№1) Алгоритм: нах число : vec.

Записываем ~~цифры~~ цифры числа в vector и сортируем его с помощью Counting Sort (получим sortedDigits)

- Что бы найти max:

Нужно reverse (sortedDigits) (что бы в порядке ~~увеличения~~ уменьш знач)

И index1 - ~~нах. эл~~ индекс первого эл-та в нач числе | sortedDigits[index1] != vec[index1]

Далее хотим найти index2 | vec[index2] == sortedDigits[index1]

после: swap (vec[index1], vec[index2]).

- Что бы найти min:

1) Если число не содержит нулей:

все то же, что и для max, только sortedDigits - отсортир в порядке возр.

2) Если число имеет нули:

X 0..0 y0..02 ... 0..00..0 ...

n_1 n_2 n_1 n_2

$n_1 \geq 0$, $n_2 \geq 0$

vec sortedDidit

① Сразу проверим совпад. ли vec[0] == ~~sortedDigits[zeroCount]~~ == sortedDigits[zeroCount]

т.к эта цифра min в числе. Если не совпад, то меняем ее.

② Если в числе: $XX_1 \dots X_m 0 \dots$, то хотим поставить 0 в vec[1]

③ Если в числе: $X0 \dots 0 X_1 X_2 \dots$ и $X_2 < X_1$, то меняем X_1 и X_2

Поиск индексов для 2) случая:
Теперь нам нужен бинпоиск indexDigit, чтобы корректно найти index2, ведь sortedDigits[indexDigit] - это наша цифра в числе, которую мы хотим поменять.

N - кол-во цифр в числе. k - кол-во тестовых чисел

Time compl: $O(N \cdot k)$

Т.к нет больш. чисел. а CountSort работает за $O(N+m)$, где $m = 16 \Rightarrow O(N)$ (для этой задачи)

Memory: $O(N)$