Kourosh Alasti

Kalasti@csu.fullerton.edu

Kevin Nguyen

oggunderscore@csu.fullerton.edu

https://github.com/kourosh-alasti/CPSC335-projectOne

Project 1 Analysis Submission

Time Complexity:

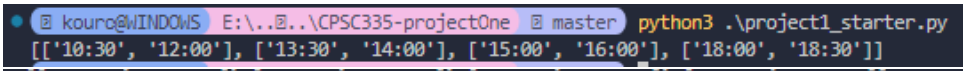
main() = O (x + 3m + 3n + (m + n)) = O(n)

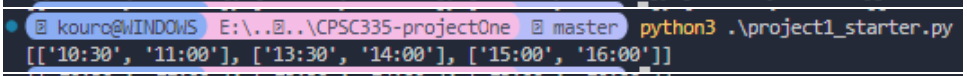
invert_time_slots() = O(n)

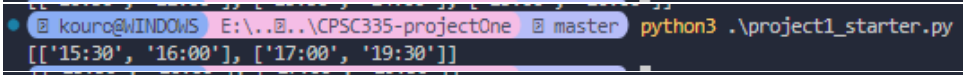
find_available_slots() = O(n)

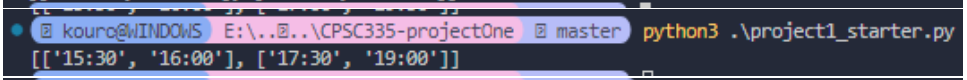
find_common_time_slots() = O (n)

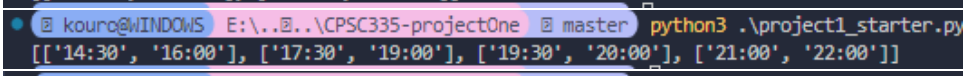
Test Cases

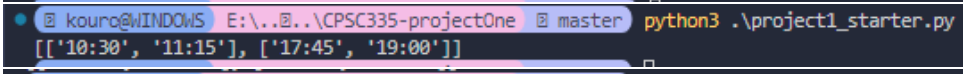
1) 

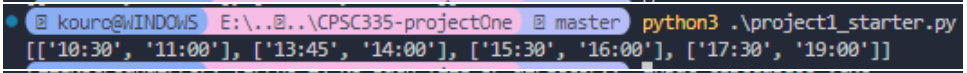
2) 

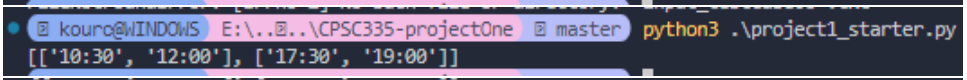
3) 

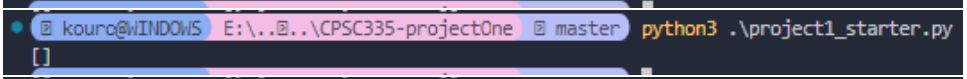
4) 

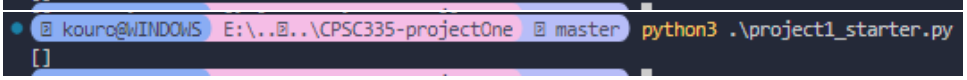
5) 

6) 

7) 

8) 

9) 

10) 

Proof

Proving our algorithms efficiency is relatively simple as our algorithm has been separated into 3 parts. Invert_time_slots, find_available_slots and find_common_time_slots. Looking at the source code of each function they all consist of 1 for loop each result in a common time complexity of O(n). Translating this to our overall time complexity is simple. Looking at our main function you will realize each function has a separate call resulting in a linear flow of our code as our variable N is related over each function. Our main functions actual time complexity is calculated as O(x + 3m + 3n + (m + n)) where x is length of our input file, m is the size of schedule1 and n is the size of schedule 2. This actual time complexity can be simplified to O(n) or Linear, as the only mathematical operation taking place is addition.

Mathematical Analysis

Pseudocode – invert_time_slots

Function inver_time_slots(schedule):

inverted_schedule = []

start_of_day = conver_to_datetime("00:00")

end_of_day = convert_to_datetime("23:59")

for each timeslot in schedule do:

start_time = convert_to_datetime(timeslot[0])

end_time = convert_to_datetime(timeslot[1])

if index is 0 and start_time > start_of_day then:

inverted_schedule.append([convert_to_string(start_of_day),

convert_to_string(start_time)])

If index is equal to length of schedule – 1 and end_time < end_of_day then:

Inverted_schedule.append([convert_to_string(end_time),

Convert_to_string(end_of_day)])

Else if index is less than length of schedule – 1 then:

Next_start_time = convert_to_datetime(schedule[index + 1][0])

If end_time < next_start_time then:

```
                inverted_schedule.append([convert_to_string(end_time),
                convert_to_string(next_start_time)])
        Return inverted_schedule
```

Step Count

2 + n + 1 = n + 3, where n is the size of the timeslot schedule

Time Complexity

O(n)

Order of Growth

Linear

Pseudocode – find_available_slots

```
Function find_available_slots(avail_schedule, daily_activity, duration):
avail_slots = [] // List to store converted busy schedule to datetime object

for each start, end in avail_schedule do:
convert start and end to datetime objects
add (start, end) to avail_slots
convert_daily_activity start and end to datetime objects
daily_available = (daily_activty[0], daily_activity[1])

merged_schedules = []

for each start, end in avail_slots do:
if start < daily_available[1] then:
if end <= daily_available[0] then:
contine // skip irrelevant slots
if start < daily_available[0] and end > daily_available[0] then:
start = daily_available[0]
```

if end > daily_available[1] then:

end = daily_available[1]

if start < end then:

add (start, end) to merged _schedules

return merged_schedule // Return the final merged available slots

Step Count

2 + n + 2 + 1 = 2n + 5, where n is the size of the timeslot schedule

Time Complexity

O(n)

Order of Growth

Linear

Pseudocode – find_common_time_slots

Function find_common_time_slots(schedule1, schedule2, duration):

Merged_schedules = [] // List to store overlapping time slots

I, j = 0 , 0 // initialize pointers for schedule 1 and schedule 2

While I < length of schedule 1 and j < length of schedule 2 do:

Start1, end1 = schedule1[I]

Start2, end2 = schedule[j]

Overlap_start = max of start 1 and start 2

Overlap_end = min of end 1 and end 2

If overlap start < overlap end then:

If overlap end – overlap start >= duration in minutes then:

Merged_schedule.append([overlap start, overlap end])

If end 1 < end 2 then:

I += 1 // move to the next time slot in schedule 1

Else:

J += 1 // move to the next time slot in schedule 2

Final schedule = []

For each start, end in merged_schedule do:

Convert start and end to human-readable string

Add [start, end] to final_Schedule

Return final_schedule

Step Count

2 + n + m + k + 1 = n + m + k + 1, where n, m are the size of both timeslot schedules and k is the size of the merged timeslots

Time Complexity

O(n)

Order of Growth

Linear

Comments

Since our algorithm runs at optimal efficiency, O(n), our most applicable improvements would come with the exporting of functions to different files to clean up our main file to allow for readable files for other developers and ourselves.

An increase in N will not result in a complexity class change as the operation between n will remain addition leading to a simplified O(N) linear efficiency. O(N + M + V), where N M and V are peoples schedules is still O(N)