Greedy-Programming-Project

COT5405 Analysis of Algorithm (AOA)

Team Members Contribution

Lohit Bhambri (lohit.bhambri@ufl.edu)

1. Implemented Strategy1
2. Implemented Strategy2
3. Implemented Strategy4
4. Implemented Strategy Bonus (equal contributor)
5. Report Compilation
6. Analysis of Algorithms (equal contributor)

Sharath Bhushan Podila (spodila@ufl.edu)

1. Implemented Strategy3
2. Implemented Strategy Bonus (equal contributor)
3. Implemented MakeFile mechanism and execution in Remote CISE Machines
4. Experimental Comparative Strategy mechanism result generation
5. Analysis of Algorithms (equal contributor)

Strategy 1

Algorithm:

```
Algorithm strat1(timelines, n, m)
Input:
    timelines: a list of m pairs of integers representing the painting timelines for each house
    n: an integer representing the number of days available for painting
    m: an integer representing the number of houses to paint
Output:
    A string containing the sequence of house numbers in the order they were painted

Initialize an empty string housePaintString
Initialize a queue houseQueue to store the painting timelines for each house
For each painting timeline in timelines, add the house number to the end of the list, and add the list to houseQueue
For each day pDay from 1 to n do the following:
    a. While houseQueue is not empty do the following:
        i. Peek the first element of houseQueue and assign it to peekHouse
        ii. If the painting timeline of peekHouse covers pDay do the following:
            1. Remove peekHouse from houseQueue
            2. Append the house number of peekHouse to housePaintString
            3. Append a space to housePaintString
            4. Break the loop
        iii. Else, if the painting timeline of peekHouse has already passed pDay remove it from houseQueue and continue
        the loop
        iv. Else, if the painting timeline of peekHouse starts after pDay break the loop
    b. If houseQueue is empty, break the loop
Return housePaintString with leading and trailing whitespaces removed
```

Proof of Completion:

As we are concerned with (n) i.e. painters availability from (1..n), our loop will surely terminate if we exhaust the painter's availabilty and we have future listing. In this scenario (n)<(m) where (m) is the number of listed houses for painting.
If we have less listed houses compared with painters day i.e. (m)<(n), we will exhaust all the available houses & the remaining painter's days will be idle.

In both the scenario's our loop will terminate based upon the exhausting the painters availability thus our algorithm will surely terminate.

Proof of Correctness:

Lets assume that our greedy approach generates output order as
G = {g1, g2, g3, g4, g5.....gN}

Lets assume that our optimal approach generates output order as
O = {o1, o2, o3, o4, o5.....gM}

Cardinality |O| >= |G|; therefore M>=N.
Lets assume till (k-1), both greedy and optimal approaches are producing the same output.
At kth point we have conflict.
Since our greedy approach is picking the earlist listing, we can state that gK.startDay <= oK.startDay.
Therefore we will start-replacing gK with oK inside (0) set via exchange argument.

Our updated optimal (0) set will be
0 = {o1,o2,o3,o4...oK-1,gk,gk+1,gk+2,gk+3...gN,oN+1,oN+2 ..... ,oM}
Therefore our greedy will become optimal is M=N strictly.

Time Complexity Analysis:

For each pDay ranging from (1..n) we are checking if the the head of the queue's startDay and endDay are following within threshold criteria or not. If yes, we are paining removing the head pair from the queue & painting the house else we are just removing the pair from the queue. Since we cannot exceed the (n) buffer, our algorithm will be linear
Time Complexity: O(n)

Strategy 2

Algorithm:

```
Algorithm strat2(timelines, n, m)
Input:
    timelines: a list of m pairs of integers representing the painting timelines for each house
    n: an integer representing the number of days available for painting
    m: an integer representing the number of houses to paint
Output:
    A string containing the sequence of house numbers in the order they were painted

Initialize an empty string housePaintString
Initialize a priority queue latestHouses to store the painting timelines for each house, sorted by decreasing start day
and increasing end day

Initialize housePtr to 0
For each day pDay from 1 to n do the following:
    a. While housePtr < m do the following:
        i. Peek the house timeline at index housePtr in timelines and assign it to pair
        ii. If the start day of pair equals pDay do the following:
            Add the index of the house (housePtr+1) to the end of pair
            Add pair to latestHouses
            Increment housePtr by 1
        iii. Else, break the loop
    b. If latestHouses is empty, continue the loop
    c. While latestHouses is not empty do the following:
        i. Peek the first element of latestHouses and assign it to peekHouse
        ii. If pDay is between the start and end days of peekHouse do the following:
            Remove peekHouse from latestHouses
            Append the house number of peekHouse to housePaintString
            Append a space to housePaintString
            Break the loop
        iii. Else, if pDay is greater than the end day of peekHouse remove it from latestHouses and continue the loop
        iv. Else, if pDay is less than the start day of peekHouse, break the loop
Return housePaintString with leading and trailing whitespaces removed
```

Proof of Completion:

As we are concerned with (n) i.e. painters availability from (1..n), on every pDay we are checking for new listing. If any we will add that inside out priority queue (max-heap on the basis of startDay) and paint the latest listing.
If no listing is avaialble we will paint the best option provided to us by the priority queue. Our program will surely terminate if we are left with no painter's availability efvne if we have previously availble listings present inside the priority queue. If we are out with all the listing, our program will surely terminate in that scenario.

Proof of Correctness:

Lets assume that our greedy approach generates output order as
G = {g1, g2, g3, g4, g5.....gN}

Lets assume that our optimal approach generates output order as
O = {o1, o2, o3, o4, o5.....gM}

Cardinality |O| >= |G|; therefore M>=N.
Lets assume till (k-1), both greedy and optimal approaches are producing the same output.
At kth point we have conflict.
Since our greedy approach is picking the latest listing i.e. latest start day (maximum), we can state that gK.startDay >= oK.startDay.
Therefore we will start-replacing gK with oK inside (G) set via exchange argument.

Our updated greedy (G) set will be
G = {g1,g2,g3,g4...gK-1,ok,ok+1,ok+2,ok+3.....oN}

Therefore our greedy will become optimal is M=N strictly.

Time Complexity Analysis:

For each pDay ranging from (1..n) we are checking if there is an available listing for that day or not. If yes, we are adding it to priority-queue (aka max-heap) and extracting the earliest available listing. If that listing is satisfying

```
the threshold, we are painting that house.
Time Complexity: O(n+logm)
```

Strategy 3

Algorithm:

```
Algorithm strat3(timelines, n, m)
Input:
    timelines: a list of m pairs of integers representing the painting timelines for each house
    n: an integer representing the number of days available for painting
    m: an integer representing the number of houses to paint
Output:
    A string containing the sequence of house numbers in the order they were painted

Initialize an empty string housePaintString
Initialize a priority queue latestHouses to store the painting timelines for each house

The priority queue latestHouses is sorted based on the duration of the painting timeline, and if the duration is the
same, based on the end date of the painting timeline

Initialize housePtr to 0
For each day pDay from 1 to n do the following:
    a. If pDay is greater than 1, do the following:
        i. Initialize an empty list previousBuffer
        ii. While latestHouses is not empty do the following:
            Remove the head of latestHouses and assign it to priorityQueueHead
            Decrement the duration of priorityQueueHead by 1
            If the duration of priorityQueueHead is still greater than or equal to 0, add it to previousBuffer and
            continue the loop
        iii. For each painting timeline in previousBuffer, add it back to latestHouses
    b. While housePtr is less than m and the start date of the painting timeline at housePtr is equal to pDay do the
    following:
        i. Create a list pair consisting of the painting timeline at housePtr, the duration of the painting timeline,
```

```
        and the house number
        ii. Add pair to latestHouses
        iii. Increment housePtr by 1
    c. If latestHouses is empty, continue to the next iteration of the loop
    d. While latestHouses is not empty do the following:
        i. Peek the head of latestHouses and assign it to peekHouse
        ii. If the painting timeline of peekHouse covers pDay do the following:
            Remove peekHouse from latestHouses
            Append the house number of peekHouse to housePaintString
            Append a space to housePaintString
            Break the loop
        iii. Else, if the painting timeline of peekHouse has already passed pDay remove it from latestHouses and
        continue the loop
        iv. Else, if the painting timeline of peekHouse starts after pDay break the loop
Return housePaintString with leading and trailing whitespaces removed
```

Proof of Completion:

As we are concerned with (n) i.e. painters availability from (1..n), on every pDay we are checking for new listing. We will calculate the duration for evey listing and enque those listings inside our priority queue. Since we are using a min heap priority queue, the listing with the shortest duration will be extracted from the queue and will be painted by the painter.
We will move on towards the next pDay and if the pDay's are exhausted, the algorithm will surely stop.

*** Additional Step implemented inside the algorithm (although not needed according to the problem)
Before moving on to the next day listing, we are making sure that the duration of our previous listing are updated. We are updating the duration of our previous listings and if we ran out of time, we will not enqueu it back inside our priority queue. This logic will increase the time complexity by the 0(klogk) where (k) is surely less than or equal to (m).

Proof of Correctness:

```
Lets assume that our greedy approach generates output order as
G = {g1, g2, g3, g4, g5.....gN}

Lets assume that our optimal approach generates output order as
O = {o1, o2, o3, o4, o5.....gM}

Cardinality |O| >= |G|; therefore M>=N.
Lets assume till (k-1), both greedy and optimal approaches are producing the same output.
At kth point we have conflict.
Since our greedy approach is picking the listing with minimum duration, we can state that
gK.duration <= oK.duration.
Therefore we will start-replacing gK with oK inside (O) set via exchange argument.

Our updated greedy (G) set will be
O = {o1,o2,o3,o4...oK-1,gk,gk+1,gk+2,gk+3...gN,oN+1,oN+2 ..... ,oM}

Therefore our greedy will become optimal is M=N strictly.
```

Time Complexity Analysis:

```
For each pDay ranging from (1..n) we are checking if there is an available listing for that day or not. If yes, we are
adding it to priority-queue (aka min-heap) by calculating the duration and extracting the listing with minimum duration.
If that listing is satisfying the threshold, we are painting that house.
Time Complexity: O(n+logm)
```

Strategy 4

Algorithm:

```
Algorithm strat4(timelines, n, m)
Input:
    timelines: a list of m pairs of integers representing the painting timelines for each house
```

```
    n: an integer representing the number of days available for painting
    m: an integer representing the number of houses to paint
Output:
    A string containing the sequence of house numbers in the order they were painted

Initialize an empty string housePaintString

Initialize a priority queue latestHouses to store the painting timelines for each house, with a custom comparator
function that compares based on the end date and then the start date of the painting timeline

Initialize housePtr to 0
For each day pDay from 1 to n do the following:
    a. While housePtr is less than m and the start date of the painting timeline at housePtr is equal to pDay, do the
    following:
        i. Add the house number (housePtr+1) to the end of the painting timeline at housePtr
        ii. Add the painting timeline at housePtr to latestHouses
        iii. Increment housePtr by 1
    b. If latestHouses is empty, continue the loop
    c. While latestHouses is not empty, do the following:
        i. Peek the first element of latestHouses and assign it to peekHouse
        ii. If the painting timeline of peekHouse covers pDay do the following:
            1. Remove peekHouse from latestHouses
            2. Append the house number of peekHouse to housePaintString
            3. Append a space to housePaintString
            4. Break the loop
        iii. Else, if the painting timeline of peekHouse has already passed pDay remove it from latestHouses and
        continue the loop
        iv. Else, if the painting timeline of peekHouse starts after pDay break the loop
Return housePaintString with leading and trailing whitespaces removed
```

Proof of Completion:

As we are concerned with (n) i.e. painters availability from (1..n), on every pDay we are checking for new listing. We are utilizing the priority queue (max heap) on the basis of deadline. The listing with earliest deadline will be painted

first.
If the enqued listing satifies the conditions, it will be painted by the painter and the algorithm will move on to the next day. If we are out of pDay, the algorithm will surely terminate as we have exhauted the painter's day and any listing inside the priority queue won't be dequeued.

Proof of Correctness:

Lets assume that our greedy approach generates output order as
G = {g1, g2, g3, g4, g5.....gN}

Lets assume that our optimal approach generates output order as
O = {o1, o2, o3, o4, o5.....gM}

Cardinality |O| >= |G|; therefore M>=N.
Lets assume till (k-1), both greedy and optimal approaches are producing the same output.
At kth point we have conflict.
Since our greedy approach is picking the listing with minimum endDate, we can state that
gK.endDate <= oK.endDate.
Therefore we will start-replacing gK with oK inside (O) set via exchange argument.

Our updated greedy (G) set will be
O = {o1,o2,o3,o4...oK-1,gk,gk+1,gk+2,gk+3...gN,oN+1,oN+2 ..... ,oM}

Therefore our greedy will become optimal is M=N strictly.

Time Complexity Analysis:

For each pDay ranging from (1..n) we are checking if there is an available listing for that day or not. If yes, we are adding it to priority-queue (aka max-heap) extracting the listing with earliest end date.If that listing is satisfying the threshold, we are painting that house.
Time Complexity: O(n+logm)

Strategy Bonus

Algorithm:

```
Algorithm stratBonus(timelines, n, m)
Input:
    timelines: a list of m pairs of integers representing the painting timelines for each house
    n: an integer representing the number of days available for painting
    m: an integer representing the number of houses to paint
Output:
    A string containing the sequence of house numbers in the order they were painted

Initialize an empty string housePaintString to store the sequence of house numbers in the order they were painted.

Initialize a priority queue latestHouses to store the painting timelines for each house in the order of their completion
time, with the house that finishes painting first being at the front of the queue. Each element in the queue is an
ArrayList with three integers: the start time, the end time, and the house number.

Initialize a variable housePtr to keep track of the current house being processed in timelines.

For each day pDay from 1 to n do the following:
    a. While housePtr is less than m and the start time of the house at housePtr in timelines is equal to pDay, do the
    following:
        i. Create an ArrayList pair containing the start time, end time, and house number.
        ii. Add the house number to pair and add pair to latestHouses.
        iii. Increment housePtr.
    b. If latestHouses is empty and housePtr is less than m:
        Set pDay to the start time of the house at housePtr in timelines minus 1,
        Continue to the next iteration of the loop.
    c. While latestHouses is not empty, do the following:
            i. Peek the first element of latestHouses and assign it to peekHouse.
            ii. If pDay is between the start time and end time (inclusive) of peekHouse, do the following:
                1. Remove peekHouse from latestHouses.
                2. Append the house number of peekHouse to housePaintString.
                3. Append a space to housePaintString.
```

4. Break the loop.
iii. Else, if the end time of peekHouse is less than pDay, remove peekHouse from latestHouses and continue the loop.
iv. Else, if the start time of peekHouse is greater than pDay, break the loop.
Return housePaintString with leading and trailing whitespaces removed.

Proof of Completion:

As we are concerned with (n) i.e. painters availability from (1..n), on every pDay we are checking for new listing. We are utilizing the priority queue (max heap) on the basis of deadline. The listing with earliest deadline will be painted first.
If the enqued listing satifies the conditions, it will be painted by the painter and the algorithm will move on to the next day. If we are out of pDay, the algorithm will surely terminate as we have exhauted the painter's day and any listing inside the priority queue won't be dequeued.
The optimality for this strategy in comparison with strategy 4 is w.r.t. the gap days. Suppose our priority queue is empty and the next listing is "far" ahead. The algorithm will jump to the startDay of the upcoming listing (condition startDay <= n) and we will follow the same above mentioned step.

Proof of Correctness:

Lets assume that our greedy approach generates output order as
G = {g1, g2, g3, g4, g5.....gN}

Lets assume that our optimal approach generates output order as
O = {o1, o2, o3, o4, o5.....gM}

Cardinality |O| >= |G|; therefore M>=N.
Lets assume till (k-1), both greedy and optimal approaches are producing the same output.
At kth point we have conflict.
Since our greedy approach is picking the listing with minimum duration, we can state that gK.duration <= oK.duration.
Therefore we will start-replacing gK with oK inside (O) set via exchange argument.

Our updated greedy (G) set will be
O = {o1,o2,o3,o4...oK-1,gk,gk+1,gk+2,gk+3...gN,oN+1,oN+2...,oM}

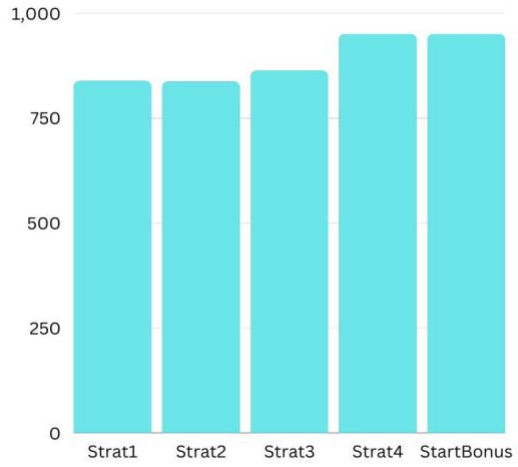
Therefore our greedy will become optimal is M=N strictly.

Time Complexity Analysis:

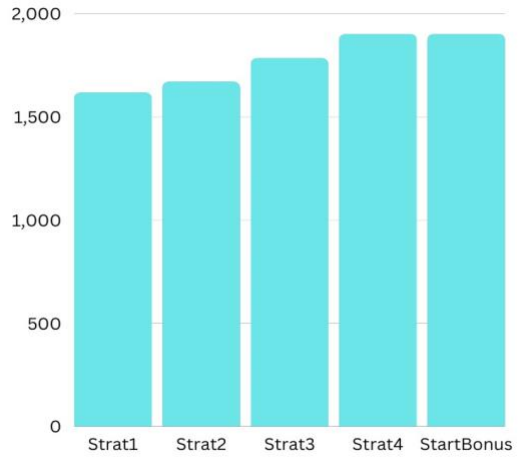
For each pDay ranging from (1..n) we are checking if there is an available listing for that day or not. If yes, we are adding it to priority-queue (aka max-heap) extracting the listing with earliest end date.If that listing is satisfying the threshold, we are painting that house. Additionally, if our priority-queue is empty, and we have the next avaialble listing startDay way ahead i.e. startDay of new listing > pday, we will update our pDay to the next available startDay and continue the processing in a similar way
Time Complexity: O(m+logm)

Tabular Experimental Analysis for the strategies

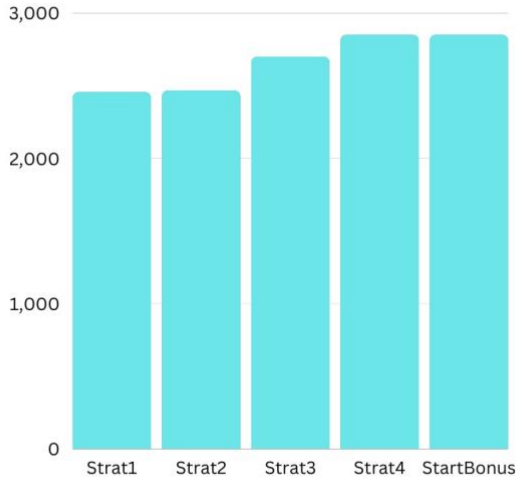
n = 1000 & m = 950



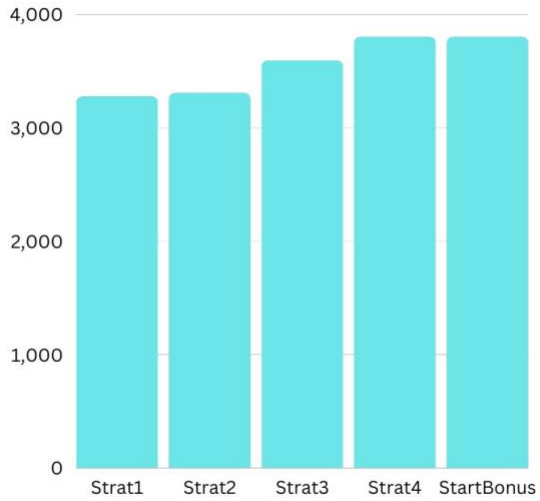
n = 2000 & m = 1900



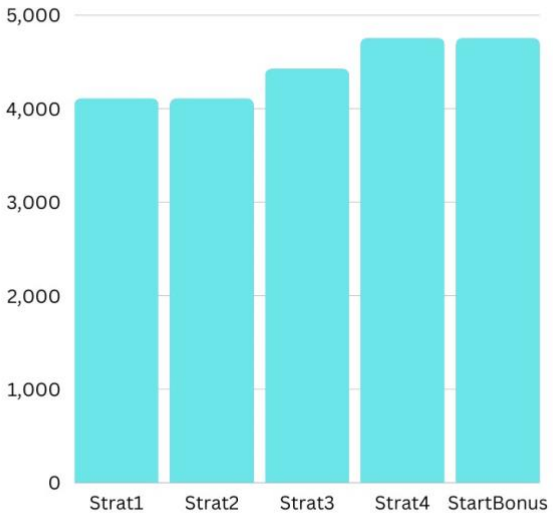
n = 3000 & m = 2850



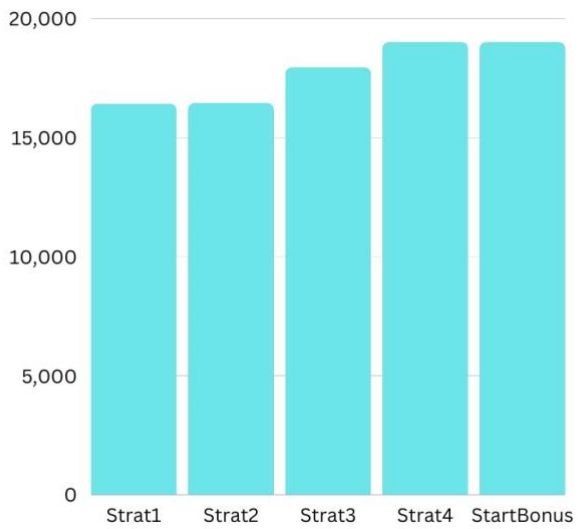
n = 4000 & m = 3800



n = 5000 & m = 4750



n = 20000 & m = 19000



Analysis Strategy 4 v/s Strategy Bonus

In strategy-4 we are iterating the pDays if the priority queue is empty & either we don't have any available listing or the available listing startDay is quite far ahead (condition startDay <= n).
So the time complexity is O(n+logm).

In strategy-bonus we are jumping to the next available startDay if our queue is empty. Afterwards we are continuing the same process in which pDay will be updated to the upcoming listing startDay. This will save us the difference: *`(startDay-of-upcoming-listing - current-pDay)`*
So the time complexity will be O(m+logm).

Tabular Time Comparision

Strat	[10000,95000]	[100000,950000]	[1000000,9500000]
Strat4	112	1092	31420
Bonus	106	937	28702

Graph Comparision