# CS2040 Mock MidTerms Solutions

By Matthew, Zhi Jian and Kevin

# Q1) Time Complexity Analysis

False.

func2 runs in **O(N)** time

func calls func2(N / 2) twice (**O(N)** time) and recursively calls itself log(N) times

One method call to func(n) runs in **O(N)** time

# Q2) Sorting

False.

After one partition, insertion sort is called on two size N / 2 blocks of the array

Effectively an obfuscated insertion sort

Runtime **$O(N^2)$**

# Q3) Stack

False.

Elements in a linked list can only be accessed in O(n) time since we have to traverse the linked list.

# Q4) Hashing

False.

Linear probing will always find an available bucket as long as there is vacancy

Quadratic problem may possibly probe forever if the load factor of the hash table is greater than 50%

Double hashing may produce a step count that is a factor of the table size and may possibly result in infinite probing

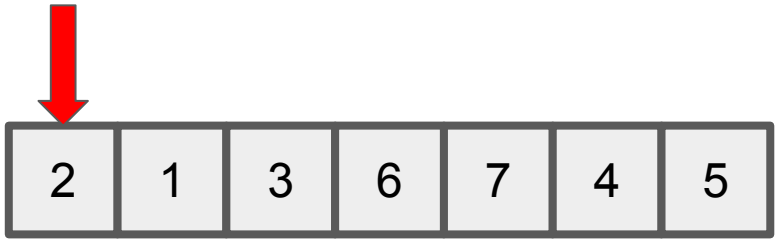
# Q5a) Optimized Bubble Sort

General case algorithm:

```
answer = 0

for i from 1 to N:

    if i > array[i]:

        answer = max(answer, i - array[i])

return min(answer + 1, n - 1)
```

# Q5a) Optimized Bubble Sort

Example run:



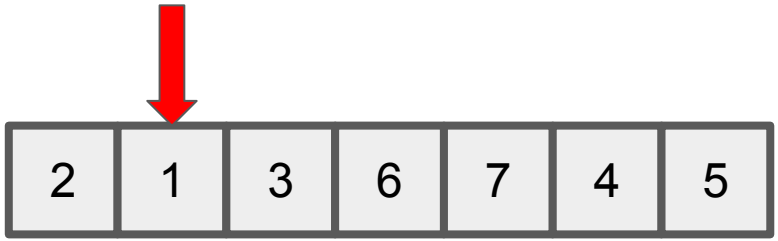

i = 1

array[i] = 2

ans = 0

# Q5a) Optimized Bubble Sort

Example run:



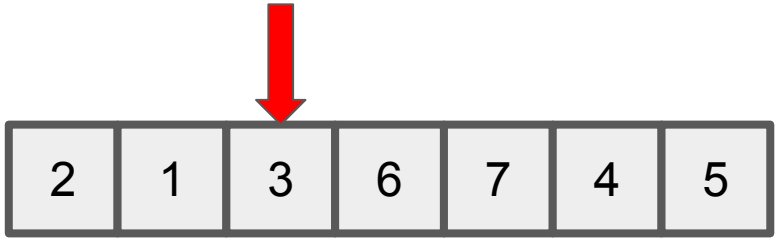

i = 2

array[i] = 1

ans = 2

# Q5a) Optimized Bubble Sort

Example run:



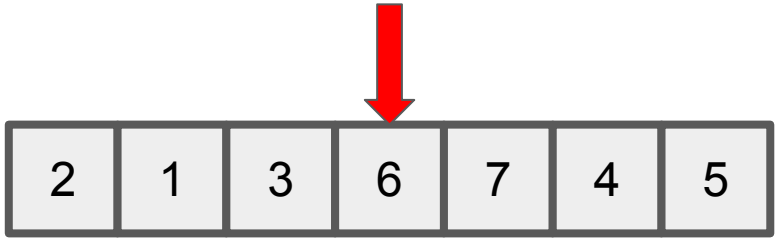

i = 3

array[i] = 3

ans = 2

# Q5a) Optimized Bubble Sort

Example run:



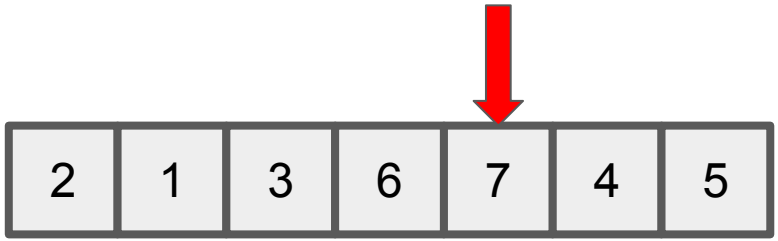

i = 4

array[i] = 6

ans = 2

# Q5a) Optimized Bubble Sort

Example run:



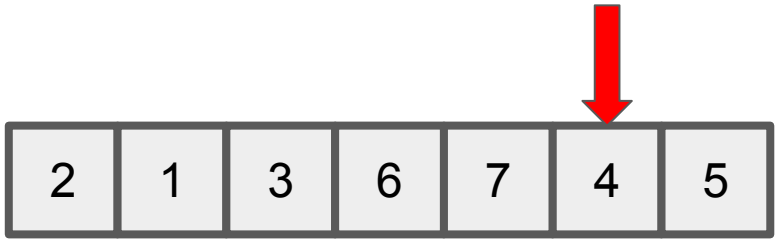

i = 5

array[i] = 7

ans = 2

# Q5a) Optimized Bubble Sort

Example run:



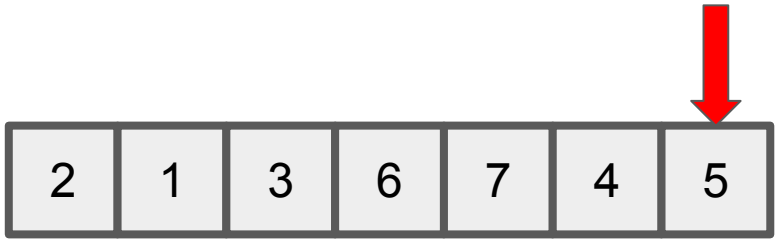

i = 6

array[i] = 4

ans = 2

# Q5a) Optimized Bubble Sort

Example run:




i = 7

array[i] = 5

ans = 2

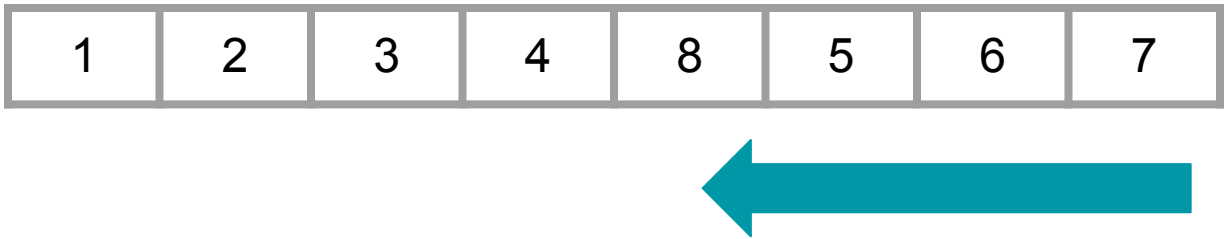
# Q5b) Optimized Bubble Sort

Start with a sorted array of *n* elements.

# Q5b) Optimized Bubble Sort

Start with a sorted array of *n* elements.

Observation: Shifting the largest element *p* positions left generates *p* swaps.



# Q5b) Optimized Bubble Sort

Observation:

Largest element in array can generate at most $n$ - 1 swaps

Second largest element in array can generate at most $n$ - 2 swaps

…

Smallest element in array cannot generate any swaps

# Q5b) Optimized Bubble Sort

Algorithm

```
A = [ ]
last = n
for i = n to 1
        if k >= i - 1:
                k = k - (i - 1)
                append i to A
        else last = i, break
for i = 1 to last - 1
        if last - i == k: append last to A, last = -1
        Append i to A
if last != -1: append last to A
```

# Q6) Most Frequent Element

You have an array of N elements. It is guaranteed that there is exactly one such element that appears more than half the time. Find that element in **O(N)** time.

# Q6) Most Frequent Element: O(N) memory

Insert all n elements into a hash table, value = frequency of that key

Iterate through the hash table and report the most frequent element

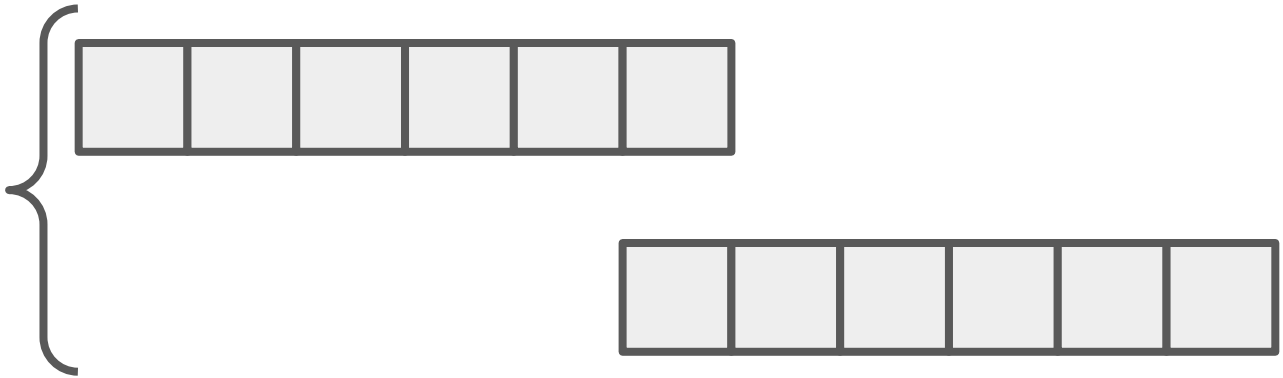
Memory: Up to (N / 2 - 1) distinct elements. O(N) space required.

# Q6) Most Frequent Element: O(1) memory, Method 1

**Quick select** the median element of the array

**Expected O(N)** runtime

No matter what the other elements are, the most frequent element must occupy the middle position



# Q6) Most Frequent Element: O(1) memory, Method 2

Maintain a candidate for the most frequent element, starting with the left-most element, and create a counter

If the next element matches the current candidate, increment the counter

Otherwise, decrement the counter

If the counter is zero, the next element on the right is the candidate

When the last element is reached, the candidate element is the most frequent element

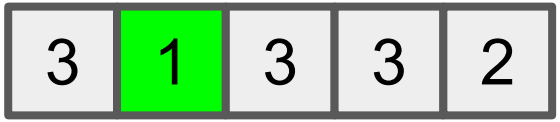
# Q6) Most Frequent Element: O(1) memory, Method 2

Candidate = 3, Frequency = 1

# Q6) Most Frequent Element: O(1) memory, Method 2

Candidate = 3, Frequency = 1

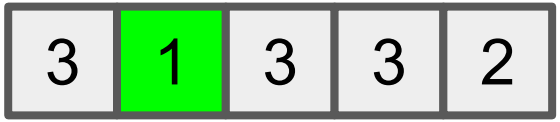

Candidate = ?, Frequency = 0



# Q6) Most Frequent Element: O(1) memory, Method 2

Candidate = 3, Frequency = 1


Candidate = ?, Frequency = 0

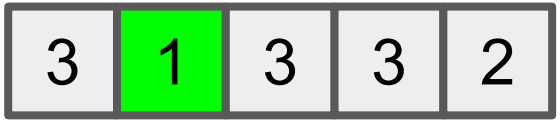



Candidate = 3, Frequency = 1

# Q6) Most Frequent Element: O(1) memory, Method 2

Candidate = 3, Frequency = 1


Candidate = ?, Frequency = 0



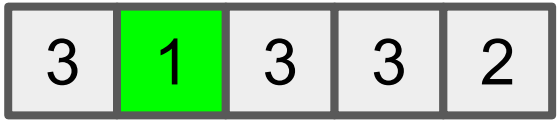

Candidate = 3, Frequency = 1


Candidate = 3, Frequency = 2

# Q6) Most Frequent Element: O(1) memory, Method 2

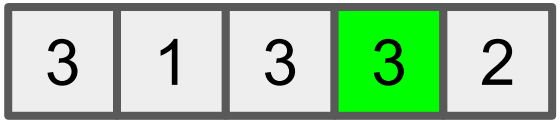
Candidate = 3, Frequency = 1


Candidate = ?, Frequency = 0




Candidate = 3, Frequency = 1


Candidate = 3, Frequency = 2

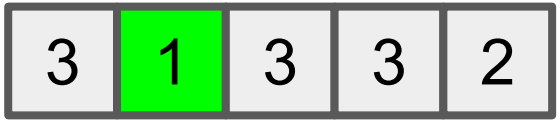



Candidate = 3, Frequency = 1

# Q6) Most Frequent Element: O(1) memory, Method 2

Candidate = 3, Frequency = 1


Candidate = ?, Frequency = 0




Candidate = 3, Frequency = 1

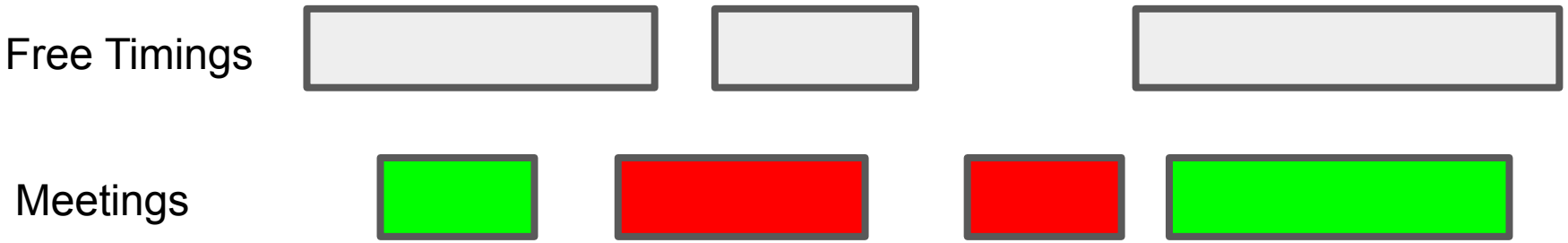

Candidate = 3, Frequency = 2


Candidate = 3, Frequency = 1


Most Frequent Element: 3

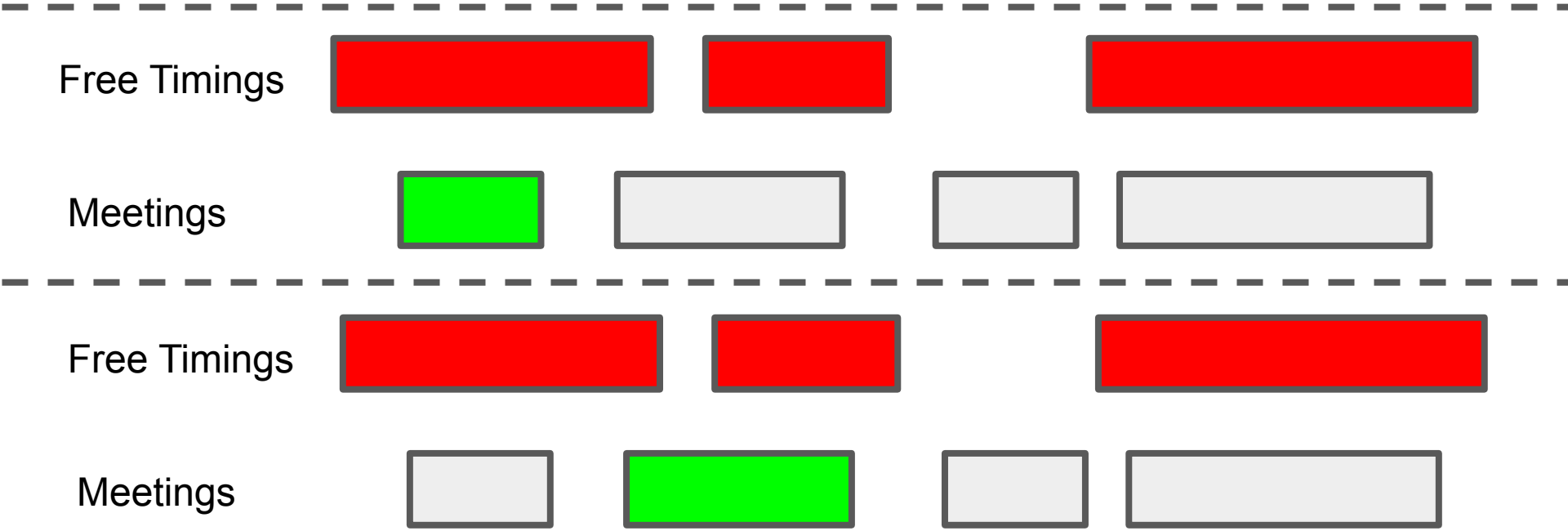
# Q7) Meetings, Part 1

Given 2 list of disjoint intervals, find the number of intervals from the second list that fits into one of the intervals from the first list



# Q7) Meetings, Part 1, O(NM)

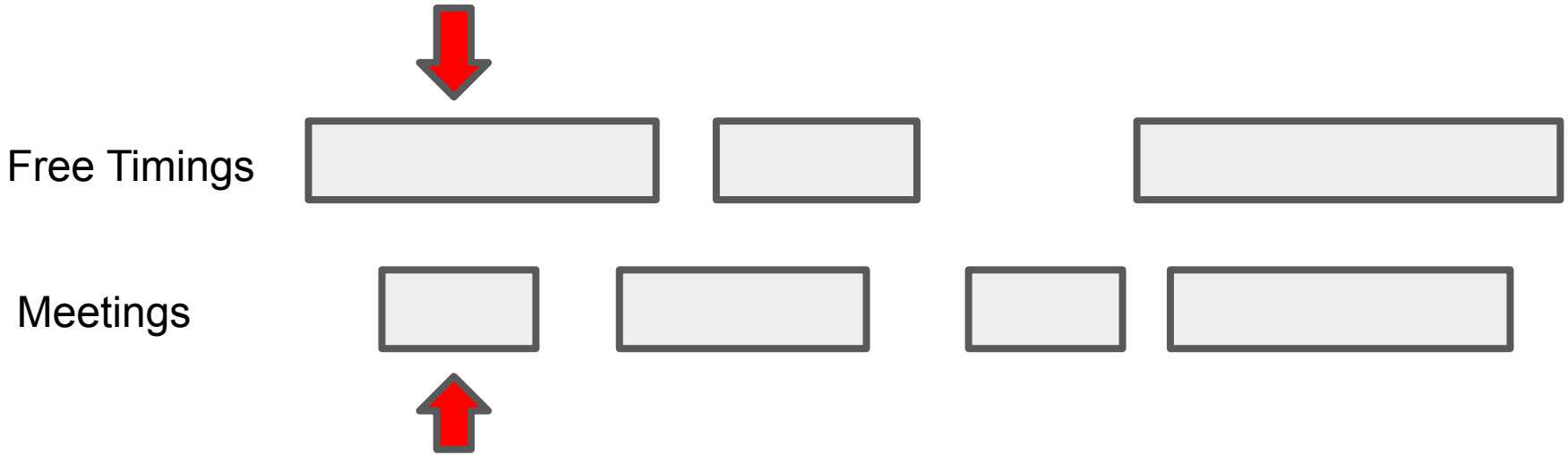
For each of the M meetings, check all N free timings and see if it fits in any of the M free time intervals



# Q7) Meetings, Part 1, O(NlogN + MlogM)

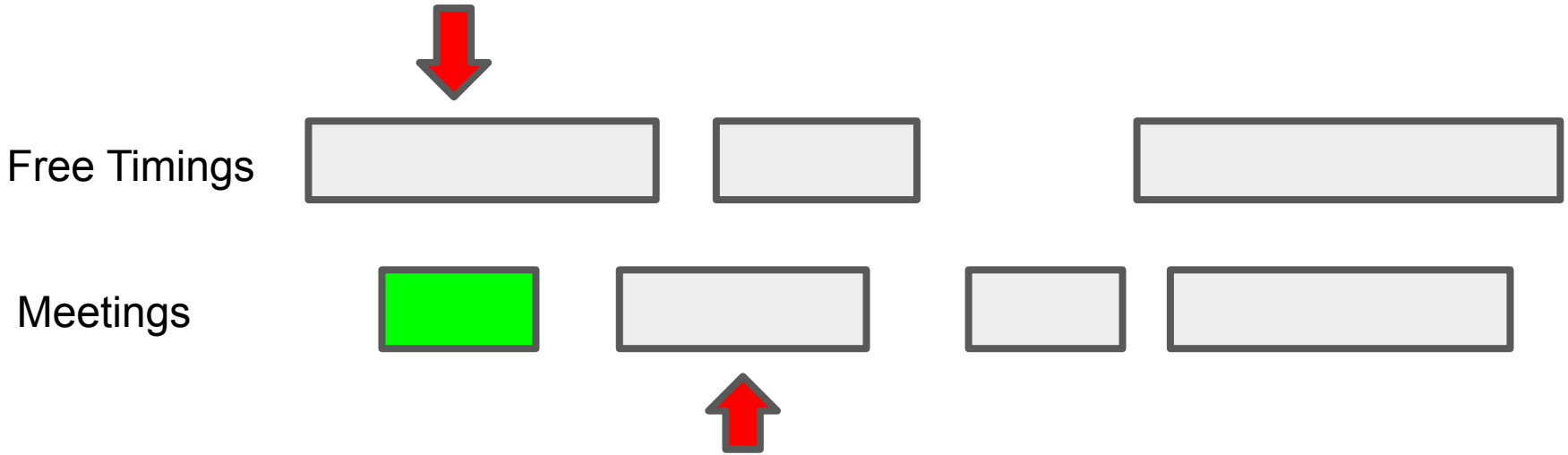
Sort the intervals by their start times

With 2 pointers, perform a merge routine on both list



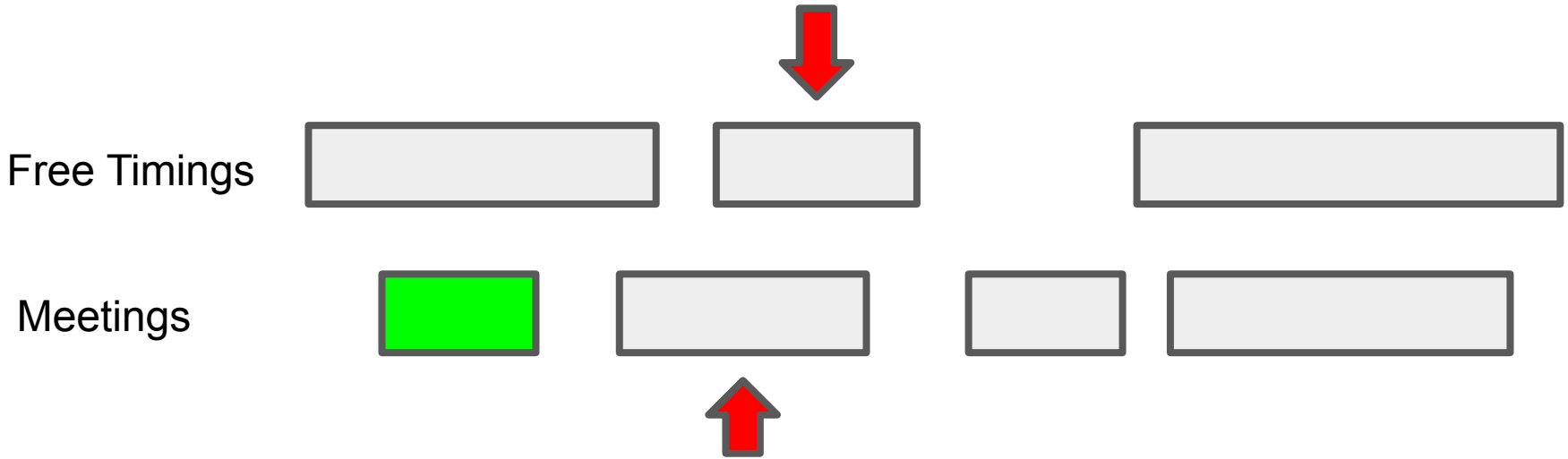
# Q7) Meetings, Part 1, O(NlogN + MlogM)

If it fits, keep that meeting interval, increment the meeting pointer



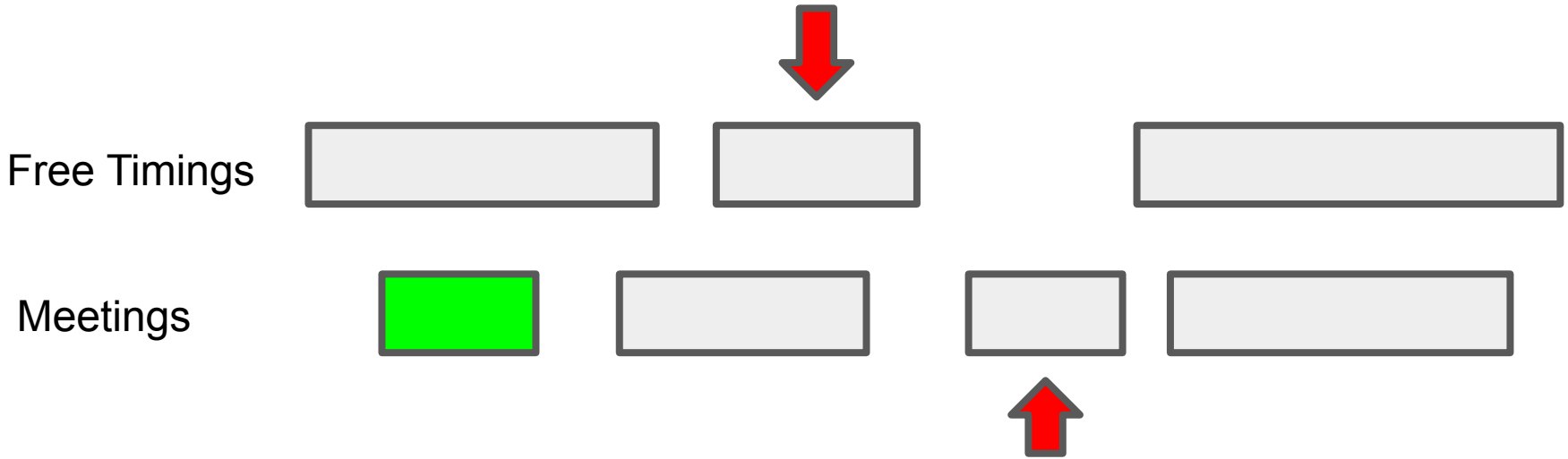
# Q7) Meetings, Part 1, O(NlogN + MlogM)

If it doesn’t fit, and the meeting time starts later than the free interval, increment the first pointer

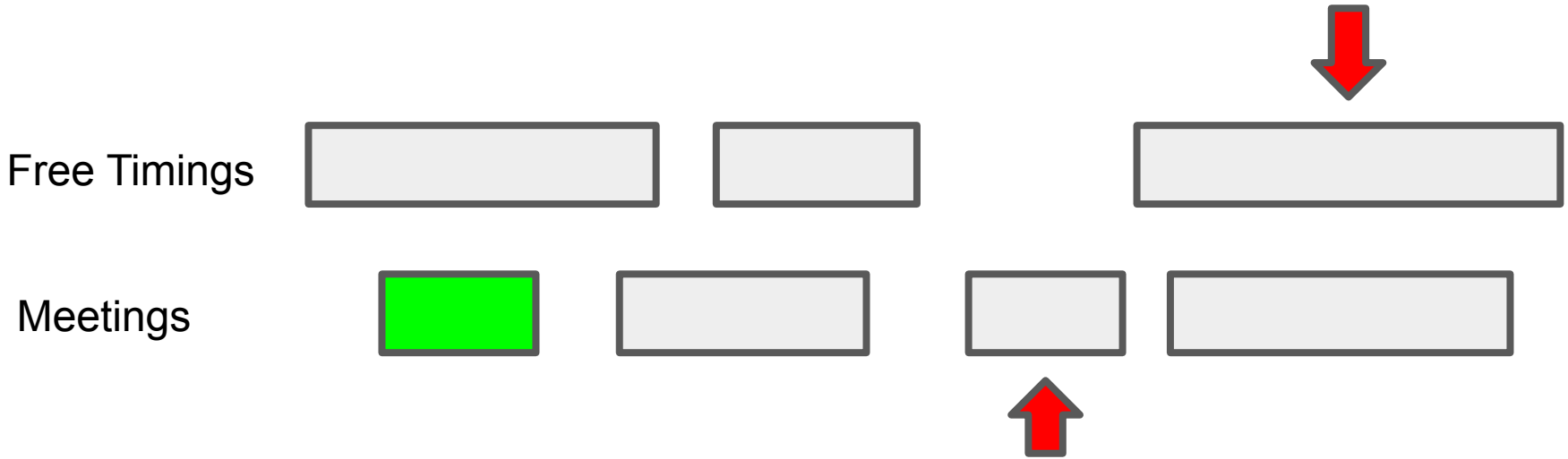


# Q7) Meetings, Part 1, O(NlogN + MlogM)

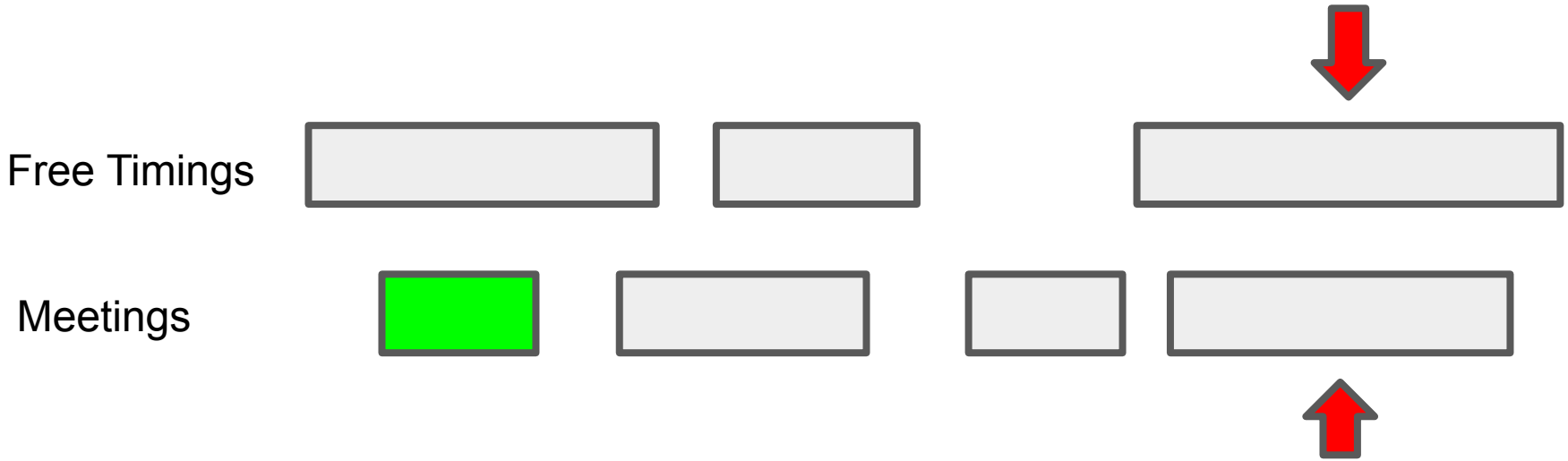
If it doesn’t fit, and the meeting time starts earlier than the free interval, increment the second pointer



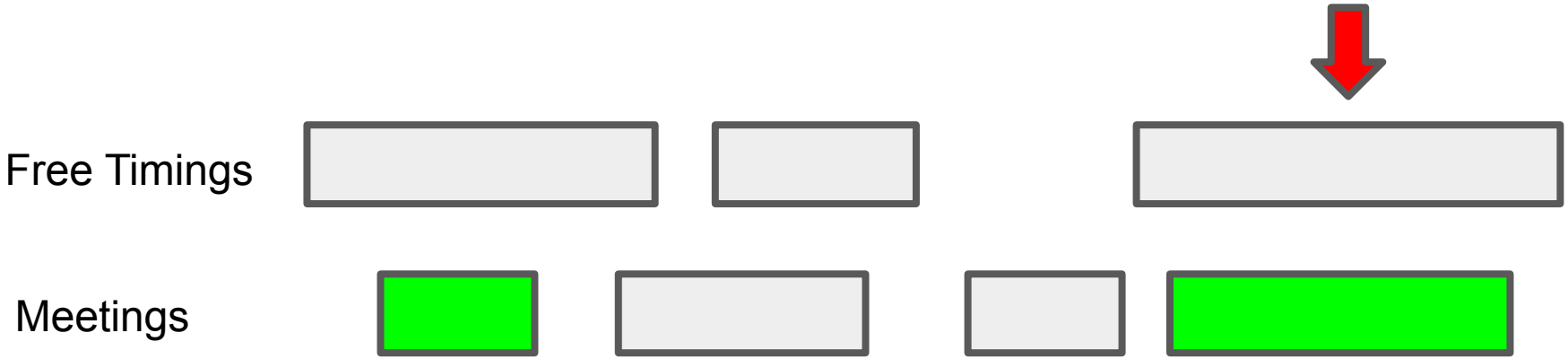
# Q7) Meetings, Part 1, O(NlogN + MlogM)



# Q7) Meetings, Part 1, O(NlogN + MlogM)

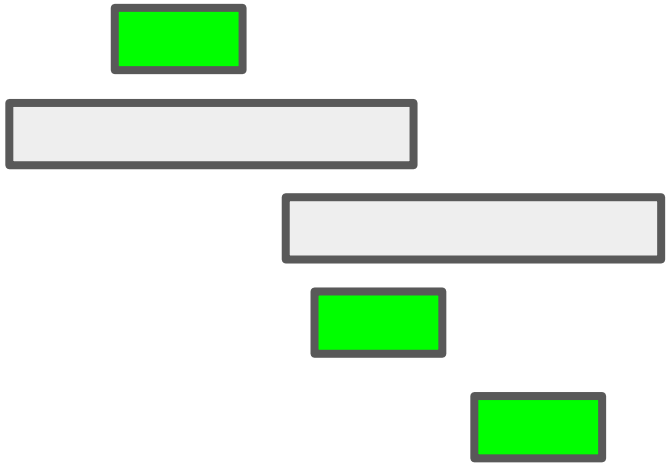


# Q7) Meetings, Part 1, O(NlogN + MlogM)



# Q7) Meetings, Part 2

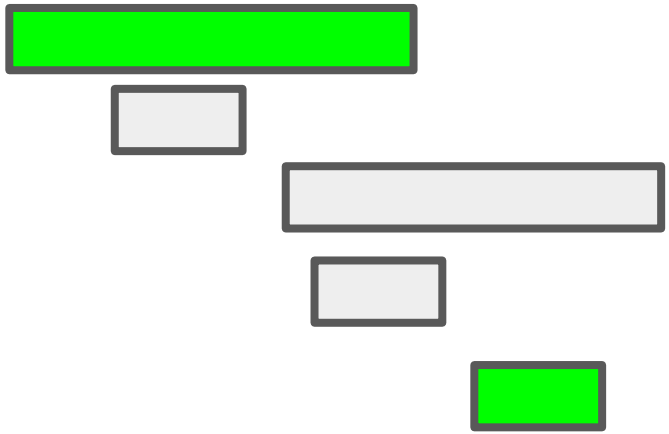
Given a list of possibly overlapping intervals, maximise the number of pair-wise disjoint intervals.



# Q7) Meetings, Part 2, wrong answer

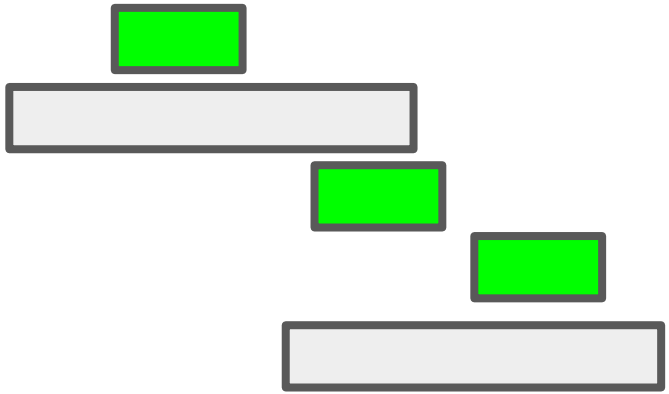
Sort by start time, greedily pick the next available meeting

**Wrong answer**



# Q7) Meetings, Part 2

Sort by **end** time, greedily pick the next available meeting



# Q7) Meetings, Part 2

Intuition: Changing the choice of a meeting to one that ends later is never a good idea