Discussion and Solution to 2021 Credit Suisse Online Coding Challenge - CodeIT (Entry Challenge)

Abstract

This document serves to discuss, explain and comment the theories, assumptions, principles, procedure and methods to obtain the solution to the entry challenge problem statement. The solution relax the limitations and can take any unit of time of the rate to work on, as well as the duration of unpaid break and maximum continuously working time, any number of time slots, any time boundaries for the time slots and any rates associated with them. The solution is mainly approached by iteration. The method of determining the rate function and the ideas behind the iteration steps in this setup have been explained. The method of finding an O(1) solution that accounts for daylight saving times has been outlined. It has also been argued an O(1) solution is not necessary the optimal solution.

1 Introduction

It is the year of 2038 and robot rights are finally part of our concerns. As an employer, we are obliged to pay for the hard work the robots beings have been working for. The robots can work at almost anytime, regardless it being daytime or at night, on weekdays or during public holidays. The problem statement specified the **minutely rate**[1] for the robots during different times of the week.

It does not take long for some of the most fundamental questions to pop out. *What does these number tell us? How long is a minute?*[2] *Do robots get paid for a minute of work or two when they actually worked for 61 seconds? Can human and the robots both agree on the shift timestamps? How confident can human and robots trust and rely on the results from the rate calculator?*

It is pretty obvious we need to set up an adequate number of boundary conditions *(or one would call it "assumptions")* in order to tackle this challenge in a more precise and clearer manner. In this document, we first discuss theories and the definitions of time calibration and synchronization, and proceed to explore the underlying assumptions and define our working principles, and eventually the solution using iteration on **Python** programming and the limitation of the application. The results have also been checked by manual calculation of some examples.

2 Theory

2.1 Calibration of time

Time is a human construct and thus requires calibration. From some ancient sages looking into the sky to a famous physicist with a wacky tongue photo, the definitions and interpretations of a second has been evolving throughout the previous centuries. The most common understandings of time [3] is outdated. The Earth's daily rotation speed has been slowing, [1] and thus the duration of a day has been increasing, as well as the duration of each second.[4] Upon discovering basing our definition of the duration of a second on the ever-changing Earth's self rotation speed is unreliable, scientists met and defined a second to be

[1] We stick to the convention of the problem statement and drop out the unit of the minutely rate.

[2] Yes, Physicists love to question the definition of time.

[3] Earth completes a rotation about its own rotation axis in 24 hours; there are 24 hours a day, under a sexagesimal subdivision we get the duration of each minute, once more for the duration of a second.

[4] A second under this definition is becoming longer just because there is comparison between different frame of references.

The Second
The second, symbol s, is the SI unit of time. It is defined by taking the fixed numerical value of the caesium frequency, $\Delta\nu_{Cs}$, the unperturbed ground-state hyperfine transition frequency of the caesium 133 atom, to be 9 192 631 770 when expressed in the unit Hz, which is equal to s^{-1} [2]

With this new definition of a second, scientists are able to create atomic clocks [3] that have an accuracy to about one second in 20 million years. [4] Paving our way to develop the

International Atomic Time (TAI)
About 420 clocks contribute as in April 2013 to the construction of TAI at the BIPM. Some 87% of these clocks are either commercial caesium clocks or active, auto-tuned hydrogen masers. To improve the stability of EAL, a weighting procedure is applied to clocks where the maximum relative weight each month depends on the number of participating clocks. About 14% of the participating clocks have been at the maximum weight, on average, per year. This procedure generates a time scale which relies upon the best clocks. [5]

Having developed a more precise time taking algorithm, we still do not wish to fully abandon the method of calibration by subdividing a solar day sexagesimally as it gives the general public a better intuition of the notion of time. To account for the difference in time passed a day in an atomic clock and that in a solar day. Scientists further introduced the concepts of

Leap Second
Leap seconds were implemented in 1972 as an attempt to ensure synchronization between clock time and Earth rotation. [6]

Coordinated Universal Time (UTC)
UTC is the time-scale maintained by the BIPM, with assistance from the IERS, which forms the basis of a coordinated dissemination of standard frequencies and time signals. It corresponds exactly in rate with TAI but differs from it by an integer number of seconds. The UTC scale is adjusted by the insertion or deletion of seconds (positive or negative leapseconds) to ensure approximate agreement with UT1. [7]

The International Organization for Standardization (ISO) [8] further specified a standard set of date and time format representations for information interchange under ISO 8601. [9], where the Proleptic Gregorian calendar and UTC has been adopted as the standard for civil uses for time and date, providing the building blocks of human perception of time scales. It is worth noting that the non-SI unit "minute" is accepted for use with the SI unit "second" [10].

And this concludes most of the international accredited standard of time scale. However this is not perfect yet, people over America do not experience daytime simultaneously with those here in Hong Kong, they experience an apparent time-lag from the self rotation of Earth.[5] To account for that, countries referenced to the UTC and take an offset[6] to retain our comprehension of solar daytime and nighttime while keeping our clocks synchronized. If this has not been confusing enough, some areas of higher latitude use daylight saving time for part of the year, typically by adding one hour to local time during spring and summer. [11]. Clocks are then created and synchronized to the time constructed above. But how do clocks in computers synchronize themselves?

[5]There is no actual time lag across the globe, different parts on the Earth receive different amount of sunlight at different times.
[6]Either positive or negative given their positions relative to the Greenwich

2.2 Synchronizing clocks in computers

In the world of Network Time Protocol (NTP) [12], reference clocks(usually atomic clocks in UTC) are known as stratum 0 servers [13]. They distribute Coordinated Universal Time (UTC) to other devices across a network. Devices connected directly to a reference clock are referred to as stratum 1 servers. Ordinary computers receiving time package from these servers are stratum 2 servers [7]. Real time clock *(usually in the form of quartz clock)* [14] built in devices are responsible for keeping track of the passage of time, the clocks themselves cannot tell what time is it but how long it has been since the synchronization with a time server. Upon retrieving a time package from the time server, it is now able to tell the time by a simple count of seconds transpired since the Epoch. *(or Unix time)* The time package also includes the information of the timezone the particular clock it is ticking in and the information of the daylight saving time. The computer can then easily translate a Unix time to the ISO format for human use.

The Epoch
The epoch is the point where the time starts, and is platform dependent. For Unix, the epoch is January 1, 1970, 00:00:00 (UTC). [15]

We can now all synchronize our clocks in computers to UTC [8] to perform international trades, cryptocurrencies trading, time-sensitive services and transport to facilitate the connection and growth of the international society, overcoming the natural occurring spatial obstacles between human societies. And most importantly, to calculate the rates of robots.

3 Approach

3.1 Assumptions

The periodicity of the addition of leap seconds has been very irregular given the unpredictability of the nature of the slowing of Earth's own rotation speed. The decision whether human will continue to follow the notion of leap seconds will be made in the World Radiocommunication Conference in 2023. [16] I do not come from the future and thus have no information whether leap seconds have been ruled out in the future. During the most recent leap second (2016/12/31T23:59:60) [17] and the immediate second after it 2017/1/1T00 : 00 : 00, Unix systems are unable to resolve the two different seconds properly, and giving the same timestamps for the two instances.

```
>>> time.mktime(time.strptime('2016-12-31T23:59:60', "%Y-%m-%dT%H:%M:%S"))
1483200000.0
>>> time.mktime(time.strptime('2017-01-01T00:00:00', "%Y-%m-%dT%H:%M:%S"))
1483200000.0
```

Assumption I
All leap seconds are ignored.

With this assumption we are happily to say one minute is equal to 60 seconds without worrying about the possibility of the positive or negative leap seconds to mess up our calculations.

Assumption II
All robots have perfectly synchronized and localized with a stratum 1 time server.

[7] Stratum 2 or above, indicating they have a larger distance from the reference clock, and less superior in telling the time
[8] Or an offset with respect to the UTC

Assumption III
All times are exact.

Under this assumption, when the robot is assigned to start working from (2038/1/1T00:00:00+00:00) it actually starts working at that very moment without any ambiguities. The real time experienced by the robots is the same the schedule tells. Not any time delay, its real clocks have prefect resolutions and precision in time. We also neglect the effects of relativity.[9]

Assumption IV
The robot acts exactly according to the timed schedule.

Under this assumption, we do not have to worry about the process time of the computers and argue with the robots whether they have done enough duration time of work to earn their deserved money. All we have to do is to localize and synchronize the robots' times with a stratum 1 time server. They will then infer the timestamps and adjust their own timezones and daylight saving times. **And thus, the results of this program are actual indicators of how much the robots should get paid.**

Assumption V
The robots have fully rested before starting a shift.

The robots only take an unpaid break after working continuously for the maximum duration of time they are entitled to.

Assumption VI
Everything work in the 64-bit.

The problem of 2038 Epochalypse can be easily avoided. [18] For a signed 64-bit integer, the maximum non-negative number is $2^{63} = 9,223,372,036,854,775,808$, which takes about 2.9×10^{11} years to overflow. Pretty sure we will have another programming language by that time.

Assumption VII
Daylight saving times have been ingored.

3.2 Principles

We would like to write a program that takes a variety of different initial conditions and output correspondingly. Before we can explore the limitations of the program, we first have to set up several principles that the program works on.

Principle I
The default unit of time is the unit of time at which the rate is given.

In the problem statement, the rate is a minutely rate and thus the default unit of time is a minute.

Principle II
The working time in each time slots will be rounded up to the nearest unit of time.

Principle III
Time boundaries are [inclusive, exclusive)

[9] Partial effects of the relativity will actually be accounted if the stratum 1 time server are the atomic clocks from the GPS.

Principle IV
No unit of time is counted across any time boundaries of the time slots.

For example, for a daytime [07:00:00-23:00:01) on a weekday a with minutely rate of 20, and a night time [23:00:01-07:00:00) with a minutely rate of 25, if a robot is to work from 22:59:59-23:00:02 *(a total of 3 seconds)*, it is entitled to earn a total of 45. *(20 from the session 22:59:01-23:00:01 and 25 from the session 23:00:01-23:01:01)*

Principle V
Each distinct moment in time is assigned one and only one time slot.

In this solution, it has been relaxed such that there can be more than two time slots per day, or any number of time slots larger or equal to one.

Principle VI
Each distinct moment in time is assigned one and only one rate.

The day time start time has to coincides with the night time end time, and the day time end time has to coincide with the night time start time. And the day time start time cannot coincide with its own end time. Therefore, the start time and end time of a day time contains information about the start time and end time of a night time too.

Principle VII
The robot has to take a unpaid break for every some hours of work.

In this solution, the two time duration is an input and can be varied.

4 Solution

4.1 Input

The program can take a total of 13 different variables as input. **(timedata.py)**

- The unit of time at which the rate is given (time width)
- The maximum time duration the robot can work continuously before taking a break.
- The time duration of each unpaid break.
- Shift Start time in ISO8601 format *(e.g.'2021-10-31T01:59:59')*
- Shift End time in ISO8601 format
- Start time of the day time of a standard day (weekday)
- End time of the day time of a standard day (weekday)
- Start time of the day time of an extra day (weekend)
- End time of the day time of an extra day (weekend)
- Minutely rate during the day time on a standard day
- Minutely rate during the night time on a standard day
- Minutely rate during the day time on an extra day
- Minutely rate during the night time on an extra day

timedata.py extracts information from the input **(timedata.json)**. The functions **datetime.fromisoformat()** and **datetime.strptime()** from the **datetime** module parses the strings from the input to datetime objects.

4.2 Data extraction

In **timedata.py**, a list has been appended by the information from the time slots, the first index 0 holds the start time of a particular time slot, 1 holds the end time and 2 holds the rate during that time. In general the last time slot of the day may extend across the midnight. I approached it by separating it into two time slots with one starting from 00:00:00 to the original end time and the second one with the original starting time and end time at 00:00:00. Note that the date of this end time will always be replaced by one date later during the iteration. And we are good to go to sort the list right away in ascending order of the start times. In *line 35* of **timedata.py** the additional checks for midnight is responsible for not creating a time slot with zero time duration (i.e. starts from 00:00:00 to 00:00:00).

4.3 Initial Checks

check.py stores a function **int_check** to check whether the time data has been input correctly. The program can only function when **Principle V** and **Principle VI** have been satisfied. This check also denied the possibility of back time travelling (shift ended before it has started).

4.4 Rate function

This part is about how the rate function $R(t)$ mentioned in section 5 is constructed. How much a robot can earn during a particular session should only depends on which time slot does it fall into, but not the name attributed to that particular session. In general, two distinct times T_α and T_β will be given as the counterpart of each other as the time boundary for a particular day. **Interchanging the name *"standardDay"* with *"standardNight"* and/or interchanging the name *"extraDay"* with *"extraNight"* in the input should not give a different result of how much the robot should get paid.** In this solution, we went one step ahead and relaxed this limitation. The script is to take an arbitrary number of time slots per day as long as it makes physical senses. The rate function takes the current dummy time variable, information of the latest standard day and extra day time slots as input. The rate function looks which time slots does that particular dummy time falls into and return the corresponding rate.

4.5 Implementation of Principle I and Principle II

Denote ΔT_{rate} the time duration of each session the robot entitled to earn rate once, which is also the default unit of time. T_{start} to be the start time of the shift, T_{end} to be the end time of the shift, ΔT_{work} be the time duration of each working session, ΔT_{break} the time duration of each unpaid break, ${}^nT_{break} = T_{start} + (n+1)\Delta T_{work} + n\Delta T_{break}), n = \{n \in \mathbb{Z}^{0+}\}$ the start time of the $(n+1)^{th}$ break. In general we should expect $\Delta T_{work} \geq \Delta T_{rate}$, since the case $\Delta T_{work} < \Delta T_{rate}$ actually reduces to $\Delta T_{work} = \Delta T_{rate}$. The default width of each iterative time step is ΔT_{rate}. Define the set of total time T, the set of break time T_B, and the set of working time T_{work} by:

$$\begin{aligned} T &= \{T_d \in T_{all} : T_{start} \leq T_d \leq T_{end}\} \\ T_B &= \{T_d \in T : ({}^nT_{break} + \Delta T_{break}) > T_d \geq^n T_{break}\} \\ T_{work} &= T \backslash T_{break} \end{aligned}$$

Define alpha (time about to cross the time boundaries), beta (shift about to end) and gamma (about to take a break) condition to be the set of time:

$$\begin{aligned} A &= \{T_d \in T_{work} : 0 < (T_{\alpha/\beta} - T_d) < \Delta T_{rate}\} \\ B &= \{T_d \in T_{work} : 0 < (T_{end} - T_d) < \Delta T_{rate}\} \\ \Gamma &= \{T_d \in T_{work} : 0 < m\Delta T_{work} + (m-1)\Delta T_{break}) - T_d < \Delta T_{rate}\}, m = \{m \in \mathbb{Z}^+\} \end{aligned}$$

For the set $T_{work} \backslash (A \cup B \cup \Gamma)$, we can just happily iterate one step forward with a time step of ΔT_{rate} without any concerns.

For the set $A\backslash(B \cup \Gamma)$ **(pure alpha condition)**, we iterate one step forward with a time step of $\Delta T_{A\backslash(B\cup\Gamma)} = T_{\alpha/\beta} - T_d$, add the corresponding minutely rate to its total earning by the if-else conditions.

For the set $B\backslash(A \cup \Gamma)$ **(pure beta condition)**, we simply iterate towards T_{end}, add the corresponding rate and end the shift.

For the set $\Gamma\backslash(A \cup B)$ **(pure gamma condition)**, if $\Delta T_{break} \geq \Delta T_{rate}$, we can simply iterate a time step of ΔT_{rate} forward and add the corresponding rate and enters the breaking phase.
If $({}^nT_{break} + \Delta T_{break}) - T_d < \Delta T_{rate}$, or $T_d > .{}^nT_{break} + \Delta T_{break} - \Delta T_{rate}$, we iterate a step with width of $({}^nT_{break} + \Delta T_{break} - T_d)$ and add the corresponding rate. It is OK to iterate over the break period since it does not contribute to any earnings. **The compulsory condition for gamma condition to occur is** $\Delta T_{break} < \Delta T_{rate}$. And gamma condition will not happen if the duration of an unpaid break is an hour which is much larger than the time width of one minute.

For the set of $(A \cap B)\backslash\Gamma$, one might think there are two cases to consider, which are $T_{\alpha/\beta} > T_{end}$ or $T_{\alpha/\beta} \leq T_{end}$ respectively. However we can just drop out the first since iterating a time step ΔT_{over} over T_{end} means $(T_d + \Delta T_{over}) \notin T$. And this reduces to a simple two steps of first iterating towards $T_{\alpha/\beta}$, and then iterate towards T_{end} in their corresponding time steps and add up their corresponding rates. The resultant condition is thus a pure alpha condition followed by a pure beta condition.

For the set of $(B \cap \Gamma)\backslash A$, if ${}^nT_{break} \geq T_{end}$, we can just iterate towards T_{end} and add the corresponding rate. If ${}^nT_{break} + \Delta T_{break} \geq T_{end}$, we treat it the same way. If ${}^nT_{break} < .{}^nT_{break} + \Delta T_{break} < T_{end}$, we first iterate a step of $({}^nT_{break} + \Delta T_{break} - T_d)$, and then another step towards T_{end}, and add up the two corresponding rates. The resultant condition is thus a pure gamma condition followed by a pure beta condition.

For the set of $(A \cap \Gamma)\backslash B$, first consider $T_{\alpha/\beta} < .{}^nT_{break}$, we first iterate a step of $(T_{\alpha/\beta} - T_d)$ and add the corresponding rate, it then becomes a pure gamma condition. If $({}^nT_{break} + \Delta T_{break}) > T_{\alpha/\beta} > .{}^nT_{break}$, it is a pure gamma condition. If $T_{\alpha/\beta} > ({}^nT_{break} + \Delta T_{break})$, it is a pure gamma condition followed by a pure alpha condition.

For the set of $A \cap B \cap \Gamma$, it is actually not that different from the other cases. We should not iterate across T_{end} and thus $T_{\alpha/\beta}, {}^n T_{break} \leq T_{end}$. Having identified this, this case is almost identical to the previous case. Thus, this is a three step iteration, first a pure gamma condition, followed by a pure alpha condition, and eventually a pure beta condition.

It is much easier to iterate through the breaking period, we only have to replace the time step whenever the robot has to start working again, which is the case if $T_d + \Delta T_{rate} > .{}^nT_{break} + \Delta T_{break}$, the time step is replaced by $({}^nT_{break} + \Delta T_{break} - T_d)$. Let's call this the ξ condition.

Altogether, we can conclude any rounding up scenarios when the robot is working can be treated by a three steps iteration. First a pure gamma condition, then a pure alpha condition and finally the pure beta condition. The gamma condition should be given the highest priority, then the alpha condition and the least to the beta condition during the iteration.

4.6 Iteration

The process of the iteration can be found in **main.py**. This process made use of the results from the previous modules and principles. Checks for alpha and gamma conditions have been infused into the iteration. Priorities for checks have been given to gamma and alpha conditions, this can be seen in their authorities to alter the width of the time steps, but none have been given to beta conditions. The gamma conditions have a higher authority than alpha conditions since their conditions will be checked before the

alpha conditions. When gamma condition is met in a particular step the gamma value will increase to 1. The alpha conditions will not be run when $\gamma = 1$. The alpha value will be checked next. The ordinary condition will not be run when either γ or $\alpha = 1$. The dummy time variable has to be refresh in order for all the conditions to run properly. The γ and α value will be reset to 0 after the time has been refreshed and we are good to continue the iteration.

5 In the search of an optimal solution

This section discuss the framework of how my solution to the problem statement was obtained. We start off by first ignoring the DST rules and then generalizing to the case when DST is present. It is then explained why an O(1) solution was not chosen to be my final solution but a combined solution between O(1) and O(n).

5.1 O(1) solution without DST

In the Proleptic Gregorian calendar, the occurrences of weekdays and weekends are periodic in nature. In this problem, a full session is $(8 + 1)$ hours in length. The **least common multiple** of a full session with a week time is a time length of 3 weeks. For every 3 weeks, there are 56 full sessions fitted within it. An arbitrary shift can be thought as the addition of two sub-shifts. The first sub-shift will be of a duration of $n\times$ 3 weeks, where $n \in \mathbb{Z}^{0+}$, and the second sub-shift will be of the length of the remaining duration. Note that the start time of sub-shift 1, $T_{start(S1)} = T_{start}$ and the start time of sub-shift 2, $T_{start(S2)} = n \times \Delta T_{3weeks} + T_{start}$. Similarly, the end time of sub-shift 1, $T_{end(S1)} = T_{start(S1)} + n \times \Delta T_{3weeks}$ and the end time of sub-shift 2, $T_{end(S2)} = T_{end}$. The integer n can be easily calculated by the following command:

```
>>> n = (shift_end_time - shift_start_time) // datetime.timedelta(weeks=3)
```

Let $R(t)$ be the minutely rate function and $B(t)$ be a break function such that $B(t) = 0$ when the robot is taking a break and $B(t) = 1$ when it is working. From the argument above, we can see that both $R(t)$ and $B(t)$ are a periodic piece-wise functions of a mutual period of 3 weeks. When we are to find how much to pay to the robot, we are actually doing an summation of $S = \sum_{T_{start}}^{T_{end}} R(t) \times B(t)\Delta t$. By the the basic theorems of periodic functions,

$$\begin{aligned} S &= \sum_{T_{start(S1)}}^{T_{end(S1)}} R(t) \times B(t)\Delta t + \sum_{T_{start(S2)}}^{T_{end(S2)}} R(t) \times B(t)\Delta t \\ &= n \times \sum_{T_{start}}^{T_{start}+\Delta T_{3weeks}} R(t) \times B(t)\Delta t + \sum_{T_{start(S2)}}^{T_{end}} R(t) \times B(t)\Delta t \\ &= n \times \sum_{T_{start}}^{T_{start}+\Delta T_{3weeks}} R(t) \times B(t)\Delta t + \sum_{T_{start}}^{T_{end}-(n\times\Delta T_{3weeks})} R(t) \times B(t)\Delta t \end{aligned}$$

where $n \in \mathbb{Z}^{0+}$, if we look at the inequality

$$T_{start} + \Delta T_{3weeks} \geq T_{end} - (n \times \Delta T_{3weeks})$$

which in turns give us

$$(n + 1)\Delta T_{3weeks} \geq T_{end} - T_{shift}$$

always holds given the definition of n. This tells us the value of the second sum is contained within the first one. While we iterate to get the value of the first sum, we can first stop at $T = T_{end} - (n \times \Delta T_{3weeks})$ to register the value of the second sum, and continue to iterate until we reach the full duration of 3 weeks to get the value of the first sum. Altogether, we only have to do iteration once to get the value of the two sums as well as the total earnings of the robot. **This is an O(1) solution to the problem statement where no**

daylight saving times have been observed. This solution can be further generalized to account for the different inputs (e.g. different break duration/ different default time width). We simply replace the duration of 3 weeks **by the least common multiple of the total duration of a session (maximum continuously working time + duration of break), the default time width and the time duration of one week.**

5.2 O(1) solution with DST

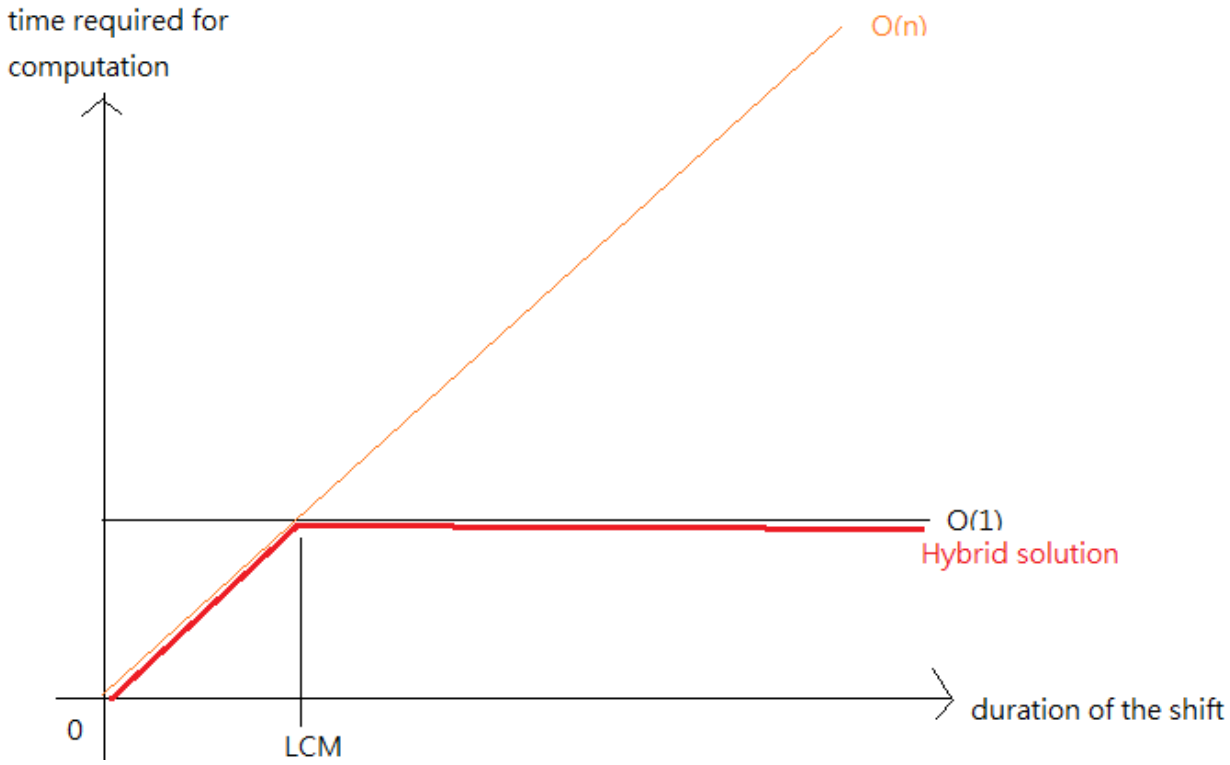
The shift from normal time to daylight saving time occurs once per year and so does shifting back to normal time. Denote $T_{DST(start)}$ to be the starting time of the daylight saving time in local time when daylight saving time is off, $T_{DST(end)}$ to be the end time of the daylight saving time in local time when the daylight saving time is off, ΔT_{DST} be the time difference when a daylight saving time starts or ends, and it is usually half an hour or an hour in length. It does not differ from 5.1) by that much, we only have to look at a much longer time span. By the same argument as that made in section 5.1, we should expect the existence of the least common multiple of all the time duration to be a length of η years. However, one may naively think the period of daylight saving times is once per year. This is not the case. In the Gregorian calendar, the smallest repeating unit in a year basis should be of a period of 400 years. As we relaxed the flexibility of the duration of the duration of each time input, after 400 years of iterations, a time step do not necessary return to its original starting point and offset by exactly 400 years. Thus η is a multiple of 400 years and is the least common multiple of the time-width, time interval of each working session and 400 years. We then write the total earning of a general shift to be

$$S = m \times \sum_{T_{start}}^{T_{start}+\eta\Delta T_{400years}} R(t) \times B(t)\Delta t + \sum_{T_{start}}^{T_{end}-m\eta\Delta T_{400years}} R(t) \times B(t)\Delta t$$

where $m \in \mathbb{Z}^{0+}$ and all times are registered in the local time when daylight saving time is not active.

5.3 Optimal solution

We have found an O(1) solution to the problem, but that does not necessary means this solution is always the fastest in computing. Trivially, when the shift lasts shorter than the duration as computed by the least common multiple of all the required factors, an O(n) solution gives the answer faster. But for any shift that last longer than the least common multiple, the O(1) solution is always faster in giving us the answer.



My solution has been coded in such a way that when the input shift is shorter than the LCM time, the O(n) way is preferred. When the input shift is longer than the threshold, the program will first cut the shift into two sub-shifts, find all the upper and lower time limits for the summations, memorize the result of the second sub-shift when it has finished calculating it, and proceed to find the sum of the first sub-shift. The figure above depicted the time complexity of my solution compared to a pure O(1) and a pure O(n) solution. A combination of two gives the fastest performance and thus chosen as my preferred solution.

6 Results

This part serves to check the results of the program manually by looking at different examples. **Only parts of the code are copied for each example. The layout has also been edited.**

Example I
Basically the set up of the first test case but with the end time of night time shift one second later.

```
>>>"shift": {"start": "2038-01-01T20:15:00","end": "2038-01-02T05:16:01"},
"standard_day": {
                "slot1": {"start": "07:00:00","end": "23:00:01","value": 20},
                "slot2": {"start": "23:00:01","end": "06:00:00","value": 25},
                "slot3": {"start": "06:00:00","end": "07:00:00","value": 25}}
The robot will earn a total of 13780 in this shift.
```

The additional 1 second of working time before the time boundary is rounded up to 1 minute and thus the robot earned 20 more.

Example II
The setup violated principle V.

```
>>>"extra_day": {
```

```
                "slot1": {"start": "07:00:00","end": "23:00:01","value": 20},
                "slot2": {"start": "23:00:00","end": "06:00:00","value": 25},
                "slot3": {"start": "06:00:00","end": "07:00:00","value": 25}}
```

Please check all extra day times have been specified a unique non-empty rate.
repl process died unexpectedly: exit status 1

Example III
A special setup

```
>>>"shift": {"start": "2037-12-31T23:59:59", "end": "2038-01-01T00:00:02"},
   ""standard_day":␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"slot1":␣{"start":␣"00:00:02","end":␣"00:00:00","value":␣20},
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"slot2":␣{"start":␣"00:00:00","end":␣"00:00:01","value":␣25},
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"slot3":␣{"start":␣"00:00:01","end":␣"00:00:02","value":␣25}}
The␣robot␣will␣earn␣a␣total␣of␣70.0␣in␣this␣shift.
```

The shift lasted for 3 seconds, but the three seconds will be rounded up to three different minutes. The robot get paid $20 + 25 + 25 = 70$.

Example IV
This setup has the same time slots as example VI, but end time of the shift, the break duration, maximum working time and the time-width has been changed.

```
>>>"end": "2038-01-01T00:00:05"},
   break_duration = datetime.timedelta(seconds=1)
   maximum_working_time = datetime.timedelta(seconds=3)
   timewidth=datetime.timedelta(seconds=2)
The robot will earn a total of 90.0 in this shift.
```

The robot earned 20 from 23:59:59 to 00:00:00, 25 from 00:00:00 to 00:00:01, 25 from 00:00:01 to 00:00:02, took a break from 00:00:02 to 00:00:03, earned 20 from 00:00:03 to 00:00:05. The robot earned a total of $20 + 25 + 25 + 20 = 90$

7 Conclusion

My script to the entry challenge composes of 7 assumptions, and works on 8 principles. The script can take 13 different variables as input, The solution relax the limitations and can take any unit of time of the rate to work on, as well as the duration of unpaid break and maximum continuously working time, any number of time slots, any time boundaries for the time slots and any rates associated with them. The solution is mainly approached by iteration. The discussion of O(1) time complexity and optimal solution has also been included.

References

[1] J. P. Laboratory, *Earth's daily rotation slowing*, https://www.jpl.nasa.gov/news/earths-daily-rotation-slowing, 1988.

[2] BIPM., *Le système international d'unités*, https://www.bipm.org/documents/20126/41483022/SI-Brochure-9.pdf/fcf090b2-04e6-88cc-1149-c3e029ad8232, p.130, 2019.

[3] J. Nelson, *What is an atomic clock?* https://www.nasa.gov/feature/jpl/what-is-an-atomic-clock, 2019.

[4] NIST., *A brief history of atomic clocks at nist*, https://www.nist.gov/pml/time-and-frequency-division/time-services/brief-history-atomic-clocks-nist, 2016.

[5] B. I. des Poids et Mesures (BIPM)– Time Department –, *Commission 1 : Reference frames*, https://iag.dgfi.tum.de/fileadmin/IAG-docs/Travaux2013/08_BIPM.pdf, p.3, 2011.

[6] B. Luzum, *The role of the iers in the leap second*, https://www.iers.org/SharedDocs/Publikationen/EN/IERS/Documents/IERS_Leap_Seconds.pdf?__blob=publicationFile&v=1, p.2.

[7] I. R. Assembly, *Standard-frequency and time-signal emissions*, https://www.itu.int/dms_pubrec/itu-r/rec/tf/R-REC-TF.460-6-200202-I!!PDF-E.pdf, Rec. ITU-R TF.460-6, 2002.

[8] *About iso*, https://www.iso.org/about-us.html, 2007.

[9] I. 154, *Date and time — representations for information interchange — part 1: Basic rules*, https://www.iso.org/standard/70907.html, ISO 8601-1:2019(en), 2019.

[10] BIPM., *Le système international d'unités*, https://www.bipm.org/documents/20126/41483022/SI-Brochure-9.pdf/fcf090b2-04e6-88cc-1149-c3e029ad8232, p.145, 2019.

[11] NIST., *Daylight saving time rules*, https://www.nist.gov/pml/time-and-frequency-division/popular-links/daylight-saving-time-dst, 2021.

[12] S. Kostecke, *What is ntp (network time protocol) ?* https://support.ntp.org/bin/view/Main/WebHome, 2003.

[13] D. Waldron, *Stratum levels & ntp explained — welcome to the 'stratumopshere'*, https://www.galsys.co.uk/news/ntp-stratum-levels-explained, 2014.

[14] D. (https://cs.stackexchange.com/users/755/d-w), *How do computers keep track of time?* https://cs.stackexchange.com/q/54935, 2016.

[15] T. P. S. Library, *Time access and conversions*, https://docs.python.org/3/library/time.html.

[16] *Coordinated universal time (utc) to retain "leap second"*, https://www.itu.int/net/pressoffice/press_releases/2015/53.aspx, 2015.

[17] NIST, *Leap second and ut1-utc information*, https://www.nist.gov/pml/time-and-frequency-division/time-realization/leap-seconds, 2021.

[18] A. Bergmann, *The end of an era*, https://www.linaro.org/blog/the-end-of-an-era/, 2020.