COT 5405 ANALYSIS OF ALGORITHMS

PROGRAMMING ASSIGNMENT 1

Spring 2023

Group No: 20	
Team Members	
Vedant Patil	3948-5964
Shivani Patil	5354-9503

Table of Content

1. Problem Definition:

You are a house painter that is available from day 1 . . . n (inclusive). You can only paint one house in a day. It also only takes one day to paint a house. You are given m houses. For each house i, you are also given startDayi and endDayi for i = 1, . . . , m. The house i can only be painted on a day between startDayi and endDayi (inclusive). The given houses are already sorted primarily on startDay and secondarily on endDay (in case of equality of the startDay). You are tasked to find the maximum number of houses that you can paint.

We have discussed all the given 4 strategies, their design in required time complexity and implementation in the below sections, followed by comparative study of each of the implementation. We have also implemented the STRAT4 which we found to be optimal greedy in Θ(m logm)

Steps to run:

1. Go to Source Folder
2. Execute "make" command
3. Execute "make run#" command to run particular strategy.
   e.g. "make run1" to execute STRAT1
   "make run5" to execute optimal implementation of STRAT4

2. STRATEGY 1

## STRAT 1:

Iterate over each day starting from day 1 . . . n. For each day, among the unpainted houses that are available to be painted on that day, paint the house that started being available the earliest.

2.1 DESIGN

1. Since the input is already sorted as per startDay_i, we can iterate through all the days and find the first house compatible such that it can be scheduled on the day. Since the algorithm is supposed to be θ(n), we can schedule the first house that is compatible since the input is already sorted by startDay.
2. In case we are at a house i with startDay>day, it means no other house j such that j>i in the list will be eligible to be scheduled on the day. In that case, we can't paint any house on that day.
3. In case we are at a house i where endDay<day, which means this house can’t be scheduled on the day, but maybe we can schedule other houses.

Taking these pointers in mind we devise the following algorithm

2.2 PSEUDO CODE

```
EARLIEST-AVAILABLE-FIRST ( n, $startDay_1$ , $endDay_1$ , .... $startDay_m$ , $endDay_m$ )

day ← 1
house_number ← 1
schedule ← $\phi$
WHILE ( day <= n or house_number <= m )
        IF ( house house_number is compatible with day day )
           Schedule house j for day d
           day ← day + 1
           house_number ← house_number + 1

        ELSE IF ($startDay_j$ > day)
               day  ← day + 1

        ELSE IF ($endDay_j$ < day)
               house_number ← house_number + 1

END WHILE
RETURN schedule
```

2.3 EXAMPLE AND COUNTEREXAMPLE

Consider the input sequence -

```
n = 10  m = 6
m1 = [1 2]
m2 = [3 4]
m3 = [3 8]
m4 = [5 6]
m5 = [7 9]
m6 = [8 10]
```

Iteration 1:
Day = 1
House = 1 -> [1 2]

Since 1 is eligible in interval [1 2],
Day 1 -> house 1
Therefore,
Day++
House++

Iteration 2:
Day = 2
House = 2 ->[3 4]
Since 2 is not eligible in the interval
Also startDay(3)>2
Therefore,
Day++

Iteration 3:
Day =3
house=2 ->[3 4]
Since 3 is eligible in interval [3 4]
Day 3 -> house 3
Therefore,
Day++
House++

Going so on our schedule would be like

Day	House Number	Interval
1	1	[1 2]
2	-	-
3	2	[3 4]
4	3	[3 8]
5	4	[5 6]
6	5	[5 7]
7	-	-
8	6	[8 10]

Even though the solution is optimal in this case where we are able to schedule all the houses on some or the other days, the algorithm is not optimal. We prove that using the following counterexample –

n = 10 m = 9
m1 = [1 2]
m2 = [3 4]
m3 = [3 8]
m4 = [3 9]
m5 = [4 5]
m6 = [5 6]
m7 = [5 7]
m8 = [7 9]
m9 = [9 10]

The Algorithm generates the following output

day	Interval scheduled in STRAT 1	Optimal case scheduling
1	[1 2]	[1 2]
2	-	-
3	[3 4]	[3 4]
4	[3 8]	[4 5]

5	[3 9]	[5 6]
6	[5 6]	[5 7]
7	[5 7]	[3 8]
8	[7 9]	[3 9]
9	[9 10]	[7 9]
10	-	[9 10]

2.4 ANALYSIS

The above algorithm iterates for all the days and picks the first eligible house for that day. For any day where there is no suitable house, the algorithm skips the day. Similarly for the houses which are past the deadline in the list, the algorithm will skip those houses. Since each iteration, we can assign a house for that day, skip the house or skip the day, in the best case the algorithm will run in O(n) complexity. In some cases the algorithm would have to skip over a large number of houses and the complexity might go above O(n), but still the main variable component for the algorithm is the parameter n, as small n means early termination and large n means we have to go over all houses to make sure we schedule something on maximum number of days. Therefore we can assume the algorithm is θ(n). Apart from the input (Space - O(m)), the algorithm uses a list variable to store indices of the scheduled houses, therefore the space complexity is O(m).

3. STRATEGY 2

## STRAT 2:
Iterate over each day starting from day 1 . . . n. For each day, among the unpainted houses that are available that day, paint the house that started being available the latest.

3.1 DESIGN

1. We will first initialize a priority queue that will keep track of unpainted houses that are available to be painted for the current day. Here the priority is the highest startDay. The time complexity to insert an element in a priority queue is O(log m). Every element will be added to the priority queue only once. Therefore, the time to add all the m houses in PQ will be O(m log m).

2. Since the input is already sorted as per startDayi, we can iterate through all the days and add all the houses that are compatible to be scheduled for that day to the priority queue if startDay>=day and endDay<=day. In case we come across a house that is not compatible with the day, that means none of the houses after that will be compatible with that day.

3. In case we are at house i and all compatible houses are in the priority queue and the priority queue is not empty, pop the house with the highest priority and schedule it for the day. Increment the day.

4. In case if multiple houses have similar startDay, we poll the one with the highest endDay as that is the house which became available the latest.

5. If the priority queue is not empty and the endDay of the peek house is less than the current day, that means the peek house is not compatible with the day, remove it from the priority queue. We will keep on removing the top element from the priority queue till we find a compatible house at peek.

6. Every house therefore will be offered and polled from the PQ which brings the complexity to O(m log m). No house will be visited twice to insert on any particular day and all the m houses will be offered and polled exactly once.

7. At the same time we can either schedule a house on a day or skip the based on whether a house is eligible or not.

8. Iteration over all days requires O(n) and priority queue operations are O(m log m). Therefore, the time complexity of this strategy is O(n + m log m).

3.2 PSEUDO CODE

```
LATEST-AVAILABLE-FIRST ( n, $startDay_1$ , $endDay_1$ , .... $startDay_m$ , $endDay_m$ )

schedule ← $\phi$
// Create a priorityQueue which polls based on the highest startDay
priorityQueue pq ← $\phi$
day ← 1
house_number ← 1
WHILE ( day <= n or house_number <= m )
        // Find all the houses eligible to start on the current day
        While( house_number <= m  and $startDay_{house\_number}$ <= day)
                add house house_number to pq with startDay as priority
                house_number ← house_number + 1

      // Peek the pq to check if the rear has a compatible house
      // Poll until pq is either empty or has a compatible house at rear
      // Polling ensures removing tasks that won't be compatible with current
and any forthcoming day
      While(pq is not empty and $endDay_{pq.peek()}$ < day)
            pop pq
      // From the eligible houses, schedule the one with highest priority
        IF ( pq is not Empty)
            pop house h' with the highest priority from pq
            schedule house h' for day day
        day ← day + 1

END WHILE
RETURN schedule
```

3.3 EXAMPLE AND COUNTEREXAMPLE

Consider the input sequence -

n = 10 m = 6

m1 = [1 2]

m2 = [3 4]

m3 = [3 8]

m4 = [5 6]

m5 = [7 9]

m6 = [8 10]

Initialize Empty Priority Queue PQ
house_number = 1

Iteration 1: Day 1 house_number = 1 house_number = 2 Day 1 -> [1 2]	 PQ -> [1 2] PQ -> [1 2] PQ -> []
Iteration 2: Day 2 house_number = 2 Day 2 -> -	 PQ -> []

Iteration 3:
 	
Day 3
house_number = 2
house_number = 3
house_number = 4
 	
Day 3 -> [3 8]
	
	PQ -> [3 4]
	PQ -> [3 4], [3,8]
	PQ -> [3 4], [3,8]
	
	PQ -> [3 4]

Iteration 4: Day 4 house_number = 4 Day 4 -> [3 4]	 PQ -> [3 4] PQ -> []
Iteration 5: Day 5 house_number = 4 house_number = 5 Day 5 -> [5 6]	 PQ -> [5 6] PQ -> [5 6] PQ -> []
Iteration 6: Day 6 house_number = Day 6 -> -	 PQ -> []

Going so on our schedule would be like

Day	House Number	Interval
1	1	[1 2]
2	-	-
3	3	[3 8]
4	2	[3 4]
5	4	[5 6]
6	-	-
7	5	[7 9]
8	6	[8 10]
9	-	-
10	-	-

Even though the solution is optimal in this case where we are able to schedule all the houses on some or the other days, the algorithm is not optimal. We prove that using the following counterexample -

n = 10 m = 9
m1 = [1 2]
m2 = [3 4]
m3 = [3 8]
m4 = [3 9]
m5 = [4 5]
m6 = [5 6]
m7 = [5 7]
m8 = [7 9]
m9 = [9 10]

The Algorithm generates the following output

day	Interval scheduled in STRAT 2	Optimal case scheduling
1	[1 2]	[1 2]
2	-	-
3	[3 9]	[3 4]
4	[4 5]	[4 5]
5	[5 7]	[5 6]
6	[5 6]	[5 7]
7	[7 9]	[3 8]
8	[3 8]	[3 9]
9	[9 10]	[7 9]
10	-	[9 10]

3.4 ANALYSIS

The above algorithm iterates for all the days and picks the first eligible house for that day from the priority queue. For every iteration, the algorithm adds all the houses that can be scheduled on the day. Since a house can only be added to the queue once, it takes O(mlog(m)) time to add houses to the priority queue. Now since for each day we poll the best house from the priority queue, it takes O(log(m)) time to remove a house. Since every house can be polled at max once from the priority queue, the complexity of polling is O(log(m)). Therefore in the worst case, we would end up adding all the houses to the priority queue on the first day, but since no new add operation will happen for the following days there would be just poll operations. The overall worst complexity would be O(n+m*log(m)+m*log(m)), which would be O(n+m*log(m)).

Apart from the input (Space - O(m)), the algorithm uses a list variable to store indices of the scheduled houses and a priority queue which at max would have m houses in it, therefore the space complexity is O(m).

4. STRATEGY 3

## STRAT 3:

Iterate over each day starting from day 1 . . . n. For each day, among the unpainted houses that are available that day, paint the house that is available for the shortest duration.

4.1 DESIGN

1. We can use the Algorithm described in STRAT 2 , just changing the priority of the priority queue to duration of all the eligible instances i.e endDay - startDay, instead of their startDay. Here the house with the smallest duration will get the priority during polling.
2. Since the algorithm remains the same, the complexity remains the same and only the output changes.
3. Iteration over all days requires O(n) and priority queue operations are O(m log m). Therefore, the time complexity of this strategy is O(n + m log m).

Taking these pointers in mind we devise the following algorithm

4.2 **PSEUDO CODE**:

```
SHORTEST-DURATION-FIRST ( n, $startDay_1$ , $endDay_1$ , .... $startDay_m$ , $endDay_m$ )

schedule ← $\phi$

// Create a priorityQueue which polls based on the Shortest duration ie
(endDay - startDay)

priorityQueue pq ← $\phi$

day ← 1

house_number ← 1

WHILE ( day <= n or house_number <= m )
        // Find all the houses eligible to start on the current day
        While( house_number <= m  and $startDay_{house\_number}$ <= day)
                add house house_number to pq with Shortest duration as priority
                house_number ← house_number + 1

        // Peek the pq to check if the rear has a compatible house
        // Poll until pq is either empty or has a compatible house at rear
        // Polling ensures removing tasks that won't be compatible with
    current and any forthcoming day

        While(pq is not empty and $endDay_{pq.peek()}$ < day)
            pop pq

        // From the eligible houses, schedule the one with highest priority
        IF ( pq is not Empty)
            pop house h' with highest priority from pq
            schedule house h' for day day
        day ← day + 1

END WHILE
RETURN schedule
```

4.3 EXAMPLE AND COUNTEREXAMPLE

n=5 m = 6
m1 = [1 5]
m2 = [2 4]
m3 = [4 5]
m4 = [4 6]
m5 = [6 9]
m6 = [9 10]

Initialize Empty Priority Queue PQ
house_number = 1

Iteration 1: Day 1 house_number = 1 house_number = 2 Day 1 -> [1 5]	 PQ -> [1 5] PQ -> [1 5] PQ -> []

Iteration 2:
 	
Day 2
house_number = 2
house_number = 3
	
	PQ -> [2 4]
	PQ -> [2 4]
	
	PQ -> []

Day 2 -> [2 4]	
Iteration 3: Day 3 house_number = 3 Day 3 -> -	 PQ -> [] PQ -> []

Iteration 4:
 	
Day 4
house_number = 3
house_number = 4
house_number = 5
 	
Day 4 -> [4 5]
	
	PQ -> [4 5]
	PQ -> [4 5] , [4 6]
	
	PQ -> [4 6]

Iteration 5: Day 5 house_number = 5 Day 5 -> [4 6]	 PQ -> [4 6] PQ -> []
Iteration 6: Day 6 house_number = 5 house_number = 6 Day 6 -> [6 9]	 PQ -> [6 9] PQ -> [6 9] PQ -> []

Going so on our schedule would be like

Day	**House Number**	**Interval**
1	1	[1 5]
2	2	[2 4]
3	-	-
4	3	[4 5]
5	4	[4 6]
6	5	[6 9]
7	-	-
8	-	-

9	6	[9 10]
10	-	-

Even though the solution is optimal in this case where we are able to schedule all the houses on some or the other days, the algorithm is not optimal. We prove that using the following counterexample -

n = 10 m = 9
m1 = [1 2]
m2 = [3 4]
m3 = [3 8]
m4 = [3 9]
m5 = [4 5]
m6 = [5 6]
m7 = [5 7]
m8 = [7 9]
m9 = [9 10]

The Algorithm generates the following output

day	Interval scheduled in STRAT 3	Optimal case scheduling
1	[1 2]	[1 2]
2	-	-
3	[3 4]	[3 4]
4	[4 5]	[4 5]
5	[5 6]	[5 6]
6	[5 7]	[5 7]
7	[7 9]	[3 8]
8	[3 8]	[3 9]
9	[9 10]	[7 9]
10	-	[9 10]

4.4 ANALYSIS

The algorithm is similar to the one discussed in strat 2, only the priority in the priority queue is different. Therefore, the overall worst complexity would be O(n+m*log(m)+m*log(m)), which would be O(n+m*log(m)).

Apart from the input (Space - O(m)), the algorithm uses a list variable to store indices of the scheduled houses and a priority queue which at max would have m houses in it, therefore the space complexity is O(m).

5. STRATEGY 4

## STRAT 4:

Iterate over each day starting from day 1 . . . n. For each day, among the unpainted houses that are available that day, paint the house that will stop being available the earliest

5.1 DESIGN

1. We can use the Algorithm described in STRAT 2 , just changing the priority of the priority queue to endDay of all the eligible instances instead of their startDay. Here the house with the earliest endDay will get the priority during polling.
2. Since the algorithm remains the same, the complexity remains the same and only the output changes.
3. Iteration over all days requires O(n) and priority queue operations are O(m log m). Therefore, the time complexity of this strategy is O(n + m log m).
4. Strata4 is the **OPTIMAL ALGORITHM** out of the 4 mentioned starts. We will argue its correctness in the following sections.

5.2 PSEUDO CODE

```
EARLIEST-ENDTIME-FIRST ( n, $startDay_1$ , $endDay_1$ , .... $startDay_m$ , $endDay_m$ )
schedule ← $\phi$
    // Create a priorityQueue which polls based on the earliest deadline ie (endDay)
    priorityQueue pq ← $\phi$
    **day** ← 1
    **house_number** ← 1
    WHILE ( **day** <= n or **house_number** <= m )
        // Find all the houses eligible to start on the current day
        While( **house_number** <= m  and $startDay_{house\_number}$ <= **day**)
```

```
                add house **house_number** to pq with earliest deadline as priority
                **house_number** ← **house_number** + 1
    // Peek the pq to check if the rear has a compatible house
    // Poll until pq is either empty or has a compatible house at rear
    // Polling ensures removing tasks that won't be compatible with current and any forthcoming day
        While(pq is not empty and $endDay_{pq.peek()}$ < day)
            pop pq

    // From the eligible houses, schedule the one with highest priority
        IF ( pq is not Empty)
            pop house h' with highest priority from pq
            schedule house h' for day **day**
        **day** ← **day** + 1

END WHILE
RETURN schedule
```

5.3 EXAMPLE AND PROOF OF CORRECTNESS

1. Let's take the example where the other algorithms fail to generate the optimal solution

   n = 10 m = 9
   m1 = [1 2]
   m2 = [3 4]
   m3 = [3 8]
   m4 = [3 9]
   m5 = [4 5]
   m6 = [5 6]
   m7 = [5 7]
   m8 = [7 9]
   m9 = [9 10]

   Initialize Empty Priority Queue PQ
   house_number = 1

Iteration 1: Day 1 house_number = 1 house_number = 2 Day 1 -> [1 2]	 PQ -> [1 2] PQ -> [1 2] PQ -> []

Iteration 2:	
Day 2	PQ -> []
house_number = 2	
	PQ -> []
Day 2 -> -	
Iteration 3:	
Day 3	PQ -> [3 4]
house_number = 2	PQ -> [3 4], [3 8]
house_number = 3	PQ -> [3 4], [3 8], [3 9]
house_number = 4	PQ -> [3 4], [3 8], [3 9]
house_number = 5	
	PQ -> [3 8], [3 9]
Day 3 -> [3 4]	
Iteration 4:	
Day 4	PQ -> [3 8], [3 9], [4 5]
house_number = 5	PQ -> [3 8], [3 9], [4 5]
house_number = 6	
	PQ -> [3 8], [3 9]
Day 4 -> [4 5]	
Iteration 5:	
Day 5	PQ -> [3 8], [3 9], [5 6]
house_number = 6	PQ -> [3 8], [3 9], [5 6], [5 7]
house_number = 7	PQ -> [3 8], [3 9], [5 6], [5 7]
house_number = 8	
	PQ -> [3 8], [3 9], [5 7]
Day 5 -> [5 6]	
Iteration 6:	
Day 6	PQ -> [3 8], [3 9], [5 7]
house_number = 8	
	PQ -> [3 8], [3 9]
Day 6 -> [5 7]	
Iteration 7:	
Day 7	PQ -> [3 8], [3 9], [7 9]
house_number = 8	PQ -> [3 8], [3 9], [7 9]
house_number = 9	
	PQ -> [3 9], [7 9]
Day 7 -> [3 8]	

Going so on our schedule would be like

day	Interval scheduled in STRAT 4	Optimal case scheduling
1	[1 2]	[1 2]
2	-	-
3	[3 4]	[3 4]
4	[4 5]	[4 5]
5	[5 6]	[5 6]
6	[5 7]	[5 7]
7	[3 8]	[3 8]
8	[3 9]	[3 9]
9	[7 9]	[7 9]
10	[9 10]	[9 10]

Therefore, we argue that the algorithm described in STRAT 4 is the optimal algorithm.

2. **Proof of Correctness:**

Loop Invariant **:**
During every iteration of the while loop, we add all the compatible houses which can be scheduled on that particular day i.e houses with startDay same as the currentDay. Then amount those houses we poll the house with the earliest deadline. This process continues for all the days.

Initialization **:**
We start with day 1 and consider the first house. Depending on compatibility we add to the pq and move to the next day or either skip the day or the house.

Maintenance **:**
At every loop we move to the next day, which means either newer houses are added to the pq or a house will be polled from the pq.

Termination **:**

The program will terminate when day>n or house_number>n meaning after scheduling houses for all the days or going through all the possible houses available, we will end our loop. At the end the **optimalSet** would give the indices of houses that are scheduled.

Proof by Contradiction **:**

- Let's assume that the greedy strategy cannot generate an optimal solution. Let G represent the solution set generated by the greedy approach and O represent the one generated by the optimal approach.
- Let i_1, i_2,...., i_k be the set of houses that are scheduled at days between 1 to n(not all counted) by the greedy algorithm. Let |G| = x
- Let $j_{1,1}$, j_2,...., j_l be the houses scheduled at days between 1 to n (not all counted) by the optimal strategy. Let |O| = y
- Now as per our assumption y>=x. We will prove that it's not the case.
- Let the solution sets match between optimal and greedy strategy uptill a maximum day r such that $i_1 = j_1$, $i_2 = j_2$, ... $i_r = j_r$.
- Which means the greedy strategy and optimal strategy schedule the same houses on the same days till the maximum day r.
- Now on day r, the greedy strategy among the eligible houses, picks the house with the earliest finish time, Whereas the optimal strategy picks a house with O($endDay_r$) >= G($endDay_r$).
- Holding off a house with an earlier deadline makes it very probable that the house cannot be scheduled in the following day since the next day might be over the endDay of the house. Meaning the house with higher deadline day can be scheduled later on but house with lower deadline day might not be scheduled later.
- Therefore, the house with the earlier deadline day can safely replace a house with higher deadline day for any eligible day r.
- This for day r, we can replace j_r such that $i_r = j_r$ without changing the solution set
- This contradicts our assumption for maximum day r.

- We can therefore replace all the houses in our optimal strategy with the ones from the greedy strategy and conclude O=G.

Any other task that the optimal strategy can have, Greedy will have it by default as the houses would have $\text{endDay}_G <= \text{endDay}_O$

5.4 ANALYSIS

The algorithm is similar to the one discussed in strat 3, only the priority in the priority queue is different. Therefore, the overall worst complexity would be O(n+m*log(m)+m*log(m)), which would be O(n+m*log(m)).

Apart from the input (Space - O(m)), the algorithm uses a list variable to store indices of the scheduled houses and a priority queue which at max would have m houses in it, therefore the space complexity is O(m).

6. EXPERIMENTAL COMPARATIVE STUDY

We created different data sets For n = 1000 to 5000 and m = n/4 to 5n/4 for each n and ran different strategies on those datasets. Following figures 1 to 4 show respective graphical representation of results obtained.

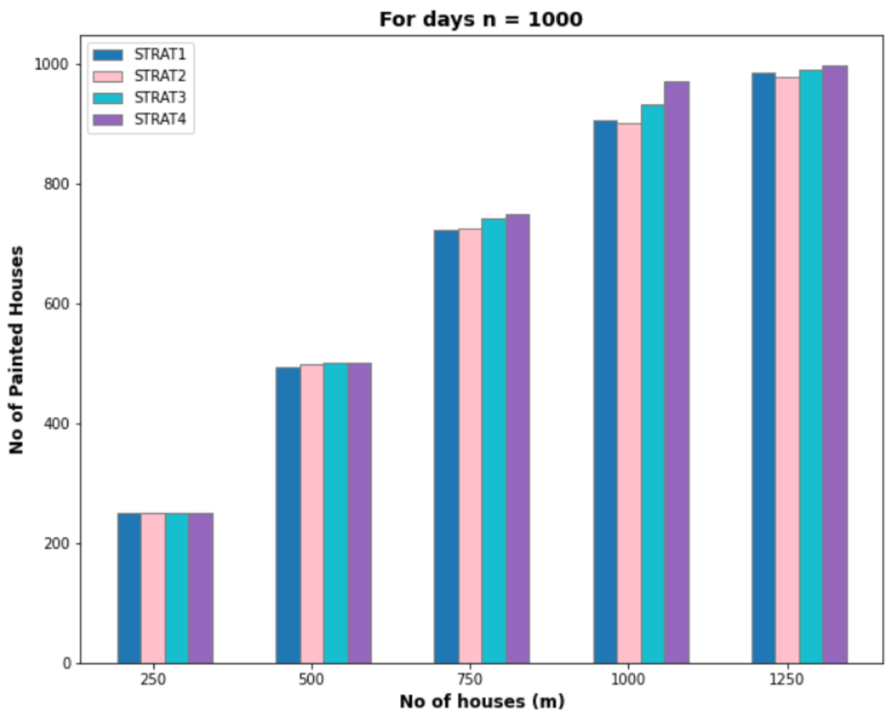

Figure 1: *No of Houses(m) Vs No of Painted Houses For n=1000*

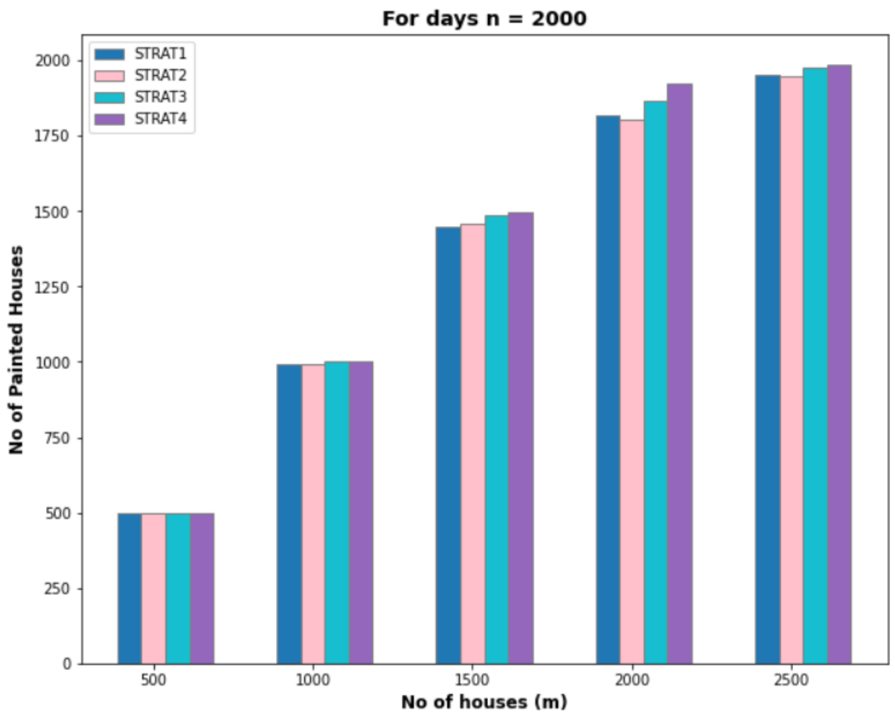

Figure 2: *No of Houses(m) Vs No of Painted Houses For n=2000*

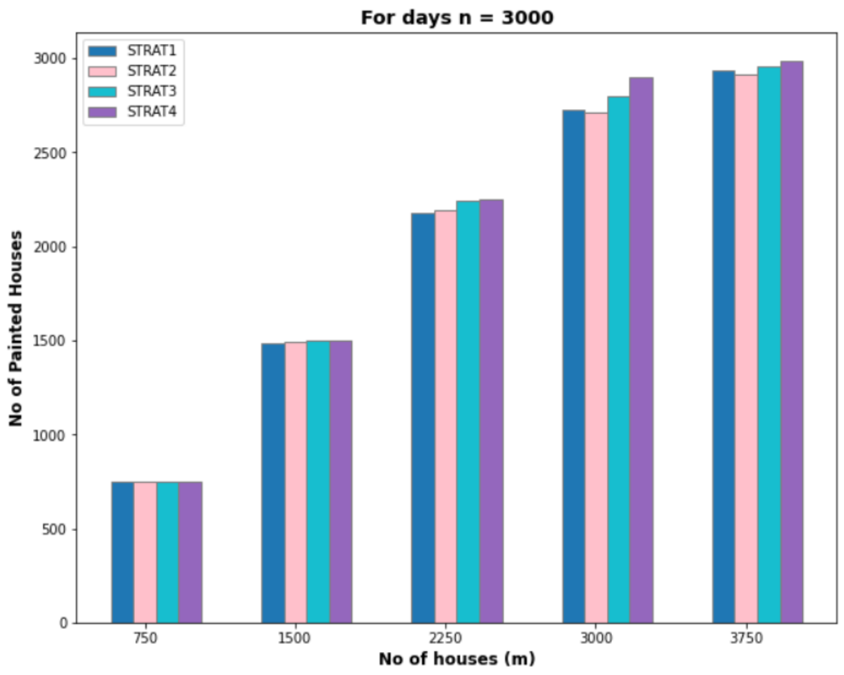

Figure 3: *No of Houses(m) Vs No of Painted Houses For n=3000*

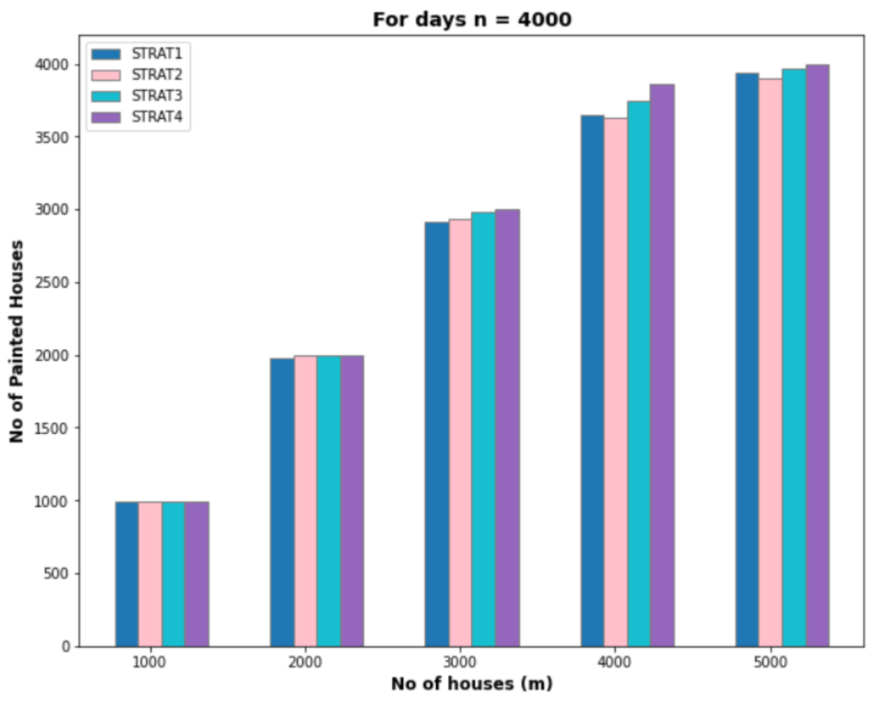

Figure 4: *No of Houses(m) Vs No of Painted Houses For n=4000*

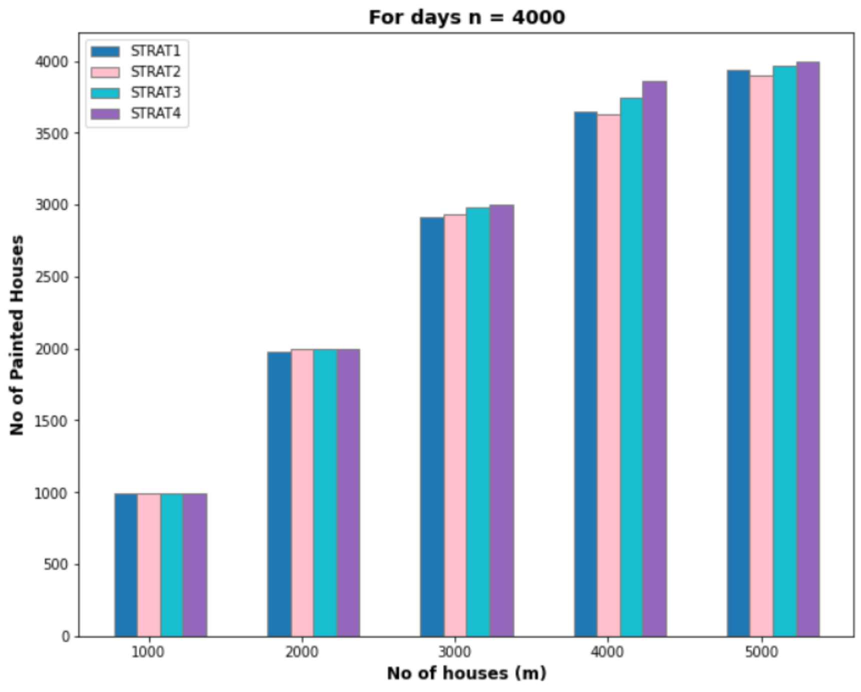

Figure 5: *No of Houses(m) Vs No of Painted Houses For n=5000*

To summarize above finding, we created datasets for n =1000 to 5000 and ran different strategies on those datasets for n = m. The results can be seen in the following figure 6, that strategy 4 schedules the maximum number of houses for painting in each set of inputs. No clear relationship between the other strategies is found but the results of this extended study on a larger dataset support our hypothesis that strategy 4 indeed is an optimal strategy.

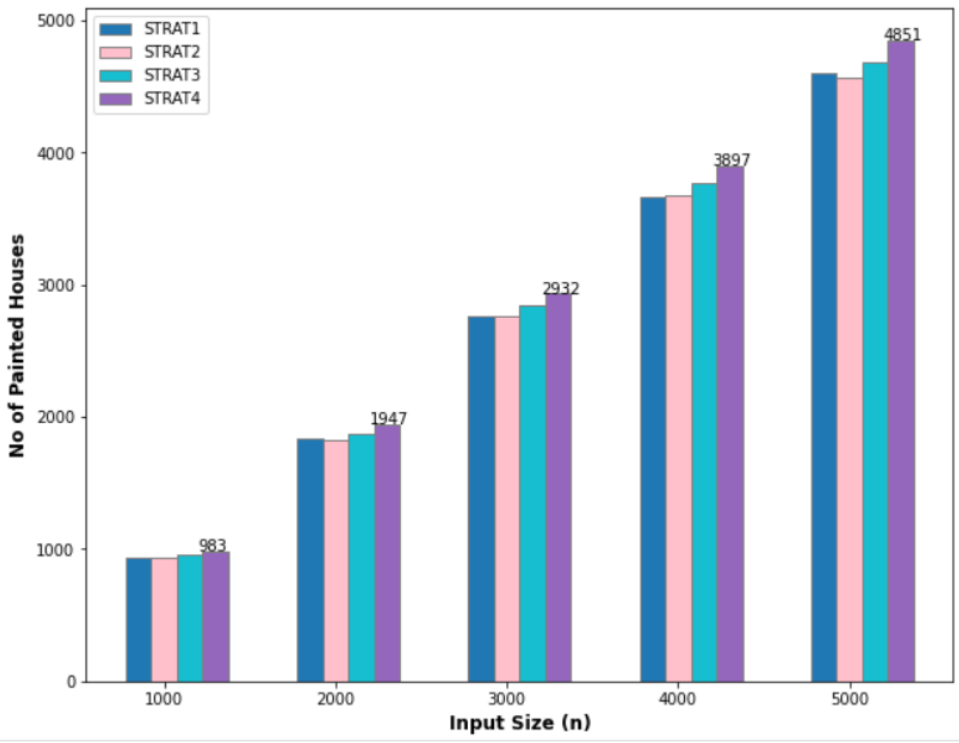

Figure 6: *Grouped Histogram (No of Houses Vs Input Size)*

Figure 7 represents the running time of each strategy for input size n = 1000 to 5000. Figure 2 shows that Strat1 with time complexity **Θ**(n) requires less time than the other 3 strategies with a time complexity of **Θ**(n + mlogm). Also, Strat4 is a time efficient implementation compared to Strat2 and Strat3.

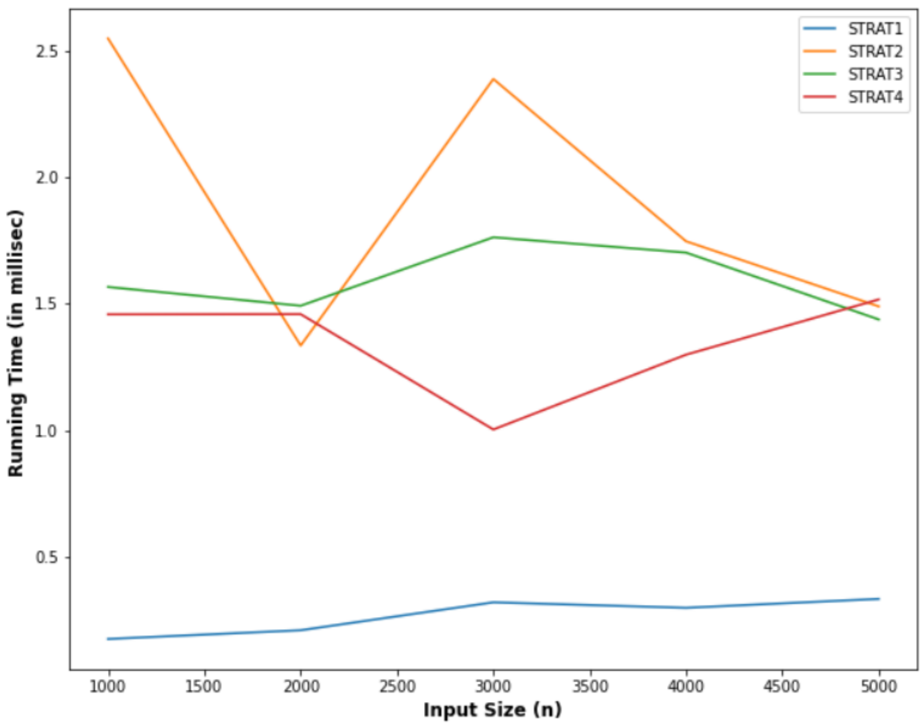

Figure 7: *Running Time Vs Input Size*

7. OPTIMAL SOLUTION

Design and analyze (time, space and correctness) of an Θ(mlog(m)) algorithm based on the greedy strategy

7.1 DESIGN

1. The difference between strat4 and the optimal strategy is the O(n) time we spend trying to find if we can schedule a house on that particular day.
2. In Optimal Solution we make jumps between houses rather than days and find all the houses which are eligible to be painted on the same startDay as the current house.
3. Upon arriving at house j with a different startDay as the current house i, we take the difference of the $\text{startDay}_j - \text{startDay}_i$. The difference is the number of days we have between where houses can be scheduled for painting.
4. Then we poll difference number of houses until the priority queue is empty or days are all filled by houses.
5. Starting from the startDay of the last house we poll until we can either fill all the days with compatible houses with valid deadlines or we exhaust our priority queue.
6. Once all the houses are exhausted, we poll until all the remaining days are filled

7.2 PSEUDO CODE

```
EARLIEST-ENDTIME-FIRST-OPTIMAL ( n, $\text{startDay}_1$ , $\text{endDay}_1$ , .... $\text{startDay}_m$, $\text{endDay}_m$ )

schedule ← $\phi$
// Create a priorityQueue which polls based on the earliest deadline ie (endDay)
priorityQueue pq ← $\phi$
curr_house_num ← 1
curr_startDay ← startDay of first house
```

```
    //Loop till you go through all the houses or arrive at a house with startDay>n
WHILE ( curr_house_num <= m  and curr_startDay <= n)

        //ADD all the houses whose startDay is same as curr_startDay to PQ
        IF  ( curr_house_num < m  and startDay_{curr_house_num} == curr_startDay)
            add house curr_house_num to pq with earliest deadline as priority

        ELSE
            //we encountered a house with a different startDay than the ones we found till now
                // we schedule houses for all the days between startDay_{curr_house_num} (included) and curr_startDay and store count in diff

            diff ← curr_startDay - startDay_{curr_house_num}
                //We can only poll houses with a deadline > curr_startDay. Rest are not schedulable. We store it in a variable
            endDayOfEligibleHouses ← curr_startDay

                WHILE (pq is not empty and diff > 0)
                // Poll the house from priority queue
                House h' ← pq.poll()
                // Polling ensures removing tasks that won't be compatible with current and any forthcoming day
                // We can only schedule houses whose endDay >= endDayOfEligibleHouses
                IF ( endDay_{house_polled} >= endDayOfEligibleHouses)
                    schedule house h'
                    diff ← diff – 1
                    endDayOfEligibleHouses ← endDayOfEligibleHouses + 1

            curr_startDay ← startDay_{curr_house_num}
```

7.3 ANALYSIS

- The above algorithm would one by one pick a house and find all the eligible houses which are eligible to be painted on the same day.
- It then adds them to a min heap, and picks the house with the lowest deadline.
- Since we don't fret over what day it would be scheduled, we decide it by the gap between two consecutive houses.
- Adding houses to a min heap cost Log(m) and since m houses at max would be added, the time taken by the algorithm would be O(m*log(m))
- Similarly while polling at max m houses will be polled from the pq and therefore O(m*log(m)) is the polling time complexity.

- Overall the time complexity si O(2*m*log(m)) which is O(m*log(m))
- We only maintain a list of houses scheduled therefore the space complexity is O(m).
- In the best case the heap size will always be 1,i.e only open ended houses are present in our dataset. Which certainly brings down the overall time but the complexity would still be Θ(m log(m)) as we have

Proof of Correctness -

Loop Invariant **:**
During every iteration of the while loop, we hop from house to house.We add houses with the same startDay in the PQ. If we encounter a house of different startDay than previous houses, we try to schedule the houses on the days represented by the gap between startDays of the two houses. In every iteration, we atleast add a house to PQ or schedule a few.

Initialization **:**
We start with house number 1 and keep on moving to the next one. Our currDay is the startDay of the first house, then it changes if we are able to schedule a house on the currDay.

Maintenance **:**
At every loop we move to the next house, or we schedule the previous houses. Which means either newer houses are added to the PQ or houses will be polled from the PQ, during each iteration.

Termination **:**
The program will terminate when we go through all the houses. At the end the **optimalSet** would give the indices of houses that are scheduled.

7.4 EXPERIMENTAL STUDY

- We analyzed the run time of both the algorithms, and observed that for sufficiently larger input size i.e n, we get 2 times the efficiency as compared to the Strat4 algorithm.
- We plot a line graph for n=m when n varies from 1000-10000 and then a plot when n varies from 10^5 to 10^6

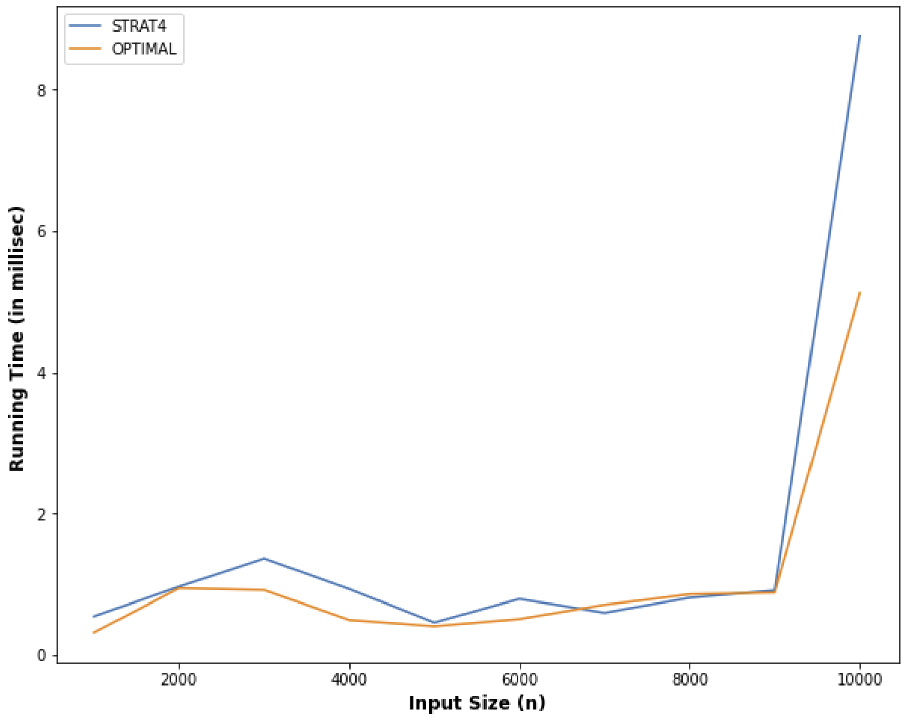

Figure 8: *Input Size n (1000 to 10000) Vs Running Time*

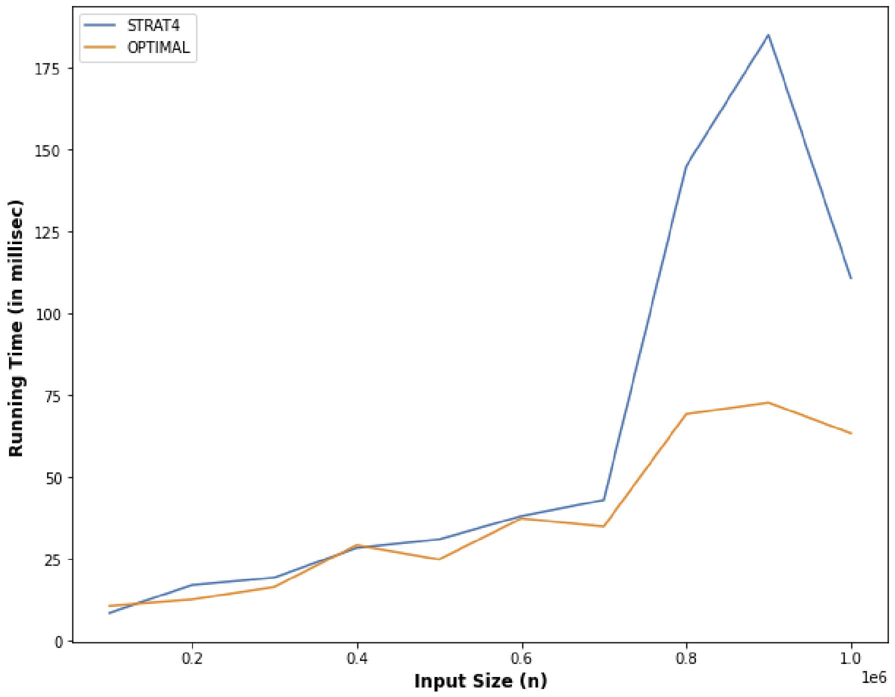

Figure 9: *Input Size n (1 lakh to 10 lakh) Vs Running Time*

- As the input size increases, the difference in the time taken to run by the algorithms widens and conclude that our optimal algorithm performs better as compared to algorithm described in strat4.

- In order to further analyze the time difference, we plot a graph where we keep m as constant and we change our input sizes, so that both the algorithms now have increasingly higher search space for similar amounts of houses.

- Since both the algorithms output the same houses, the optimal algorithm is able to navigate the search space better and schedule the jobs as it only goes through the constant number of houses every time regardless of the size of n.

- On the other hand, the strat4 algorithm loops through the n houses, which makes it more time bound on n and hence loses time as n increases.

- This trend can be seen from the following plots figure 10 to 13 where we keep m as constant and change n on the x axis.

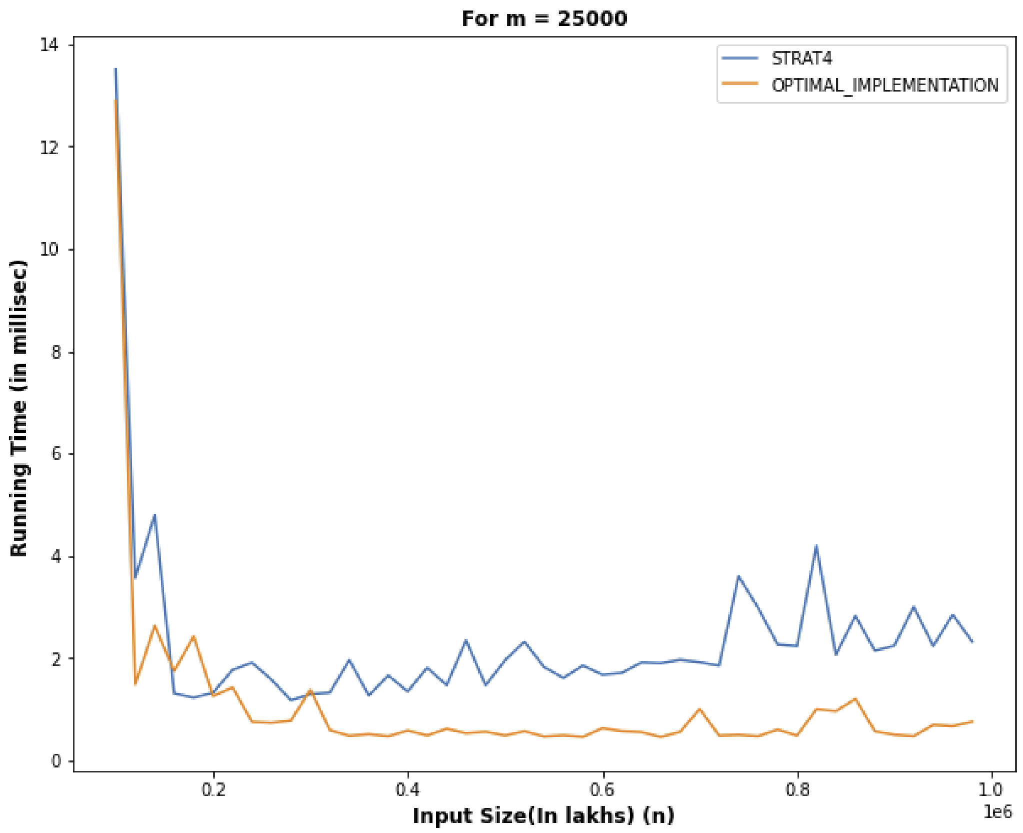

Figure 10*: Input Size n (1 lakh to 10 lakh) Vs Running Time (For m=25 k)*

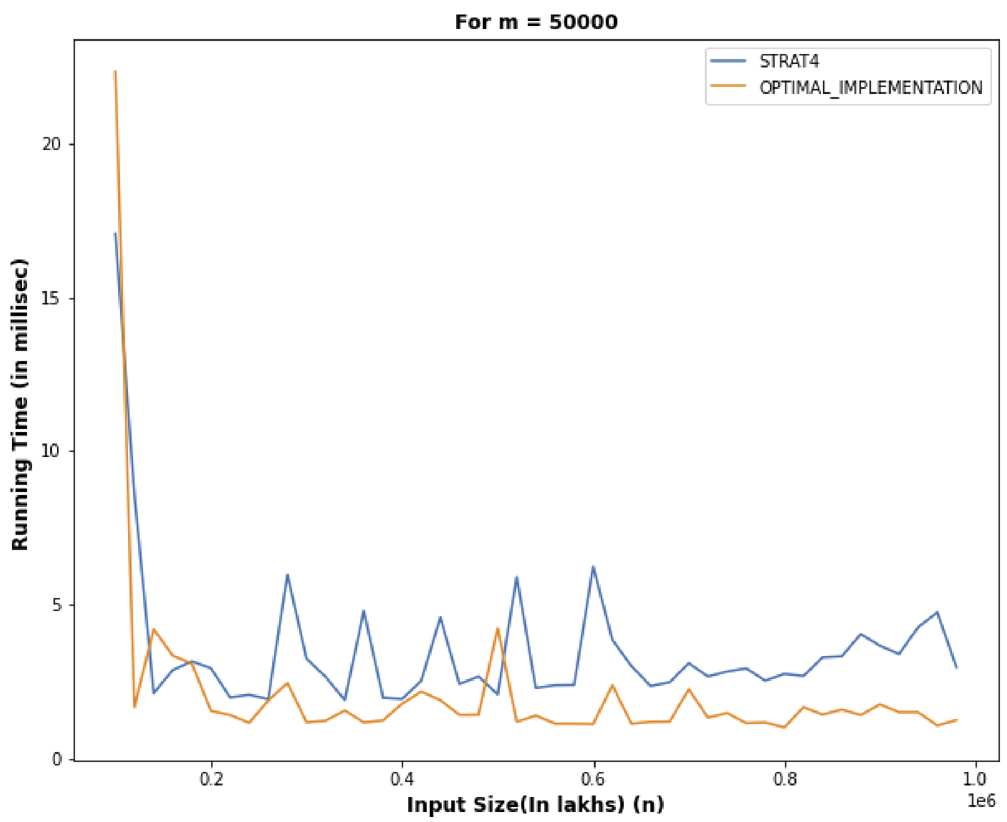

Figure 11: *Input Size n (1 lakh to 10 lakh) Vs Running Time (For m=50 k)*

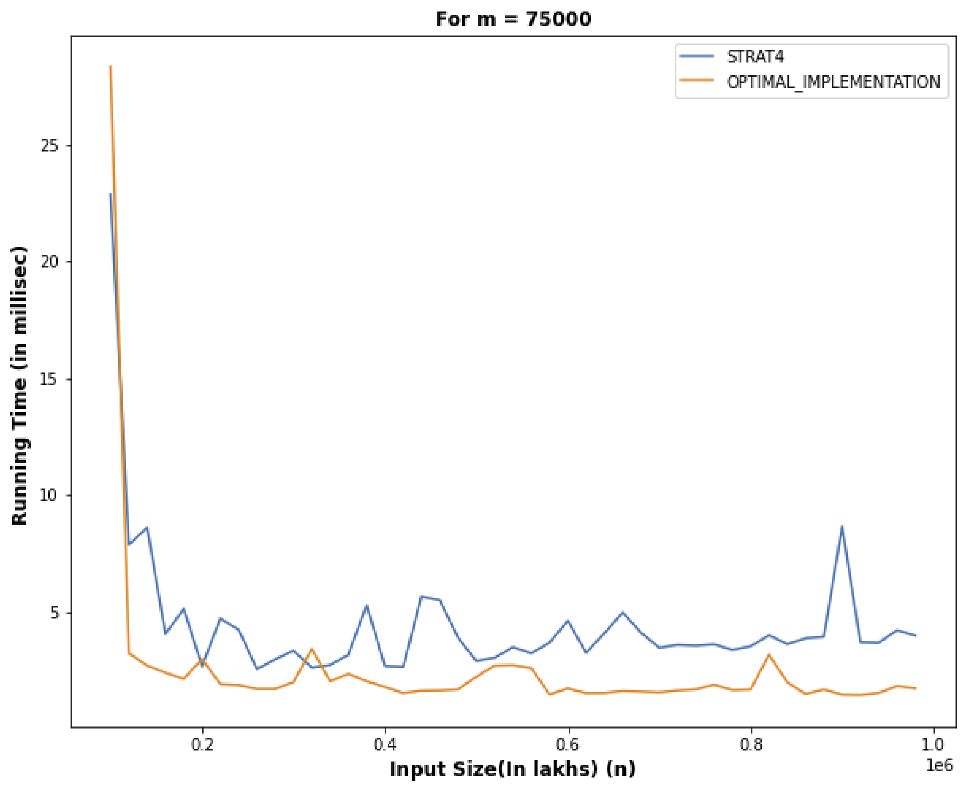

Figure 12: *Input Size n (1 lakh to 10 lakh) Vs Running Time (For m=75 k)*

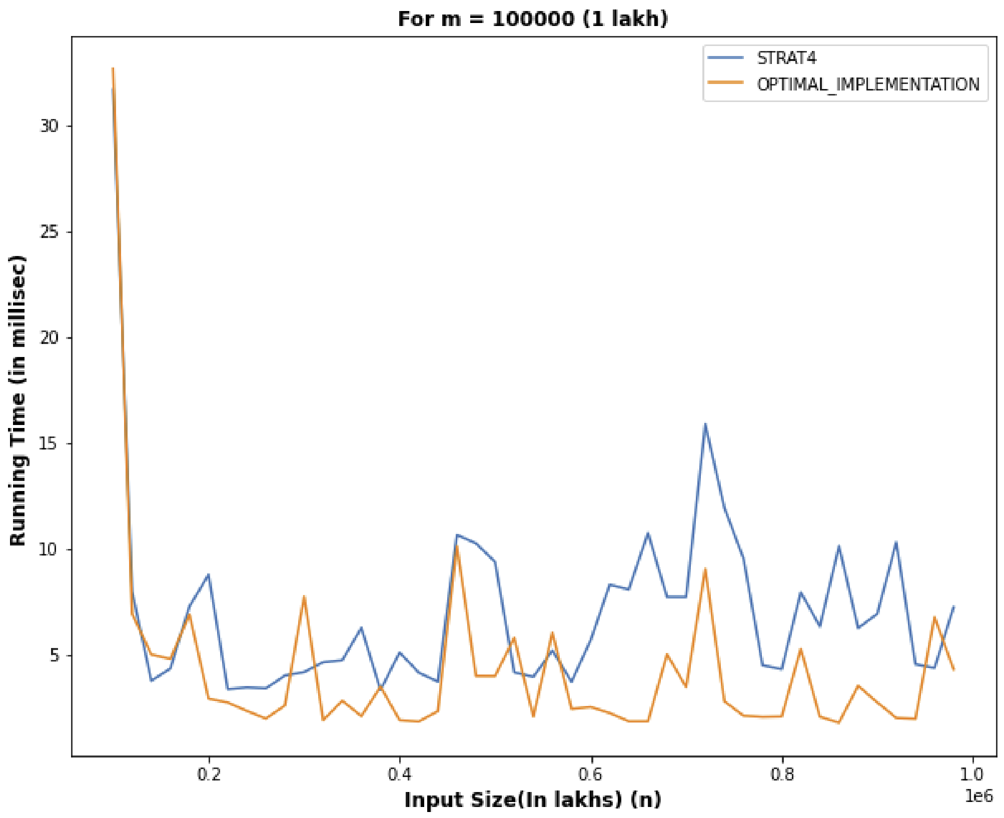

Figure 13: *Input Size n (1 lakh to 10 lakh) Vs Running Time (For m = 1lakh)*

8. CONCLUSION

- The project assignment helped us in understanding the greedy algorithms in depth. We were able to grasp on how different greedy choices can lead to different solutions.
- The need for a counterexample for each strategy enabled us to brainstorm on scenarios where each of the greedy approaches would fail.
- This helped us further in identifying the optimal strategy as our greedy choices later on revealed the flaws in the greedy design.
- We were able to visualize using the histograms that strat4 was actually the optimal strategy as it painted the most number of tasks for different input datasets.
- Programming assignments enabled us to implement the ideas into code.

Ease of Implementation **:**

- Implementing the programming task 1 was pretty straight-forward as the already sorted input handled the priority, so we didn't need to do much to establish the priority of the houses.
- Implementing the programming task 2 was tricky as we needed to find a way to understand all the eligible houses for the current day in the loop. We figured out a way to devise a moving priority queue and implemented it using java to keep optimal case working complexity.
- Once Strat2 was defined we had the structure for the other strategies as we just changed the way we make our heap in priority queue. The Implementations for strat3 and strat4 were trivial after strat2.
- There were other challenges of how start date are handled and accommodating edge cases in our algorithm.
- The trickiest part of the project was the optimal solution where we had to somehow make our algorithm dependent on just the number of houses and independent of the days.
- We figured out different ways to do so but none of them were as optimal as the one mentioned in the next section.
- Overall, a great learning exercise.