Теоретическое решение задачи B.

Алгоритм решения и доказательство его правильности.

Данная задача ставит перед собой довольно легкие задания: хранение ряда из школьников с уже определенным ростом для каждого, функция переворота определенного отрезка ряда и функция подсчета суммарного роста на определенном отрезке. Для этого можно использовать самую простую структуру данных – массив, с функциями подсчета суммы с временем O($\sqrt{n}$) при использовании Sqrt-декомпозиции и переворотом отрезка массива. Переворот такой структуры данных как массив по времени займет не менее чем O(n). И следовательно данный вариант даст асимптотику O(nm), при которой при экстремальных значениях n и m в размере 200 000, количество операций будет $>10^{10}$. Данное решение явно не вложится в отведенное задаче временное ограничение в размере 5с.

В связи с этим, требуется искать другую структуру данных, которая будет производить данные две операции с меньшей асимптотикой. В качестве данной структуры было выбрано декартово дерево. Декартово дерево – структура, объединяющая в себе двоичное дерево и пирамиду. В роли значений для пирамиды (двоичной кучи), которые в нашем дереве будут выполнять роль приоритетов используются рандомные значения, которые дают возможность однозначно задать дерево, и при этом задать его приблизительно с высотой log(n). Ключ же, по которому строится двоичное дерево, будет задан неявно и будет иметь значение порядкового номера студента в ряду. Явно же мы хранить ключ не будем, потому что при изменении порядка в массиве, придётся пересчитывать ключ для всех вершин. Для определения ключа будем хранить переменную count которая является количеством узлов в данном поддереве. Также одна из задач была подсчет суммы на отрезке. Для этого к нашему дереву добавим дерево отрезков, для каждого узла приписано значение: суммарный рост всех учеников поддерева.

Нахождение суммы происходит так: дерево обрезается функцией Split по левому граничному значению, а затем правое результирующее дерево функции обрезаем по правому граничному значению. В итоге получаем нужный нам отрезок, в вершине которого будет храниться значение суммарного роста. В конце деревья склеиваются обратно функциями Merge.

Переворот отрезка происходит так же: дерево обрезается с двух сторон, переменной reverse корня присваивается значение true, после чего при использовании функции Merge вызывается функция Push, которая при значении reverse = true, меняет левое и правое поддерево местами. Так происходит при каждом рекурсивном вызове функции Merge, таким образом отрезок переворачивается.

Временная сложность.

Наш алгоритм включает такие функции:

1. **Split.** Рекурсивно вызывается одна операция Split для дерева меньшей высоты (хотя бы на 1) и делается еще O(1) операций. В итоге получаем оценку O(h), где h – высота дерева, которая в среднем случае равняется O(logn), и следовательно время выполнения функции равняется O(logn).
2. **Merge.** Рекурсивно вызывается одна операция Merge для дерева меньшей высоты (хотя бы на 1) и делается еще O(1) операций. В итоге получаем оценку O(h), где h – высота дерева, которая в среднем случае равняется O(logn), и следовательно время выполнения функции равняется O(logn).
3. **Insert.** Вызывает методы: Split, Merge, Merge, каждый из которых имеет временную оценку O(logn), что в конечном итоге и даст временную сложность O(logn)
4. **Push.** Все операции - условные или присваивания - выполняются за O(1), как и вся функция.

И конечные функции, которые требуются в условии задачи:

1. **Reverse.** Вызывает методы: Split, Split, Merge, Merge, каждый из которых имеет временную оценку O(logn), что в конечном итоге и даст временную сложность O(logn)
2. **SumFromAToB** Вызывает методы: Split, Split, Merge, Merge, каждый из которых имеет временную оценку O(logn), что в конечном итоге и даст временную сложность O(logn)

Видим, что все функции реализованы с одной асимптотикой O(logn).

Непосредственно в нашей задаче задается ряд учеников, а для того что бы мы могли делать операции с этим рядом, нам надо заполнить наше декартово дерево. Делаем мы это с помощью вызова n-го количества метода Insert с оценкой O(logn), что дает асимптотику O(nlogn). А также требуется, произвести m запросов на переворот определенного участка ряда из учеников или получить суммарный рост определенного участка ряда учеников, это выполняется двумя методами: Reverse и SumFromAToBc оценками O(logn), что дает асимптотику O(mlogn).

Общая асимптотика решения задачи O(nlogn+mlogn) = O((m+n)logn).

Затраты памяти.

Для реализации описанного выше алгоритма, требуется постоянно хранить только n узлов декартового дерева каждый из которых хранит в себе количество узлов в поддереве, приоритет, рост одного ученика, сумму роста учеников поддерева, ссылку на левое и правое поддерево и булевую переменную, отвечающую за переворот отрезка

Таким образом, итоговые затраты памяти —O(7n) = O(n).