Wolf

Calvin Fung

August 3, 2025

1 Problem

Wolf has found n sheep with tastiness values $p_1, p_2, \ldots, p_n$ where p is a permutation*. Wolf wants to perform binary search on p to find the sheep with tastiness of k, but p may not necessarily be sorted. The success of binary search on the range $[l, r]$ for k is represented as $f(l, r, k)$, which is defined as follows:

If $l > r$, then $f(l, r, k)$ fails. Otherwise, let $m = \lfloor \frac{l+r}{2} \rfloor$, and:

- If $p_m = k$, then $f(l, r, k)$ is **successful**,
- If $p_m < k$, then $f(l, r, k) = f(m+1, r, k)$,
- If $p_m > k$, then $f(l, r, k) = f(l, m-1, k)$.

Cow the Nerd decides to help Wolf out. Cow the Nerd is given q queries, each consisting of three integers l, r, and k. Before the search begins, Cow the Nerd may choose a non-negative integer d, and d indices $1 \le i_1 < i_2 < \ldots < i_d \le n$ where $p_{i_j} \ne k$ over all $1 \le j \le d$. Then, he may re-order the elements $p_{i_1}, p_{i_2}, \ldots, p_{i_d}$ however he likes.

For each query, output the **minimum** integer d that Cow the Nerd must choose so that $f(l, r, k)$ can be **successful**, or report that it is impossible. Note that the queries are independent and the reordering is not actually performed.

2 Solution

We present a $O(n + q \lg n)$ time solution where the input size is $n + q$.

2.1 analysis

Let a_k be the index of the sheep with tastiness values k in p. Given l, r, k, notice the indices $l, l+1, ..., r$ is sorted so we can do a binary search on it (not on p!) to find a_k e.g. in the first iteration, let $m = \lfloor \frac{l+r}{2} \rfloor$, if $a_k = m$ halt, else if $a_k < m$ recurse on $(l, m-1)$ otherwise recurse on $(m+1, r)$. Suppose $a_k \in [l, r]$, then binary search on $l, l+1, ..., r$ to find a_k recursed on a sequence of intervals $(l_1, r_1), ..., (l_q, r_q)$ with corresponding mid-points $m_i = \lfloor \frac{l_i + r_i}{2} \rfloor$ where $(l_1, r_1) = (l, r)$ and $m_q = a_k$.

Let $P(i)$ be the proposition that $f(l_i, r_i, k)$ is successful and either ($a_k < m_i$ and $k < p_{m_i}$) or ($a_k > m_i$ and $k > p_{m_i}$).

Suppose $P(i)$ holds and $i \le q - 2$ (so that $m_{i+1} \ne a_k$). Then we have 2 cases :

- $a_k < m_i$ and $k < p_{m_i}$: As $a_k < m_i$, by definition we have $l_{i+1} = l_i, r_{i+1} = m_i - 1$. As $k < p_{m_i}$, we have $f(l_i, r_i, k) = f(l_{i+1}, r_{i+1}, k)$. Combining with the fact that $f(l_i, r_i, k)$ is successful, we know $f(l_{i+1}, r_{i+1}, k)$ must be successful.

  If $a_k < m_{i+1}$, then we must have $k < p_{m_{i+1}}$. To see why, suppose $k > p_{m_{i+1}}$ then $f(l_{i+1}, r_{i+1}, k) = f(m_{i+1} + 1, r_{i+1}, k)$, however $a_k \notin [m_{i+1} + 1, r_{i+1}]$ which means $f(m_{i+1} + 1, r_{i+1}, k)$ fails, a contradiction as we have just shown above that $f(l_{i+1}, r_{i+1}, k)$ must be successful. Moreover, because $a_k < m_{i+1}$, we know $k \ne p_{m_{i+1}}$.

Otherwise $a_k > m_{i+1}$, then we must have $k > p_{m_{i+1}}$. To see why, suppose $k < p_{m_{i+1}}$ then $f(l_{i+1}, r_{i+1}, k) = f(l_{i+1}, m_{i+1} - 1, k)$, however $a_k \notin [l_{i+1}, m_{i+1} - 1]$ which means $f(l_{i+1}, m_{i+1} - 1, k)$ fails, a contradiction as we have just shown above that $f(l_{i+1}, r_{i+1}, k)$ must be successful. Moreover, because $a_k > m_{i+1}$, we know $k \neq p_{m_{i+1}}$.

- $a_k > m_i$ and $k > p_{m_i}$: The only difference from the above case is that as $a_k > m_i$, by definition we have $l_{i+1} = m_i + 1, r_{i+1} = r_i$. Also as $k > p_{m_i}$, we have $f(l_i, r_i, k) = f(l_{i+1}, r_{i+1}, k)$. The rest of the argument ie. "Combining with the fact that ..." is exactly same as above.

Therefore, given $P(i), i \leq q - 2$ holds $P(i + 1)$ also holds. For the base case, given $f(l, r, k)$ is successful and $q \geq 2$ (so that $m_1 \neq a_k$), it's easy to check that $P(1)$ holds (the argument is the same as above). The correctness of the below lemma follows :

Lemma 1. If $f(l, r, k)$ is successful and $q \geq 2$, then for all $i \leq q - 1$ we have either ($a_k < m_i$ and $k < p_{m_i}$) or ($a_k > m_i$ and $k > p_{m_i}$).

Notice lemma 1 holds for any permutation, just remember that the $a_k, q, (l_2, l_2), (l_3, l_3), ...$ values should be defined w.r.t that permutation.

Now suppose $a_k \in [l, r]$ and $q \geq 2$. Also suppose Cow the Nerd can change p to p' so that $f(l, r, k)$ on p' is successful. Let

$$\mathcal{M} = \{m = m_1, ..., m_{q-1} : (a_k < m \text{ and } k > p_m) \text{ or } (a_k > m \text{ and } k < p_m)\}$$

and partition $\mathcal{M} = \mathcal{M}^+ \cup \mathcal{M}^-$ such that $\mathcal{M}^+ = \{m \in \mathcal{M} : p_m > k\}$ and $\mathcal{M}^- = \{m \in \mathcal{M} : p_m < k\}$. Notice the index of k in p' will still be a_k, it follows if we redefine every thing in the first paragraph for p', the $(l_1, r_1), ..., (l_q, r_q)$ and q will not change. By lemma 1 and the fact that $f(l, r, k)$ is successful on p', it's easy to see that $\forall m \in \mathcal{M}^+$ we have $p'_m < k$ and $\forall m \in \mathcal{M}^-$ we have $p'_m > k$. This means all $p_m, m \in \mathcal{M}$ must participate in the re-ordering, in particular every $p_m \in \mathcal{M}^+$ must be replaced by an element less than k and every $p_m \in \mathcal{M}^-$ must be replaced by an element larger than k. It follows that if $|\mathcal{M}^+| = |\mathcal{M}^-|$ then any bijection from $\mathcal{M}^+$ to $\mathcal{M}^-$ will work and the number of elements involved in the re-ordering is $|\mathcal{M}^+| + |\mathcal{M}^-|$ which is also optimal. Otherwise suppose $|\mathcal{M}^+| \geq |\mathcal{M}^-|$ (the other case is similar), let

$$\mathcal{U} = \{m = m_1, ..., m_{q-1} : (a_k < m \text{ and } k < p_m) \text{ or } (a_k > m \text{ and } k > p_m)\}$$

and partition $\mathcal{U} = \mathcal{U}^+ \cup \mathcal{U}^-$ such that $\mathcal{U}^+ = \{u \in \mathcal{U} : p_u > k\}$ and $\mathcal{U}^- = \{u \in \mathcal{U} : p_u < k\}$. Notice indices in $\mathcal{U}$ must not participate in the re-ordering. If there exist at least $|\mathcal{M}^+|$ elements less than k and not belong to $\mathcal{U}^-$, then notice $2|\mathcal{M}^+|$ will be the optimal solution, else there're fewer than $|\mathcal{M}^+|$ elements less than k and not belong to $\mathcal{U}^-$, then it's infeasible.

2.2 algorithm

For each query, the algorithm visit each of $m_1, ..., m_{q-1}$ using binary search and calculate the size of $\mathcal{M}^+, \mathcal{M}^-$, finally printing a value as we have described above. The runtime is logarithmic for each query, thus total run time is $O(n + q \lg n)$ where the input size is $n + q$.

```
f(P,n,Q){
        A = array of size n
        for(i=1;i<=n;++i)          A[P[i]] = i;
        for each ((l,r,k) in Q) {
                m = (l+r)/2
                if(A[k] < l or A[k] > r){
                        print "impossible";
                }else if(A[k] == m){
                        print 0;
                }else{
                        print(g(P,A,n,l,r,m,k));
```

```
            }
        }
}

//Assume a_k ∈ [l,r] and q ≥ 2
g(P,A,n,l,r,m,k){
        x,y = 0 //x = |M+|, y = |M-|
        u,v = 0 //u = |U+|, v = |U-|
        //binary search on index l,l+1,...,r for A[k]
        while(m != A[k]){
                if(A[k] < m){
                        if(P[m] < k) y++;
                        else v++;
                        r = m-1;
                }else{
                        if(P[m] > k) x++;
                        else u++;
                        l = m+1;
                }
                m = (l+r)/2;
        }
        if(x > y){
                return (x <= k-1-v) ? 2*x : impossible;
        }else if (x < y){
                return (y <= n-k-u) ? 2*y : impossible;
        }else{
                return x+y;
        }
}
```