A4. Значительные инверсии

Демченко Георгий Павлович , БПИ-235

1. DaC-алгоритм CInV (CountInversions)

CountInversions.cpp

```
int32_t MergeAndCountInversions(std::vector<int32_t>& array, int32_t left,
int32_t middle, int32_t right) {
    int32_t left_size = middle - left + 1;
    int32_t right_size = right - middle;

    std::vector<int32_t> left_part;
    left_part.reserve(left_size);
    std::vector<int32_t> right_part;
    right_part.reserve(right_size);

    for (int32_t i = 0; i < left_size; ++i) {
        left_part.emplace_back(array[left + i]);
    }
    for (int32_t i = 0; i < right_size; ++i) {
        right_part.emplace_back(array[middle + 1 + i]);
    }

    int32_t inversionsCount = 0;
    int32_t arrowFromRightCount = 0;

    int32_t left_idx = 0;
    int32_t right_idx = 0;
    int32_t main_idx = left;

    while (left_idx < left_size && right_idx < right_size) {
        if (left_part[left_idx] <= right_part[right_idx]) {
            array[main_idx] = left_part[left_idx];
            left_idx++;
            inversionsCount += arrowFromRightCount;
        } else {
            arrowFromRightCount++;
            array[main_idx] = right_part[right_idx];
            right_idx++;
        }
        main_idx++;
    }

    while (left_idx < left_size) {
        inversionsCount += arrowFromRightCount;
        array[main_idx] = left_part[left_idx];
        left_idx++;
        main_idx++;
    }
```

```
    while (right_idx < right_size) {
        array[main_idx] = right_part[right_idx];
        right_idx++;
        main_idx++;
    }

    return inversionsCount;
}

int32_t CountInversions(std::vector<int32_t>& array, int32_t left, int32_t
right) {
    if (left == right) {
        return 0;
    }
    int32_t middle = left + (right - left)/ 2;

    // DIVIDE
    int32_t leftNumOfInversions = CountInversions(array, left, middle);
    int32_t rightNumOfInversions = CountInversions(array, middle + 1,
right);

    // CONQUER & COMBINE
    int32_t numOfConnectiveInversions = MergeAndCountInversions(array,
left, middle,  right);

    return leftNumOfInversions + rightNumOfInversions +
numOfConnectiveInversions;
}
```

Понял поставленную задачу, как подсчет кол-ва перестановок элементов до состояние отсортированности, а не поиск количества инверсий в массиве.

Суть алгоритма - модифицированный MERGE SORT

1.1 Шаг Divide

- **Разбиваем входной массив на 2 подмассива по (примерно - ± 1) половине элементов в каждом.**

Шаг CONQUER & COMBINE

- **Рекурсивно выполняем подсчёт кол-ва необходимых перестановок между полученными подмассивами во время их сортировки (модифицированный MERGE из MERGE SORT)**
- **Объединяем 2 подмассива в один**

1.2 $T(n) = O(n \cdot \log_2(n))$

- **MergeAndCountInversions**

$T_{maci}(n) = O(n)$ - линейное слияние и подсчёт кол-ва непобходимых перестановок.

- **CountInversions**

$$\Rightarrow T(n) = 2 \cdot T(\frac{n}{2}) + T_{maci}(n) = 2 \cdot T(\frac{n}{2}) + O(n)$$

Согласно мастер-теореме

$$\log_b(a) = \log_2(2) = 1 = k$$

$\Rightarrow T(n) = O(n \cdot \log_2(n))$ - **Соответсвует требуемой ассимтотической верхней границе временной сложности**

1.3 Минимальное количество необходимых инверсий.

Данный алгоритм не возвращает минимальное количество необходимых инвсерий до состояния отсортированности

Так как подсчет количества необходимых инверсий ведется рекурсивно между двумя последовательными подмамассивами изначального массива во время их слияния на каждом уровне рекурсии, то мы не можем гарантировать, что некоторая перестановка с элементом, не входящим в эти подмассивы, не будет оптимальней нашей (в плане кол-ва затраченных перестановок в дальнейшем). Также, при поднятии по уровням рекурсии,во время слияния мы не можем оставить перестановки, которые были необходимы для полученния текущих отсортированных подмассивов и обязаны их учитывать.

В связи с этими факторами мы не всегда будем получать минимальное количество необходимых перестановок.

2. DaC-алгоритм CSINV (CountSignificantInversions)

Понял поставленную задачу, как подсчет кол-ва значительных инверсий (всех таких пар элементов), а не кол-ва перестановок как в предыдущем случае

- **Изменена логика подсчета инверсий в MergeAndCountSignificantInversions**

CountSignificantInversions.cpp

```
int32_t MergeAndCountSignificantInversions(std::vector<int32_t> &array,
int32_t left, int32_t middle, int32_t right) {
    int32_t left_size = middle - left + 1;
    int32_t right_size = right - middle;

    std::vector<int32_t> left_part;
    left_part.reserve(left_size);
    std::vector<int32_t> right_part;
    right_part.reserve(right_size);

    for (int32_t i = 0; i < left_size; ++i) {
        left_part.emplace_back(array[left + i]);
    }
    for (int32_t i = 0; i < right_size; ++i) {
        right_part.emplace_back(array[middle + 1 + i]);
```

```
    }

    int32_t inversionsCount = 0;

    int32_t left_idx = 0;
    int32_t right_idx = 0;
    int32_t main_idx = left;

    while (left_idx < left_size && right_idx < right_size) {
        if (left_part[left_idx] <= right_part[right_idx]) {
            array[main_idx] = left_part[left_idx];
            left_idx++;
        } else {
            if (left_part[left_idx] > 2 * right_part[right_idx]) {
                inversionsCount += left_size - left_idx;
            } else if (left_idx + 1 < left_size && left_part[left_idx + 1]
> 2 * right_part[right_idx]) {
                inversionsCount += left_size - (left_idx + 1);
            }
            array[main_idx] = right_part[right_idx];
            right_idx++;
        }
        main_idx++;
    }

    while (left_idx < left_size) {
        array[main_idx] = left_part[left_idx];
        left_idx++;
        main_idx++;
    }

    while (right_idx < right_size) {
        array[main_idx] = right_part[right_idx];
        right_idx++;
        main_idx++;
    }

    return inversionsCount;
}

int32_t CountSignificantInversions(std::vector<int32_t> &array, int32_t
left, int32_t right) {
    if (left == right) {
        return 0;
    }
    int32_t middle = left + (right - left) / 2;

    int32_t leftNumOfInversions = CountSignificantInversions(array, left,
middle);
    int32_t rightNumOfInversions = CountSignificantInversions(array,
middle + 1, right);

    int32_t numOfConnectiveInversions =
MergeAndCountSignificantInversions(array, left, middle, right);
```

```
    return leftNumOfInversions + rightNumOfInversions +
numOfConnectiveInversions;
}
```

2.1 $T_{new}(n) = O(n \cdot \log_2(n))$

- **MergeAndCountSignificantInversions**

$T_{macsi}(n) = O(n)$ - линейное слияние и подсчёт кол-ва значительных инверсий.

- **CountSignificantInversions**

$\Rightarrow T(n) = 2 \cdot T(\frac{n}{2}) + T_{macsi}(n) = 2 \cdot T(\frac{n}{2}) + O(n)$

Согласно мастер-теореме

$\log_b(a) = \log_2(2) = 1 = k$

$\Rightarrow T_{new}(n) = T(n) = O(n \cdot \log_2(n))$ - **Соответсвует требуемой ассимтотической верхней границе временной сложности**