(Array)

Problem-1

Given an integer array `nums` of length `n`, you want to create an array `ans` of length `2n` where `ans[i] == nums[i]` and `ans[i + n] == nums[i]` for `0 <= i < n` (**0-indexed**).

Specifically, `ans` is the **concatenation** of two `nums` arrays.

Return *the array* `ans`.

Example 1:

```
Input: nums = [1,2,1]

Output: [1,2,1,1,2,1]

Explanation: The array ans is formed as follows:

- ans = [nums[0],nums[1],nums[2],nums[0],nums[1],nums[2]]

- ans = [1,2,1,1,2,1]
```

Example 2:

```
Input: nums = [1,3,2,1]

Output: [1,3,2,1,1,3,2,1]

Explanation: The array ans is formed as follows:

- ans = [nums[0],nums[1],nums[2],nums[3],nums[0],nums[1],nums[2],nums[3]]

- ans = [1,3,2,1,1,3,2,1]
```

Constraints:

- `n == nums.length`
- `1 <= n <= 1000`
- `1 <= nums[i] <= 1000`

Problem-2

Given a **zero-based permutation** `nums` (**0-indexed**), build an array `ans` of the **same length** where `ans[i] = nums[nums[i]]` for each `0 <= i < nums.length` and return it.

A **zero-based permutation** `nums` is an array of **distinct** integers from `0` to `nums.length - 1` (**inclusive**).

Example 1:

```
Input: nums = [0,2,1,5,3,4]

Output: [0,1,2,4,5,3]

Explanation: The array ans is built as follows:

ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]], nums[nums[4]], nums[nums[5]]]

    = [nums[0], nums[2], nums[1], nums[5], nums[3], nums[4]]

    = [0,1,2,4,5,3]
```

Example 2:

```
Input: nums = [5,0,1,2,3,4]

Output: [4,5,0,1,2,3]

Explanation: The array ans is built as follows:

ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]], nums[nums[4]], nums[nums[5]]]

    = [nums[5], nums[0], nums[1], nums[2], nums[3], nums[4]]

    = [4,5,0,1,2,3]
```

Constraints:

- `1 <= nums.length <= 1000`
- `0 <= nums[i] < nums.length`
- The elements in `nums` are **distinct**.

Problem-3

Given an array `nums`. We define a running sum of an array as `runningSum[i] = sum(nums[0]…nums[i])`.

Return the running sum of `nums`.

Example 1:

```
Input: nums = [1,2,3,4]

Output: [1,3,6,10]

Explanation: Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].
```

Example 2:

```
Input: nums = [1,1,1,1,1]

Output: [1,2,3,4,5]

Explanation: Running sum is obtained as follows: [1, 1+1, 1+1+1, 1+1+1+1,
1+1+1+1+1].
```

Example 3:

```
Input: nums = [3,1,2,10,1]

Output: [3,4,6,16,17]
```

Constraints:

- `1 <= nums.length <= 1000`
- `-10^6 <= nums[i] <= 10^6`

Problem-4

Given the array `nums` consisting of `2n` elements in the form $[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n]$.

Return the array in the form $[x_1, y_1, x_2, y_2, \ldots, x_n, y_n]$.

Example 1:

```
Input: nums = [2,5,1,3,4,7], n = 3

Output: [2,3,5,4,1,7]

Explanation: Since x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 then the answer is
[2,3,5,4,1,7].
```

Example 2:

```
Input: nums = [1,2,3,4,4,3,2,1], n = 4

Output: [1,4,2,3,3,2,4,1]
```

Example 3:

```
Input: nums = [1,1,2,2], n = 2

Output: [1,2,1,2]
```

Constraints:

- `1 <= n <= 500`
- `nums.length == 2n`
- `1 <= nums[i] <= 10^3`

Problem-5

There is a programming language with only **four** operations and **one** variable `X`:

- `++X` and `X++` **increments** the value of the variable `X` by `1`.
- `--X` and `X--` **decrements** the value of the variable `X` by `1`.

Initially, the value of `X` is `0`.

Given an array of strings `operations` containing a list of operations, return *the **final** value of* `X` *after performing all the operations*.

Example 1:

```
Input: operations = ["--X","X++","X++"]

Output: 1

Explanation: The operations are performed as follows:

Initially, X = 0.

--X: X is decremented by 1, X =  0 - 1 = -1.

X++: X is incremented by 1, X = -1 + 1 =  0.

X++: X is incremented by 1, X =  0 + 1 =  1.
```

Example 2:

```
Input: operations = ["++X","++X","X++"]

Output: 3

Explanation: The operations are performed as follows:

Initially, X = 0.

++X: X is incremented by 1, X = 0 + 1 = 1.

++X: X is incremented by 1, X = 1 + 1 = 2.

X++: X is incremented by 1, X = 2 + 1 = 3.
```

Example 3:

```
Input: operations = ["X++","++X","--X","X--"]

Output: 0

Explanation: The operations are performed as follows:
```

```
Initially, X = 0.

X++: X is incremented by 1, X = 0 + 1 = 1.

++X: X is incremented by 1, X = 1 + 1 = 2.

--X: X is decremented by 1, X = 2 - 1 = 1.

X--: X is decremented by 1, X = 1 - 1 = 0.
```

Constraints:

- `1 <= operations.length <= 100`
- `operations[i]` will be either `"++X"`, `"X++"`, `"--X"`, or `"X--"`.