

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 7
«2136. Earliest Possible Day of Full Bloom - LeetCode»

Выполнил работу

Смирнов Александр

Академическая группа №J3111

Принято

Ментор Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель работы: попрактиковаться в решении задач на жадные алгоритмы, решить одну из hard задач на LeetCode.

Задачи:

1. Выбрать задачу для решения;
2. Разобраться в условии, придумать применение метода жадного алгоритма;
3. Реализовать алгоритм;
4. Провести анализ полученных результатов и оформить отчёт.

2. Теоретическая подготовка

В решении я использовал векторы, кортежи и int-ы.

Использовал метод жадного алгоритма, на который не многозначительно намекает тематика лабы и тэг задачи.

3. Реализация

Условие задачи следующее:

У вас есть n цветочных семян. Каждое семя должно быть сначала посажено, прежде чем оно сможет начать расти, а затем зацвести. Посадка семени требует времени, как и рост семени. Вам предоставляются два целочисленных массива `plantTime` и `growTime`, длиной n каждый:

Время посадки $[i]$ - это количество полных дней, необходимых для посадки i -го семени. Каждый день вы можете сажать ровно одно семечко.

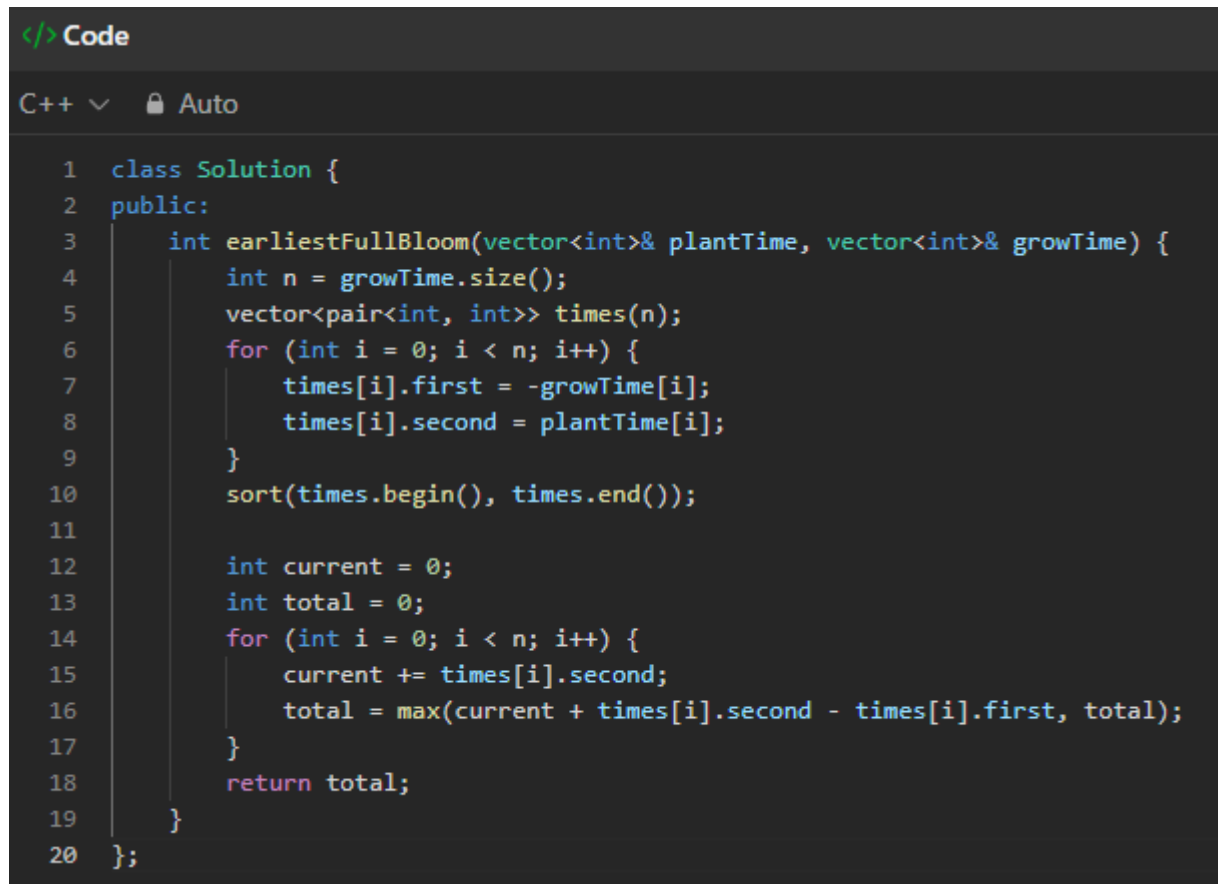
Время прорастания $[i]$ - это количество полных дней, которое требуется i -му семени для прорастания после полной посадки.

Начиная с 0-го дня, вы можете высаживать семена в любом порядке.

Верните наиболее ранний день, когда все семена распустятся

Для её решения я использовал библиотеки `iostream`, `time`, `algorithm` и `vector`.

Решение задачи представлено на рис. 1:



```
</> Code
C++ v Auto

1  class Solution {
2  public:
3      int earliestFullBloom(vector<int>& plantTime, vector<int>& growTime) {
4          int n = growTime.size();
5          vector<pair<int, int>> times(n);
6          for (int i = 0; i < n; i++) {
7              times[i].first = -growTime[i];
8              times[i].second = plantTime[i];
9          }
10         sort(times.begin(), times.end());
11
12         int current = 0;
13         int total = 0;
14         for (int i = 0; i < n; i++) {
15             current += times[i].second;
16             total = max(current + times[i].second - times[i].first, total);
17         }
18         return total;
19     }
20 };
```

Рисунок 1. Код решения задачи

Концептуально решение следующее:

1. Сортируем семена по времени их прорастания в порядке убывания. При сортировке нас не интересует время посадки.
2. Просматривая отсортированные семена, следим за текущим временем наибольшего полного цветения и текущим временем окончания посева.

Обновляем их следующим образом:

$total = \max(total, current + seed[i] + seed[i])$ (Время выращивания)

$current += seed[i]$ (Время посадки)

После просмотра всех семян, $total$ - это ответ.

4. Экспериментальная часть

Оценочная сложность решения – $n \cdot \log(n)$, так как самым затратным по времени действием в алгоритме является сортировка массива длины n .

Такую же сложность дает решению и ЛитКод, при этом скорость довольно высокая относительно других решений (представлено на рис.2)

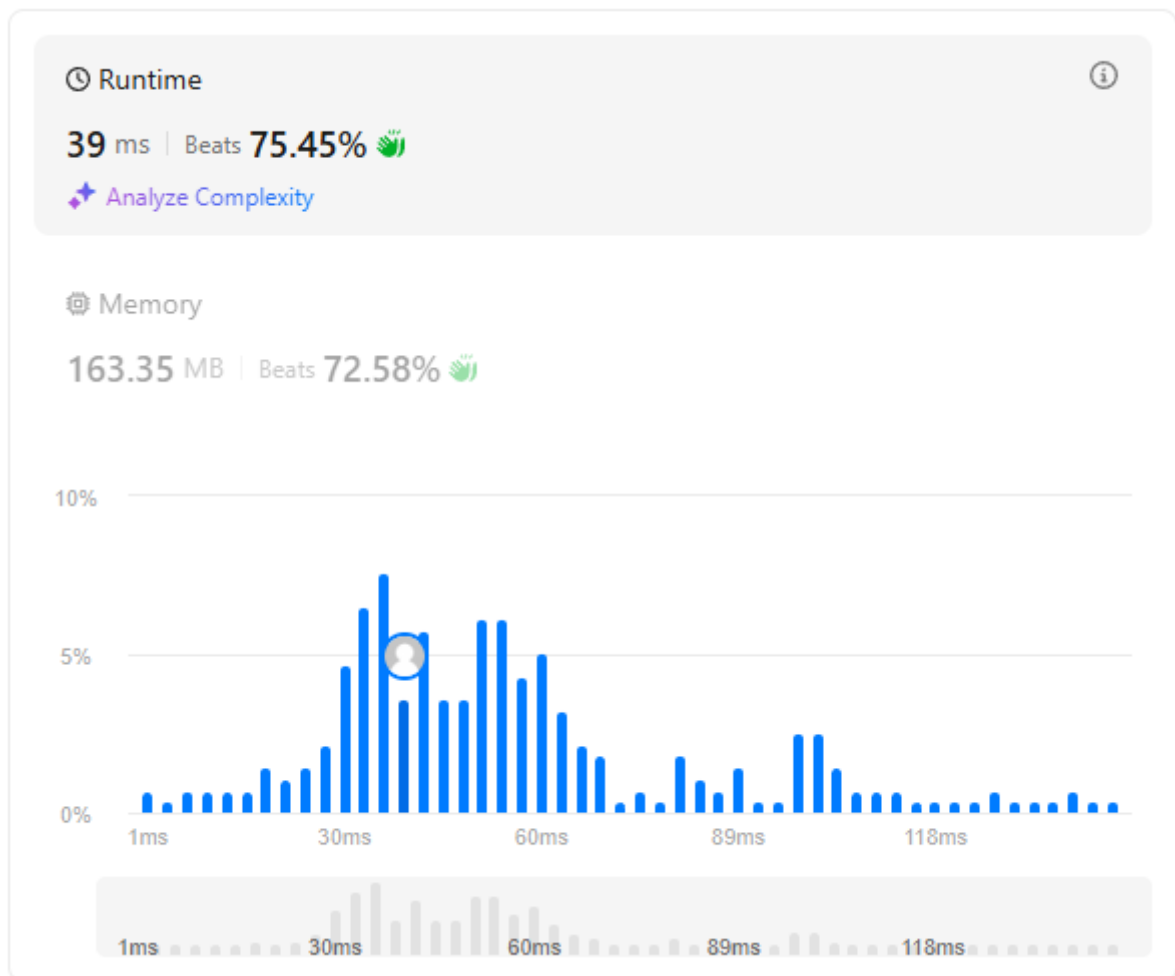


Рисунок 2. Скорость решения в сравнении с другими решениями этой задачи

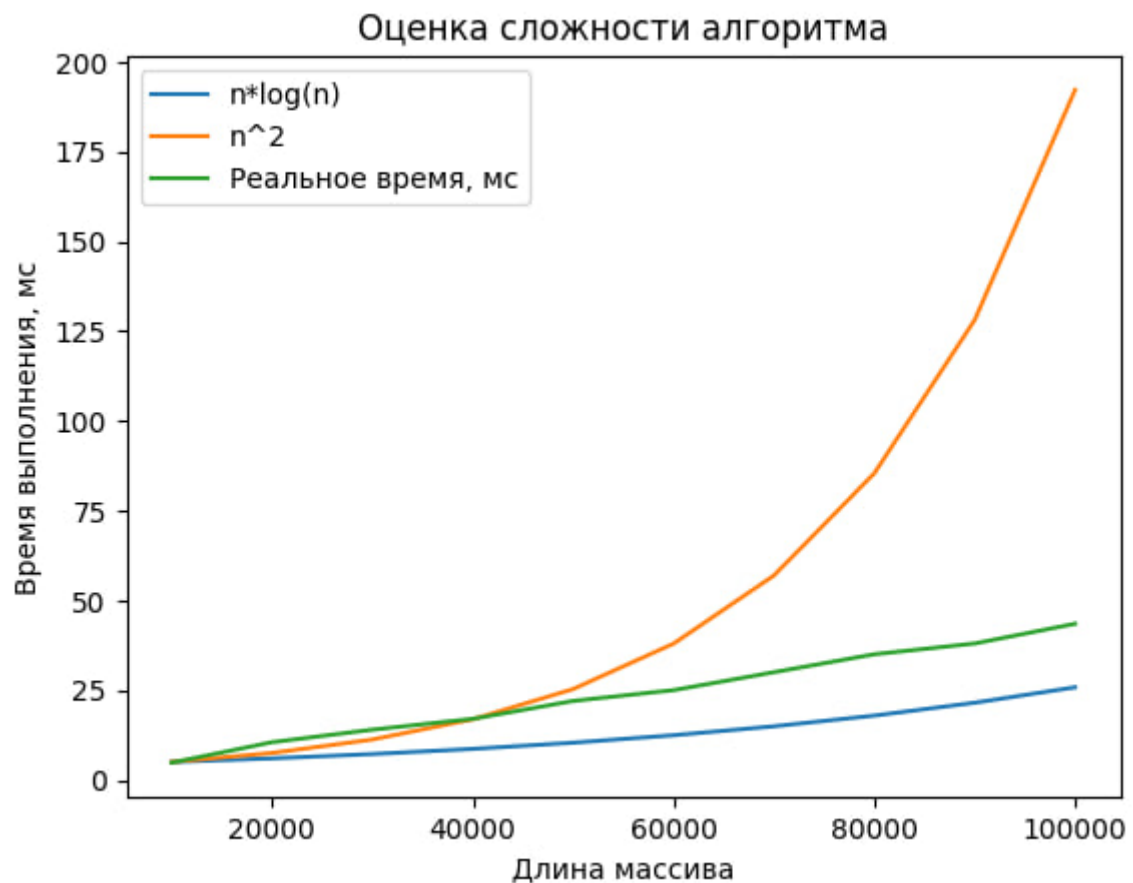
Расход памяти: 12 байт уходит на хранение трёх целочисленных переменных, и $24 + n \cdot 8$ байт (вектор пар интов длины n). На вектор входных значений передается ссылка, так что функция не тратит на его хранение дополнительную память.

Проверим теоретическую сложность алгоритма, сравнив с $O(N^2)$ и $O(N \cdot \log N)$ (таблица 1)

N	10000.0	20000.0	30000.00	40000.000	50000.0000	60000.00000	70000.000000	80000.000000	90000.000000	100000.000000
$n \cdot \log(n)$	5.0	6.0	7.20	8.640	10.3680	12.44160	14.929920	17.915904	21.499085	25.798902
n^2	5.0	7.5	11.25	16.875	25.3125	37.96875	56.953125	85.429688	128.144531	192.216797
Реальное время, мс	4.8	10.5	14.00	17.000	22.0000	25.00000	30.000000	35.000000	38.000000	43.500000

Таблица 1. Результаты замеров реального времени исполнения, $O(N^2)$ и $O(N^3)$

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №3.



Изображение №3 - График работы алгоритма

Таким образом, видим, что реальная сложность алгоритма действительно очень близка к $N \cdot \log(N)$, значит мы верно оценили сложность.

5. Заключение

В ходе выполнения работы мною был реализован алгоритм для решения задачи 2136. Earliest Possible Day of Full Bloom на LeetCode. Цель работы была достигнута путём тестирования на массивах с различным количеством элементов. Полученные результаты также совпадают с теоретическими оценками сложности алгоритма.

В качестве дальнейших исследований можно рассмотреть варианты других решений от пользователей данного сайта, перенять от них идеи по оптимизации алгоритма с точки зрения используемой памяти.