- The role of adversary is to prove that no algorithm for the given problem can take lesser time[2] to the lower bound to be established. If an algorithm claims to do the job while beating the said lower bound, adversary outputs an input instance on which the algorithm is guaranteed to output an incorrect solution. The adversary has access to the contents of the input array and it modifies the values in the array as the algorithm progresses. The algorithm has only access to keys while the values are hidden from it. The algorithm provides keys corresponding to any operation together with that operation to the adversary. Since the adversary maintains values corresponding to keys, the adversary does the operation on the values corresponding to keys supplied by the algorithm, and returns the correct outcome to the algorithm. At every stage of the algorithm's execution, if necessary, adversary modifies the input values so that the algorithm acquires minimal amount of information, forcing the algorithm to work hard, so that the algorithm mimics its worst-case behaviour. Significantly, at every stage, the adversary is ensured to have at least one set of key-value pairs that are consistent with its previous replies to the algorithm. Since the adversary's responses never contradict its own replies earlier, the algorithm does not recognize changing values as the algorithm gets executed. The adversary does not make any assumptions about the order in which the algorithm performs these operations. If an algorithm terminates prematurely, before acquiring enough information, the adversary reveals an input that is consistent with its answers while depicting algorithm's output is incorrect. More importantly, if the algorithm is again run on the values output by the adversary, then that algorithm indeed outputs an incorrect solution. In the below examples, we assume algorithm does only pairwise comparison of input elements.

- Given a set S of n distinct integers in array S, finding an i such that $S[i]$ is maximum in S needs at least $n-1$ comparisons in the worst-case.

  Consider any algorithm $\mathcal{A}$ for this problem. The $\mathcal{A}$ places a mark on every key of S, where a mark denotes 'may be the value stored corresponding to this key is the maximum among all the values stored in S'. Essentially, $\mathcal{A}$ must remove $n-1$ of these marks before it can claim that a particular value is the maximum among values in S - hence the stated lower bound. Whenever $\mathcal{A}$ gets to know $S[i]$ is less than $S[j]$ for any i and j, $\mathcal{A}$ is assumed to unmark key i (since $S[i]$ cannot be the maximum of S). If the algorithm terminates after $r < n-1$ number of comparisons, then there must exist at least two keys, say ℓ' and ℓ'', which continue to be marked, that is, $S[\ell']$ and $S[\ell'']$ not yet been concluded as non-maximum elements. Further, if $S[\ell']$ (resp. $S[\ell'']$) is not the output produced by $\mathcal{A}$ then adversary assigns a maximum value to $S[\ell']$ (resp. $S[\ell'']$) and the resultant array S has values to prove that $\mathcal{A}$ is incorrect. Significantly, the adversary will always be able to find a set of values consistent with its replies to $\mathcal{A}$. The adversary maintains a directed graph G with a node for each key. Whenever adversary replies to $\mathcal{A}$ that $S[i]$ is less than $S[j]$, adversary introduces an arc from i to j into G. Since G cannot have any cycles, topological sorting of nodes of G lets the adversary assign values to entries of S. If $\mathcal{A}$ had not acquired information, there exist at least two nodes in G, say one corresponding to key i and the other corresponding to key j, that do not have outgoing arcs. If $\mathcal{A}$ outputs $S[i]$ (resp. $S[j]$) is the maximum, the adversary assigns maximum value to $S[j]$ (resp. $S[i]$) and discloses the modified array to $\mathcal{A}$.

- Given a set S of n distinct integers with n even, finding i and j such that $S[i]$ and $S[j]$ are the minimum and maximum values among elements of S needs at least $\lceil \frac{3n}{2} \rceil - 2$ comparisons in the worst-case.

[1] or information-theoretic
[2] or any other resource

Consider any algorithm $\mathcal{A}$ for this problem. It suffices to show that if $\mathcal{A}$ makes less than $\lceil \frac{3n}{2} \rceil - 2$ comparisons, then it is guaranteed to output an incorrect maximum, an incorrect minimum, or both. The $\mathcal{A}$ places M and m marks on every key of S, where M denotes 'may be the value stored corresponding to this key is the maximum among all the values stored in S' and m denotes 'may be the value stored corresponding to this key is the minimum among all the values stored in S'. Essentially, $\mathcal{A}$ must remove $2n-2$ of these marks before it can correctly claim that a particular value is the maximum or minimum in S, provided the two leftover marks are on distinct keys. It is immediate that a key i has both the marks associated to it if and only if i is yet to participate in any comparison. The adversary adjusts values in S so that it minimizes the number of units of information $\mathcal{A}$ acquires in any iteration. However, if $\mathcal{A}$ requests adversary to compare $S[i]$ and $S[j]$ wherein i and j have both the marks, upon receiving a reply from the adversary, $\mathcal{A}$ is guaranteed to acquire two units of information: if the adversary outputs $S[i] < S[j]$, M mark can be removed from i and m mark from j; if the adversary outputs $S[i] > S[j]$, m mark can be removed from i and M mark from j. In every other case, adversary adjusts the values in S so that $\mathcal{A}$ acquires at most one unit of information. If $\mathcal{A}$ requests adversary to compare $S[i]$ and $S[j]$ wherein i has both marks and j has only m mark, the adversary adjusts the value of $S[i]$ so that $S[j]$ is smaller among these two; thus by getting rid of m mark on i the algorithm acquires one unit of information. In yet another case in which $\mathcal{A}$ seeks to compare $S[i]$ and $S[j]$ wherein i is marked with M and j is marked with m, adversary responds $S[i] > S[j]$, leading $\mathcal{A}$ to acquire 0 units of information. (An exhaustive case analysis could be done here.)

Similar to the strategy of adversary in the above problem, by maintaining a directed acyclic graph G, the adversary assigns a set of values to entries of S that are consistent with the replies that it had given to $\mathcal{A}$. Besides, this graph helps in inferring whether $S[i]$ is less than $S[j]$ from the adversary's earlier replies. Also, it is ensured that no response from the adversary causes a cycle in G. When a node v in G is reachable from all other nodes of G, the value corresponding to v can be finalized as the maximum value. Analogously, every other node of G is reachable from a node v, then the value corresponing to v can be finalized as the minimum value. If $\mathcal{A}$ had not acquired information, there exist at least three marks on keys; by using these, the adversary constructs a witness w to prove A is incorrect on w. Note that none of the adversary's responses cause a cycle in G.

Since there are only $\lfloor \frac{n}{2} \rfloor$ pairs of keys with both marks unerased on each key, there are only $\lfloor \frac{n}{2} \rfloor$ number of pairwise comparisons each of which let $\mathcal{A}$ to acquire two units of information. And, $\mathcal{A}$ acquires the rest of $(2n-2) - 2\lfloor \frac{n}{2} \rfloor = 2\lceil \frac{n}{2} \rceil - 2$ units of info by doing one comparison per unit of info. Therefore, $\mathcal{A}$ must perform at least $\lfloor \frac{n}{2} \rfloor + 2\lceil \frac{n}{2} \rceil - 2 = n + \lceil \frac{n}{2} \rceil - 2 = \lceil \frac{3n}{2} \rceil - 2$ comparisons.

- Given a set S of n distinct integers with n odd, finding the median among elements of S needs at least $\frac{3}{2}(n-1)$ comparisons in the worst-case.

  The order relation between every element with the median is needed before declaring a particular value in S as the median of S: For any key k whose relation to median is not determined, the adversary could change the value of k, moving it to the other side of the median without contradicting the previous comparisons. A comparison involving a key k' is said to be an essential comparison for k' if it is the first comparison where $A[k']$ is found to be greater than (resp. smaller than) $A[k'']$ for some $A[k'']$ known to be greater than the median (resp. less than the median). It is immediate the worst-case lower bound on the number of essential comparisons in $n-1$.

  Further, comparing x with y where $x > median$ and $y < median$ is termed an inessential comparison. Naturally, the adversary tries to maximize the number of inessential comparisons. Whenever a key that is not yet assigned a value is compared, the adversary assigns a value to it. Each entry in S is marked with larger than the median, smaller than the median, and not been in any comparison. When both the comparands are being compared for the first time, the adversary makes one larger than the median, the other smaller than the median. When only one of the comparand is being compared for the first time, the adversary makes that comparand smaller (resp. larger) to the median if the other comparand's value is larger (resp. smaller) than the median. However, if there are already $\frac{n-1}{2}$ keys with values smaller (resp. larger) than the median, the adversary ignores all of these rules and assigns values larger (resp. smaller) than median to comparand that is being compared for the first time. Whenever algorithm compares two keys with any other marking combination (labeled larger to median with larger to median, smaller to median with smaller to median, or larger to median with smaller to median), the adversary simply gives the correct response based on the values it has already assigned to the keys. When only one key without a value remains, the adversary assigns the value median to that key. With these, it is immediate the adversary cannot assure forcing more than $\frac{n-1}{2}$ inessential comparisons.