Задача B

1) Алгоритм

Для представления числа x используется вектор x_bits размером 101.000(с запасом) элементов (отвечающий за биты от 0 до 100000), где каждый элемент может быть равен 0 или 1. Переменная ones_count будет отслеживать количество единичных битов в числе.

Прибавление или вычитание к числу x значения 2^S эквивалентно установке или сбросу бита в индексе S в единицу или ноль. Однако нужно учесть ситуацию, когда понадобится обращаться к соседним индексам и менять уже их значения. Так по правилам двоичного сложения если мы прибавляем 1 в индекс, где уже установлена 1, нужно обнулить ячейку и перенести 1 в старший бит. По правилам двоичного вычитания реализуем работу операции – S.

Таким образом каждая операция над x происходит точно в соответствии с двоичной арифметикой. В случае сложения и вычитания учтены все возможные "переносы" и "заимствования" по двоичным правилам.

2) Сложность решения

В худшем случае, при выполнении операции может потребоваться пройти по всем битам числа x, начиная с бита S. Таким образом, сложность одной операции O(N).

Амортизированное время выполнения каждой операции прибавления или вычитания в среднем составляет O(1).

Память используется для хранения массива x_bits из 101000 элементов, что требует O(N) памяти.

3) Распечатка кода

```
#include <cstdint>
#include <iostream>
#include <vector>

const int SizeOfX = 101000;
std::vector<uint8_t> x_bits(SizeOfX, 0);

int main() {
  size_t quant_req = 0;
  char operation = '0';
  size_t bit_index = 0;
  int ones_count = 0;

  std::cin >> quant_req;

  for (size_t i = 0; i < quant_req; ++i) {
    std::cin >> operation >> bit_index;
```

```
    if (bit_index >= SizeOfX) {
      std::cout << ones_count << std::endl;
      continue;
    }

    if (operation == '+') {
      while (bit_index < SizeOfX) {
        if (x_bits[bit_index] == 0) {
          x_bits[bit_index] = 1;
          ++ones_count;
          break;
        }
        x_bits[bit_index] = 0;
        --ones_count;
        ++bit_index;
      }
    } else if (operation == '-') {
      while (bit_index < SizeOfX) {
        if (x_bits[bit_index] == 1) {
          x_bits[bit_index] = 0;
          --ones_count;
          break;
        }
        x_bits[bit_index] = 1;
        ++ones_count;
        ++bit_index;
      }
    }
    std::cout << ones_count << std::endl;
  }
  return 0;
}
```