COT 5405 ANALYSIS OF ALGORITHMS

Algorithms Programming Project I

Greedy Algorithms

Spring 2023

Team Members
Abdul Samadh - 2213-7268
Sheel Taskar - 2753-4463

Team Members

This programming project is an original work done by Sheel Taskar and Abdul Samadh. Sheel was primarily responsible for designing, analyzing, and programming STRATEGY 1, as well as developing a bonus algorithm to improve STRATEGY 4. Sheel also conducted a comparative study of all four greedy strategies, experimenting and generating a histogram grouping implementation for each. Meanwhile, Abdul focused on developing and designing Strategies 2, 3, and 4, and contributed to the comparative study of the bonus algorithm by generating time graphs to compare the two implementations of STRATEGY 4. We equally shared the responsibilities of producing testcases which resulted in optimal and sub-optimal result for strategy 1,2 and 3. We also worked together on writing the proof of correctness for strategy 4 and the efficient implementation of strategy 4.

Greedy Algorithms and Analysis

STRATEGY 1:

DESIGN:

- STRATEGY 1 requires iterating through all the houses, for each day we must check if any of the houses can be painted on that day.
- As our input is already sorted primarily on Start Day and secondarily on End Day, we can keep two pointers.
- One pointer to keep track of day (starting from 1) and another to keep track of current house to be painted (initially first house).
- We move from day 1 to n, for each day we check if current house can be painted or not.
- If it can be painted, we increment the current house (by one) to point next house, and we also increment the day (by one) to the next day.
- If house cannot be painted, it means either the Start Day of the house is later, or the Deadline of the house is missed.
- If Start Day of the house is late, we simply increment the day.
- If Deadline of the house is missed, we point the next house.
- Since this algorithm only requires us to iterate from day 1 to n, decision to paint the house as well as scheduling the house takes Θ (1). In the worst case at some point all remaining houses could have missed the deadline for the current day. In this case we will keep incrementing houses till we run out of houses, or we get a house which can be painted on the current day. So, in the worst case our algorithm will have iterating through all the days and all the house and hence the time complexity would be Θ (n+m).

ALGORITHM:

```
INITIALIZE paintedIndexList to an empty list.

INITIALIZE currentDay to 1 and currentHouse to 1.

WHILE currentDay <= n and currentHouse <=m:

  IF (currentHouse can be painted)
    ADD current house to paintedIndexList
    INCREMENT both currentDay and currentHouse

  ELSE

    IF (currentDay is earlier than currentHouse Start Date)
      INCREMENT currentDay
    ELSE
      INCREMENT currentHouse

RETURN paintedIndexList
```

INPUT FOR OPTIMAL SOLUTION:
3 3
1 2
2 2
2 4

n = 3
m = 3
houseList = {[1,2], [2,2], [2,4]}

STRATEGY 1 Solution:
1 2 3
{[1,2], [2,2], [2,4]}

Optimal Solution:
1 2 3
{[1,2], [2,2], [2,4]}

INPUT FOR SUB-OPTIMAL SOLUTION:
4 5
1 1
1 3
1 4
2 2
2 3

n = 4
m = 5
houseList = {[1,1], [1,3], [1,4], [2,2], [2,3]}

STRATEGY 1 Solution:
1 2 3
{[1,1], [1,3], [1,4]}

Optimal Solution:
1 4 2 3
{[1,1], [2,2], [1,3], [1,4]}

STRATEGY 2:

DESIGN

- STRATEGY 2 requires iterating through all the houses, for each day we must check all the houses that can be painted on that day as well as the day before.
- As you need to keep track of the current day as well as all the previous day houses, we need a priority queue to store the houses encountered till the current day.
- Regarding assignment of the house, algorithm tells us to paint the house that starts being available the latest.
- Hence, we need to maintain a max heap according start date of the houses.
- For each day 1 to n, for each day, we check which houses can be painted, and we push all the houses to max heap.
- Once all the houses are pushed to heap which became available in the current day, we pop one house from the heap and paint it if valid and continue to next day.
- As we are iterating from 1 to n, we push and pop each house in the house list exactly once. The time complexity of push and pop operation in a heap is log(m). Since we do exactly two log(m) (push and pop) operations on m houses and process n days we have a time complexity of Θ (n + m*log(m)).

ALGORITHM:

```
INITIALIZE paintedIndexList to an empty list and an empty max heap indexed on start date.

INITIALIZE currentDay to 1 and currentHouse to 1.

FOR 1 to n:

   WHILE (start date of currentHouse is less than currentDay):
      PUSH house in the heap with start date as a key
```

```
        INCREMENT currentHouse

    WHILE (heap is not empty):
        POP the house from the heap

        IF (popped house is valid):
            ADD popped house to paintedIndexList
            BREAK from the loop

RETURN paintedIndexList
```

INPUT FOR OPTIMAL SOLUTION:
4 5
1 1
1 3
1 4
2 2
2 3

n = 4
m = 5
houseList = [[1,1],[1,3],[1,4],[2,2],[2,3]]

STRATEGY 2 Solution:
1 4 5 3
{ [1,1] [2,2] [1,3] [1,4] }

Optimal Solution:
1 4 2 3
{ [1,1] [2,2] [1,3] [1,4] }

INPUT FOR SUB-OPTIMAL SOLUTION:
12 7
1 1
1 2
1 3
1 4
2 10
2 11
2 12

n = 12
m = 7
houseList = [[1,1],[1,2],[1,3],[1,4],[2,10],[2,11],[2,12]]

STRATEGY 2 Solution:
1 5 6 7
{ [1,1] [2,10] [2,11] [2,12] }

Optimal Solution:
1 2 3 4 5 6 7
{ [1,1] [1,2] [1,3] [1,4] [2,10] [2,11] [2,12]}

STRATEGY 3:

DESIGN

- STRATEGY 3 requires iterating through all the houses, for each day we must check all the houses that can be painted on that day as well as the day before.
- As you need to keep track of the current day as well as all the previous day houses, we need a priority queue to store the houses encountered till the current day.
- Regarding assignment of the house, algorithm tells us to paint the house that is available for shortest duration.
- Hence, we need to maintain a min heap according to the difference between the end and start times of the houses.
- For each day 1 to n, for each day, we check which houses can be painted, and we push all the houses to the min heap.
- Once all the houses that became available on the current day, are pushed to heap, we pop the house from the heap and paint it if valid and continue to next day.
- As we are iterating from 1 to n, we push and pop each house in the house list exactly once. The time complexity of push and pop operation in a heap is log(m). Since we do exactly two log(m) (push and pop) operations on m houses and process n days we have a time complexity of Θ (n + m*log(m)).

ALGORITHM:

```
INITIALIZE paintedIndexList to an empty list and an empty min heap indexed on end date - start date
.

INITIALIZE currentDay to 1 and currentHouse to 1.

FOR 1 to n:

   WHILE (start date of currentHouse is less than currentDay):
      PUSH house in the heap with end date - start date as key
      INCREMENT currentHouse

   WHILE (heap is not empty):
      POP the house from the heap

      IF (popped house is valid):
```

```
        ADD popped house to paintedIndexList
        BREAK from the loop

RETURN paintedIndexList
```

INPUT FOR OPTIMAL SOLUTION:
12 7
1 1
1 2
1 3
1 4
2 10
2 11
2 12

n = 12
m = 7
houseList = {[1,1],[1,2],[1,3],[1,4],[2,10],[2,11],[2,12]}

STRATEGY 3 Solution:
1 2 3 4 5 6 7
{ [1,1] [1,2] [1,3] [1,4] [2,10] [2,11] [2,12]}

Optimal Solution:
1 2 3 4 5 6 7
{ [1,1] [1,2] [1,3] [1,4] [2,10] [2,11] [2,12]}

INPUT FOR SUB-OPTIMAL SOLUTION:
6 6
1 1
1 3
1 4
2 2
2 3
3 5

n = 6
m = 6
houseList = [[1,1],[1,3],[1,4],[2,2],[2,3],[3,5]]

STRATEGY 3 Solution:
1 4 5 6
{ [1,1] [2,2] [2,3] [3,5] }

Optimal Solution:
1 4 2 3 6
{ [1,1] [2,2] [1,3] [1,4] [3,5] }

STRATEGY 4:

DESIGN

- STRATEGY 4 requires iterating through all the houses, for each day we must check all the houses that can be painted on that day as well as the day before.
- As you need to keep track of the current day as well as all the previous day houses, we need a priority queue to store the houses encountered till the current day.
- Regarding assignment of the house, algorithm tells us to paint the house that stops being available the earliest.
- Hence, we need to maintain a min heap according end date of the houses.
- For each day 1 to n, for each day, we check which houses can be painted, and we push all the houses to min heap.
- Once all the houses, that became available that day, are pushed to the heap, we pop the house from the heap and paint it if valid and continue to the next day.
- As we are iterating from 1 to n, we push and pop each house in the house list exactly once. The time complexity of push and pop operation in a heap is log(m). Since we do exactly two log(m) (push and pop) operations on m houses and process n days we have a time complexity of Θ (n + m*log(m)).

ALGORITHM:

```
INITIALIZE paintedIndexList to an empty list and an empty min heap indexed on end date.

INITIALIZE currentDay to 1 and currentHouse to 1.

FOR 1 to n:

   WHILE (start date of currentHouse is less than currentDay):
      PUSH house in the heap with end date as key
      INCREMENT currentHouse

   WHILE (heap is not empty):
      POP the house from the heap

      IF (popped house is valid):
         ADD popped house to paintedIndexList
         BREAK from the loop

RETURN paintedIndexList
```

INPUT FOR OPTIMAL SOLUTION:
6 6
1 1
1 3
1 4
2 2
2 3
3 5

n = 6
m = 6
houseList = [[1,1],[1,3],[1,4],[2,2],[2,3],[3,5]]

STRATEGY 4 Solution:
1 4 2 3 6
{ [1,1] [2,2] [1,3] [1,4] [3,5] }

Optimal Solution:
1 4 2 3 6
{ [1,1] [2,2] [1,3] [1,4] [3,5] }

PROOF OF CORRECTNESS OF STRATEGY 4:

Let S be the greedy strategy and O be the optimal strategy.
For r = 1,
There is only one house so both S and O will choose the only job.

For r = n,
Let us assume that first n elements in S and O are the same.

For r = n+1,
Let's say S (greedy algorithm) chooses house A and O (Optimal algorithm) chooses house B to be painted on day d. Then by definition,

endDate(A) < endDate(B)

Let us see if we can replace B with A in O without changing the total number of houses that get painted.

Since house A is available to be painted on day d,

endDate(A) >= d

In the worst case endDate(A) will be exactly equal to d, which means endDate(B) will be > d (There is at least 1 day after d where B can still be painted). If we replace B with A in O, we can schedule A on day d in the worst case and paint house B on d+1. Hence by exchange argument we can say that O (optimal solution) can be transformed to greedy solution, hence greedy solution is optimal.

PROOF OF TERMINATION FOR STRATEGY 4:

We iterate through n days which will be the outer most loop. Since number of days n is finite, it is bound to terminate. Inside this loop, on each day we iterate through houses and if day ∈ [startDate(house), endDate(house)] we push this house to the priority queue. Now we need to paint a house on this day, for this we pop from the priority queue. Eventually each house is pushed once and popped once from the priority queue, and since m is finite this algorithm will terminate after all n days and m houses have been processed.

EXPERIMENTAL COMPARATIVE STUDY:

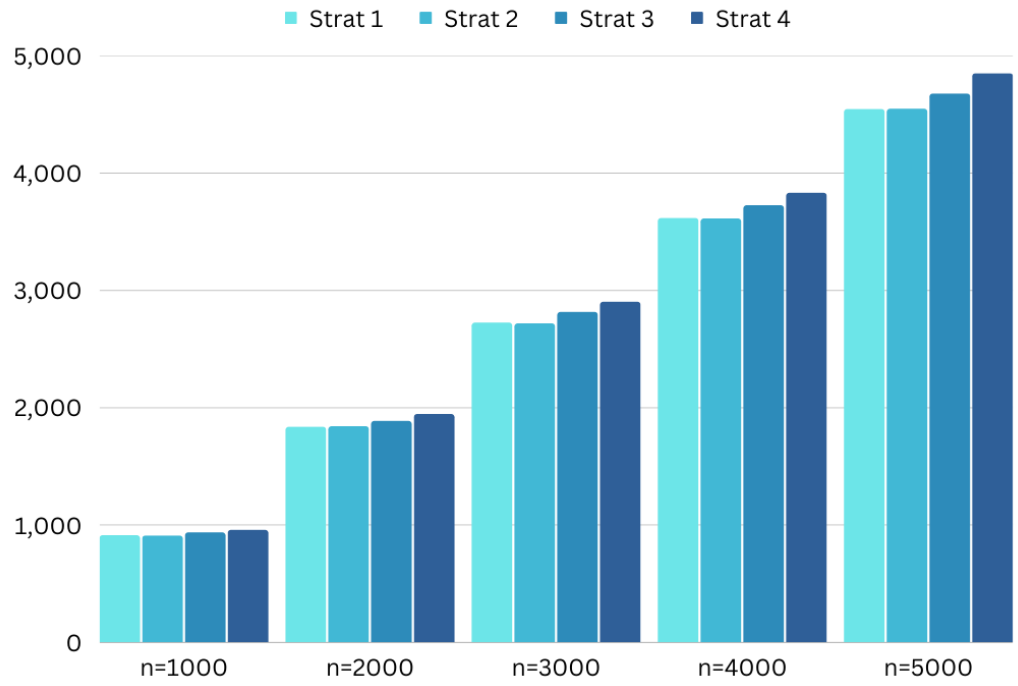


Figure: Number of houses scheduled by each STRATEGY grouped together for input size

For the experimentation we generated 5 input files populated with random house list for n = 1000,2000,3000,4000,5000

We repeated the process for multiple files for different values of m & n. In the above graph, we kept the number of houses (m) the same as number of days (n). Though changing the value of will not change the trend highlighted by the above graph which is as follows:

1. For every value of n, STRATEGY 4 yields the maximum count of scheduled houses among all the other strategies.
2. As we earlier deduced, STRATEGY 4 is an optimal solution, number of houses scheduled by Start 4 will always be greater than or equal to other strategies.

CONCLUSION:

We started our assignment with the implementation of STRATEGY 1. STRATEGY 1 was by far the easiest implementation of all four strategies as it required only incrementing the day and

current house pointer based on the validity of the current house. Unlike other implementations, it does not require additional data structures like priority queues. Hence, time and space complexity of this STRATEGY was better, but it fails in the main part, i.e., correctness. The strategy fails to give an optimal solution.

Next, we started STRATEGY 2 which is not as trivial as STRATEGY 1, which required to keep track of seen houses in a priority queue. Hence, as another data structure is required and push and pop operations were also involved, both time and space complexity are increased. Even though, the performance overhead is being increased in this STRATEGY, the STRATEGY 2 does not necessarily give better results. In our experimentation, we found out that there are several instances where STRATEGY 1 gives better results (paints more houses) then STRATEGY 2. Hence, the increased execution costs in STRATEGY 2 don't necessarily account for better results...

Next, we went to STRATEGY 3. STRATEGY 3 is quite like STRATEGY 2, just the priority queue key is different. Hence, implementing STRATEGY 3 didn't take long. One point to note for STRATEGY 3 is that higher values of n or m, STRATEGY 3 generally gives better results than both STRATEGY 1 and STRATEGY 2 (exceptions are possible (only if we handcraft such cases) but highly unlikely). Hence, we can say that increased overhead can be justifiable in case of STRATEGY 3.

Finally, STRATEGY 4 is also very similar to STRATEGY 2 & 3, with some minor changes in priority queue. Implementing Start 4 was also very straight forward once we implemented STRATEGY 2 & 3. The minor change in priority queue results in Stra 4 giving optimal results. Hence, we can confidently say that Start 4 will always paint the same of higher number of houses compared to STRATEGY 1,2 & 3.

Bonus

AIM:
Efficient implementation of STRATEGY 4

TARGET:
To run the strategy in $\Theta(m \log(m))$ time

INSIGHT:

- Input of houses is sorted primarily according to Start Date and secondarily on End Date.
- Given STRATEGY 4 implementation requires us to iterate through day 1 to n and schedule the houses that stop being available the earliest.
- As long as we iterate through each day, time complexity will always have a factor of "n" in time complexity.
- In order to achieve time complexity $\Theta(m*\log(m))$, we need to eliminate the dependency of days and instead just focus on painting each house.
- We should iterate through m houses and schedule each house $\Theta(\log(m))$ time.

DESIGN:
As we want to just focus on scheduling houses just by looking at their start date and end date, we will start iterating through m houses.

Input being in sorted order according to start date and end date makes our life easier.

We know that whenever we start iterating, we will see houses that have same or higher start date.

So, our aim is to group all the houses together, mark them as seen, so compare them to get the one with the lowest end date.

We do so my pushing houses with the same start date into a min-heap indexed on the end date. Pushing items on min heap will give us the privilege of also comparing the houses that have been added earlier. (This is quite like STRATEGY 4)

While iterating through the houses, whenever we detect a change in start date, it means that now we have pushed all the houses into the min-heap that can be painted till the current day. Hence, we pop the element in the heap (with lowest end date) and paint it (if valid).

There are two caveats in the above steps. The first being, if we detect a change in start date and the change is greater than one, it means we must do more than one pop operations to account for the days missed.

Note that we are not iterating through the dates, just doing more pop operations (Note that that we only push each house into the queue exactly once and pop them exactly once). The number of pops we do is equal to the difference in current day and the start date of next house which is different. This way, make sure we are not missing any day's work.
The second caveat is, once we iterate through all the houses, we might still have elements in the heap. We must pop all the elements from the heap and schedule them one by one (if valid).

ALGORITHM:

```
INITIALIZE paintedIndexList to an empty list and an empty min heap indexed on end date.

INITIALIZE currentDay to start date of 1st house and currentHouse to 1.

FOR house 1 to m:

   IF (start date of currentHouse is equal to currentDay):
      PUSH house in the heap with end date as key

   ELSE ():
      WHILE (currentDay is less than start day of currentHouse and
                  there are elements in the heap):
         POP the house from the heap

         IF (popped house is valid):
            ADD popped house to paintedIndexList
            INCREMENT currentDay
```

```
    PUSH the currentHouse to the heap


  WHILE (heap is not empty):
    POP the house from the heap

    IF (popped house is valid):
      ADD popped house to paintedIndexList
      INCREMENT currentDay

RETURN paintedIndexList
```

TIME AND SPACE COMPLEXITY ANALYSIS:

The above design involves pushing m houses to min heap and popping m houses from the min heap. A push and a pop operation takes log(m) time each. Since, we are doing m pushes and m pops, over overall time complexity will be 2m*log(m) which is Θ (m*log(m)).

Our design requires a min heap of size m to keep track of visited houses. We only need one heap; all the push and pop operations will be done in the same heap and at one time there cannot be more than m houses in the heap. Therefore, the space complexity of the algorithm will be Θ (m).

PROOF OF TERMINATION:

Optimal version requires iterating through houses and making m pushes and m pops along the way.

Since, the number of houses are finite, the loop will end. Also, as the number of houses is finite, if we keep popping elements from the heap, it will eventually be empty (after at max m pops), hence the loop will terminate.

As both for loop for iterations and while loop for popping both ends, the algorithm will terminate.

PROOF OF CORRECTNESS:

We gave the proof of correctness for Strategy 4. Since, Strategy 5 and Strategy 4 have the same approach (scheduling house with lowest end date for pushed house till current date), we can say if Strategy 5 yields the same solution as Strategy 4, Strategy 5 will give optimal solution.

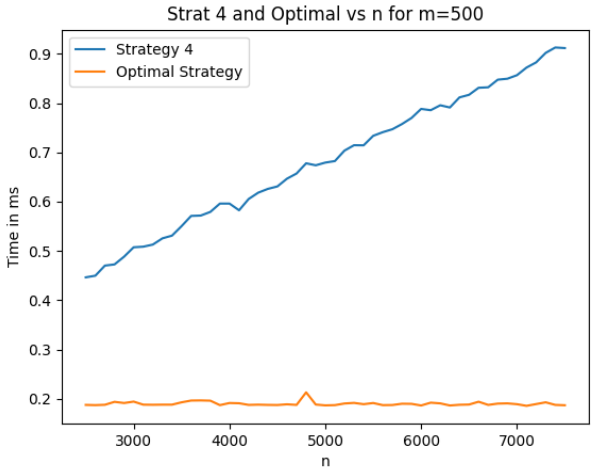
Strategy 5 iterates through houses (instead of days) and pushes houses with the same start day in the heap. Once that's done, the Strategy 5 pops the element from the heap and paints it. Also, Strategy 5 accounts for missed days by making multiple pops (while loop in else condition).

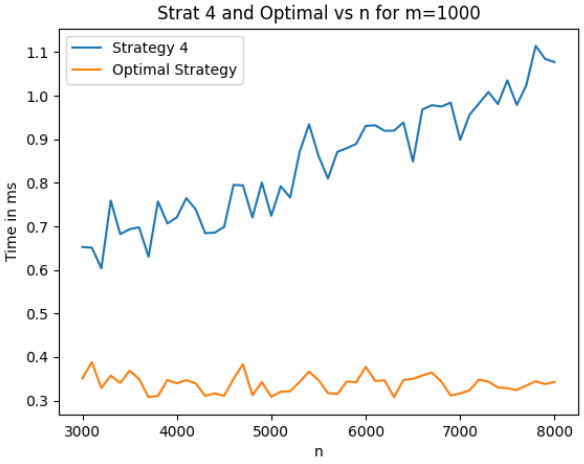
Strategy 4 iterates through days, for each day it pushes the elements in the heap. Once all the houses are pushed for the current day, we make a pop and increment the day.

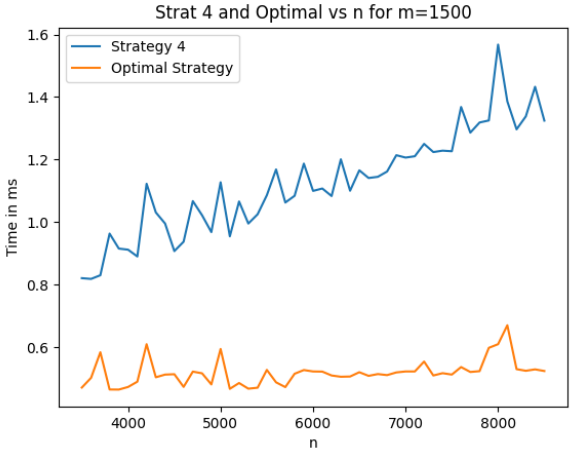
Strategy 4 and Strategy 5 both push elements with the same start day in the heap and pops the element when all houses for the given day are pushed. Both strategies account for all the days. Since, these two are the same approach, they will yield exactly same solution.

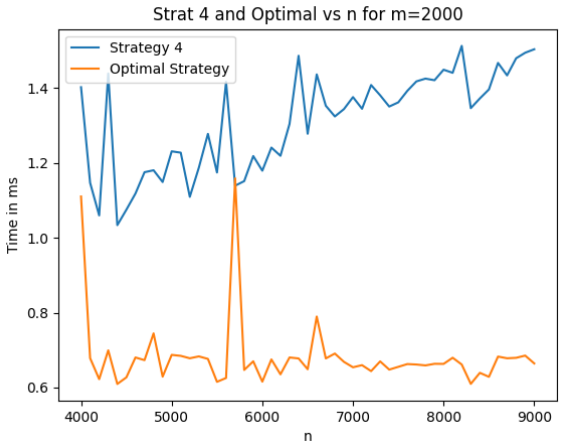
EXPERIMENTAL COMPARATIVE STUDY:

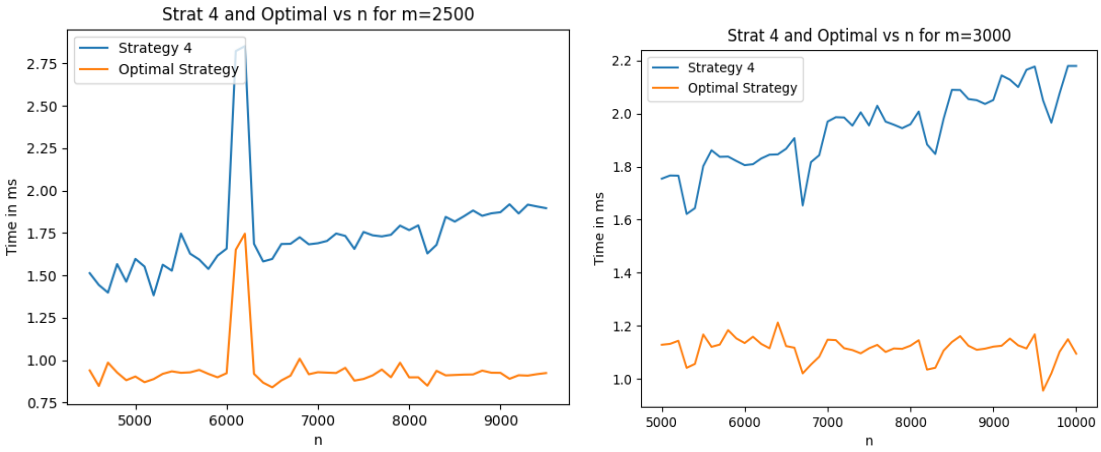
The time complexity of STRATEGY 4 is ***Θ (n + m log(m))*** and the time complexity of STRATEGY 5 is ***Θ (m log(m))***. This would imply that for a constant value of ***m*** when ***n*** increases the time graph for STRATEGY 4 will increase linearly with ***n***, on the other hand, the time graph for STRATEGY OP will remain constant since it does not depend on. To prove this, we plot 5 graphs for m values 500, 1000, 1500, 2000, 2500, and 3000. We will vary n from m+1000 to m+7000.











From these graphs, we can observe that when ***m*** is comparatively smaller ***(m=500)*** than ***n (n= 1500 to 7500)***, the time graph for STRATEGY 4 increases linearly with ***n***. As we take larger values of ***m (m=3000)***, the time graph for STRATEGY 4 starts to flatten since the ***m log(m)*** component of ***Θ (n + m log(m))*** starts to dominate. The time graph for STRATEGY OP remains constant for a given ***m*** irrespective of the value of ***n*** which proves it is independent of ***n***.