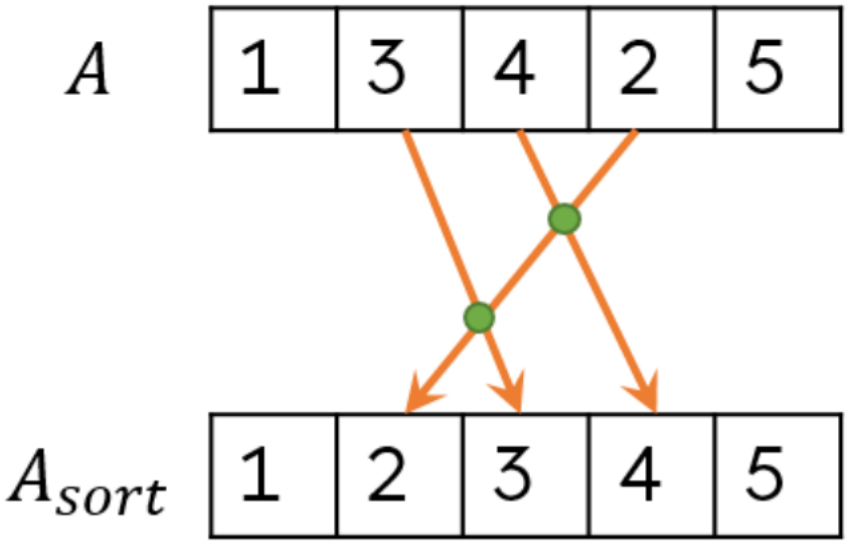
Задача A4. Значительные инверсии

Рассмотрим механизм подсчета так называемой «степени упорядоченности» некоторого целочисленного массива $A = [a_1, a_2, a_3, \ldots, a_n]$, заполненного уникальными значениями.

Элементы a_i и a_j массива A назовем инвертированными, если $i < j$, но $a_i > a_j$. Например, в массиве $A = [1, 3, 4, 2, 5]$ достаточно выполнить две инверсии, а именно $3 \leftrightarrow 2$ и $4 \leftrightarrow 2$, чтобы получить отсортированный массив $A' = [1, 2, 3, 4, 5]$.



1. Разработайте DaC-алгоритм CINV, временная сложность которого должна соответствовать $O(nlogn)$, для подсчета степени упорядоченности массива путем вычисления количества необходимых перестановок. Описание алгоритма представьте в любом удобном формате. Опишите суть шагов *DIVIDE*, *CONQUER* и *COMBINE*, а также представьте рекуррентное соотношение для $T(n)$ и обоснуйте соответствие требуемой асимптотической верхней границе временной сложности. Проанализируйте, возвращает ли разработанный вами алгоритм CINV минимальное количество необходимых инверсий.

```python
def CINV(arr):
    def mergeSort(arr):
        n = len(arr)
        if n <= 1:
            return arr, 0
        else:
            mid = n // 2
            left, invLeft = mergeSort(arr[:mid])
            right, invRight = mergeSort(arr[mid:])
            merged, invSplit = merge(left, right)
            return merged, invLeft + invRight +
invSplit
    def merge(left, right):
        merged = []
        invCount = 0
        i = 0
        j = 0
        lenLeft = len(left)
        while i < lenLeft and j < len(right):
            if left[i] <= right[j]:
                merged.append(left[i])
                i += 1
            else:
                merged.append(right[j])
                invCount += lenLeft - i
                j += 1
        merged.extend(left[i:])
        merged.extend(right[j:])
        return merged, invCount

    _, totalInversions = mergeSort(arr)
    return totalInversions
```

DIVIDE

Разбиваем массив A на две приблизительно равные части. Разделение проблемы на более мелкие подзадачи, которые легче решить рекурсивно.

CONQUER

Рекурсивно сортируем каждую половину массива и подсчитываем количество инверсий внутри каждой половины. Решая проблему для меньших массивов, мы можем построить решение для более крупного массива.

COMBINE

Объединяем две отсортированные половины в один отсортированный массив, одновременно подсчитывая количество *сквозных* инверсий (инверсий, где элемент из левой половины больше элемента из правой половины). Комбинирование решений подзадач для решения исходной задачи.

Рекуррентное соотношение $T(n)$

Временная сложность алгоритма описывается рекуррентным соотношением:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)$$

При каждом рекурсивном вызове массив разделяется на две части, что дает $2 \cdot T\left(\frac{n}{2}\right)$. Шаг слияния занимает $O(n)$ времени.

Анализ временной сложности

Применим мастер-теорему для решения данного рекуррентного соотношения.

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n),$$

где $a = 2$, $b = 2$, $f(n) = O(n)$
$\log_b a = \log_2 2 = 1$
Сравниваем $f(n)$ с $n^{\log_b a}$
$n^{\log_b a} = n^1 = n$
Значит, $f(n) = O(n) = \Theta(n^{\log_b a})$
Это второй случай мастер-теормы ($f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$ с $k = 0$).
$\Longrightarrow T(n) = \Theta\left(n^{\log_b a} \cdot \log^{k+1} n\right) = \Theta\left(n \cdot \log n\right)$

Алгоритм CINV имеет временную сложность $O(n \log n)$.

Минимальное количество необходимых инверсий

Алгоритм точно подсчитывает количество инверсий, что соответствует минимальному количеству перестановок, необходимых для сортиров-

ки массива, если можно менять местами любые два элемента.

2. Элементы a_i и a_j массива A назовем *значительно инвертированными*, если $i < j$, но $a_i > 2 \cdot a_j$. Какие изменения и доработки необходимо внести в алгоритм CINV, разработанный на предыдущем шаге, чтобы в качестве степени упорядоченности велся подсчет количества пар значительно инвертированных элементов? Например, в массиве $A = [1, 3, 4, 2, 5]$ нет значительно переставленных элементов, а в массиве $A = [5, 3, 2, 4, 1]$ всего 4 пары значительно переставленных элементов: $5 \leftrightarrow 1$, $5 \leftrightarrow 2$, $4 \leftrightarrow 1$ и $3 \leftrightarrow 1$. Асимптотическая верхняя граница временной сложности измененного алгоритма должна остаться неизменной.

Основное изменение заключается в том, что во время шага слияния мы должны подсчитывать количество элементов в правом подмассиве, которые удовлетворяют условию $a_i > 2 \cdot a_j$ для каждого элемента a_i из левого подмассива.

CONQUER

Рекурсивно применяем алгоритм к каждой половине массива.

COMBINE

В процессе слияния двух отсортированных половин мы модифицируем процесс подсчета инверсий.

Для каждого элемента a_i из левого подмассива подсчитываем количество элементов a_j в правом подмассиве, для которых выполняется $a_i > 2 \cdot a_j$.

Объяснение изменений в коде

Функция mergeSort

После рекурсивной сортировки левой и правой половин мы подсчитываем значительно инвертированные пары перед слиянием массивов.

Подсчет значительно инвертированных пар:

```
1 j = 0
2 for i in range(len(left)):
3     while j < len(right) and left[i] > 2 *
4 right[j]j += 1
5     invCount += j
```

Итерируемся по каждому элементу a_i из левого подмассива.

Для каждого a_i увеличиваем j, пока $left[i] > 2 \cdot right[j]$.

После завершения внутреннего цикла j содержит количество элементов в правом подмассиве, которые удовлетворяют условию для текущего a_i.

Суммируем j в общий счетчик инверсий invCount.

Функция merge

Сливает два отсортированных подмассива в один отсортированный массив.

Процесс слияния остается таким же, как в стандартном алгоритме сортировки слиянием.

Обоснование временной сложности

Шаги разделения и объединения:

Каждый вызов mergeSort разделяет массив на две половины, что создает $\log n$ уровней рекурсии.

Подсчет значительно инвертированных пар:

На каждом уровне рекурсии, перед слиянием, мы выполняем подсчет значительно инвертированных пар.

Внешний цикл по i выполняется $O(n)$ раз (по длине левого подмассива).

Внутренний цикл по j в сумме по всем вызовам работает за $O(n)$ времени, поскольку j не уменьшается.

Общее время на подсчет инверсий на каждом уровне: $O(n)$.

Слияние массивов:

Операция слияния двух отсортированных массивов выполняется за $O(n)$.

Общая временная сложность:

Имеем $\log n$ уровней рекурсии.

На каждом уровне затрачивается $O(n)$ времени.

Итого: $O(n \log n)$.