

**BINARY**

# Binary

- In C++, **std::bitset** represents a fixed-size sequence of N bits

- Example:

```
std::bitset<8> bitmask;
```

```
bitmask.reset(1)
```

```
bitmask.set(1)
```

```
if (bitmask.test(1)) { // true
```

```
...
```

- **reset** : set bit to false
- **set** : set a specific bit
- **test** : check a specific bit
- **count** : return the number of bits set to true
- **flip** : toggle the value of the bits (if true, set to false and vice-versa)

# Problem 3 – Longest Substring Without Repeating Characters

Medium

 [leetcode.com/problems/longest-substring-without-repeating-characters](https://leetcode.com/problems/longest-substring-without-repeating-characters)

Given a string, the goal is to find the longest substring without repeating characters.

For example, given a string “abcbd”, it should return 4:

abcd since “b” is repeated

# Solution 3 – Longest Substring Without Repeating Characters

Medium

- Using bitset: create a bitmask with 128 bits where each bit represent a character
- Use sliding window using **left** and **right**
- Set the bit to true for each character found
- Check if it is repeated and reset the bit

```
int lengthOfLongestSubstring(string s) {
    std::bitset<128> bitmask;
    uint32_t left = 0;
    uint32_t maxLength = 0;

    for (uint32_t right = 0; right < s.length(); ++right) {
        uint32_t bitIndex = s[right];
        // if char is already in the bitmask, move left until we reset the bits
        while (bitmask.test(bitIndex)) {
            bitmask.reset(s[left]);
            ++left;
        }

        bitmask.set(bitIndex);
        maxLength = std::max(maxLength, right - left + 1);
    }
    return maxLength;
}
```

# Negabinary

- Non-standard positional numeral system that uses base of -2
- Allow representing negative numbers in binary
- Example:

$1101_{-2}$

$$(-2)^3 + (-2)^2 + 0 + (-2)^0 = -8 + 4 + 0 + 1 = -3$$

## Summing Negabinary

- Add as a regular binary number, but with **negative carry**

$$0 + 0 = 0$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \quad \text{with a negative carry 1}$$

$$\mathbf{1} + 1 = 0 \quad (\text{subtract})$$

$$\mathbf{1} + 0 = 1 \quad \text{with a positive carry 1}$$

# Negabinary

## Example 1

$$\begin{array}{r} 11\ 1 \\ 1011 \\ + 1110 \\ \hline = 110001 \end{array}$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ with negative carry } 1$$

$$1 + 1 = 0$$

$$1 + 1 = 0 \text{ with negative carry } 1$$

$$1 + 0 = 1 \text{ with positive carry } 1$$

$$1 + 0 = 1$$

red 1 = negative carry

green 1 = regular carry

## Example 2

$$\begin{array}{r} 1111 \\ 101010 \\ + 101100 \\ \hline = 11110110 \end{array}$$

## Reference

<https://math.stackexchange.com/questions/3251605/how-to-add-negabinary-numbers>

# Problem 1073 – Adding Two Negabinary Numbers

Medium

<https://leetcode.com/problems/adding-two-negabinary-numbers>

Given two numbers `arr1` and `arr2` in base -2, return the result of adding them together.

Each number is given in *array format*: as an array of 0s and 1s, from most significant bit to least significant bit. For example, `arr = [1,1,0,1]` represents the number  $(-2)^3 + (-2)^2 + (-2)^0 = -3$ . A number `arr` in array, format is also guaranteed to have no leading zeros: either `arr == [0]` or `arr[0] == 1`.

Return the result of adding `arr1` and `arr2` in the same format: as an array of 0s and 1s with no leading zeros.

## Example 1

Input: `arr1 = [1,1,1,1,1]`, `arr2 = [1,0,1]`

Output: `[1,0,0,0,0]`

Explanation: `arr1` represents 11, `arr2` represents 5, the output represents 16.

## Example 2

Input: `arr1 = [0]`, `arr2 = [0]`

Output: `[0]`

## Example 3

Input: `arr1 = [0]`, `arr2 = [1]`

Output: `[1]`

# Solution 1073 – Adding Two Negabinary Numbers

Medium

<https://leetcode.com/problems/adding-two-negabinary-numbers>