МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

АНАЛИЗ СЛОЖНОСТИ СОРТИРОВОК, НЕ ИСПОЛЬЗУЮЩИХ СРАВНЕНИЕ ЭЛЕМЕНТОВ

ОТЧЁТ

студента 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Карасева Вадима Дмитриевича

Проверено:
доцент, к. ф.-м. н. _____ М. И. Сафрончик

Саратов 2025

СОДЕРЖАНИЕ

1 Сортировки

1.1 Сортировка подсчетом

```
vector<int> arr(n);
int max_value = INT_MIN; // инициализируем минимально
    // возможное значение
int min_value = INT_MAX; // инициализируем максимально
    // возможное значение

// вводим элементы и находим min и max значения
for (int i = 0; i < n; i++) {
    cout << i + 1 << ": ";
    cin >> arr[i];
    max_value = max(max_value, arr[i]); // обновляем максимум
    min_value = min(min_value, arr[i]); // обновляем минимум
}

// создаем массив для подсчета, размером от min до max значения
vector<int> count_arr(max_value - min_value + 1, 0);

// подсчитываем количество каждого элемента
for (int i : arr) {
    count_arr[i - min_value]++; // увеличиваем счетчик для
            // текущего числа
}

// восстанавливаем отсортированный массив из счетчиков
int j = 0; // индекс для заполнения исходного массива
for (int i = 0; i < count_arr.size(); i++) {
    while (count_arr[i] > 0) {
        arr[j++] = i + min_value; // записываем число
                    // обратно в массив
        count_arr[i]--; // уменьшаем счетчик
    }
}
```

1.2 Поразрядная сортировка (LSD)

```
// функция для определения максимального кол-ва
// разрядов в числах вектора
int k(vector<int>& vec) {
    int retK = 0; // переменная для хранения максимального
        // кол-ва разрядов

    // перебираем все числа в векторе
    for (int n : vec) {
        int cnt; // счетчик разрядов текущего числа

        // особый случай для числа 0 (у него 1 разряд)
        if (n == 0){
            cnt = 1;
        } else {
            cnt = 0;
        }

        // считаем кол-во разрядов в числе
        while (n != 0) {
            n /= 10; // удаляем младший разряд
            cnt++;
        }

        // обновляем максимальное кол-во разрядов,
                // если текущее число имеет больше разрядов
        if (cnt > retK) {
            retK = cnt;
        }
    }
    return retK;
}

// функция поразрядной сортировки (LSD - Least Significant Digit)
```

```
void radixSort(std::vector<int>& vec) {
    int dig = k(vec); // получаем максимальное кол-во разрядов
    vector<vector<int> > p(10); // вектор векторов для цифр от 0 до 9

    // проходим по всем разрядам, начиная с младшего
    for (int i = 0; i < dig; i++) {
        // распределяем числа по корзинам в
                // соответствии с текущей цифрой
        for (int j = 0; j < vec.size(); j++) {
            // вычисляем текущую цифру (i-й разряд числа)
            int dig_cnt = (abs(vec[j]) / static_cast<int>(pow(10,i)) % 1
            p[dig_cnt].push_back(vec[j]); // помещаем число в
                        // соответствующую корзину
        }

        // собираем числа из корзин обратно в исходный вектор
        int ind = 0; // индекс для вставки в исходный вектор
        for (int i = 0; i < 10; i++) {
            for (int n : p[i]) {
                vec[ind++] = n; // последовательно берем
                                // числа из всех корзин
            }
            p[i].clear(); // очищаем корзину для следующего разряда
        }
    }
}
```

2 Анализ сложности сортировок

2.1 Сортировка подсчетом

Функция нахождения `max_value` и `min_value` выполняется за время $O(n)$, где n — размер исходного массива, так как в одном цикле происходит и ввод, и поиск минимума и максимума.

Вычисление размера счётного массива $(\texttt{max\_value} - \texttt{min\_value} + 1)$ и передача его в `vector<int> count_arr(...)` происходит за время $O(1)$.

Инициализация массива `count_arr` размером $k = \texttt{max\_value} - \texttt{min\_value} + 1$ выполняется за время $O(k)$, где k — диапазон возможных значений элементов массива.

Массив `arr` уже был создан ранее, поэтому дополнительных затрат на его инициализацию не происходит. Однако логически восстановление отсортированного массива эквивалентно созданию нового, что учитывается как $O(n)$.

Первый цикл, отвечающий за подсчёт частот элементов в `count_arr`, выполняется за время $O(n)$, так как каждый элемент массива `arr` обрабатывается один раз.

В данной реализации **отсутствует этап построения префиксной суммы**, но если бы он был, его сложность составила бы $O(k)$. Однако, в текущем коде сразу восстанавливается отсортированный массив из счётчиков, без вычисления префиксной суммы.

Второй цикл (восстановление массива) проходит по `count_arr`, и каждый элемент записывается столько раз, сколько он встречался. Общее количество операций записи в `arr` равно n. Таким образом, цикл работает за $O(n + k)$: k — длина `count_arr`, n — количество итераций в общей сумме `while`-циклов.

Итог:

- n — количество элементов в исходном массиве.
- k — диапазон значений $(\texttt{max\_value} - \texttt{min\_value} + 1)$.
- **Общая сложность:** $O(n + k)$.

2.2 Поразрядная сортировка (LSD)

Функция `k(vec)`, определяющая максимальное количество разрядов в числах вектора, выполняется за время $O(n \cdot d)$, где n — размер исходного массива, а d — максимальное количество разрядов в числе (в худшем случае $d = \log_{10}(\text{max})$).

Вычисление параметров, включая создание массива корзин (вектор из 10 векторов), происходит за время $O(1)$.

Каждая итерация внешнего цикла, проходящего по всем разрядам (от младшего к старшему), выполняется d раз, где d — количество разрядов.

Во внутреннем цикле каждое число помещается в соответствующую корзину, что осуществляется за $O(n)$ на каждой итерации.

Сборка чисел из корзин обратно в исходный массив также осуществляется за $O(n)$ на каждой итерации.

Очистка всех корзин за одну итерацию выполняется за $O(n)$, так как в сумме через корзины проходит n элементов.

Таким образом, каждый проход по одному разряду выполняется за $O(n)$, а таких проходов d, следовательно, общая временная сложность составляет (согласно правилу суммы):

$$O(n \cdot d) + O(1) + O(n \cdot d) + O(n \cdot d) + O(n \cdot d) = O(n \cdot d)$$

Итог:

— n — количество элементов в массиве.

— d — количество разрядов (в десятичной системе — $d = \lfloor \log_{10}(\max) \rfloor + 1$).

— **Общая временная сложность:** $O(n \cdot d)$.