# Solution Approach

- The maximum number is 0 to 127. That means the first 7 bits of each byte are used to store the number. We can use the 8th bit for our compression process
- It's common that the series has repeated numbers, so we can compress those repeated numbers, thus compressing the entire array
- If a number is not repeated we do nothing for that number. If it is repeated, we count the number of repetition for that number. Then we store the count along with the number itself. The big question is, how can we distinguish a count index from a number index?
    - As the 8th bit of each number is unused. We can use that to denote the count index. If the 8th bit of a particular index value is set in the compressed array, it will indicate that it's a count variable and the next value is repeated that many times
    - As we can only use the first 7 bits to store the counter. The maximum repeated count will be 127. If for an input, the counter exceeds 127, we can store the current count and value in the array, and store the rest of the counts in the next indexes using the same approach
      **(NOTE: In the code we assumed the max repeated count would be less than 127 and didn't split the counter as explained here)**
- To decompress an array, first we count the total elements in the array. Then we set the write pointer to the end of the total element and read pointer to the end of the current elements
    - write_ptr = total_elements - 1
    - read_ptr = cur_elements - 1
    - scan and decompress the array from the end (so that it doesn't replace the important informations at the beginning of the array)
    - while decompressing, we check if the previous index of a number is a counter. If yes we add the number that many times, if not we just copy that number once
- This solution approach is good, if we have an array with repeated values. Then all those repeated values will be compressed into 2 indexes(counter and value)
- If the array has only unique values, this compression technique will be useless
- Also, if the repeated count is always 2, this compression technique won't be able to compress the total data size. Because for each repeated segment, ultimately we will create two indexes. So we will be ending up with the same array size after compression
- We are not resizing the array in the code after compression, so the actual allocated memory for the array will be unchanged and we don't have to allocate memory during decompression. If we decide to free the memory after compression, we need to allocate memory for the extra elements before decompression.

## Compressor-Input

An array of bytes and the data size:

```
uint8_t data_ptr[] = { 0x03, 0x74, 0x04, 0x04, 0x04, 0x35, 0x35, 0x64,
                       0x64, 0x64, 0x64, 0x00, 0x00, 0x00, 0x00, 0x00,
                       0x56, 0x45, 0x56, 0x56, 0x56, 0x09, 0x09, 0x09 }
uint8_t data_size = 24
```

## Compressor-Output

An array of compressed bytes and new data size:

```
{ 0x03, 0x74, 0x83, 0x04, 0x82, 0x35, 0x84, 0x64,
  0x85, 0x00, 0x56, 0x45, 0x83, 0x56, 0x83, 0x09 }
new_size = 16
```

Here, the bold entries are the compressed bytes. Each compressed segment has two corresponding indexes (counter and value). For example:

- data_ptr[2-4] is compressed to **0x83 0x04**
- first index represents the repetition count of the number and second index represents the number itself
- the 8th bit of each index represents if it's a counter index(set) or number index(reset) 0x83: **1**000 0011 (8th bit is set, so it's a counter index)
- That means the next index value(0x04) is repeated (0x83 **XOR** ~(1u<<7) ) = 3 times

## Decompressor-Input

An array of compressed bytes and data size:

```
uint8_t data_ptr[] = { 0x03, 0x74, 0x83, 0x04, 0x82, 0x35, 0x84, 0x64,
                     0x85, 0x00, 0x56, 0x45, 0x83, 0x56, 0x83, 0x09 }
uint8_t new_size = 16
```

## Decompressor-Output

An array of original bytes and the original data size:

```
{ 0x03, 0x74, 0x04, 0x04, 0x04, 0x35, 0x35, 0x64,
  0x64, 0x64, 0x64, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x56, 0x45, 0x56, 0x56, 0x56, 0x09, 0x09, 0x09 }
data_size = 24
```