COT 5405 - Analysis of Algorithms

Programming Project 1

Group Members

Koushik Chanda - UFID 28719927

Adithya Koti Narasimhachar Guruachar - UFID 93422539

Spring Semester 2023

Team members

Koushik Chanda (UFID: 28719927)

- Analyzed on design and analysis of algorithms for TASK1, TASK2, TASK3, TASK4 and TASK 5.
- Implemented coding and tested the code of TASK1, TASK2 and TASK5.
- Created counter examples for TASK1 and TASK2.
- Created a random Input File to get experimental input data of size n = 1000, 10000, 25000, 50000, 10000 and verified correctness of Algorithm for TASK3, TASK4 and TASK 5.
- Worked on writing and reviewing the report.

Adithya KNG (UFID: 93422539)

- Analyzed on design and analysis of algorithms for TASK1, TASK2, TASK3 and TASK4 and TASK5.
- Implemented coding and tested the code of TASK3, TASK4 and TASK 5.
- Created counter example for TASK3 and proof of correctness for TASK4 and TASK5.
- Created a random Input File to get experimental input data of size n = 1000, 10000, 25000, 50000, 10000 and verified correctness of Algorithm for TASK1, TASK2.
- Worked on writing and reviewing the report.

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 $startDay_i$ and $endDay_i$ for i = 1, . . . , m. The house i can only be painted on a day between $startDay_i$ and $endDay_i$ (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.

Greedy Strategies

For each of the following greedy strategies, you must

i) Design a O(n + m log(m)) algorithm;

ii) Provide an instance where the strategy yields an optimal solution;

iii) Provide an instance where the strategy does not yield an optimal solution or prove that it is an optimal strategy. Your algorithm for Strat1 must have a Θ(n) running time. The remaining strategies must have a Θ(n + m log(m)) running time.

TASK1

Strat1: 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.

Algorithm: Θ (n+m) algorithm for solving TASK1.

Function paintHouses takes the number of days(n), number of houses(m) and startDay and endDay of m houses(input[][]) and prints the index of painted houses.
input[k][0] contains the startDay of k^{th} house and input[k][1] contains the endDay of k^{th} house.

```
function paintHouses(n, m, input[][]):
  currentDay <- 1
  currentHouseIndex <- 0

  WHILE currentDay <=n  AND  currentHouseIndex <m:
      IF currentDay lies between [startDay , endDay] of input[currentHouseIndex]:
            currentDay++
            currentHouseIndex++
            PRINT currentHouseIndex
      ELSE IF startDay of input[currentHouseIndex] > currentDay:
            currentDay = startDay of input[currentHouseIndex]
      ELSE:
            currentHouseIndex++
```

Analysis:

- **Time Complexity:** In this approach each house is iterated at most once, the while loop runs for min(n,m) times and the IF conditions take a constant amount of time. So, the overall time complexity is O(n+m) or precisely Θ(n+m).
- **Space Complexity:** There is no extra space used. Therefore, space complexity will be a constant which is O(1).
- **Counter Example:**
  Consider the following date ranges of houses with n=7 and m=7.
  A (1,2)
  B (2,3)
  C (3,4)
  D (3,5)
  E (3,6)
  F (3,10)
  G (4,5)

If we follow the algorithm of choosing to paint the house that started being available the earliest, we start with painting house A on day 1, house B on day 2, house C on day 3, house D on day 4, house E on day 5 and house F on day 6. As house G is not available on day 7, we cannot paint it. So, a total of 6 {A->B->C->D->E->F} houses are painted.

However, an optimal solution can be achieved by painting houses A to D each on a day, which takes 4 days in total. On the fifth day instead of painting E we can paint G, followed by E on sixth day and F on the seventh day.
So, we can paint all the seven houses in the order {A->B->C->D->G->E->F} order.

- **Conclusion:**
  Using the strategy of selecting and painting the house that started being available the earliest, we can achieve best time and space complexity, but it is not possible to get an optimal solution. Among all the given strategies, this algorithm utilizes very less space(constant space of O(1)).
- **Output:**

```
thunder:~/AOA/AoAProject1> make run1
java Task1
7 7
1 2
2 3
3 4
3 5
3 6
3 10
4 5
1 2 3 4 5 6 thunder:~/AOA/AoAProject1>
```

TASK2

Strat2: 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.

Algorithm: Θ (n + m log(m)) algorithm for solving TASK2.

Define HOUSE:

startDay,
endDay,
index

Function paintHouses takes the number of days(n), number of houses(m) and startDay and endDay of m houses(input[][]) and prints the index of painted houses.
input[k][0] contains the startDay of k^{th} house and input[k][1] contains the endDay of k^{th} house.

```
function paintHouses(n, m, input[][]):
    currentDay <- 1
    currentHouseIndex <- 0
    pq <- EMPTY PRIORITY QUEUE

    WHILE currentDay <= n:
      WHILE currentHouseIndex < m AND input[currentHouseIndex][0] <= currentDay
        pq.add(HOUSE(input[currentHouseIndex]))
        currentHouseIndex++
      WHILE(pq.size > 0 AND pq.top.endDay < currentDay)
        pq.removeTop
      IF(pq.size > 0)
        PRINT pq.removeTop.index
      currentDay++
```

IMPLEMENT PRIORITY QUEUE IN THE FOLLOWING WAY:

Compare two houses (h1,h2) in the following way using priority queue:

1. IF h1.startDay > h2.startDay, give more priority to h1
2. IF h1.startDay < h2.startDay, give more priority to h2

Tie Breaker:

IF h1.startDay == h2.startDay:

1. IF h1.endDay> h2.endDay, give more priority to h2
2. IF h1.endDay<h2.endDay, give more priority to h1
3. IF h1.endDay == h2.endDay, give more priority to house with highest index

Analysis:

- **Time Complexity:** To implement this approach, we have a maximum of 'm' houses that needs to be added into priority queue and adding each house into the priority queue comes at an expense of log(m) which sums the total cost to mlog(m). Since a maximum of m houses can be taken out of the priority queue, the total removal time is mlog(m). Additionally, the outer while loop has n iterations. Therefore, the total time complexity for the algorithm is O(n+mlog(m)) or precisely Θ(n +mlog(m)).
- **Space Complexity:** As we are not using any additional space except for the priority queue whose maximum size can be 'm', the space complexity is O(m).
- **Counter Example:**
  Consider the following date ranges of houses with n=8 and m=8
  A (1,2)
  B (2,3)
  C (3,4)

D (3,5)
E (3,7)
F (4,5)
G (5,6)
H (6,8)

Current Strategy: 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.

If we follow the above algorithm, then on day 1 we paint house A, on day 2 we paint house B, on day 3 we paint house C and on day 4 we paint house F, followed by G and H on day 5 and 6 respectively. Next on day 7 we paint house E and house D remains unpainted. So, 7 houses will be painted in the following order, 1 each day. {A->B->C->F->G->H->E}

However, an optimal solution can be painting house A on day 1, house B on day 2, house C on day 3, house D on day 4, house F on day 5, house G on day 6, house E on day 7 and finally house H on day 8.
So, all 8 houses will be painted in the following order 1 each day. {A->B->C->D->F->G->E->H}

- **Conclusion:**
  Strat2 has a greater time complexity [Θ(n+mlogm)] than Strat1 [Θ(n+m)]. However, both strategies do not provide an optimal solution, and this is demonstrated by giving a counterexample. Picking the house that started being available the earliest and picking the house that started being available the latest doesn't guarantee an optimal solution.
- **Output:**

```
thunder:~/AOA/AoAProject1> make run2
java Task2
8 8
1 2
2 3
3 4
3 5
3 7
4 5
5 6
6 8
1 2 3 6 7 8 5 thunder:~/AOA/AoAProject1>
```

TASK3

Strat3: 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.

Algorithm: Θ (n + m log(m)) algorithm for solving TASK3.

Define HOUSE:

startDay,
endDay,
index

Function paintHouses takes the number of days(n), number of houses(m) and startDay and endDay of m houses(input[][]) and prints the index of painted houses.
input[k][0] contains the startDay of k^{th} house and input[k][1] contains the endDay of k^{th} house.

```
function paintHouses(n, m, input[][]):
    currentDay <- 1
    currentHouseIndex <- 0;
    pq <- EMPTY PRIORITY QUEUE

    WHILE currentDay <= n:
      WHILE currentHouseIndex < m AND input[currentHouseIndex][0] <= currentDay
        pq.add(HOUSE (input[currentHouseIndex]))
        currentHouseIndex++;
      WHILE(pq.size > 0 AND pq.top.endDay < currentDay)
        pq.removeTop
      IF(pq.size > 0)
        PRINT pq.removeTop.index
      currentDay++
```

IMPLEMENT PRIORITY QUEUE IN THE FOLLOWING WAY:

Compare two houses (h1, h2) in the following way using priority queue:

Compute the duration for which house is available as stated below
h1Diff = h1.endDay - h1.startDay
h2Diff = h2.endDay - h2.startDay

1. IF h1Diff <h2Diff, give more priority to h1
2. IF h1Diff > h2Diff, give more priority to h2

Tie Breaker:
IF h1Diff == h2Diff:

1. IF h1.endDay< h2.endDay, give more priority to h1
2. IF h1.endDay>h2.endDay, give more priority to h2
3. IF h1.endDay == h2.endDay, give more priority to house with highest index

Analysis:

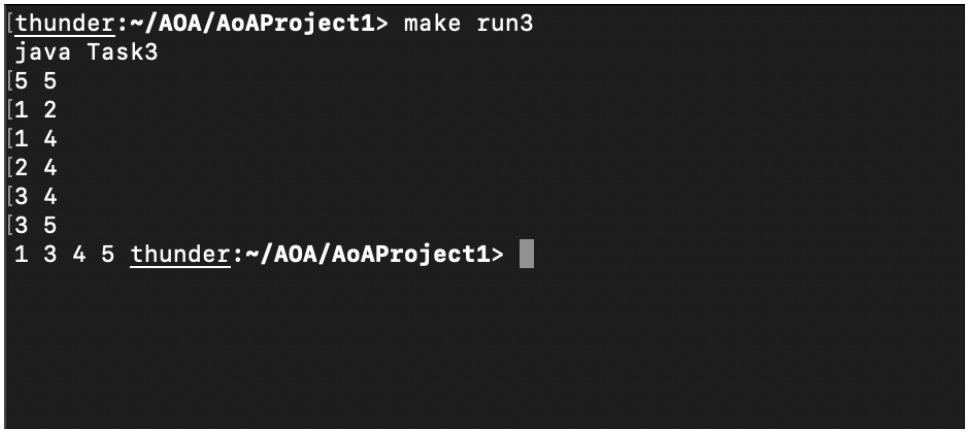
- **Time Complexity:** To implement this approach, we have a maximum of 'm' houses that needs to be added to the priority queue and adding each house into the priority queue comes at an expense of log(m) which sums the total cost to mlog(m). Since a maximum of m houses can be taken out of the priority queue, the total removal time is mlog(m). Additionally, the outer while loop has n iterations. Therefore, the total time complexity for the algorithm is O(n+mlog(m)) or more precisely Θ(n+mlog(m)).
- **Space Complexity:** As we are not using any additional space except for the priority queue whose maximum size can be 'm', the space complexity is O(m).
- **Counter Example:**
  Consider the following date ranges of houses with n=5 and m=5:

  A (1,2)
  B (1,4)
  C (2,4)
  D (3,4)
  E (3,5)

  Current Strategy: 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.

  If we follow the above algorithm, for day 1 we have both A & B available. As A is available for the shortest duration (1 day) than B (3 days), we choose to paint A. On day 2, we have B and C available to paint and C is available for 2 days while B is available for 3 days. Hence, we choose C (shortest available). And on day 3, we are left with houses B, D, E available and each of them are available for 3,1,2 days respectively. We choose to paint D on day 3. On day 4 we have B and E houses available to paint. We choose E to paint as it is available for 2 days while B is available for 3 days. We cannot paint the remaining house B on fifth day because it's endDay is 4.
  So, the houses are painted in the following order {A->C->D->E}.

  However, an optimal solution exists, paint A on day 1, paint C on day 2, paint D on day 3, paint B on day 4 and finally paint E on day 5.
  So, all the houses can be painted in the following order {A->C->D->B->E}
- **Conclusion:**
  This is a different approach than Strat2 but with the same running time [Θ(n+mlogm)]. Selecting houses that are available for shortest duration does not yield an optimal solution and this is clearly demonstrated using a counter example.
- **Output:**



TASK4

Strat4: 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.

Algorithm: Θ (n + m log(m)) algorithm for solving TASK4.

Define HOUSE:
START DAY,
END DAY,
INDEX

Function paintHouses takes the number of days(n), number of houses(m) and startDay and endDay of m houses(input[][]) and prints the index of painted houses.
input[k][0] contains the startDay of k^{th} house and input[k][1] contains the endDay of k^{th} house.

```
function paintHouses(n, m, input[][]):
    currentDay <- 1
    currentHouseIndex <- 0;
    pq <- EMPTY PRIORITY QUEUE

    WHILE currentDay <= n:
      WHILE currentHouseIndex < m AND input[currentHouseIndex][0] <= currentDay
            pq.add(HOUSE(input[currentHouseIndex]))
            currentHouseIndex++;
      WHILE(pq.size > 0 AND pq.top.endDay < currentDay)
        pq.removeTop
      IF(pq.size > 0)
        PRINT pq.removeTop.index
      currentDay++
```

IMPLEMENT PRIORITY QUEUE IN THE FOLLOWING WAY:

Compare two houses (h1,h2) in the following way using priority queue:

1. IF h1.endDay< h2.endDay, give more priority to h1
2. IF h1.endDay > h2.endDay, give more priority to h2

Tie Breaker:

IF h1.endDay == h2.endDay:

1. IF h1.startDay < h2.startDay, give more priority to h1
2. IF h1.startDay > h2.startDay, give more priority to h2
3. IF h1.startDay == h2.startDay, give more priority to house with highest index

Analysis:

- **Correctness:**
  Proof by contradiction:
  Assume that the current greedy strategy is not optimal and there exists a better strategy i.e., better strategy will result in higher number of painted houses than current greedy strategy.
  Let A_1 A_2 A_3 A_4 ....A_i be the set of houses painted by current greedy strategy.
  Let B_1 B_2 B_3 B_4 ....B_j be the set of houses painted by the better strategy.

  Until someday let both strategies paint the same house (A_1=B_1 A_2=B_2 and so on). On the day D let both strategies paint different houses. Let B_r be the house painted by better strategy on that day D and A_r be the house painted by the current greedy strategy.
  Note that both houses A_r and B_r are available on the day D as they are painted by these strategies, and as the current greedy strategy chooses A_r instead of B_r it must be the point that the endDate of A_r must be less than that of B_r.
  However, we can now say that the better strategy can choose to paint house A_r on day D and paint B_r in the future as endDate(B_r) > endDate(A_r). This means that the better strategy can paint more houses by following the greedy strategy of choosing the house with the earliest end date. This contradicts our assumption that the better strategy would result in maximizing the number of houses to be painted.
  Hence the current greedy solution is optimal.
- **Time Complexity:** To implement this approach, we have a maximum of 'm' houses that needs to be added and adding each house into the priority queue comes at an expense of log(m) which sums the total cost to mlog(m). Since a maximum of m houses can be taken out of the priority queue, the total removal time is mlog(m). Additionally, the outer while loop has n iterations. Therefore, the total time complexity for the algorithm is O(n+mlog(m)) or more precisely Θ(n+mlog(m)).
- **Space Complexity:** As we are not using any additional space except for the priority queue whose maximum size can be 'm', the space complexity is O(m).
- **Conclusion:**
  This strategy gives the optimal solution out of all strategies implemented. Selecting and painting houses that will stop being available the earliest can be achieved in

Θ(n+mlog(m)) which is the tighter bound and this strategy gives the maximum number of houses that can be painted.

- **Output:**

Experimental Study:

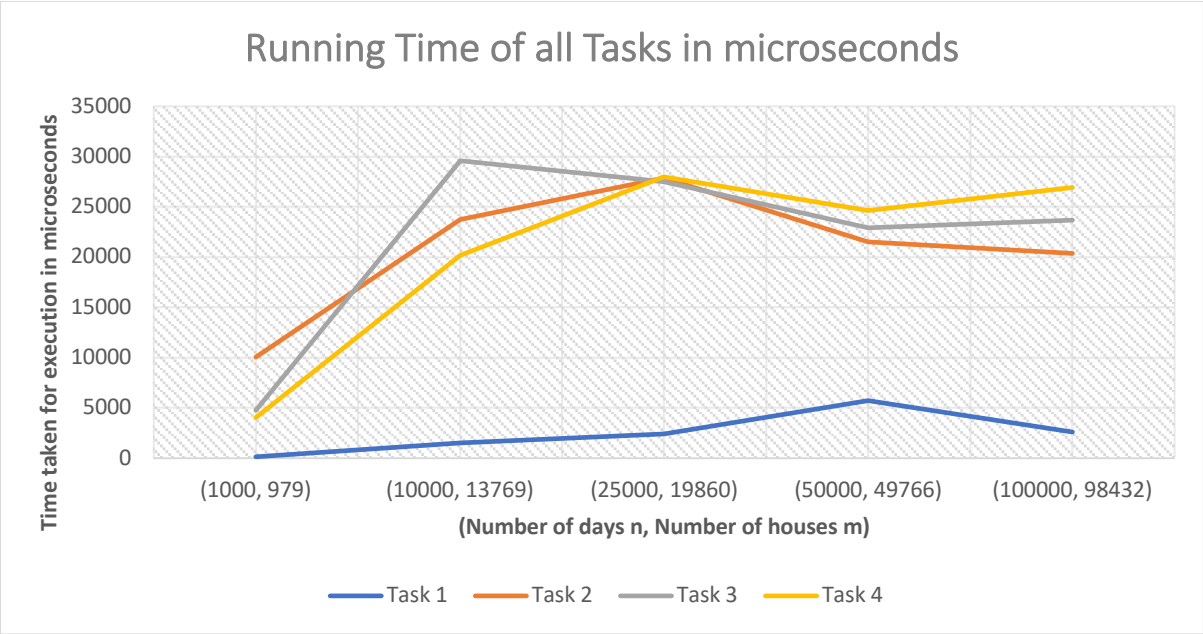
We have developed a python script that generates input files of different sizes. The python file randomly generates startDay and endDay for given **n**. **m** will be generated randomly and m value will be in the interval [1, 10^5). For n= 1000, 10000, 25000, 50000, 100000 we are checking the number of houses that can be painted by implementing each of the strategies. Additionally, we are also computing the total time taken in nano seconds for executing each of the tasks. Below attached is the screenshot that demonstrates running time and total number of houses painted for each of the tasks. A line graph is plotted with **(Number of days n, Number of houses m)** on x-axis and **execution time** on y-axis. Furthermore, a bar graph is plotted with **Number of Houses Painted** vs **Number of Days (n), Number of Houses (m).**

```
thunder:~/AOA/AoAProject1> java CompareTasks
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 1000
m value: 979
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 1

Total Number of houses painted: 877
Total Time taken in nano seconds: 137000
————————————————
TASK 2

Total Number of houses painted: 882
Total Time taken in nano seconds: 10054000
————————————————
TASK 3

Total Number of houses painted: 903
Total Time taken in nano seconds: 4761000
————————————————
TASK 4

Total Number of houses painted: 925
Total Time taken in nano seconds: 4038000
————————————————
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 10000
m value: 13769
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 1

Total Number of houses painted: 9979
Total Time taken in nano seconds: 1549000
————————————————
TASK 2

Total Number of houses painted: 9973
Total Time taken in nano seconds: 23734000
————————————————
TASK 3

Total Number of houses painted: 9987
Total Time taken in nano seconds: 29584000
————————————————
TASK 4

Total Number of houses painted: 9987
Total Time taken in nano seconds: 20173000
————————————————
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 25000
m value: 19860
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 1

Total Number of houses painted: 18723
Total Time taken in nano seconds: 2440000
————————————————
TASK 2

Total Number of houses painted: 18742
Total Time taken in nano seconds: 27870000
————————————————
TASK 3

Total Number of houses painted: 19023
Total Time taken in nano seconds: 27497000
————————————————
TASK 4

Total Number of houses painted: 19502
Total Time taken in nano seconds: 27982000
————————————————
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 50000
m value: 49766
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 1

Total Number of houses painted: 47158
Total Time taken in nano seconds: 5726000
————————————————
TASK 2

Total Number of houses painted: 47286
Total Time taken in nano seconds: 21516000
————————————————
TASK 3

Total Number of houses painted: 48026
Total Time taken in nano seconds: 22897000
————————————————
TASK 4

Total Number of houses painted: 49246
Total Time taken in nano seconds: 24637000
————————————————
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 100000
m value: 98432
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 1

Total Number of houses painted: 93800
Total Time taken in nano seconds: 2596000
————————————————
TASK 2

Total Number of houses painted: 94081
Total Time taken in nano seconds: 20377000
————————————————
TASK 3

Total Number of houses painted: 95660
Total Time taken in nano seconds: 23695000
————————————————
TASK 4

Total Number of houses painted: 97969
Total Time taken in nano seconds: 26935000
————————————————
thunder:~/AOA/AoAProject1>
```

Execution times for all tasks in microseconds:

Task Number/Inputs(n,m)	(1000, 979)	(10000, 13769)	(25000, 19860)	(50000, 49766)	(100000, 98432)
Task 1	137	1549	2440	5726	2596
Task 2	10054	23734	27870	21516	20377
Task 3	4761	29584	27497	22897	23695
Task 4	4038	20173	27982	24637	26935

Graphical Representation 1:
Below is the graph with **(Number of days n, Number of houses m)** on x-axis and **execution time in microseconds** on y-axis.



Inference from Graphical Representation 1:

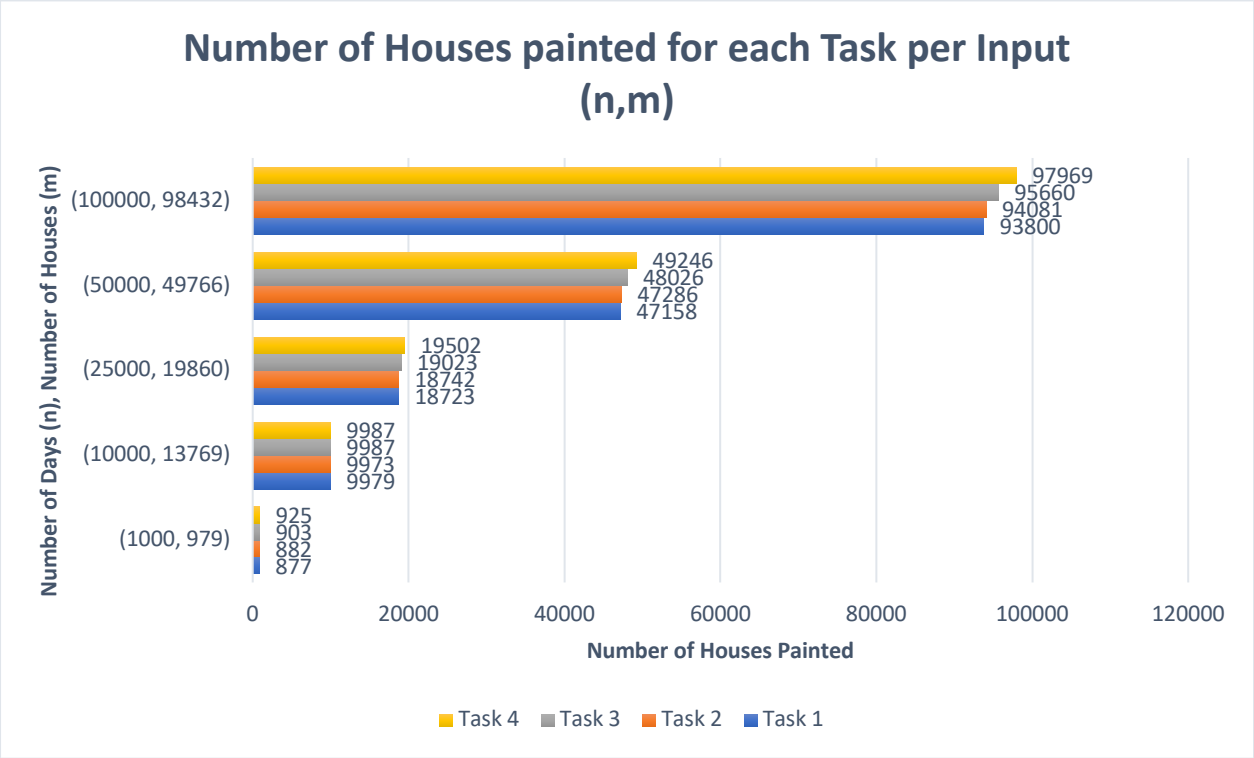
From the graph we can see that Task 1 Θ (n+m) approach has the lowest running time when compared to other tasks. Task 2, Task3 and Task4 have distinct running times for different inputs because of the difference in the comparator operations.

Number of houses painted:

Task Number/(n,m)	(1000, 979)	(10000, 13769)	(25000, 19860)	(50000, 49766)	(100000, 98432)
Task 1	877	9979	18723	47158	93800
Task 2	882	9973	18742	47286	94081
Task 3	903	9987	19023	48026	95660
Task 4	925	9987	19502	49246	97969

Graphical Representation 2:

Below is the graph with **Number of Houses Painted** vs **Number of Days (n), Number of Houses (m) for all four tasks.**



Inference from Graphical Representation 2:

Among the four tasks, Task 4 which is optimal has the highest painted houses for all the input sizes.

Conclusion:

In this project, we understood the importance of efficiency of algorithms and the role they play in developing an optimal solution. This project made us to think in different ways which constructively led us to develop an efficient algorithm. Now, we have a deeper understanding of various concepts that should be followed when designing an algorithm. Finding the perfect counter example to prove that the given strategy doesn't yield an optimal solution made us find out different underlying mistakes of the given strategy which periodically made us to build logic and write error free code. Different input sizes implied the way how each strategy's execution time varies with that of other strategies. We also got to know about different debugging strategies when we tried to solve the errors during code development. Developing the logic to write code given the time complexity was the crucial part and it enhanced our learning curve.

Bonus:

Task5: Design and analyze (time, space and correctness) of an Θ(m log(m)) algorithm based on the greedy strategy that you proved to be Optimal in Section 2.

Algorithm: Θ (m log(m)) algorithm for solving TASK5.

Define HOUSE:

startDay,
endDay,
index

Function paintHouses takes the number of days(n), number of houses(m) and startDay and endDay of m houses(input[][]) and prints the index of painted houses.
input[k][0] contains the startDay of k^{th} house and input[k][1] contains the endDay of k^{th} house.

```
function paintHouses(n, m, input[][]):
    currentDay <- 1
    currentHouseIndex <- 0;
    pq <- EMPTY PRIORITY QUEUE

    WHILE currentHouseIndex< m:
        IF input[currentHouseIndex][0] > n:
            Break
        WHILE currentDay < input[currentHouseIndex][0]:
            IF pq.size == 0:
                currentDay = input[currentHouseIndex][0]
            ELSE:
                House topHouse = pq.removeTop
                IF topHouse.endDay >= currentDay:
                    PRINT topHouse.index
                    currentDay++
        IF input[currentHouseIndex][0] <= currentDay:
            pq.add( HOUSE(input[currentHouseIndex]))
        currentHouseIndex++

    WHILE currentDay <=n AND pq.size >0 :
        House topHouse = pq.removeTop
        IF topHouse.endDay >= currentDay:
            PRINT topHouse.index
            currentDay++
```

IMPLEMENT PRIORITY QUEUE IN THE FOLLOWING WAY (same as Task 4):
Compare two houses (h1,h2) in the following way using priority queue:

1. IF h1.endDay< h2.endDay, give more priority to h1
2. IF h1.endDay > h2.endDay, give more priority to h2

Tie Breaker:
IF h1.endDay == h2.endDay:

1. IF h1.startDay < h2.startDay, give more priority to h1
2. IF h1.startDay > h2.startDay, give more priority to h2
3. IF h1.startDay == h2.startDay, give more priority to house with highest index

Analysis:

- **Time Complexity:** The algorithm just iterates through each and every house and performs at most m pushes into the priority queue and at most m pops from the priority queue, each push/pop takes log(m) time. Therefore, the total time complexity of the above algorithm is Θ(mlog(m) + mlog(m)) which is equivalent to Θ(mlog(m)).

- **Space Complexity:** As we are not using any additional space except for the priority queue whose maximum size can be 'm', the space complexity is O(m).

- **Correctness:** The above algorithm and Task 4 just vary in the implementation. The greedy strategy remains the same. While in the Task 4 we iterate through each day n and push the available houses into priority queue. Whereas in task 5, we implicitly do that when there is a difference in the currentDay and startDate of the currentHouse to be processed.

- **Output:**

```
[thunder:~/AOA/AoAProject1> make run5
 java Task5
[8 8
[1 2
[2 3
[3 4
[3 5
[3 7
[4 5
[5 6
[6 8
 1 2 3 4 6 7 5 8 thunder:~/AOA/AoAProject1>
```

Experimental Study:

In order to prove that Task 5 has a better time complexity than Task 4, we have tested the run times of both Task 4 and Task 5 with input files generated from a python script. For n= 1000, 10000, 25000, 50000, 100000 and a random m value we are checking the number of houses that can be painted by implementing these two strategies. Additionally, we are also computing the total time taken in nano seconds for executing each of the tasks. Below attached is the screenshot that demonstrates running time and total number of houses painted for each of the tasks. A line graph is plotted with **(Number of days n, Number of houses m)** on x-axis and **execution time** on y-axis.

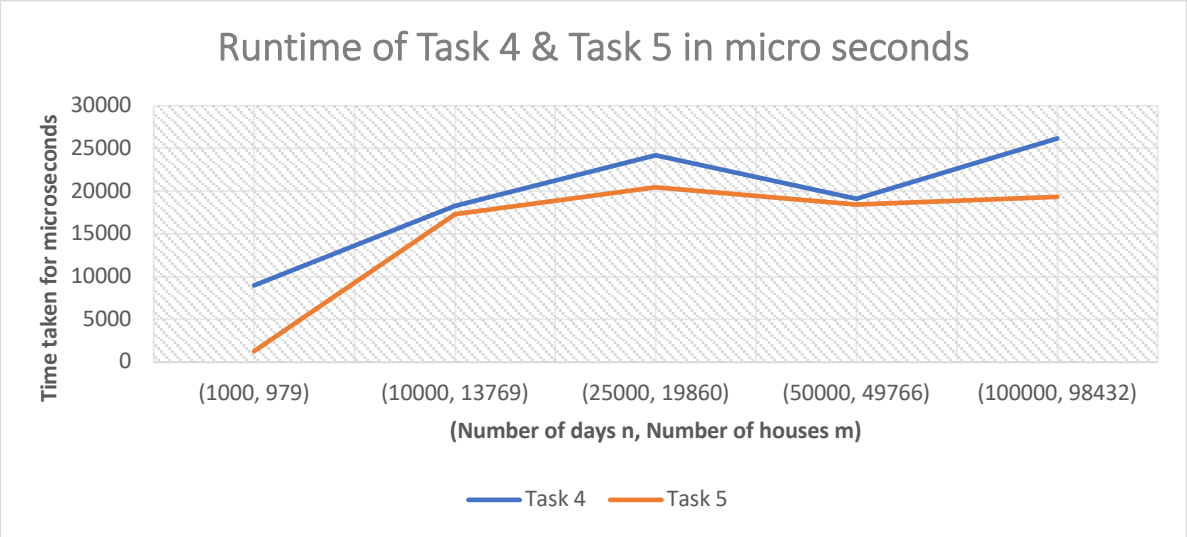
Number of Houses Painted:

Task Number/(n,m)	(1000, 979)	(10000, 13769)	(25000, 19860)	(50000, 49766)	(100000, 98432)
Task 4	925	9987	19502	49246	97969
Task 5	925	9987	19502	49246	97969

Execution Time in microseconds:

Task Number/Inputs(n,m)	(1000, 979)	(10000, 13769)	(25000, 19860)	(50000, 49766)	(100000, 98432)
Task 4	8987	18313	24230	19141	26180
Task 5	1287	17285	20467	18460	19331

Graphical Representation 5:



Output of Comparision Study of Task 4 and Task 5:

```
thunder:~/AOA/AoAProject1> java CompareTasks
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 1000
m value: 979
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 4

Total Number of houses painted: 925
Total Time taken in nano seconds: 8987000
--------------------
TASK 5

Total Number of houses painted: 925
Total Time taken in nano seconds: 1287000
--------------------
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 10000
m value: 13769
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 4

Total Number of houses painted: 9987
Total Time taken in nano seconds: 18313000
--------------------
TASK 5

Total Number of houses painted: 9987
Total Time taken in nano seconds: 17285000
--------------------
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 25000
m value: 19860
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 4

Total Number of houses painted: 19502
Total Time taken in nano seconds: 24230000
--------------------
TASK 5

Total Number of houses painted: 19502
Total Time taken in nano seconds: 20467000
--------------------
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 50000
m value: 49766
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 4

Total Number of houses painted: 49246
Total Time taken in nano seconds: 19141000
--------------------
TASK 5

Total Number of houses painted: 49246
Total Time taken in nano seconds: 18460000
--------------------
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
n value: 100000
m value: 98432
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TASK 4

Total Number of houses painted: 97969
Total Time taken in nano seconds: 26180000
--------------------
TASK 5

Total Number of houses painted: 97969
Total Time taken in nano seconds: 19331000
--------------------
thunder:~/AOA/AoAProject1>
```

Conclusion:

From the above graphs, we can conclude that run time of Task 5 Θ(mlog(m)) is less than that of Task 4 Θ(n+mlog(m)) while both adopt the same strategy of choosing the house with the earliest end date.