# Google Farewell Round 2$^{\text{nd}}$ Training

## CS Chapter IEEE NTUA SB

### April 6, 2023

## Combination Lock

A combination lock has $W$ wheels, each of which has the integer values 1 through $N$ on it, in ascending order. At any moment, each wheel shows a specific value on it. $X_i$ is the initial value shown on the $i$-th wheel. You can use a single move to change a wheel from showing the value $X$ to showing either $X + 1$ or $X - 1$, wrapping around between 1 and $N$. For example, if a wheel currently shows the value 1, in one move you can change its value to 2 or $N$. Given all wheels' initial values, what is the minimum number of moves to get all wheels to show the same value?

**Input:** The first line of the input gives the number of test cases, $T$. $T$ test cases follow. The first line of each test case contains the two integers $W$ and $N$. The second line contains $W$ integers. The $i$-th integer is $X_i$.

**Output:** For each test case, output one line containing Case #$x$: $y$, where $x$ is the test case number (starting from 1) and $y$ is the minimum number of moves to get all wheels to show the same value.

| Limitations | Sample Input | Sample Output |
|---|---|---|
| Time limit: 40 seconds. | 2 | Case #1: 2 |
| Memory limit: 1 GB. | 3 5 | Case #2: 8 |
| $1 \le T \le 100$ | 2 3 4 | |
| $1 \le X_i \le N$ | 4 10 | |
| | 2 9 3 8 | |

*Test Set 1:*
$1 \le W \le 1000$
$1 \le N \le 1000$

*Test Set 2:*
$1 \le W \le 1000$
$1 \le N \le 10^9$

*Test Set 3:*
$1 \le W \le 10^5$
$1 \le N \le 10^9$

In Sample Case #1, the best solution is to get all wheels to show value 3, which would take a total of 2 moves: the first wheel would move once (from value 2 to value 3), the second wheel would not move (it already shows value 3), and the third wheel would move once (from value 4 to value 3). For reference, it would take 5 moves to get all wheels to show value 1, 3 moves to get all wheels to show value 2, 3 moves to get all wheels to show value 4, and 5 moves to get all wheels to show value 5. In Sample Case #2, the best solutions are to get all wheels to show either value 1, 2, 9 or 10, which would take a total of 8 moves.

We can see that if we decide a final value at which all wheels should be in the end, moves for each wheel to reach that value are independent of the moves performed on other wheels. Thus, we can calculate number of moves for each wheel separately. Consider a wheel $i$ which is at value $x$ currently. We want the wheel to finally reach the value y. There are 2 cases here:

- Case 1: $x \leq y$. Increasing the value of the wheel $i$ would take $y - x$ steps. Decreasing the value of wheel $i$ would take $N - y + x$ steps. Hence, the minimum number of moves in this case would be minimum of $y - x$ and $N - y + x$.

- Case 2: $x > y$. Decreasing the value of the wheel $i$ would take $x - y$ steps. Increasing the value of wheel i would take $N + y - x$ steps. Hence, the minimum number of moves in this case would be minimum of $x - y$ and $N + y - x$.

**Test Set 1**

We can solve this test set by trying all possible $N$ values that all the wheels should have in the end. For each value, we calculate the total number of moves to get all of the wheels to this value using Case 1 and Case 2. The answer is the minimum possible moves performed over all such $X$. There are $N$ possible values and for each value, we perform $O(W)$ operations. Thus, the time complexity of the solution would be $O(W \cdot N)$.

**Test Set 2**

A major observation is that we could always get the minimum possible moves by finally bringing all the wheels to one of the initial values of the given wheels. We can prove this by considering a value $V$ which is not among the initial values of wheels and then showing that we can get a same or better answer by moving all the wheels to one of the initial values. Consider the initial values of the wheels in the sorted order. et the index of the wheel with smallest initial value greater than $V$ be $j$, if no such value exists, then we can consider $j = 1$ as it is first wheel next to value $V$ in cyclic order. Let the index of wheel with largest initial value smaller than $V$ be $i$, if no such value exists, then we can consider $i = W$ as it is the first wheel before value $V$ in cyclic order. Now to reach the value $V$, each value will either reach $X_i$ or $X_j$. Now say there are $y$ wheels at value $X_i$ currently which leaves us with $W - y$ wheels at value $X_j$. The number of moves for all wheels to reach value $V$ would be $y \cdot (V - X_i) + (W - y) \cdot (X_j - V)$. We can get the same or better result than this.

- $y \leq W - y$. If we choose to bring all the wheels to value $X_j$, we will have number of moves as $y \cdot (V - X_i) + $y$ \cdot (X_j - V)$, which is never large than the number of moves required for all wheels to reach value $V$. Hence, we have a better answer if we bring all the wheels to value $X_j$.

- $y \leq W - y$. If we choose to bring all the wheels to value $X_i$, we will have number of moves as $(W - y) \cdot (V - X_i) + (W - y) \cdot (X_j - V)$, which is never large than the number of moves required for all wheels to reach value $V$. Hence, we have a better answer if we bring all the wheels to value $X_i$.

Now instead of trying all possible values from 1 to $N$, we would try the initial values of the wheels and calculate moves required for all wheels to reach that value. For each value, we take $O(W)$ time to calculate the moves required for all wheels to reach that value. There are $W$ values. Hence the complexity is $O(W^2)$.

The implementation of the above algorithm on c++ can be found in maximumCoins.cpp file.

**Test Set 3**

We have already proved that we only need to consider one of the initial values of the wheels. We need to calculate the number of moves required for all wheels to reach a particular value efficiently. We can do the following. Sort the initial values of the wheels. Maintain a prefix sum array $Pre$. $Pre[i]$ denotes the sum of initial values of wheels from 1 to $i$. We define a method $GetSum(i, j)$ which gives the sum of values between indexes $i and j$. This can be caluated in $O(1)$ using $Pre$ array.

Suppose that currently we are caluating the number of moves required for all wheels to reach $X_i$. Consider any wheel $j$ from index 1 to $i$. Minimum number of moves required for wheel $j$ to reach value $X_i$ would

be minimum of $X_i - X_j$ and $N - X_i + X_j$. Consider 2 indexes $k$ and $l$ such that $1 \leq k < l \leq i$. We can prove that it is not possible to have $X_i - X_k < N - X_i + X_k$ and $N - X_i + X_l < X_i - X_l$ simultaneously. This is because if we add the two inequalities, we get $N - X_k + X_l < N + X_k - X_l$ which implies $X_l < X_k$. But it is not possible to have such condition because we know that $Xk \leq X_l$. Thus, we can say that there exists an index $p$ such that $1 \leq p \leq i$ and for each wheel $q$ such that $p \leq q \leq i$ will have minimum number of moves as $X_i - X_q$ and for each wheel $r$ such that $1 \leq r \leq p - 1$ will have minimum number of moves as $N - X_i + X_r$. We can find this index $p$ using binary search on wheels 1 to $i$. Now we need to find sum of $X_i - X_q$ for each $q$. This can be calculated in $O(1)$ by $(i - p + 1) \cdot X_i - GetSum(p, i)$. Now we need to find sum of $N - X_i + X_r$ for each $r$. This can be calculated in $O(1)$ by $(p - 1) \cdot (N - X_i) + GetSum(1, p - 1)$. We have calculated number of moves required for wheels 1 to $i$ to reach value $X_i$.

Similarly, we can say that there exists an index $b$ such that $i \leq b \leq W$ and for each wheel $c$ such that $i \leq c \leq b$ will have the minimum number of moves as $X_c - X_i$ and for each wheel d such that $b + 1 \leq d \leq W$ will have the minimum number of moves as $N - X_d + X_i$. We can find this index $b$ by using binary search on wheels 1 to $W$. Now we need to find sum of $X_c - X_i$ for each $c$. This can be calculated in $O(1)$ by $GetSum(i + 1, c) - (c - i) \cdot X_i$. Now we need to find sum of $N - X_d + X_i$ for each $d$. This can be calculated in $O(1)$ by $(W - b) \cdot (N + X_i) - GetSum(b + 1, W)$.

We have calculated the number of moves required for all wheels to reach a particular value in $O(\log W)$. Now we need to take the minimum over all such values. Thus we can calculate moves for all the initial values of all wheels in $O(W \cdot \log W)$ time complexity. Note that instead of using binary search to find indexes $p$ and $b$, we can use two pointers approach to find them. We can prove that indexes $p$ and $b$ will keep on increasing as we increment $i$. The overall complexity of the solution remains same due to the sorting part.