COT5405 - Algorithms Programming Project I

5.1 Team Members:

Shashank Reddy Boyapally, UFID: 1923-9618
Sri Maruti Keerthana Ponnuru, UFID: 3394-5082

Contributions:
Shashank Reddy Boyapally:
Brainstormed on designing the supporting and counter examples
Code design and Analysis of start 1 and 4
strat1 and strat4 code implementations,
Experimental Study of strat 1-4 and Graphs,
Bonus strat5, proof of correctness, Experimental Study.

Sri Maruti Keerthana Ponnuru:
Brainstormed on designing the supporting and counter example
Code design and analysis of strat 2 and 3
strat2 and strat3 code implementations
Worked on the proof of correctness for strat4
Formatted and wrote the report as per the given template
Bonus strat5 strategy design analysis and code implementation

5.2 Greedy Strategies:

For implementing all the four strategies we maintained a similar code structure by defining a class called House. There are three attributes common to all the four strategies and their implementations which are ‘startdate’, ‘enddate’ and ‘painted’, an additional attribute used for the third strategy is ‘duration’. A few terms that frequently appear in the algorithm are explained below.
class attributes:

startdate : gives the start day for a house on which it can be painted
enddate : gives the end day for a house on which it can be painted (inclusive)
painted : boolean variable, False if house not yet painted and True if painted
duration : given by the difference between the enddate and startdate
n : number of days the painter is available (1 … n)
m : number of houses
houses : a list of tuples representing the start and end dates of each house
Q : priority queue
li : list of House Objects
res : list of indices of painted houses

In the code we imported the PriorityQueue from the queue module, in order to maintain a priority queue that contains the unpainted houses that are available to be painted on the current day while iterating over each day.

Strategy 1: Paint the house that started being available the earliest

Algorithm:

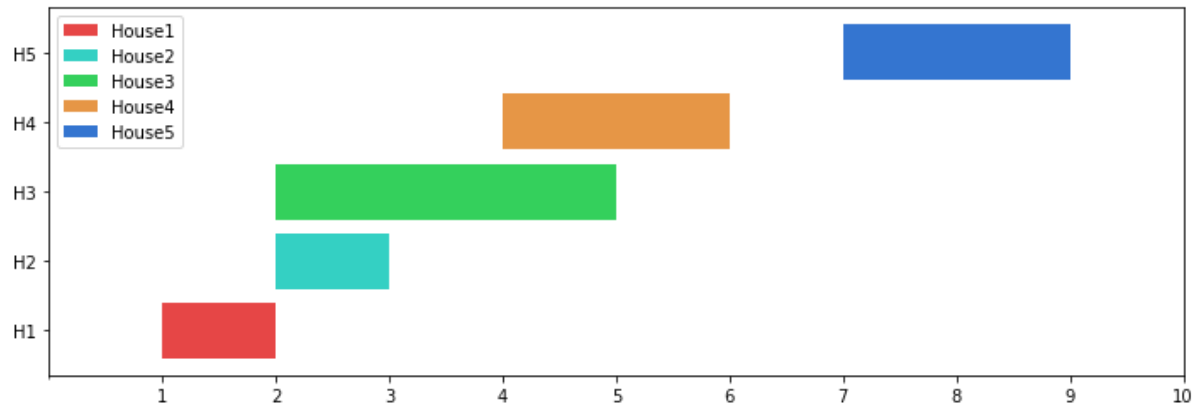
In the algorithm implemented we see that-

1. There are majorly 2 arguments n, m. n represents the number of days the painter is working and m represents the houses. Each house has its own start and end date of availability.
2. The approach we use in the strategy is to just iterate through the sorted list of houses which is given and select the house which has the earliest start date.
3. Since the houses are sorted primarily with startdate and secondarily with enddate, there is no need to use a priority queue to get the earliest available house.
4. We take the earliest available house and choose it on day c.
5. We add this house to res in order to keep track, also we change the painted to True, and jump to the next day
6. Since we are iterating through the list with n and jumping when required, this increases our efficiency. Though the correctness of the algorithm is doubted and is discussed in the further steps.

Analysis:

The idea of the algorithm is just to iterate through the list of houses but performing jumps, this shows we really iterate n elements. The algorithm gives us a time complexity of $\Theta(n)$ where n is the number of days the painter is available. We add the resultant elements to a res variable, the major time complexity is of $\Theta(n)$ which is only given by the iteration of the loop.

Example:

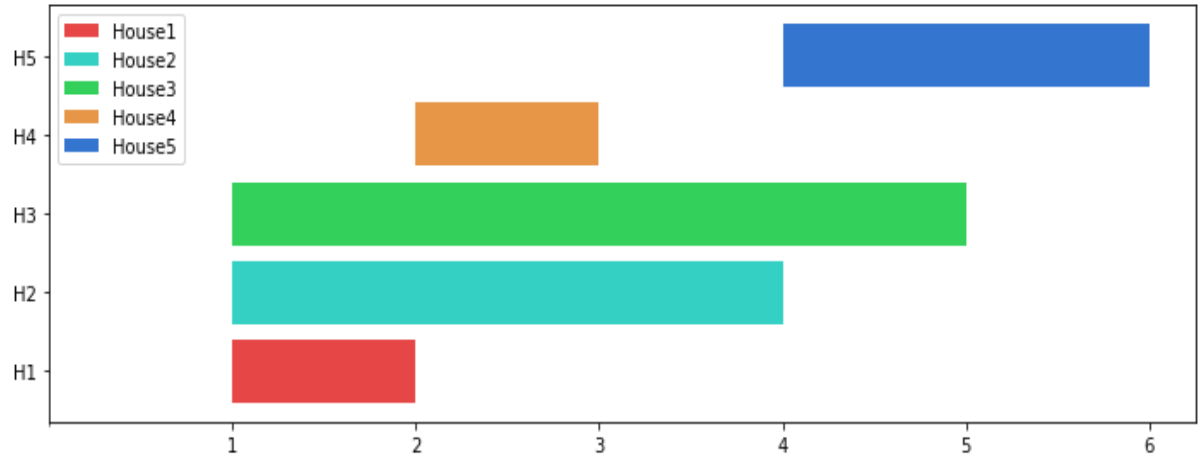


In the above mentioned example,

n=10 m=5 and houses=[(1,2),(2,3),(2,5),(4,6),(7,9)]
The algorithm gives an optimal solution, all 5 houses are painted which is the maximum number of houses that can be painted for the given example. Explanation is given as follows:
Day 1: House 1 is painted
Day 2: House 2 is painted

Day 3: House 3 is painted
Day 4: House 4 is painted
Day 7: House 5 is painted

Counter Example:



In the above mentioned example, n=6 m=5 and houses=[(1,2),(1,4),(1,5),(2,3),(4,6)], the algorithm does not give an optimal solution. The maximum number of houses painted is only 4 whereas the optimal gives 5 as the result. Explanation is given as follows:
Day 1: House 1 is painted
Day 2: House 2 is painted
Day 3: House 3 is painted
Day 4: House 5 is painted

Whereas the optimal solution would be:
Day 1: House 1 is painted
Day 2: House 4 is painted
Day 3: House 2 is painted
Day 4: House 3 is painted
Day 5: House 5 is painted

Strategy 2: Paint the house that started being available the latest

Algorithm:

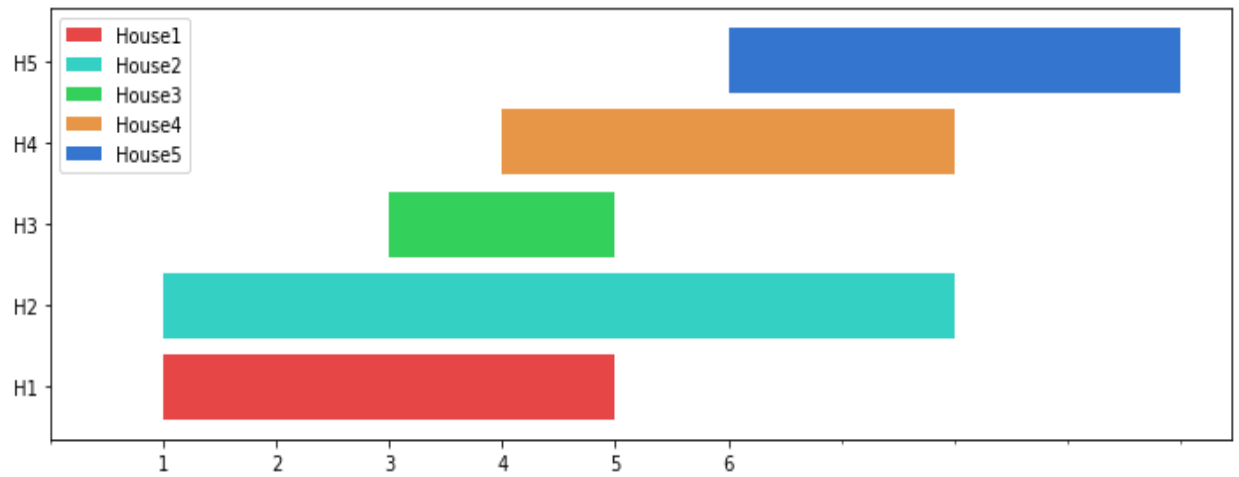
In the algorithm implemented,

1. The function takes 2 arguments: n and houses
    a.
2. __lt__ method is defined for the priority queue to determine the ordering of the Houses, in order to compare any 2 objects of the class based on their start date. This is done to ensure that while iterating over each day the 'Priority Queue' keeps track of houses in the decreasing order of start date since we need to paint the house that started being available the latest.
3. Initially given the input list of houses, the method first generates a list 'li' of House objects. Then it initializes an empty priority queue 'Q' followed by an empty list 'res' to store indices of the painted homes.
4. We iterate for each day starting from 1 up to n, and first check the while condition that the Q is not empty and also take out any houses with end dates before the current day. Following this it adds to the priority queue the houses which are unpainted and are available on the current day.
5. We check if the priority queue is not empty such that if there are unpainted houses available on the current day, then it paints the one with the latest start date and the corresponding index is appended to 'res'

Analysis:

The algorithm maintains a priority queue of unpainted houses that are available to be painted on the current day. It iterates over each day and adds any available houses to the queue which are unpainted. It pops that house with the latest start date and assigns it True as painted. The time complexity of this algorithm is $\Theta(n + m \log(m))$, where n is the number of days and m is the number of houses in the priority queue; because this algorithm iterates over n days and uses a priority queue, which has a log(m) time complexity for each insertion and removal operation, it has an overall running time complexity of $\Theta(n + m \log(m))$.

Supporting Example:



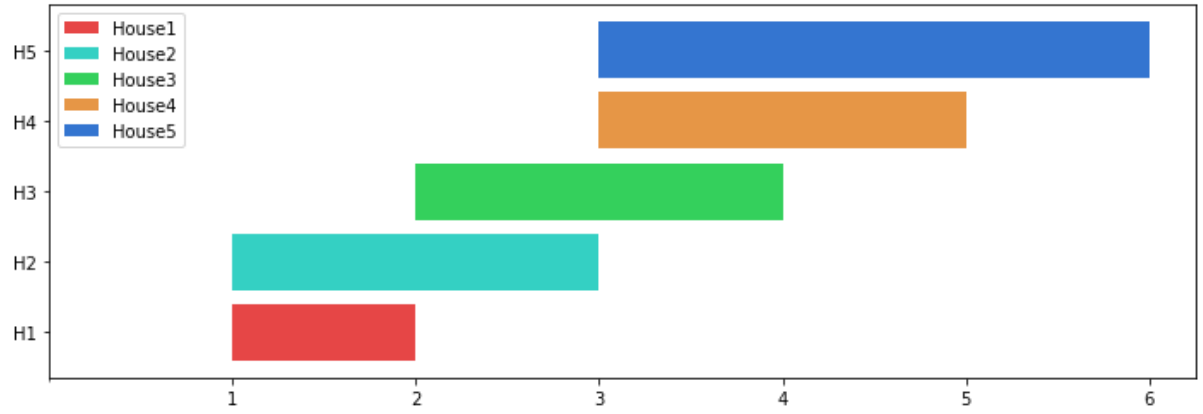
In the above mentioned example,

n=6 m=5 and houses=[(1,5),(1,8),(3,5),(4,8),(6,10)]

The algorithm gives an optimal solution, all 5 houses are painted which is the maximum number of houses that can be painted for the given example. Explanation is given as follows:

Day 1: House 1 is painted
Day 2: House 2 is painted
Day 3: House 3 is painted
Day 4: House 4 is painted
Day 5: House 5 is painted

Counter Example:



In the above mentioned example, n=6 m=5 and houses=[(1,2),(1,3),(2,4),(3,5),(3,6)], the algorithm does not give an optimal solution. The maximum number of houses painted is only 4 whereas the optimal gives 5 as the result. Explanation is given as follows:
Day 1: House 1 is painted
Day 2: House 3 is painted
Day 3: House 4 is painted
Day 4: House 5 is painted

Whereas the optimal solution would be: [0, 1, 2, 3, 4]
Day 1: House 1 is painted
Day 2: House 2 is painted
Day 3: House 3 is painted
Day 4: House 4 is painted
Day 5: House 5 is painted

Strategy 3: Paint the house that is available for the shortest duration.

Algorithm:

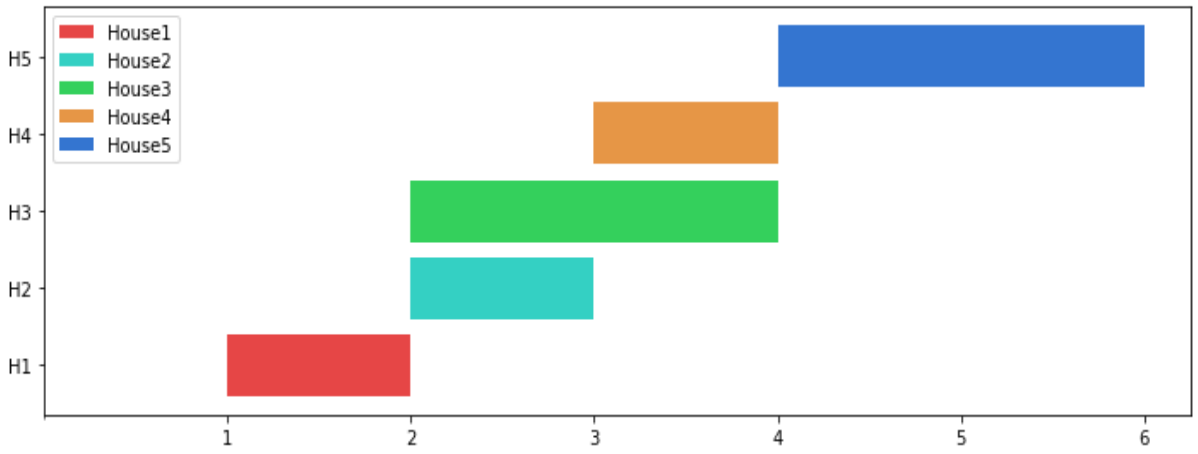
In the algorithm implemented,

1. The function takes 2 arguments: n and houses
2. __lt__ method is defined for the priority queue to determine the ordering of the Houses, in order to compare any 2 objects of the class based on their duration. This is done to ensure that while iterating over each day the 'Priority Queue' keeps track of houses in the increasing order of duration since we need to paint the house that is available for the shortest duration.
3. Initially given the input list of houses, the method first generates a list 'li' of House objects. Then it initializes an empty priority queue 'Q' followed by an empty list 'res' to store indices of the painted homes.
4. We iterate for each day starting from 1 up to n, and first check the while condition that the Q is not empty and also take out any houses with end dates before the current day. Following this it adds to the priority queue the houses which are unpainted and are available on the current day.
5. The for loop iterates and adds the houses available at the current day c to the priority Q, next check if the priority queue is not empty such that if there are unpainted houses available on the current day, then it paints the one with the shortest duration and the corresponding index is appended to 'res'

Analysis:

The algorithm maintains a priority queue of unpainted houses that are available to be painted on the current day. It iterates over each day and adds any available houses to the queue which are unpainted. It pops that house with the shortest duration and assigns it True as painted. The time complexity of this algorithm is $\Theta(n + m \log(m))$, where n is the number of days and m is the number of houses in the priority queue; because this algorithm iterates over n days and uses a priority queue, which has a log(m) time complexity for each insertion and removal operation, it has an overall running time complexity of $\Theta(n + m \log(m))$.

Example:

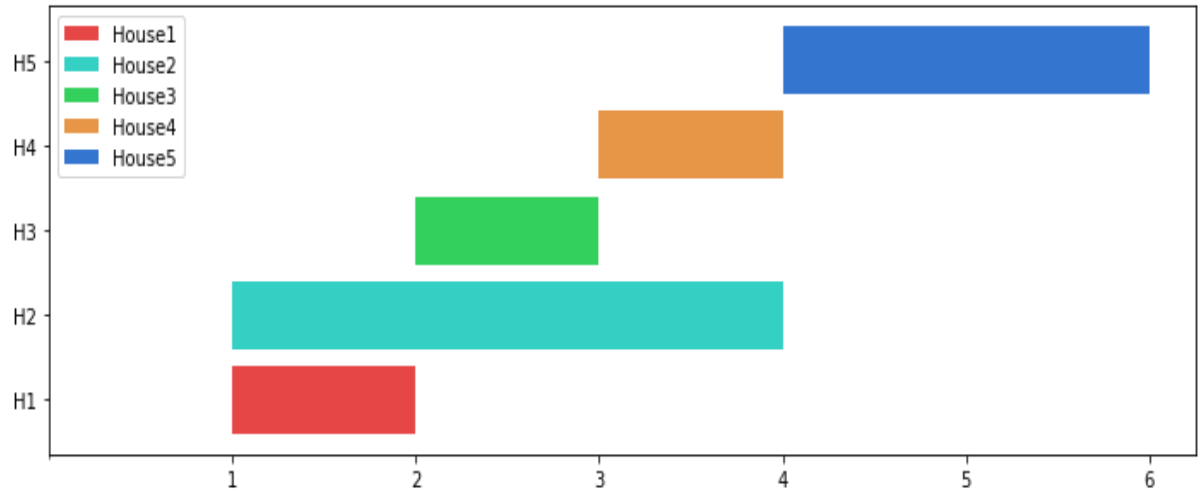


Example: In the above mentioned example

m=5, n=7 and houses =[(1,2),(2,3),(2,4),(3,4),(4,6)]

The algorithm gives an optimal solution. The maximum number of houses painted is 5 which is also the optimal result for the given example. Explanation is given as follows:
Day 1: House 1 is painted
Day 2: House 2 is painted
Day 3: House 4 is painted
Day 4: House 3 is painted
Day 5: House 5 is painted

Counter Example:



In the above mentioned example, n=6 m=5 and houses=h=[(1,2),(1,4),(2,3),(3,4),(4,6)], the algorithm does not give an optimal solution. The maximum number of houses painted is only 4 whereas the optimal gives 5 as the result. Explanation is given as follows:
Day 1: House 1 is painted
Day 2: House 3 is painted
Day 3: House 4 is painted
Day 4: House 5 is painted

Whereas the optimal solution would be: [0, 2, 3, 1, 4]

Day 1: House 1 is painted
Day 2: House 3 is painted
Day 3: House 4 is painted
Day 4: House 2 is painted
Day 5: House 5 is painted

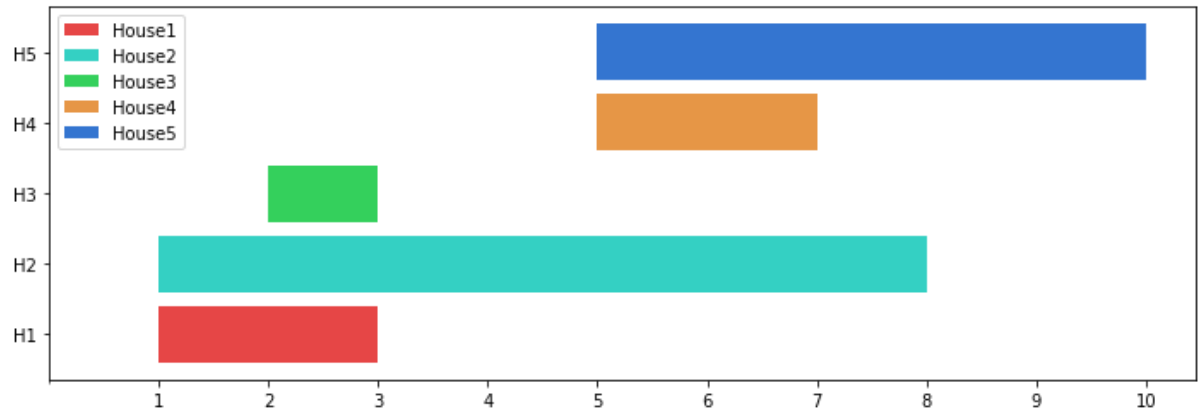
Strategy 4: Paint the house that that will stop being available the earliest

Algorithm:

In the algorithm implemented,

1. The function takes 2 arguments: n and houses
2. __lt__ method is defined for the priority queue to determine the ordering of the Houses, in order to compare any 2 objects of the class based on their duration. This is done to ensure that while iterating over each day the 'Priority Queue' keeps track of houses in the increasing order of end dates since we need to paint the house that will stop being available the earliest.
3. Initially given the input list of houses, the method first generates a list 'li' of House objects. Then it initializes an empty priority queue 'Q' followed by an empty list 'res' to store indices of the painted homes.
4. We iterate for each day starting from 1 up to n, and first check the while condition that the Q is not empty and also take out any houses with end dates before the current day. Following this it adds to the priority queue the houses which are unpainted and are available on the current day.
5. The for loop iterates and adds the houses available at the current day c to the priority Q, next check if the priority queue is not empty such that if there are unpainted houses available on the current day, then it paints the one that will stop being available the earliest and the corresponding index is appended to 'res'

Example:



In the above mentioned example
m=5, n=10 and houses =[(1,3),(1,8),(2,3),(5,7),(5,10)]

The algorithm gives an optimal solution. The maximum number of houses painted is 5 which is also the optimal result for the given example. Explanation is given as follows:
Day 1: House 1 is painted
Day 2: House 3 is painted
Day 3: House 2 is painted
Day 5: House 4 is painted
Day 6: House 5 is painted

Analysis: The algorithm maintains a priority queue of unpainted houses that are available to be painted on the current day. It iterates over each day and adds any available houses to the queue which are unpainted. It pops that house that will stop being available the earliest and assigns it True as painted. The time complexity of this algorithm is $\Theta(n + m \log(m))$, where n is the number of days and m is the number of houses in the priority queue; because this algorithm iterates over n days and uses a priority queue, which has a log(m) time complexity for each insertion and removal operation, it has an overall running time complexity of $\Theta(n + m \log(m))$.

Correctness Proof:

The solution is optimal and we need to argue about it. Let us consider O to be the optimal set of intervals. One might want to show that A is equal to a single one of them, but there may be many optimal solutions and A is not equal to a single one.

We will show that A has the same number of intervals as O and that is an optimal solution. The idea of the proof is to find a sense in which our greedy algorithm stays ahead of the solution.

We will compare the partial solutions that the greedy algorithm constructs to initial segments of the solution O, and show that the greedy algorithm is doing better in a step-by-step fashion. This proof is helped by some notation. Let i1,..., ik be the set of requests in A in the order they were added to A. Note that |A| = k.

Similarly, let the set of requests in O be denoted by j1,..., jm. The requests in O are ordered in the order of the start and finish points. The start points have the same order as the finish points, because the requests in O are compatible.
For a given day c after satisfying the first request, we wanted our resource to be free again as soon as possible. And indeed, our greedy rule guarantees that the finish time of $i_1 \leq$ finish time of j_1. The sense we want to show is that our greedy rule stays ahead, that each interval finishes as soon as the corresponding interval in the set O. The r^{th} request in the optimal schedule finishes no later than the r^{th} request in the algorithms schedule. We have finish time of i_r and finish time of j_r for all indices.

Proof.
We will prove this statement by contradiction similar to an extremality proof.

If A is not optimal, then an optimal set O has m set of houses painted,Assuming m and k are potentially different

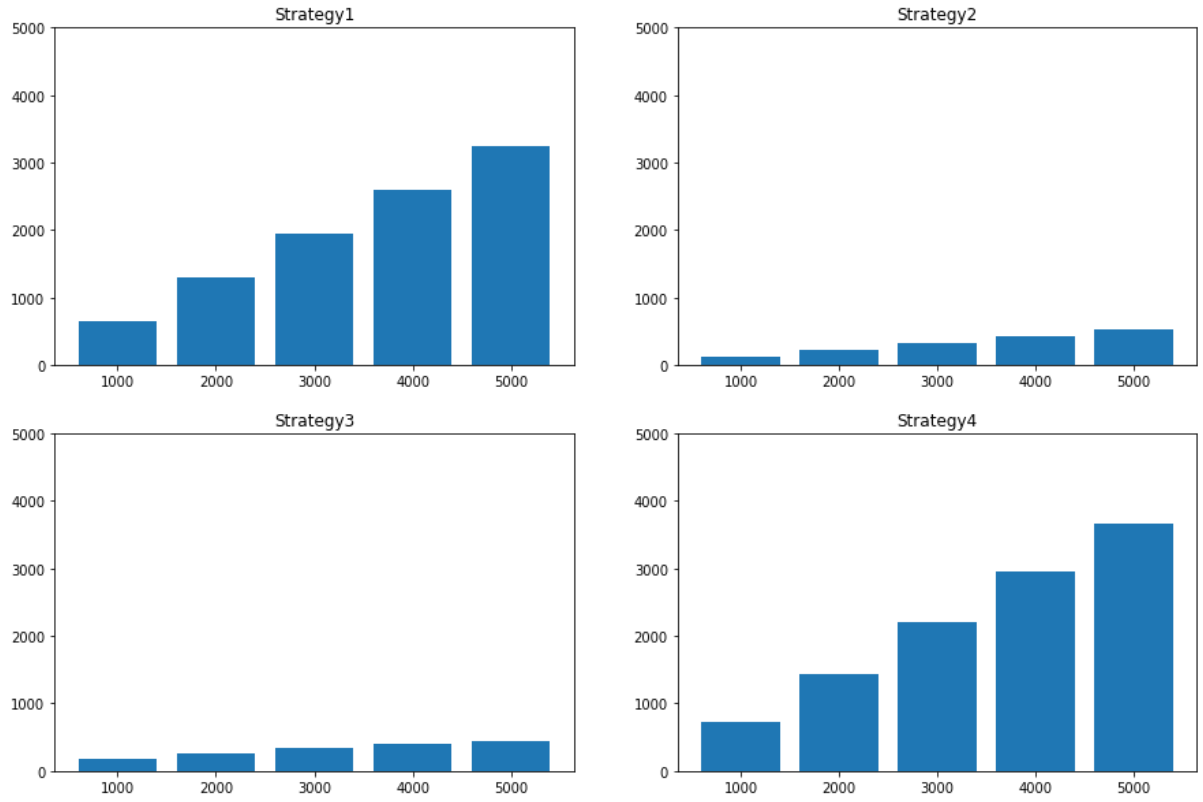
Consider h1, h2, … hk set of houses generated by our greedy approach A
Let $j_1,j_2,...j_m$ denote a set of jobs generated by optimal solution O.

Up to a certain point jobs selected could be the same, basically we are picking such that it has maximum similarity in the jobs generated by A, so for the largest possible

value of r for each day, we get $h1=j_1$, $h2=j_2$, … $hr=j_r$. Now we have two feasible solutions A and O.

The finish time of the job i_{r+1} must be less than or equal to the finish time of j_{r+1} for a given day c, up until r we have picked same set of houses and for the remaining our greedy approach chooses the one with that will stop being available the earliest i.e. the one with earliest end date among the houses available on that day where as the optimal solution O chooses something else based on the greedy approach followed by it but it cannot be earlier than than the end date of i_{r+1}. Now if we replace i_{r+1} with j_{r+1}, causing no compatibility issue as when we replace j_{r+1} with i_{r+1} it will not be conflicting with j_{r+2} this gives us another solution giving Another optimum solution that has more similarity to A. Similarly we can keep doing this recursively until both solutions become the same.

5.3. Experimental Comparative Study



Graph 1: n vs no of houses painted grouped by strategy

The above graph, Graph 1 shows the optimality of each strategy we used with respect to the number of houses in the output which was given. The x axis represents the input size n, the y-axis represents the number of houses which were painted using the current strategy. The same dataset was used to test across all strategies so that equality can be maintained.

5.4 Conclusion

Based on the implementation and understanding of the experimental comparative study done some of the interesting observations which were made are listed as follows:

- Strat 1 and Strat 4 generated more number of houses painted when compared to the other strategies which are Strat 3 and Strat 2
- The number of houses painted increases with increase in input size parameter n such as the number of days which are used to paint have increased.
- Strat 4 gives us the most number of houses painted, reason being that it is the optimal approach as discussed in the above sections the correctness of this approach.
- Strat 4 is the optimal approach with a complexity of $\Theta(n+m\log m)$ it can be further optimised which is discussed in the further sections. The optimality is reflected in the graph for further reference.

Bonus

8.1 Algorithm Design

Intuition for the algorithm design :

A greedy rule that does lead to the optimal solution is based on strategy 4, to accept first the house that finishes first, that is, to paint the houses that will stop being available the earliest. The goal is to paint a maximum number of houses following the greedy approach, so to iterate over n is not the best solution. Let us consider a case where n ismuch larger than m , here after looping through m houses and the maximum houses have been painted iterating through the rest of the days till n serves no purpose, to avoid this and thus reduce the running time to Θ(m log(m)) we have designed the algorithm as follows.

Algorithm :

In the algorithm implemented,

1. The function takes 2 arguments: n and houses
2. __lt__ method is defined for the priority queue to determine the ordering of the Houses, in order to compare any 2 objects of the class based on their duration. This is done to ensure that while iterating over each day the 'Priority Queue' keeps track of houses in the increasing order of end dates since we need to paint the house that will stop being available the earliest.
3. Initially given the input list of houses, the method first generates a list 'li' of House objects. Then it initializes an empty priority queue 'Q' followed by an empty list 'res' to store indices of the painted homes.
4. We first check the while condition to keep a track that the number of houses painted is always less than or at most equal to m and also if the current day is within the number of days the painter is available and increment c up to m so this loop runs until m and not n, then we check if the Q is not empty and also take out any houses with end dates before the current day. Following this it adds to the priority queue the houses which are unpainted and are available on the current day.
5. The loop iterates and adds the houses available at the current day c to the priority Q this is done by considering a variable i which is used for assigning index value to each house in 'li', next we check if the priority queue is not empty such that if there are unpainted houses available on the current day, then it paints the one that will stop being available the earliest and the corresponding index is appended to 'res' We increment the current day based on the condition only if its less than or equal to m limiting the number of iteration to m rather than n.

Time Analysis:

It is evident here that the algorithm always paints the house that has the earliest end date among all available houses. The priority queue is always sorted by the end date of houses in ascending order to ensure this. The time complexity of the algorithm is Θ(m log(m)), which is the time complexity of using a priority queue to maintain the available houses as it has a

log(m) time complexity for each insertion and removal operation. In this case, the outermost loop runs at most m times rather than n like the previous case of strategy 4.

Space Analysis:

In the code we have maintained 2 lists ‘li’ and ‘res’ respectively and a Priority Queue. The list ‘li’ is of the size O(m) res at most has m elements and the priority queue has at most m House Objects, so the overall space complexity is O(m).

Correctness:

The logic followed is same as that of the strategy 4 but the implementation is different, in this case, the outermost loop runs at most m times rather than n like the previous case of strategy 4. Here after looping through m houses and the maximum houses have been painted iterating through the rest of the days till n when n is larger than m serves no purpose, to avoid this and thus reduce the running time we designed this approach, the correctness is in similar lines of strategy 4 since the core logic is the same for both of these algorithms.

The solution is optimal and we need to argue about it. Let O be an optimal set of intervals. One might want to show that A is equal to a single one of them, but there may be many optimal solutions and A is not equal to a single one. We will show that A has the same number of intervals as O and that is an optimal solution. The idea of the proof is to find a sense in which our greedy algorithm stays ahead of the solution.

We will compare the partial solutions that the greedy algorithm constructs to initial segments of the solution O, and show that the greedy algorithm is doing better in a step-by-step fashion. This proof is helped by some notation. Let i1,..., ik be the set of requests in A in the order they were added to A. Note that |A| = k. Similarly, let the set of requests in O be denoted by j1,..., jm. The requests in O are ordered in the order of the start and finish points. The start points have the same order as the finish points, because the requests in O are compatible.
For a given day c after satisfying the first request, we wanted our resource to be free again as soon as possible. And indeed, our greedy rule guarantees that $f(i1) \leq f(j1)$. The sense we want to show is that our greedy rule stays ahead, that each interval finishes as soon as the corresponding interval in the set O. The rth request in the optimal schedule finishes no later than the rth request in the algorithms schedule. We have f(ir) and f(jr) for all indices.

Proof.

We will prove the statement by contradiction similar to an extremality proof.

If A is not optimal, then an optimal set O has m set of houses painted,Assuming m and k are potentially different

Consider h1, h2, … hk set of houses generated by our greedy approach A
Let $j_1,j_2,...j_m$ denote a set of jobs generated by optimal solution O.

Up to a certain point jobs selected could be the same, basically we are picking such that it has maximum similarity in the jobs generated by A, so for the largest possible value of r for each day, we get $h1=j_1$, $h2=j_2$, … $hr=j_r$. Now we have two feasible solutions A and O.

The finish time of job i_{r+1} must be <= finish time of j_{r+1} for a given day c, up until r we have picked same set of houses and for the remaining our greedy approach chooses the one with that will stop being available the earliest i.e. the one with earliest end date among the houses available on that day where as the optimal solution O chooses something else based on the greedy approach followed by it but it cannot be earlier than than the end date of i_{r+1}. Now if we replace i_{r+1} with j_{r+1}, causing no compatibility issue as when we replace j_{r+1} with i_{r+1} it will not be conflicting with j_{r+2} this gives us another solution giving Another optimum solution that has more similarity to A. Similarly we can keep doing this recursively until both solutions become the same.

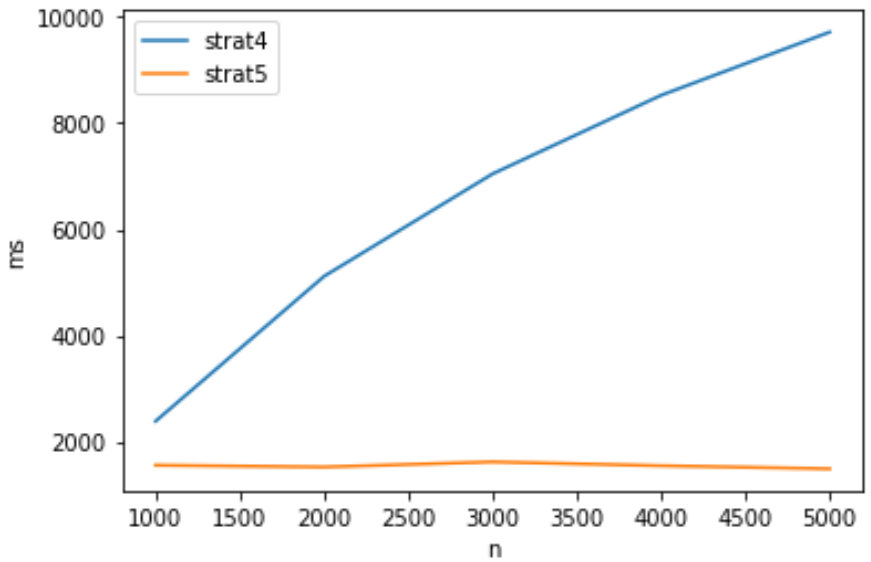
8.2 Programming Task

The code is given in the AOAbonus.py file which can be located in the zip file. It can be run by using the makefile with the command ‘make run5’.

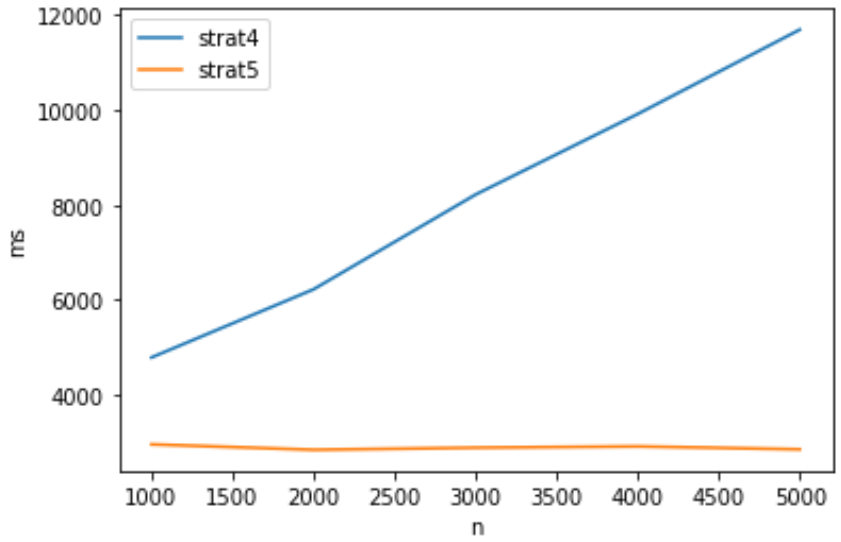
8.3 Experimental Study

The experimental study shows the increase in time of running of the algorithm with respect to increase in N. The two algorithms show expected results with the same dataset, and are aligned with the complexity of the algorithms.

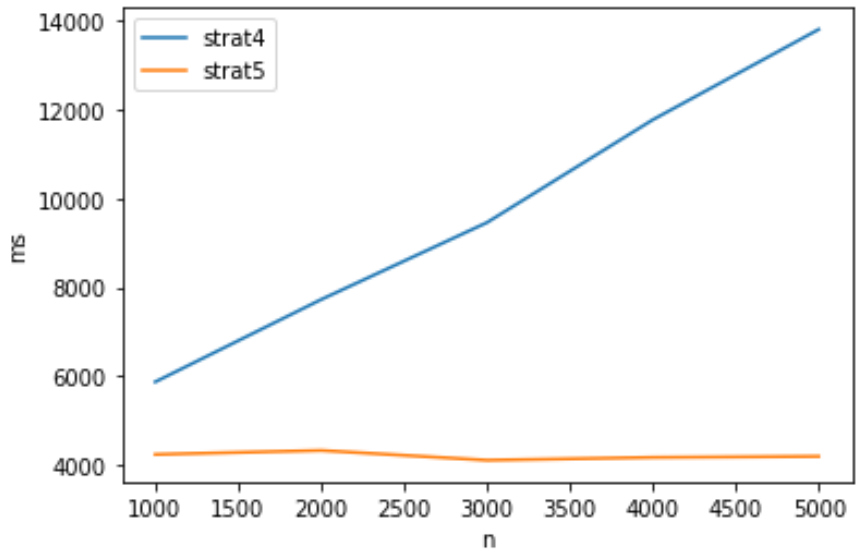
Given below are the graphs which are generated using matplotlib



Graph 2. N vs time in ms for m =200



Graph 3. N vs time in ms for m=300



Graph 4. N vs time in ms for m=400

We see that the time increases almost linearly as n increases this is due to the fact it runs on $\Theta(n + m \log(m))$ complexity, while strat5 stays the same as the complexity is $\Theta(m \log(m))$.