example 3:

   ```
   Input: s = "2[abc]3[cd]ef"

   Output: "abcabccdcdcdef"
   ```

   Example 4:

   ```
   Input: s = "abc3[cd]xyz"

   Output: "abccdcdcdxyz"
   ```

   Constraints:

   - `1 <= s.length <= 30`
   - `s` consists of lowercase English letters, digits, and square brackets `'[]'`.
   - `s` is guaranteed to be **a valid** input.
   - All the integers in `s` are in the range `[1, 300]`.

2. Given a string `s`, remove duplicate letters so that every letter appears once and only once. You must make sure your result is **the smallest in lexicographical order** among all possible results.

Example 1:

```
Input: s = "bcabc"

Output: "abc"
```

Example 2:

```
Input: s = "cbacdcbc"

Output: "acdb"
```

Constraints:

- `1 <= s.length <= 10`
- `s` consists of lowercase English letters.

3. Given string num representing a non-negative integer `num`, and an integer `k`, return *the smallest possible integer after removing* `k` *digits from* `num`.

Example 1:

```
Input: num = "1432219", k = 3

Output: "1219"
```

Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.

Example 2:

```
Input: num = "10200", k = 1

Output: "200"
```

Explanation: Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes.

Example 3:

```
Input: num = "10", k = 2

Output: "0"
```

```
Explanation: Remove all the digits from the number and it is left with nothing
which is 0.
```

Constraints:

- `1 <= k <= num.length <= 10^5`
- `num` consists of only digits.
- `num` does not have any leading zeros except for the zero itself

4. Given a string `s`, determine if it is **valid**.

A string `s` is **valid** if, starting with an empty string `t = ""`, you can **transform** `t` **into** `s` after performing the following operation **any number of times**:

- Insert string `"abc"` into any position in `t`. More formally, `t` becomes t_{left} `+ "abc" +` t_{right}, where `t ==` t_{left} `+` t_{right}. Note that t_{left} and t_{right} may be **empty**.

Return `true` *if* `s` *is a **valid** string, otherwise, return* `false`.

Example 1:

```
Input: s = "aabcbc"

Output: true

Explanation:

"" -> "abc" -> "aabcbc"

Thus, "aabcbc" is valid.
```

Example 2:

```
Input: s = "abcabcababcc"

Output: true

Explanation:

"" -> "abc" -> "abcabc" -> "abcabcabc" -> "abcabcababcc"

Thus, "abcabcababcc" is valid.
```

Example 3:

```
Input: s = "abccba"

Output: false
```

Explanation: It is impossible to get "abccba" using the operation.

Example 4:

Input: s = "cababc"

Output: false

Explanation: It is impossible to get "cababc" using the operation.

Constraints:

- `1 <= s.length <= 2 * 10^4`
- `s` consists of letters `'a'`, `'b'`, and `'c'`

5. You are given an integer array, `nums`, and an integer `k`. `nums` comprises of only `0`'s and `1`'s. In one move, you can choose two **adjacent** indices and swap their values.

Return *the **minimum** number of moves required so that* `nums` *has* `k` ***consecutive*** `1`*'s*.

Example 1:

Input: nums = [1,0,0,1,0,1], k = 2

Output: 1

Explanation: In 1 move, nums could be [1,0,0,0,1,1] and have 2 consecutive 1's.

Example 2:

Input: nums = [1,0,0,0,0,0,1,1], k = 3

Output: 5

Explanation: In 5 moves, the leftmost 1 can be shifted right until nums = [0,0,0,0,0,1,1,1].

Example 3:

Input: nums = [1,1,0,1], k = 2

Output: 0

Explanation: nums already has 2 consecutive 1's.

Constraints:

- `1 <= nums.length <= 10^5`
- `nums[i]` is `0` or `1`.
- `1 <= k <= sum(nums)`