M1830 Lazy Tutor
Anson Ho
Simplified statement

• Given a list of binary numbers.

• In each query, find the number of submasks of a given binary mask in the original list.

• $01110_2$, $01010_2$ and $00000_2$ are submasks of $01110_2$
• $01111_2$, $11010_2$ and $11111_2$ are not
Prerequisite

• bitwise operation

• https://en.wikipedia.org/wiki/Bitwise_operation
Solution

• $dp[i][j] =$ number of submasks of $j$ such that the bits other than the first $i$ bits are fixed

• counted = X, not counted = X

<table>
<thead>
<tr>
<th>j \ submask</th>
<th>0 (first i bits)</th>
<th>1 (first i bits)</th>
<th>0 (last N - i bits)</th>
<th>1 (last N - i bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Solution

• initialization (i = 0):
  • \( \text{dp}[0][j] = \text{number of } j \text{ in the input list} \)

• transition (1 ≤ i ≤ N):
  • “unfixing” the \( i^{\text{th}} \) bit
    • if the \( i^{\text{th}} \) bit of \( j \) is not set
      • \( \text{dp}[i][j] = \text{dp}[i - 1][j] \)
    • if the \( i^{\text{th}} \) bit of \( j \) is set
      • \( \text{dp}[i][j] = \text{dp}[i - 1][j] + \text{dp}[i - 1][j - 2^i] \)

• output
  • \( \text{dp}[N][y] \) if the mask in the query is \( y \)
  • \( i = N \) means no bits are fixed
The end
Play with Lines
Naive solution

- Brute force to fix two points
- A line can be fixed by fixing 2 points
- For each line, find is it cover at least V points

- Time complexity O(N^3)
Randomize solution

• Note that \( p \) is at least 20
• **Randomly** fixing two points to fix a line
• If such a line exists
  • Probability of finding such line if it exists = \( \text{Probability(both points line on that line)} \)
  • \( \text{Probability(both points line on that line)} \) \( \geq 0.2 \times 0.2 = 0.04 \)
• Repeat the process 200 times
• Probability of finding such line if it exists = \( 1 - 0.96^{200} = 99.97\% \)
• Able to pass all test data
Randomize solution

• Also able to fix 1 point and brute force all line passing through the it
• Prob of find such line = 0.2 in this method
• Repeat 100 times, probability to find such line = 1 – 0.8^{100} = 1

• Time complexity: \( O(\text{Constant} * N) \) where constant is around 200
M1837 Grid Exploration

Alex Tung
alex20030190@yahoo.com.hk

14 April 2018
Some thoughts...

- You may wonder: why move for $M$ days, instead of $M'$ seconds?

Let total displacement after one cycle be $(dx, dy)$.

Since $N \times N \geq N$, we have:

**Observation**

Let $S_1$ and $S_2$ be length-$N$ instructions with one-cycle displacements $(dx_1, dy_1)$ and $(dx_2, dy_2)$, respectively. If $(dx_1, dy_1) = (dx_2, dy_2)$, then the instructions lead the robot to different final positions after $N \times N$ seconds.

So the first step of the solution is to enumerate all $O(N^2)$ valid one-cycle displacements.
Some thoughts...

- You may wonder: why move for $M$ days, instead of $M'$ seconds?
- It is crucial that the robot moves *for a long time***!!!
Some thoughts...

- You may wonder: why move for $M$ days, instead of $M'$ seconds?
- It is crucial that the robot moves *for a long time!!!*
- Let total displacement after one cycle be $(dx, dy)$. 
You may wonder: why move for $M$ days, instead of $M'$ seconds?

It is crucial that the robot moves *for a long time***!

Let total displacement after one cycle be $(dx, dy)$.

Since $86400 \times M \geq N^2$, we have:

**Observation**

Let $S_1$ and $S_2$ be length-$N$ instructions with one-cycle displacements $(dx_1, dy_1)$ and $(dx_2, dy_2)$, respectively. If $(dx_1, dy_1) \neq (dx_2, dy_2)$, then the instructions lead the robot to different final positions after $86400 \times M$ seconds.
You may wonder: why move for \( M \) days, instead of \( M' \) seconds?

It is crucial that the robot moves \textit{for a long time}!!!

Let total displacement after one cycle be \((dx, dy)\).

Since \( 86400 \times M \geq N^2 \), we have:

**Observation**

Let \( S_1 \) and \( S_2 \) be length-\( N \) instructions with one-cycle displacements \((dx_1, dy_1)\) and \((dx_2, dy_2)\), respectively.

If \((dx_1, dy_1) \neq (dx_2, dy_2)\), then the instructions lead the robot to different final positions after \( 86400 \times M \) seconds.

So the first step of the solution is to enumerate all \( O(N^2) \) valid one-cycle displacements.
So we now narrow down to all instructions with one-cycle displacement \((dx, dy)\).
So we now narrow down to all instructions with one-cycle displacement \((dx, dy)\).

Let \(REM := (86400 \times M) \mod N\). Clearly we should just consider the prefix \(S[0..(REM - 1)]\).
So we now narrow down to all instructions with one-cycle displacement \((dx, dy)\).

Let \(REM := (86400 \times M) \mod N\). Clearly we should just consider the prefix \(S[0..(REM - 1)]\).

Reformulated problem:

**Problem**

Given \(REM, N, dx, \) and \(dy\). Consider all length-\(N\) instructions with one-cycle displacement \((dx, dy)\). Count the number of possible final positions after executing just the first \(REM\) instructions.
Solution

- So we now narrow down to all instructions with one-cycle displacement \((dx, dy)\).
- Let \(REM := (86400 \times M) \mod N\). Clearly we should just consider the prefix \(S[0..(REM - 1)]\).
- Reformulated problem:

**Problem**

Given \(REM, N, dx,\) and \(dy\).
Consider all length-\(N\) instructions with one-cycle displacement \((dx, dy)\).
Count the number of possible final positions after executing just the first \(REM\) instructions.

- By “simple arithmetic”, this problem can be solved in \(O(1)\).
So we now narrow down to all instructions with one-cycle displacement $(dx, dy)$.

Let $REM := (86400 \times M) \mod N$. Clearly we should just consider the prefix $S[0..(REM - 1)]$.

Reformulated problem:

**Problem**

Given $REM$, $N$, $dx$, and $dy$.

Consider all length-$N$ instructions with one-cycle displacement $(dx, dy)$. Count the number of possible final positions after executing just the first $REM$ instructions.

- By “simple arithmetic”, this problem can be solved in $O(1)$.
- So the overall time complexity is $O(N^2)$ per case.
Problem

Given $REM$, $N$, $dx$, and $dy$.
Consider all length-$N$ instructions with one-cycle displacement $(dx, dy)$.
Can $(x, y)$ be reached after $REM$ steps?

$-REM \leq x + y \leq REM$,
$-REM \leq x - y \leq REM$,
$dx + dy - (N - REM) \leq x + y \leq dx + dy + (N - REM)$,
$dx - dy - (N - REM) \leq x - y \leq dx - dy - (N - REM)$,
and $x + y$ and $REM$ have the same parity.
Problem

Given \( REM, N, dx, \) and \( dy. \)
Consider all length-\( N \) instructions with one-cycle displacement \((dx, dy)\).
Can \((x, y)\) be reached after \( REM \) steps?

\((x, y)\) can be reached if and only if:

- \(-REM \leq x + y \leq REM,\)
- \(-REM \leq x - y \leq REM,\)
- \(dx + dy - (N - REM) \leq x + y \leq dx + dy + (N - REM),\)
- \(dx - dy - (N - REM) \leq x - y \leq dx - dy - (N - REM),\) and
- \(x + y \) and \( REM \) have the same parity.
Here is a (rather clumsy, in hindsight) way to count the number of \((x, y)\).

Let \(PARITY := \text{REM} \mod 2\).

Let \(u := \frac{x+y-\text{PARITY}}{2}\) and \(v := \frac{x-y-\text{PARITY}}{2}\).

The constraints are of the form

- \(L_1 \leq 2u \leq R_1\)
- \(L_2 \leq 2v \leq R_2\)

for some \(L_1, R_1, L_2, R_2\).

**The Point**

Each solution \((u, v)\) corresponds to a solution \((x, y)\).

Plus, the above system is super easy to solve :)

Alex Tung

M1837 Tutorial
We can split the instruction into two parts.

Operations in the first part will be executed \( \left\lfloor \frac{86400 \times M}{N} + 1 \right\rfloor \) times.
(There are \( 86400 \times M \mod N \) such operations.)

Those in the second part will be executed \( \left\lfloor \frac{86400 \times M}{N} \right\rfloor \) times.
(There are \( N - (86400 \times M \mod N) \) such operations.)
We can split the instruction into two parts.

Operations in the first part will be executed \( \left\lfloor \frac{86400 \times M}{N} + 1 \right\rfloor \) times. (There are \( 86400 \times M \mod N \) such operations.)

Those in the second part will be executed \( \left\lfloor \frac{86400 \times M}{N} \right\rfloor \) times. (There are \( N - (86400 \times M \mod N) \) such operations.)

There is a bijection between the set of valid final positions and pairs of valid first-part and second-part displacements \((d_1, d_2)\). \((d_1 \text{ and } d_2 \text{ are vectors in } \mathbb{Z}^2; \text{ there are actually four coordinates in } (d_1, d_2)\).)
We can split the instruction into two parts.

Operations in the first part will be executed \( \lfloor \frac{86400 \times M}{N} + 1 \rfloor \) times. (There are \( 86400 \times M \mod N \) such operations.)

Those in the second part will be executed \( \lfloor \frac{86400 \times M}{N} \rfloor \) times. (There are \( N - (86400 \times M \mod N) \) such operations.)

There is a bijection between the set of valid final positions and pairs of valid first-part and second-part displacements \((d_1, d_2)\). \(d_1\) and \(d_2\) are vectors in \(\mathbb{Z}^2\); there are actually four coordinates in \((d_1, d_2)\).

Clearly, given valid \((d_1, d_2)\), one can get a valid final position.

Moreover, if \((d_1, d_2) \neq (d'_1, d'_2)\), then they give rise to different final positions. (It has to do with \(86400 \times M \geq N\).)

The Point

The first and second parts are independent.
Let $f(i)$ be the number of possible final positions after $i$ moves from the origin.

Then the answer is $f(R) \times f(N - R)$, where $R := 86400 \times M \mod N$. (Prove it!)
Let $f(i)$ be the number of possible final positions after $i$ moves from the origin.

Then the answer is $f(R) \times f(N - R)$, where $R := 86400 \times M \mod N$.

It turns out that $f(i) = (i + 1)^2$. (Prove it!)

Kudos to Charlie for finding such an elegant $O(1)$ solution!
Part 1: Ideal Turret Position

- Let $P$ be the turret position.
Part 1: Ideal Turret Position

- Let $P$ be the turret position.
- You can hit bubble $i$ if and only if segment $i$ and the ball trajectory intersect.
Part 1: Ideal Turret Position

- Let $P$ be the turret position.
- You can hit bubble $i$ if and only if segment $i$ and the ball trajectory intersect.
- That is, $l[i] \leq P + dx \times y[i] \leq r[i]$.
Part 1: Ideal Turret Position

- Let $P$ be the turret position.
- You can hit bubble $i$ if and only if segment $i$ and the ball trajectory intersect.
- That is, $l[i] \leq P + dx \times y[i] \leq r[i]$.
- Rewrite it as $l[i] - dx \times y[i] \leq P \leq r[i] - dx \times y[i]$. 
Let $P$ be the turret position.

You can hit bubble $i$ if and only if segment $i$ and the ball trajectory intersect.

That is, $l[i] \leq P + dx \times y[i] \leq r[i]$.

Rewrite it as $l[i] - dx \times y[i] \leq P \leq r[i] - dx \times y[i]$.

Think of it as adding a score of $v[i]$ to all $P$ in the range.
Part 1: Ideal Turret Position

- Let $P$ be the turret position.
- You can hit bubble $i$ if and only if segment $i$ and the ball trajectory intersect.
- That is, $l[i] \leq P + dx \times y[i] \leq r[i]$.
- Rewrite it as $l[i] - dx \times y[i] \leq P \leq r[i] - dx \times y[i]$.
- Think of it as adding a score of $v[i]$ to all $P$ in the range.
- Range update $\rightarrow$ difference “array” :)

Alex Tung  
M1838 Tutorial  
14 Apr 18  
2 / 6
Difference Array

To add $v[i]$ to all $P$ s.t. $A \leq P \leq B$, we can add $v[i]$ to $diff[A]$ and $-v[i]$ to $diff[B + 1]$.
Difference Array

- To add $v[i]$ to all $P$ s.t. $A \leq P \leq B$, we can add $v[i]$ to $\text{diff}[A]$ and $-v[i]$ to $\text{diff}[B + 1]$

- Partial sum the $\text{diff}[]$ array to get the highest obtainable score for each turret position.
To add $v[i]$ to all $P$ s.t. $A \leq P \leq B$, we can add $v[i]$ to $\text{diff}[A]$ and $-v[i]$ to $\text{diff}[B + 1]$.

Partial sum the $\text{diff}[]$ array to get the highest obtainable score for each turret position.

In this task, coordinates are too large. So we need to store all $(\text{endpoint}, \text{delta})$, then sort them, to do partial sum.
Difference Array

- To add \( v[i] \) to all \( P \) s.t. \( A \leq P \leq B \), we can add \( v[i] \) to \( \text{diff}[A] \) and \( -v[i] \) to \( \text{diff}[B+1] \).
- Partial sum the \( \text{diff}[] \) array to get the highest obtainable score for each turret position.
- In this task, coordinates are too large. So we need to store all \( (\text{endpoint}, \text{delta}) \), then sort them, to do partial sum.
- Specifically, we store \( (l[i] - dx \times y[i], v[i]) \) and \( (r[i] - dx \times y[i] + 1, -v[i]) \) for each \( i \).
Difference Array

- To add $v[i]$ to all $P$ s.t. $A \leq P \leq B$, we can add $v[i]$ to $\text{diff}[A]$ and $-v[i]$ to $\text{diff}[B + 1]$
- Partial sum the $\text{diff}[]$ array to get the highest obtainable score for each turret position.
- In this task, coordinates are too large. So we need to store all $(\text{endpoint}, \text{delta})$, then sort them, to do partial sum.
- Specifically, we store $(l[i] - dx \times y[i], v[i])$ and $(r[i] - dx \times y[i] + 1, -v[i])$ for each $i$.
- To force the turret position to be in $[L, R]$, simply add $(L, X)$ and $(R + 1, -X)$ to the list.
Does it work?

- NO!!!
Does it work?

- NO!!!
- You must not add \((endpoint, \delta)\) corresponding to negative \(v[i]\)s.
If the bubble cannot be hit no matter what, just output anything valid.
Part 2: Bubble Configuration

- If the bubble cannot be hit no matter what, just output anything valid.
- For simplicity, assume $l[i] = 0$, and write $r$ for $r[i]$, $y$ for $y[i]$. 

Say the ball will visit $(x, y)$ for some $0 \leq x \leq r$. Number from to $r$ the states $(0, \text{right})$, $(1, \text{right})$, ..., $(r−1, \text{right})$, $(r, \text{left})$, $(r−1, \text{left})$, ... , $(1, \text{left})$.

After one second, bubble state changes from $S$ to $(S + 1) (m d r)$. We want the bubble to be at state $x$ at time $y$. So it should be at $x - y \mod r$ at time 0.

Beware: if $v[i] < 0$, we actually don’t want it to be at state $x$ at time $y$. 

Alex Tung
M1838 Tutorial
14 Apr 18
Part 2: Bubble Configuration

- If the bubble cannot be hit no matter what, just output anything valid.
- For simplicity, assume \( l[i] = 0 \), and write \( r \) for \( r[i] \), \( y \) for \( y[i] \).
- Say the ball will visit \((x, y)\) for some \(0 \leq x \leq r\).
Part 2: Bubble Configuration

- If the bubble cannot be hit no matter what, just output anything valid.
- For simplicity, assume $l[i] = 0$, and write $r$ for $r[i]$, $y$ for $y[i]$.
- Say the ball will visit $(x, y)$ for some $0 \leq x \leq r$.
- Number from 0 to $2r - 1$ the states $(0, \text{right}), (1, \text{right}), \ldots, (r-1, \text{right}), (r, \text{left}), (r-1, \text{left}), \ldots, (1, \text{left})$. 
Part 2: Bubble Configuration

- If the bubble cannot be hit no matter what, just output anything valid.
- For simplicity, assume $l[i] = 0$, and write $r$ for $r[i]$, $y$ for $y[i]$.
- Say the ball will visit $(x, y)$ for some $0 \leq x \leq r$.
- Number from 0 to $2r - 1$ the states
  $(0, right), (1, right), \ldots, (r-1, right), (r, left), (r-1, left), \ldots, (1, left)$.
- After one second, bubble state changes from $S$ to $(S + 1) \pmod{2r}$.
Part 2: Bubble Configuration

- If the bubble cannot be hit no matter what, just output anything valid.
- For simplicity, assume \( l[i] = 0 \), and write \( r \) for \( r[i] \), \( y \) for \( y[i] \).
- Say the ball will visit \((x, y)\) for some \( 0 \leq x \leq r \).
- Number from 0 to \( 2r - 1 \) the states \((0, right), (1, right), \ldots, (r-1, right), (r, left), (r-1, left), \ldots, (1, left)\).
- After one second, bubble state changes from \( S \) to \((S + 1) \pmod{2r}\).
- We want the bubble to be at state \( x \) at time \( y \). So it should be at \( x - y \pmod{2r} \) at time 0.
Part 2: Bubble Configuration

- If the bubble cannot be hit no matter what, just output anything valid.
- For simplicity, assume \( l[i] = 0 \), and write \( r \) for \( r[i] \), \( y \) for \( y[i] \).
- Say the ball will visit \((x, y)\) for some \(0 \leq x \leq r\).
- Number from 0 to \(2r - 1\) the states
  \((0, \text{right}), (1, \text{right}), \ldots , (r-1, \text{right}), (r, \text{left}), (r-1, \text{left}), \ldots , (1, \text{left})\).
- After one second, bubble state changes from \(S\) to \((S + 1) \pmod{2r}\).
- We want the bubble to be at state \(x\) at time \(y\). So it should be at \(x - y \pmod{2r}\) at time 0.
- Beware: if \(v[i] < 0\), we actually \textbf{don’t want} it to be at state \(x\) at time \(y\).
First we find the ideal turret position.
First we find the ideal turret position.

This is done by storing \((\text{endpoint}, \delta)\) and sorting them.
A Short Summary

- First we find the ideal turret position.
- This is done by storing \((\text{endpoint}, \text{delta})\) and sorting them.
- Next we decide the initial configuration of the bubbles.
A Short Summary

- First we find the ideal turret position.
- This is done by storing \((\text{endpoint}, \text{delta})\) and sorting them.
- Next we decide the initial configuration of the bubbles.
- It can be calculated easily by observing the periodic behaviour.
First we find the ideal turret position. This is done by storing $(endpoint, delta)$ and sorting them.

Next we decide the initial configuration of the bubbles. It can be calculated easily by observing the periodic behaviour.

Overall time complexity: $O(N\log N)$. 

First we find the ideal turret position. This is done by storing \((\text{endpoint}, \delta)\) and sorting them.

Next we decide the initial configuration of the bubbles. It can be calculated easily by observing the periodic behaviour.

Overall time complexity: \(O(N \log N)\).

Be careful when \(v[i] < 0\).
No sort, no search solution

Lau Chi Yung

2018/04/14
Problem

- write 13 3
- write 15 5
- write 15 7
- ask 14 2 8
- ask 15 2 8
- write 16 5
- ask 15 2 8

```
  1 2 3 4 5 6 7 8 9
13
```

Problem

- write 13 3
- **write 15 5**
- write 15 7
- ask 14 2 8
- ask 15 2 8
- write 16 5
- ask 15 2 8

```
1 2 3 4 5 6 7 8 9
```

```
13 15
```

```
 1  2  3  4  5  6  7  8  9
```
Problem

- write 13 3
- write 15 5
- write 15 7
- ask 14 2 8
- ask 15 2 8
- write 16 5
- ask 15 2 8
Problem

- write 13 3
- write 15 5
- write 15 7
- ask 14 2 8
- ask 15 2 8
- write 16 5
- ask 15 2 8

14 does not exist
Problem

- write 13 3
- write 15 5
- write 15 7
- ask 14 2 8
- ask 15 2 8
- write 16 5
- ask 15 2 8

15 exists
Problem

- write 13 3
- write 15 5
- write 15 7
- ask 14 2 8
- ask 15 2 8
- write 16 5
- ask 15 2 8
Problem

- write 13 3
- write 15 5
- write 15 7
- ask 14 2 8
- ask 15 2 8
- write 16 5
- ask 15 2 8

![Diagram showing not sorted numbers: 1 2 3 4 5 6 7 8 9 with selected numbers 13, 16, 15.]

not sorted
Problem

- Given a range, determine if it is sorted
- If it is sorted, find if a number exists
  - binary search with empty slots
Determine if a range is sorted

- At any time, we remember all disjoint sorted sub-ranges
- When we write to a slot, only the sub-ranges near that slot will have to be modified

\[
\begin{array}{cccccccc}
10 & 11 & 13 & 5 & 4 & 7 & 15 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
\end{array}
\]
Determine if a range is sorted

- At any time, we remember all disjoint sorted sub-ranges
- When we write to a slot, only the sub-ranges near that slot will have to be modified

\[
\left[ L, R \right] \text{ is sorted if and only if } L \text{ and } R \text{ lie in the same sub-range}
\]

Implementation: record the right boundaries of the ranges in a set::<int>, i.e.,

\[ \text{Time complexity: } O(N \log N) \]
Determine if a range is sorted

- At any time, we remember all disjoint sorted sub-ranges
- When we write to a slot, only the sub-ranges near that slot will have to be modified
Determine if a range is sorted

- At any time, we remember all disjoint sorted sub-ranges
- When we write to a slot, only the sub-ranges near that slot will have to be modified
- $[L, R]$ is sorted if and only if $L$ and $R$ lie in the same sub-range
Determine if a range is sorted

- At any time, we remember all disjoint sorted sub-ranges.
- When we write to a slot, only the sub-ranges near that slot will have to be modified.
- \([L, R]\) is sorted if and only if \(L\) and \(R\) lie in the same sub-range.
- Implementation: record the *right boundaries* of the ranges in a set:::<int>, i.e. \(\{2, 5, 9\}\).

Time complexity: \(O(N \log N)\)
Binary search with empty slots

1. Shrink \([L, R]\) to \([L', R']\) such that both boundaries are not empty
2. Calculate \(M = \frac{L' + R'}{2}\)
3. Calculate \(M' = \text{the nearest slot to } M\) that is not empty
4. Set \(L' = M'\) or \(R' = M'\), and recurse the process
5. (you should handle many index ±1 special cases)
6. With \(\text{std::map<int, int>}\), time complexity = \(O(N \log^2 N)\)
Binary search with empty slots

1. Shrink $[L, R]$ to $[L', R']$ such that both boundaries are not empty
2. Calculate $M = \frac{L' + R'}{2}$
3. Calculate $M' = \text{the nearest slot to } M$ that is not empty
4. Set $L' = M'$ or $R' = M'$, and recurse the process
5. (you should handle many index ±1 special cases)
6. With `std::map<int, int>`, time complexity = $O(N \log^2 N)$
Binary search with empty slots

1. Shrink \([L, R]\) to \([L', R']\) such that both boundaries are not empty
2. Calculate \(M = \frac{L' + R'}{2}\)
3. Calculate \(M' = \) the nearest slot to \(M\) that is not empty
4. Set \(L' = M'\) or \(R' = M'\), and recurse the process
5. (you should handle many index ±1 special cases)
6. With \texttt{std::map<int, int>}, time complexity = \(O(N \log^2 N)\)
Binary search with empty slots

1. Shrink \([L, R]\) to \([L', R']\) such that both boundaries are not empty
2. Calculate \(M = \frac{L' + R'}{2}\)
3. Calculate \(M' = \text{the nearest slot to } M \text{ that is not empty}\)
4. Set \(L' = M'\) or \(R' = M'\), and recurse the process
5. (you should handle many index ±1 special cases)
6. With \(\text{std::map<int, int>}\), time complexity = \(O(N \log^2 N)\)

\[
\begin{array}{ccccccccc}
L & L' & M' & M & R' & R \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{array}
\]
Binary search with empty slots

1. Shrink \([L, R]\) to \([L', R']\) such that both boundaries are not empty
2. Calculate \(M = \frac{L' + R'}{2}\)
3. Calculate \(M' = \text{the nearest slot to } M \text{ that is not empty}\)
4. Set \(L' = M'\) or \(R' = M'\), and recurse the process
5. (you should handle many index \(\pm 1\) special cases)
6. With \texttt{std::map<int, int>} , time complexity = \(O(N \log^2 N)\)

\[
\begin{array}{cccccccc}
 L & L' & M' & M & R' & R \\
 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]
Overall time complexity

\[ O(N \log^2 N) \]