1. \( \theta_1 < \cdots < \theta_n \) describe the arguments of \( n \) given points (in polar coordinates) on the unit circle. First note that if \( n = 2k + 1 \) is odd, a bisecting diameter must be incident with exactly ONE \( \theta_i \) and if \( n = 2k \) is even, it is either incident with NONE, or TWO (a diametric pair) of the \( \theta_i \). In fact we only need to solve the odd case because then, for the even case, we omit one of the points - say \( \theta_j \) - and solve the resulting (odd-size) problem with a diameter through one of the \( n - 1 \) points, say \( \theta_m \). Now we put \( \theta_j \) back, and the diameter through \( \theta_m \) is no longer bisecting - the semicircle with \( \theta_j \) has one more point than the one without. So we move our diameter slightly so \( \theta_m \) is on the opposite semicircle from \( \theta_j \) and now the diameter bisects the \( 2k \) points.

We take \( \theta = \pi / 2 \) as the diameter directed to the point \((1,0)\) and count \( f(\theta) = n_{\text{left}} - n_{\text{right}} \) the signed difference between the number of points in the open left-halfspace of \( \theta \) and the number in the open right-halfspace. If \( f \) is zero we have a bisecting diameter at that \( \theta \). If not we will rotate \( \theta \) counterclockwise to \( 3\pi / 2 \). During the rotation we will encounter all the \( \theta_i \), either by the upper half of our rotating diameter, or by the lower, or by both if two \( \theta_i \)'s are diametric, and in constant time we decrease \( f \) by 1, increase it by 1, or make no change if we encounter diametric \( \theta_i \)'s. We know exactly the sequence of these events by using the sorted order and merging the events to the left of the initial \( \theta \) and those to the right, and so we have an \( O(n) \) algo.

2. 2a: If we start by sorting the \( n \) initially unsorted \( \theta_i \) and then use the above algorithm, we will have solved the problem in \( O(n \log n) \). The problem has a \( \Omega(n) \) lower bound: if you ignore one of the inputs and still give some diameter as the solution, (even if it were correct) an adversary could change the \( \theta \) you ignored by adding \( \pi \) to it, making your answer incorrect.

3. 2b: There is an (optimal) linear time algorithm that uses the prune-and-search idea and is somewhat reminiscent of the linear time algorithm for the median. SKETCH: As in 1a start with \( \theta_0 = \pi / 2 \) and count \( j \), the number of inputs in \((\pi / 2, 3 \pi / 2)\) [LEFT-side], and \( k \leq n - j \) the number in that are \( > 3 \pi / 2 \) or \( < \pi / 2 \) [RIGHT-side]; \( j + k \leq n \) and I won't worry about the fussy details for data ON a diameter). So far we made \( O(n) \) comparisons to partition inputs into LEFT side or RIGHT, and if \( j = k \) we stop with \( \theta_0 \) as a solution.

Otherwise we search. In \( O(j) \) we compute \( \theta' \), the “median” of the inputs to the left of \( \theta_0 \). Next, we compute \( f(\theta') \), to see if \( f \) changes sign between \( \pi / 2 \) and \( \theta' \), or between \( \theta' \) and \( 3 \pi / 2 \) (exactly one of these MUST occur). We know there are \( j / 2 \) inputs between \( \theta' \) and \( 3 \pi / 2 \), so we just need to count how many (call it \( m \)) fall between \( 3 \pi / 2 \) and \( \theta' + \pi \), the latter count obtained in \( O(k) = O(n) \) (and so we have \( k - m \) inputs between \( \theta' + \pi \) and \( \pi / 2 \), going counterclockwise - good idea to draw a picture).

Let's suppose \( f \) had different signs at \( \theta_0 \) and \( \theta' \) (the other case is similar so I won't discuss it) so we continue searching for a bisecting diameter in the “double-wedge” of inputs in \((\pi / 2, \theta')\). We pruned the \( j / 2 \) inputs in \((\theta', 3 \pi / 2)\) and the \( k - m \) inputs on the right side between \( 3 \pi / 2 + \theta' \)
and $\pi/2$; the cost was $O(n)$ time. Here “prune” means we will never again need to search among those contiguous sets of inputs.

But here comes a subtle point. We pruned half the inputs from the left of $\theta_0$, and we pruned $k - m$ from the right, and this might actually be ZERO points pruned, e.g., if ALL right-side inputs are in the double wedge we still need to search. Furthermore, the $j/2$ we DID prune from the left side might only be $o(n)$ if, e.g., $j = \log n$. So we MUST (and will) assure that we do prune a fixed fraction of all inputs in each PAIR of steps, first pruning from the left, as just described, and then from the right, as will be explained. In this way, in $O(n)$ every two steps will reduce the set of candidates by a fixed fraction, and thus make an $O(n)$ algorithm because the running time is then $cn[1 + a + a^2 + ...] = O(n)$, $a < 1$ the fraction “pruned” in each PAIR of steps.

To prune from the right side, compute $\theta''$, the median of the $m$ inputs between $3\pi/2$ and $\theta' + \pi$ and then evaluate $f$ at $\theta'' - \pi$ by (i) counting how many $\theta \in (\theta'' - \pi, \theta')$, (ii) adding $j/2$ for the pruned left-side inputs, and (iii) adding $m/2$ for unpruned right-side points to the left of the diameter thru $\theta'' - \pi$ (again, a drawing will help).

In every such pair of steps we prune at least $j/2 + m/2 + k - m$ inputs ($j/2$ from the left-side step and the rest from the right side step, one median computation for each) and its easily seen that this is at least $n/2$ because $k - m/2$ is at least $k/2$, (so on average at least $1/4$ of the inputs for each pair of half-steps).

(MORE COMING)