Анализ на задача Нейтън

В задачата са поставени няколко проблема, с които един програмист може да се наложи да се бори в реалния живот. Обикновено за решаването им се използват сложни библиотеки с множество функции, но в конкретния случай има и по-прости алтернативи.

Начало на ден
Определянето на началото на деня по даден момент от него става лесно, когато се сетим, че всеки един има фиксиран брой секунди - а именно, 24 часа * 60 минути * 60 секунди = **86 400** секунди. Тъй като Unix timestamp 0 се е случил в 00:00:00 (т.е. началото на деня), значи следващият ден е започнал в timestamp 86 400, по-следващият - в 172 800 (= 2 * 86 400) и т.н. Следователно за да определим началото на деня, в който е timestamp X, е необходимо да намерим най-голямото число dayTimestamp, такова че:

- **dayTimestamp <= X**
- **dayTimestamp = days * 86 000** за някое цяло число **days**.

След това решението на тази част от задачата е лесно. Можем или да сметнем **days = X / 86 400** (закръглено надолу), и от него да сметнем **dayTimestamp**; или директно да сметнем **dayTimestamp = X - X % 86 400** (т.е. вадим остатъка при деление с 86 400 и това, което остава, със сигурност се дели на 86 400).
За по-запознатите с астрономическите феномени, едно потенциално притеснение относно този подход би било наличието на така наречените "високосни" секунди. Това са секунди, които се добавят или премахват, когато въртенето на земята се забърза или забави спрямо SI дефиницията за секунда. За щастие, Unix timestamp е дефиниран така, че всеки ден е точно 86 400 секунди и съответно това не е проблем за нашето решение. (В условието не се споменава нищо за високосни секунди поради предположението, че повечето участници няма да знаят за тях и това само би ги объркало допълнително.)

Начало на седмица
За намирането на началото на седмицата можем да започнем с подобна идея. Всяка седмица продължава 7 * 86 400 = 604 800 секунди. За съжаление, 1 януари 1970 е четвъртък, и ако използваме същия подход като с дните, ще получим последния четвъртък преди timestamp X, а не последният понеделник.
Една възможна идея, която може да ни дойде на ум, след като открием проблема с четвъртъка, е просто да извадим три дни, за да получим съответния понеделник. Това звучи добре, но е грешно за половината дни от седмицата. Нека разгледаме деня 6 януари 1970 година - вторник. По описания до момента алгоритъм първо ще получим 1 януари 1970, четвъртък. След като извадим три дни, ще получим 29 декември 1969, понеделник - а правилният отговор е 5 януари. Тази грешка се получава, тъй като алгоритъмът разделя седмиците от четвъртък до сряда, вместо от понеделник до неделя.
Нека разгледаме една алтернативна система за измерване на време, започваща от 29 декември 1969 година - да я наречем MondaySystem timestamp - и нека за даден момент T да означаваме с MondaySystem(T) timestamp-а на този момент в новата ни система. Т.е. MondaySystem(T) е броят секунди, изминал от 29 декември 1969 година до T. Аналогично, нека с Unix(T) да означим Unix timestamp-а за момента T. Можем да забележим, че MondaySystem(T) = Unix(T) + 3 * 86 400, тъй като реално добавяме още три дни преди началото на Unix timestamp.
Алтернативната ни система има едно сериозно предимство - тъй като започва от понеделник, всяко следващо начало на седмица ще се дели точно на 604 800 и в нея вече може да използваме идеята за

дните. Следователно за да намерим началото на седмицата за даден момент T, можем първо да намерим MondaySystem(T), след това да намерим началото на седмицата, и след това да превърнем резултата обратно в Unix timestamp.
Една подробност, която трябва да се има предвид в тази част от задачата, е, че началото на седмицата може да е отрицателно число. Това се получава за дните от 1 до 4 януари 1970 г. - за тях началото на седмицата е на 29 декември 1969, което отговаря на Unix timestamp -259200. Това не променя алгоритъма, но може да е проблем например, ако се ползват целочислени типове без знак.

Начало на година
Въпреки че годините също имат сравнително фиксирана продължителност, наличието на високосни години затруднява използването на алгоритъм, подобен на горните два.
Един подход, който може да използваме тук, е следният:

```
int year = 1970
while timestamp > getSecondsForYear(year) {
  timestamp -= getSecondsForYear(year)
  ++year
}
```

Така за всяка година ще пресметнем правилния брой секунди в нея, използвайки функция, която се справя с високосни години. Проблемът с това решение е, че за зададените в задачата ограничения е твърде бавно и хваща около 73 точки на официалните тестове.
Един начин да се справим с този проблем е като забележим, че всеки 400 последователни години, започващи от година, която се дели на 400, имат един и същ брой секунди. Това ни позволява да използваме горният алгоритъм, докато достигнем такава година (например 2000), след което да сметнем колко пъти по 400 години да добавим, и да продължим процеса с оставащите < 400 години. По този начин алгоритъмът ще отнеме < 1000 операции вместо първоначалните няколко милиарда.

Алтернативни решения
В авторските решения е включено решение на задачата на Java - чрез него може да придобием по-добра представа как изглеждат реалните библиотеки за работа с подобни проблеми. От една страна, Java функциите сравнително ясно показват какво целим да направим - например:

```
1. Създай дата от Unix timestamp.
2. Премахни полетата, по-малки от 1 ден.
3. Върни съответстващия Unix timestamp.
```

От друга страна обаче, за да постигнем функционалността, търсена в цялата задача, е нужно да използваме 6-7 различни класа със сложни връзки между тях, а и производителността вероятно ще е по-лоша от едно деление и едно изваждане.
Накратко, работата с дати в реалния живот обикновено е най-добре да се остави на добре обмислените и тествани библиотеки, но винаги е полезно да знаем какво се случва отдолу, дори и да не ни се наложи да го използваме.