Carnegie Corporation Promotion Solution

We can first reword this problem as finding the M-th largest element of an array where one element is replaced every k comparisons.

- **Solution for $k = cn\log(n)$ (20 pts).** This solution belongs in the category of algorithms which seeks to complete the task before any replacements occur. We can first sort the array through merge-sort with $k \leq cn\log(n)$ comparisons. Then, we just pick the M-th largest array element in the sorted array.
- **Solution for $k = cn,\ c > 1$ (40 pts).** Like the former bound, we wish to solve this problem before any replacements occur. With the optimal deterministic selection algorithm, we can find the M-th largest element in $k = O(n)$ comparisons.
- **Solution for $k = n$ (70 pts).** If we treat it as a tournament graph (directed graph with exactly one edge between any two vertices), a replacement removes at most $n-1$ edges but we get n edge queries at a time. Thus, we will always gain a new edge every time until we have a complete graph at some point. Given the complete graph, we know how every element compares with every other element, so we can pick the M-th largest element as the one with exactly $M-1$ larger elements.
- **Solution for $k = n - \lfloor\frac{-1+\sqrt{3+2n}}{2}\rfloor + 1 \leq n - \lfloor\sqrt{n/2} - \frac{1}{2}\rfloor + 1$ (85 pts).**
  If we are allowed only $n - x + 1$ moves, treating it as a tournament graph again, we just keep all the degrees of vertices $\leq n - x$. Then, we will gain a new edge every time until we have a graph of all degree $n - x$. One of these vertices is now replaced. The other $n - 1$ vertices each have degree at least $n - x - 1$, so their rank in the array is a range of x. For now, consider only these $n - 1$ elements. We know at most $2x + 1$ elements can be distance x from the true M-th largest element. These are the elements of rank $M - x$ to $M + x$.

  Thus, with x comparisons for each of these elements, so a total of $x(2x + 1)$ comparisons, we can figure out all of their exact positions. If none of them are rank M, then the newly replaced element must be the rank M element. Thus, we just need $n - x + 1 \geq x(2x + 1)$ or $x \leq \frac{-1+\sqrt{3+2n}}{2}$.
- **Solution for $k = n - \lfloor\frac{-1+\sqrt{5+4n}}{2}\rfloor + 1 \leq n - \lfloor\sqrt{n} - \frac{1}{2}\rfloor + 1$ (100 pts).**
  Do the previous solution until we reach the $2x + 1$ elements that could potentially be rank M.

  The element of rank $M - x$ will take 1 additional comparison to rule out. Similarly, the element of rank $M - x + 1$ will take at most 2 additional comparisons to rule out. Thus, in total, we expect to use up at most $1 + 2 + \cdots + x + x - 1 + \cdots + 1 = x^2$ comparisons in the case where we end up finding the true M-th element last. Otherwise, if the true M-th element is never found, it must be the newly replaced element. Thus, we just need $n - x + 1 \geq x^2$ or $x \leq \frac{-1+\sqrt{5+4n}}{2}$.
- **Better Asymptotic Solution with $k = \lceil\log_2(n-2)\rceil + \lceil\log_2(n-1)\rceil$ (100 pts).** If we have a sorted sequence, removing an element keeps the rest sorted. Using binary search, we can insert 2 elements into a size $\leq l$ sorted sequence using binary search in $\lceil\log_2(l)\rceil$ comparisons. Thus, we always make progress, so eventually we'll end up with a fully sorted sequence as we end up only losing one element at a time, but insert at least 2 elements. We can also verify that this k is always less than the $n - \lfloor\sqrt{n} - \frac{1}{2}\rfloor + 1$.

  For solutions which work asymptotically but not for small n, there is a deduction of -5 points.

Scottybeard's Treasure Solution

- **Solution for $K = \frac{1}{4}$ (10 pts).**

  If we only use the instrument twice, we can never sink the island. As such, for our first use we draw a line up the middle of the island, which cuts our search space in half. We then draw a line through the middle of the remaining half, which cuts our search space down to $K = \frac{1}{4}$ in all cases.

- **Idea for $K < \frac{1}{4}$ (20+ pts).**

  In order to safely use the instrument more than twice, we need to be able to force a certain reading in the event that we receive two consecutive "left"s or two consecutive "right"s. To do this, we can simply draw an upward-oriented line off the left or right side of the island, which forces a reading of "right" or "left" respectively, taking us out of imminent danger of sinking the island.

- **Solution for $K = \frac{1}{128}$ (30 pts).**

  Equipped with the idea above, we perform the safest possible binary search by performing the following for each of 12 steps:

  - If the previous two readings were "left", draw a line off the left side of the island; and similarly for two consecutive "rights." (This guarantees that we never sink the island, as we are forcing the opposite reading.)
  - In any other case, draw a line splitting our remaining space of possibilities in half.

  We claim this yields a worst-case area of $K = \frac{1}{128}$. The key observation is that lines as in the first bullet point don't give us any new information (call them *uninformative*), while lines as in the second bullet point cut our solution space in half (call them *informative*). As such, the area of the space of possibilities is 2^{-k}, where k is the number of informative lines.

  In the worst case, we get as many uninformative lines as possible. Clearly our first two lines are informative (as there's nothing to avoid), and we cannot have two consecutive uninformative lines (as the whole point of an uninformative line is to get us out of imminent danger). Thus at worst we can have five uninformative lines, which can happen with, e.g., LLRRLLRRLLRR. This corresponds to 7 informative lines, or an area of $2^{-7} = \frac{1}{128}$.

- **Idea for $K < \frac{1}{128}$ (50+ pts).**

  The hurdle that we have faced so far is that in the event that we get unlucky and have identical consecutive readings, we have to essentially "throw away" a use of the instrument (what the previous solution called "uniformative lines"). To improve beyond this, we need to reduce the likelihood that this happens. To do this, instead of splitting our solution space in half each time, we bias our split point by a bit in order to make it less likely that we see the same reading twice in a row.

- **Solution for $K = K_{\min} = \frac{1}{466}$ (75-100 pts).**

  Claim: $K_{\min} \geq \frac{1}{466}$.

  Proof of claim: Consider the set of all possible sequences of outputs from the instrument. There is a natural correspondence between this set and the set of all LR-strings of length 12 that have no three consecutive L's and no three consecutive R's, let us call such strings *good*.

  We seek to count the number of good strings of length 12. Let A_n denote the number of good strings of length n that end in exactly one L or exactly one R (call this *type 1*), and let B_n be the number of good strings of length n that end in two consecutive L's or two consecutive R's (call this *type 2*). We can then create the recurrences

  $$A_{n+1} = A_n + B_n$$
  $$B_{n+1} = A_n.$$

  To see why, note that we can construct a type 1 string of length $n+1$ by taking any good string of length n and appending the opposite of its last character (this gives the first recurrence), and we can construct a type 2 string of length $n+1$ by taking a type 1 string and appending its last character again (note we cannot build such a string from another type 2 string, as otherwise we would sink the island).

  With these recurrences in hand, we can easily compute $A_1 = 2$ and $B_1 = 0$, and a simple inductive argument yields $A_n = 2F_n$ and $B_n = 2F_{n-1}$, where F_k denotes the kth Fibonacci number. Thus the number of good strings of length n is $2F_{n+1}$, so in particular there are $2F_{13} = 466$ good strings of length 12.

  Therefore, there are 466 possible sequences of results we can get from the instrument, so at best we can separate the island into 466 distinct pieces. To minimize the worst-case area, we would want all of these pieces to have the same area, or $\frac{1}{466}$. Thus we cannot do any better than $K = \frac{1}{466}$, which completes the proof. □

  Thus it remains to show that $K = \frac{1}{466}$ is attainable. Consider the following strategy:

  - Partition the island into 466 congruent vertical rectangles.
  - Lexically order the 466 strings from the proof above, and assign them in order to the rectangles from left to right.
  - For each of 12 steps, use the following subroutine (all mentioned lines will be directed straight up):
    - If the previous two readings were "left", draw a line off the left side of the island; and similarly for two consecutive "rights." (This guarantees that we never sink the island, as we are forcing the opposite reading.)
    - In any other case, consider the collection of rectangles whose assigned strings agree with our sequence of readings so far. By our lexicographic ordering, this is a single contigouous block of rectangles, the left portion of which has 'L' as their next character and the right portion of which has 'R' as their next character. Draw a line on the boundary between the 'L' and 'R' blocks. (This lets us narrow down our options while preserving agreement with the strings.)
  - At the end of the 12 steps, we will have narrowed the location of the treasure down to a single rectangle, the one corresponding to our sequence of readings from the instrument. (Since all of the rectangles have area $\frac{1}{466}$, this is the worst-case area of this algorithm.)

  This shows $K_{\min} \leq \frac{1}{466}$, so we have indeed obtained a worst-case area of $K = K_{\min} = \frac{1}{466}$.

CMIMC 2023

Perfect Shuffle Solution

We define $f(im+j) = i+jn$, for all $0 \le i < n, 0 \le j < m$. Here, $f(x)$ gives the location of the x-th card after one shuffle. Then, the answer is the minimum t such that $f^{(t)}(x) = x$ for all $0 \le x < nm$.

- **Solution for $k = 10^{10^{10^{10}}}$ (10 pts).** Note that each "m-perfect shuffle" is a permutation of size nm, so a brute force simulation must terminate in $(nm)!$ iterations. We need $6nm$ steps to simulate and $2nm$ steps to check if all cards are in the original place, so the total number of steps is no more than $(nm)!(8nm+1) < 10(nm)^{nm} \le 10 \cdot 10^{2000^{200}} < 10^{10^{10^{10}}}$.

- **Solution for $k = 10^{240}$ (50 pts).** We can decompose the permutation into cycle permutations, then the answer is the least common multiple of the cycle lengths. To get the list of cycle lengths, we maintain an array *visited* to track the list $x, f(x), f(f(x)), \ldots$ for all x that has not been visited yet. We need 4 arithmetic operations to compute $f(x)$ for any x, 1 step to check if it is visited, 1 step to set $visited[x] = True$, and 1 step to increment the current cycle length by 1; we need an outside loop of $2nm$ steps in total to start the inner loops that visit all cycles. So we can get the list of all cycle lengths in $9nm$ steps. We can compute the least common multiple of 2 numbers using the greatest common divisor in at most $4\lceil \log_2 x \rceil + 2$ steps where x is the smaller one of the 2 numbers. Since the number of cycle permutations is no more than nm and the length of each cycle is also no more than nm, we can get the answer in no more than $9nm + (4\lceil \log_2(nm) \rceil + 2)nm \le (11 + 800\lceil \log_2 10 \rceil)nm \le 3.211 \cdot 10^{203} < 10^{240}$ steps.

- **Solution for $k = 10^{201}$ (80 pts).** The answer is the order of n modulo $(nm-1)$. Proof: $f(im+j) = i+jn = (im+j)n - i(nm-1)$, so $f(x) \equiv nx \pmod{nm-1}$ for all $0 \le x < nm-1$. Suppose o is the order of n modulo $(nm-1)$, then for all $t < o$, $f^{(t)}(1) \equiv n^t \pmod{nm-1}$, so $f^{(t)}(1) \neq 1$; meanwhile, $\forall 0 \le x < nm$, $f^{(o)}(x) \equiv x \pmod{nm-1}$, plus that $f(0) = 0$ and $f(nm-1) = nm-1$ as the first and last card stays put, so o is the answer.

  To compute the order of n modulo $(nm-1)$, we first compute $(nm-1)$ in 2 steps, then run a loop of i from 1 to $(nm-1)$ maintaining n^i mod $(nm-1)$ until it becomes 1 where we output i as the answer. In each iteration, we perform a multiplication with n, a modulo operation, and a comparison, with a loop index increment, so the algorithm finishes in at most $4(nm-1)+2 < 10^{201}$ steps.

- **Solution for $k = 10^{101}$ (100 pts).** Note that the order of n modulo $(nm-1)$ is always a divisor of $\phi(nm-1)$, we can first factorize $(nm-1)$ to get $\phi(nm-1)$ and enumerate the divisors of $\phi(nm-1)$ to get the answer. We first compute $(nm-1)$ in 2 steps, then have an infinite loop to extract all prime divisors from $nm-1$. The outer loop terminates in at most $\lceil \sqrt{nm-1} \rceil$ iterations, and it takes 5 steps per iteration by checking if the remaining number is divisible by the loop index (2 steps), incrementing the loop index, and checking if we should break out of the loop (computing the square of the loop index and check if it is already greater than the remaining number – if so, the remaining number is the final prime divisor of $nm-1$). If the remaining number is divisible by the loop index i, we know that i is a prime number and we go into an inner loop to extract all divisors i. We first divide the remaining number by i and append a pair of the prime number i and the number 1 (the number of i in $nm-1$) to a list, then go into a while loop to check if the remaining number is still divisible by i: if it is then divide it and increment

the second element in the pair, if it is not then store the result and break out of the loop. All the inner loops take at most $4\lceil \log_2(nm-1) \rceil$ steps in total because $nm-1$ has at most $\lceil \log_2(nm-1) \rceil$ prime divisors.

Then we can compute $\phi(nm-1)$ in at most $2\lceil \log_2(nm-1) \rceil$ steps. Note that $\phi(nm-1)$ is always even, so we can factorize $\phi(nm-1)/2$ in at most $5\lceil \sqrt{\phi(nm-1)/2} \rceil + 4\lceil \log_2(\phi(nm-1)/2) \rceil$ steps then add the prime divisor 2 back in at most 3 steps. Then we can compute all the positive divisors of $\phi(nm-1)$ in at most $2d(\phi(nm-1))$ steps by searching for the number of prime divisors where $d(x)$ is the number of positive divisors of x. For each divisor t, we compute $n^t \bmod (nm-1)$ using the exponential by squaring algorithm in no more than $10\lceil \log_2 t \rceil$ steps, check if the result is 1 and if t is smaller than the current best answer, and update the answer if both conditions hold.

The total number of steps is at most

$$
\begin{aligned}
&5\lceil \sqrt{nm-1} \rceil + 4\lceil \log_2(nm-1) \rceil + 5\lceil \sqrt{\phi(nm-1)/2} \rceil + 4\lceil \log_2(\phi(nm-1)) \rceil \\
&\quad + 3 + (10\lceil \log_2(\phi(nm-1)) \rceil + 5) \cdot d(\phi(nm-1)) \\
&\leq 5\lceil \sqrt{nm} \rceil + 5\lceil \sqrt{nm/2} \rceil + 8\lceil \log_2(nm) \rceil + 3 + (10\lceil \log_2(nm) \rceil + 5) \cdot \exp\left(\frac{2(\ln 2) \ln \phi(nm-1)}{\ln \ln \phi(nm-1)} \right) \\
&\leq (5 + \frac{5\sqrt{2}}{2}) \cdot 10^{100} + 6403 + 8005 \cdot \exp\left(\frac{400(\ln 2)(\ln 10)}{\ln(200 \ln 10)} \right) \\
&\leq 8.54 \cdot 10^{100} + 10^{50} \\
&\leq 10^{101}.
\end{aligned}
$$