CS 580: Algorithm Design and Analysis

Order Statistics

The **selection problem** is the problem of computing, given a set A of n distinct numbers and a number i, $1 \leq i \leq n$, the i^{th}h **order statistics** (i.e., the i^{th} smallest number) of A.

We will consider some special cases of the order statistics problem:

- the **minimum**, i.e. the first,
- the **maximum**, i.e. the last, and
- the **median**, i.e. the "halfway point."

Order Statistics

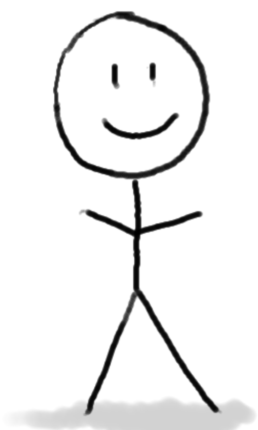
Medians occur at $i = \lfloor (n+1)/2 \rfloor$ and $i = \lceil (n+1)/2 \rceil$. If n is odd, the median is unique, and if n is even, there are two medians.

Recall

We work in the **comparison model**:

- Given input array $x = (x_1, \dots, x_n)$, we can access the array via comparison queries: "Is $x_i < x_j$?"

Is  bigger than  ?



Algorithm

YES

There is a genie that knows what the right order is.

The genie can answer YES/NO questions of the form:
is [this] bigger than [that]?

Finding the Min

How many comparisons are necessary and sufficient for finding the minimum?

Finding the Min

Algorithm:

Given vector $x = (x_1, \dots, x_n)$ *as input, consider the standard algorithm.*

Let $q = x_1$.

For each $i = 2, \dots, n$:

If $x_i < q$:

$q = x_i$

Return q

This algorithms makes $n - 1$ comparisons.

Finding the Min

Can we do better than $n-1$ comparisons?

Finding the Min

Can we do better than $n-1$ comparisons?

If not, then show a lower bound of the form:

- Every deterministic algorithm for finding the min (which is correct on every input) makes at least $n-1$ comparisons.

Finding the Min

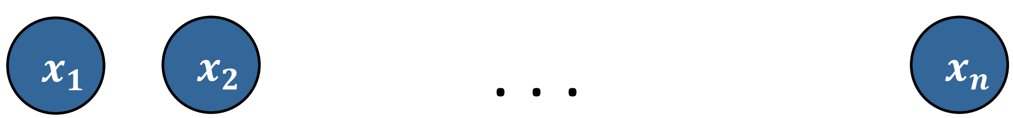
Lower Bound:

Consider any (correct) deterministic algorithm A for finding the min.

Construct a graph G with:

- vertices $x_1, \dots, x_n$.
- edge (x_i, x_j) if A compares elements x_i and x_j at some point during its execution.

What happens if the graph G is not connected at the end of A's execution?



Lower Bound:

Consider any (correct) deterministic algorithm A for finding the min.

Construct a graph G with:

- vertices $x_1, \dots, x_n$.
- edge (x_i, x_j) if A compares elements x_i and x_j at some point during its execution.

If G is not connected at the end of A's execution, then the algorithm can output the wrong answer on some inputs.

For G to be connected, it must have at least $n - 1$ edges (if it has exactly $n - 1$, it is a tree) $\Rightarrow$ A makes at least $n - 1$ comparisons.

Selection (Find s-th smallest element)

Selection is a trivial problem **if the input numbers are sorted.** If we use a sorting algorithm having $O(n \lg n)$ worst-case running time, then the selection problem can be solved in in $O(n \lg n)$ time.

But using a sorting is more like using a cannon to shoot a fly since only one number needs to computed.

Selection (Find s-th smallest element)

Selection is a trivial problem **if the input numbers are sorted.** If we use a sorting algorithm having $O(n \lg n)$ worst-case running time, then the selection problem can be solved in in $O(n \lg n)$ time.

But using a sorting is more like using a cannon to shoot a fly since only one number needs to computed.

Task: Design a selection algorithm inspired by QuickSort but with fewer comparisons.

$O(n)$ expected-time selection using the randomized partition

Idea: In order to find the s-th order statistics in a region of size n, use the **randomized partition** to split the region into two subarrays. Let $\mathsf{k} - 1$ and $n - \mathsf{k}$ be the size of the left subarray and the size of the right subarray. If $k = s$, the pivot is the key that's looked for. If $\mathsf{s} \leq \mathsf{k} - 1$, look for the **?-th element in the left subarray**.

$O(n)$ expected-time selection using the randomized partition

Idea: In order to find the s-th order statistics in a region of size n, use the **randomized partition** to split the region into two subarrays. Let $k - 1$ and $n - k$ be the size of the left subarray and the size of the right subarray. If $k = s$, the pivot is the key that's looked for. If $s \leq k - 1$, look for the **s-th element in the left subarray**. Otherwise **?**

$O(n)$ expected-time selection using the randomized partition

Idea: In order to find the s-th order statistics in a region of size n, use the **randomized partition** to split the region into two subarrays. Let $\mathsf{k}-1$ and $n-\mathsf{k}$ be the size of the left subarray and the size of the right subarray. If $k = s$, the pivot is the key that's looked for. If $\mathsf{s} \leq \mathsf{k}-1$, look for the s**-th element in the left subarray**. Otherwise, look for the $(\mathsf{s}-\mathsf{k})$**-th one in the right subarray**

Analysis

Define

- $T(n,s) =$ expected # comparisons for selection of s-th statistic
- $T(n) = \max_{s} T(n,s)$ is the expected # comparisons of selection for the worst case index s.

Analysis

Define

- $T(n, s) =$ expected # comparisons for selection of s-th statistic
- $T(n) = \max_s T(n, s)$ is the expected # comparisons of selection for the worst case index s.

Task: Write an inequality to upper bound $T(n)$ as a function of the amount of work done in Partition and in recursive selection calls.

Analysis

Recall: $T(n)$ is the expected # comparisons of selection for the worst case index s.

For each i, $0 \le i \le n-1$, the size of the left subarray is equal to i with probability $1/n$. Assuming that the larger interval is taken, for some $\alpha > 0$, $T(n)$ is at most

Work for partition

$$\alpha n + \frac{1}{n} \sum_{1 \le k \le n-1, k \ne s} T(\max(k, n-k)).$$

Expected work for recursive call

This is at most

$$\alpha n + \frac{2}{n} \left(\sum_{k=\lceil n/2 \rceil}^{n-1} T(k) \right).$$

Analysis (cont'd)

Assume that there is $c > 0$ such that $T(k) \leq ck$ for all $k < n$.

Then the sum $\sum_{k=\lceil n/2 \rceil}^{n-1} T(k)$ is at most $\sum_{k=\lceil n/2 \rceil}^{n-1} ck$. This is at most

$$
\begin{aligned}
& \sum_{k=1}^{n-1} ck - \sum_{k=1}^{\lceil n/2 \rceil - 1} ck \\
= \ & \frac{cn(n-1)}{2} - \frac{c}{2}\left(\left\lceil \frac{n}{2} \right\rceil - 1\right)\left\lceil \frac{n}{2} \right\rceil \\
\leq \ & \frac{cn(n-1)}{2} - \frac{c}{2}\left(\frac{n}{2} - 1\right)\frac{n}{2} \\
= \ & cn\left(\frac{3n}{8} - \frac{1}{4}\right).
\end{aligned}
$$

Analysis (cont'd)

So, if c is sufficiently large,

$$T(n) \leq \alpha n + c\left(\frac{3}{4}n - \frac{1}{2}\right).$$

By making the constant c at least 4α, we have that αn is at most $\frac{cn}{4}$. Then $\mathrm{T}(\mathrm{n}) \leq \mathrm{c} \cdot n$.

Min and Max

How many comparisons are necessary and sufficient for computing both the minimum and the maximum?

Min and Max

How many comparisons are necessary and sufficient for computing both the minimum and the maximum?

Well, to compute the maximum $n - 1$ comparisons are necessary and sufficient. The same is true for the minimum. So, the number should be $2n - 2$ for computing both.

Min and Max

Hint: *Actually you can do better by processing the input numbers in pairs*

Min and Max Algorithm

Simultaneous computation of max and min can be done in $\leq 3n/2$ comparisons

Assume n is even:

- Divide the numbers in pairs and find the larger and smaller one in each pair
- From the n/2 larger items, find the maximum
- From the n/2 smaller items, find the minimum

For n odd, compare the remaining (n-th item) with both the min and max.

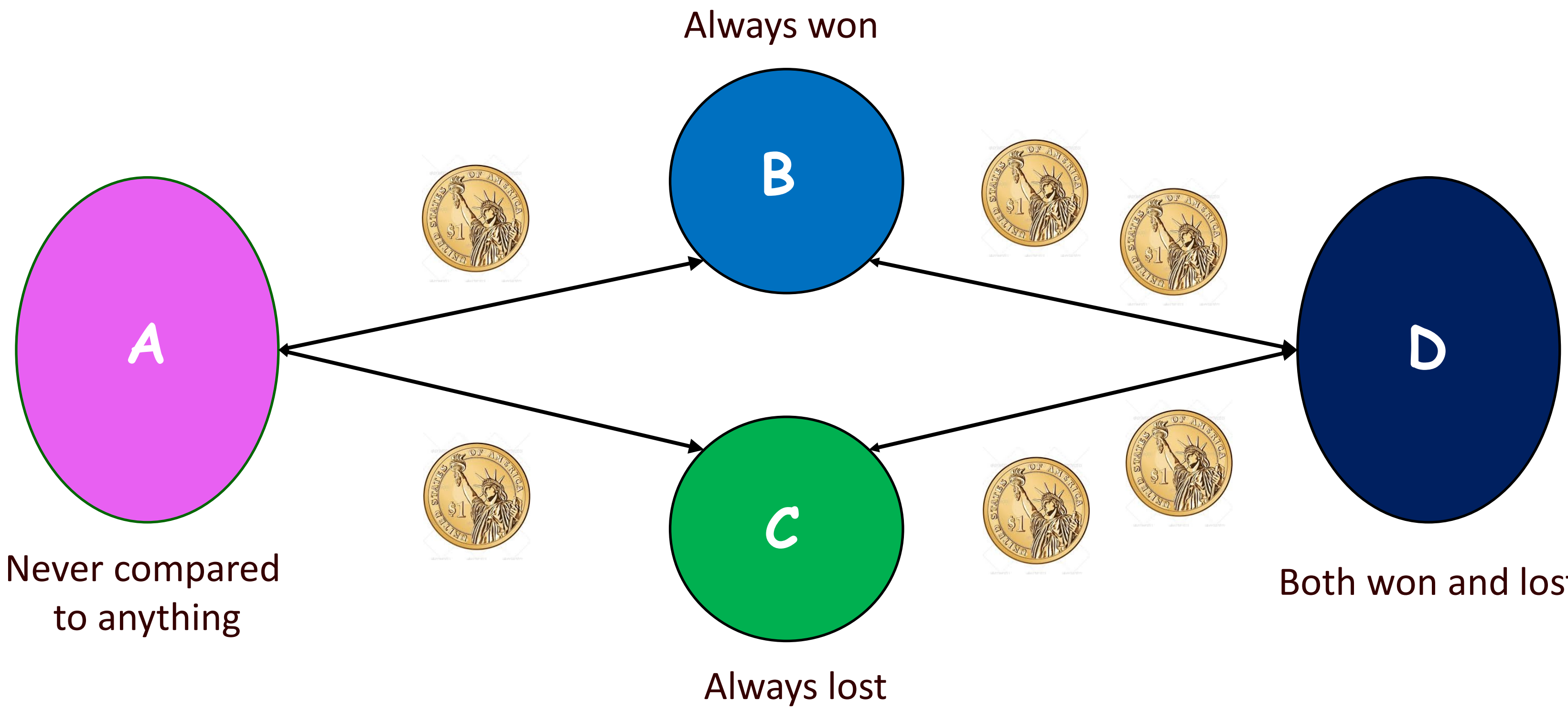
Min and Max Lower Bound

Q: Can we do better than $\frac{3n}{2} - 2$ comparisons?

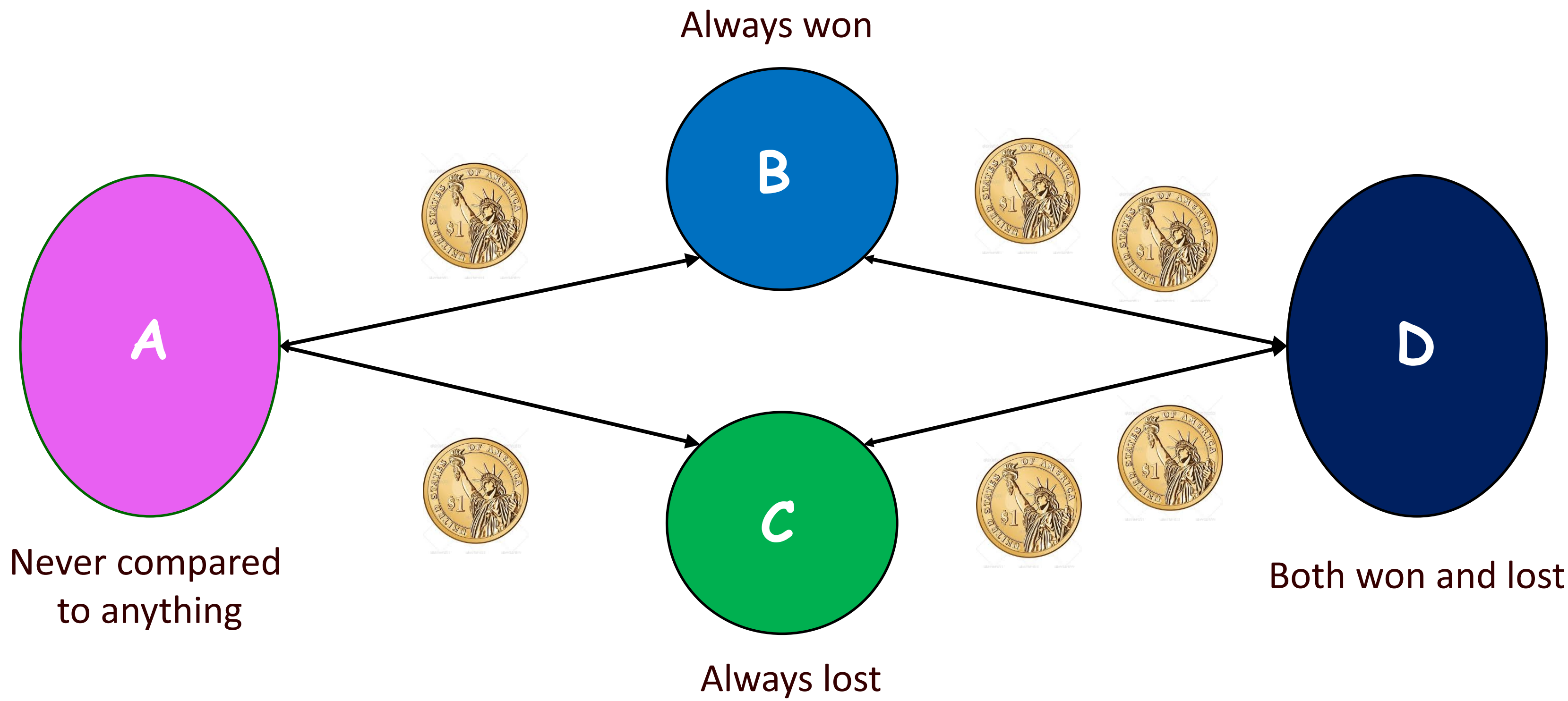
Lower bound proof: Consider any algorithm M for finding the min and the max. Assume distinct elements.

Partition the elements throughout the execution into 4 groups:

- **A:** items never compared to any other item
- **B:** items that always won their comparisons
- **C:** items that always lost
- **D:** items that both lost and won



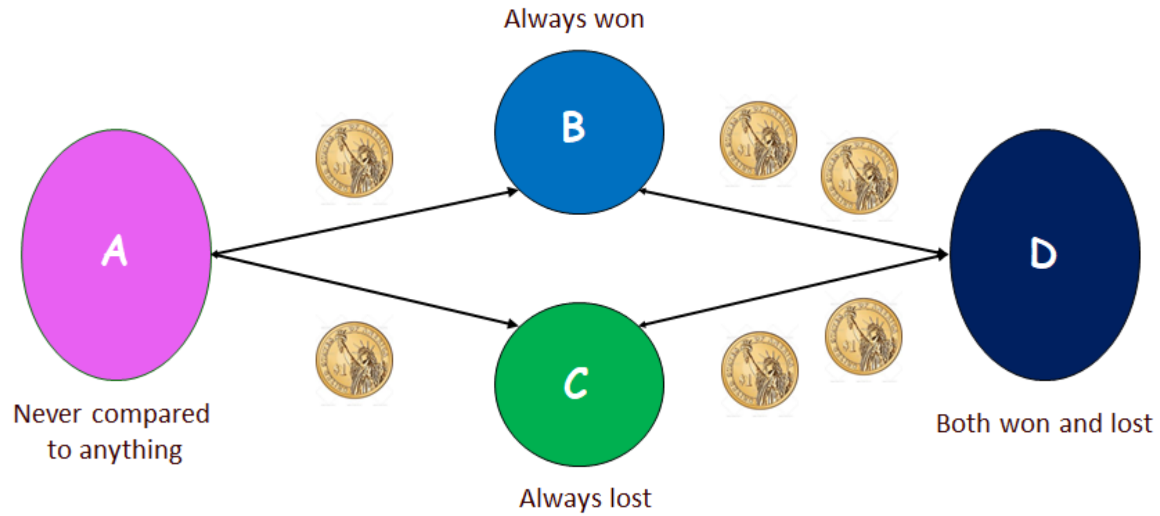
Give an item 1 point for moving from A to B or from A to C, and 2 points for moving from B to D or C to D.



Give an item 1 point for moving from A to B or from A to C, and 2 points for moving from B to D or C to D.

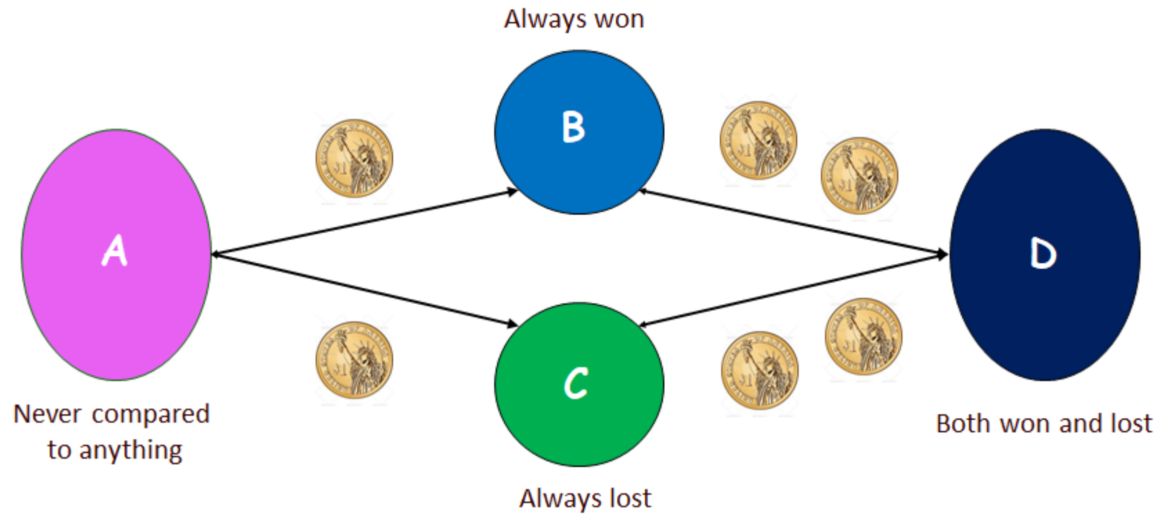
Note: each of $n-2$ items that are not min or max need 3 points at the end (otherwise they could be the min or the max), while the max and min have one point each.

Devise an adversary strategy for answering queries posed by M, so that each query does not gather more than 2 points. Say the current query asked by M is “x < y”?.



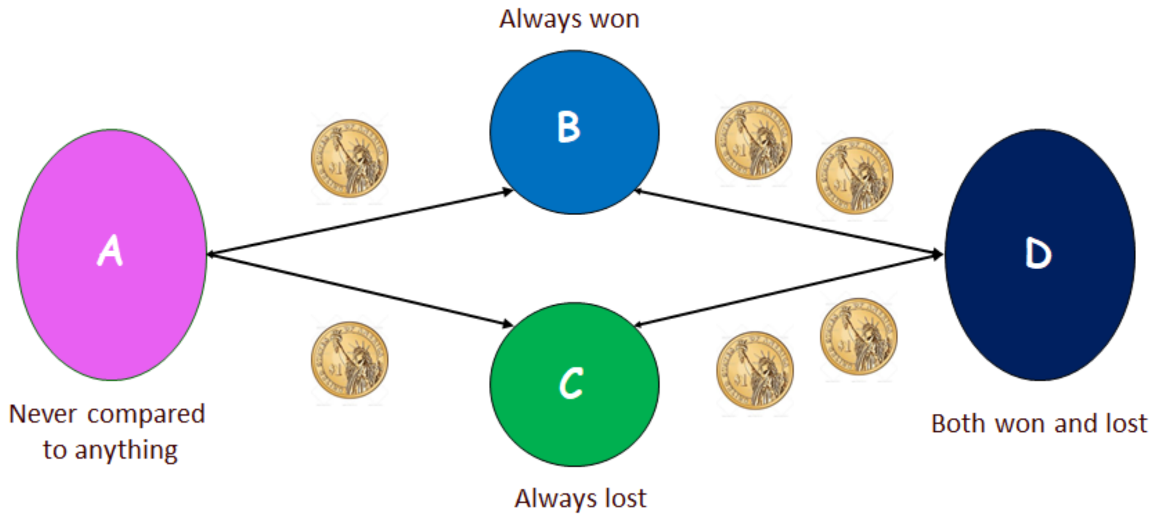
Cases:

- $x, y \in A$: Let x move to B and y to C ➔ gain 2 points



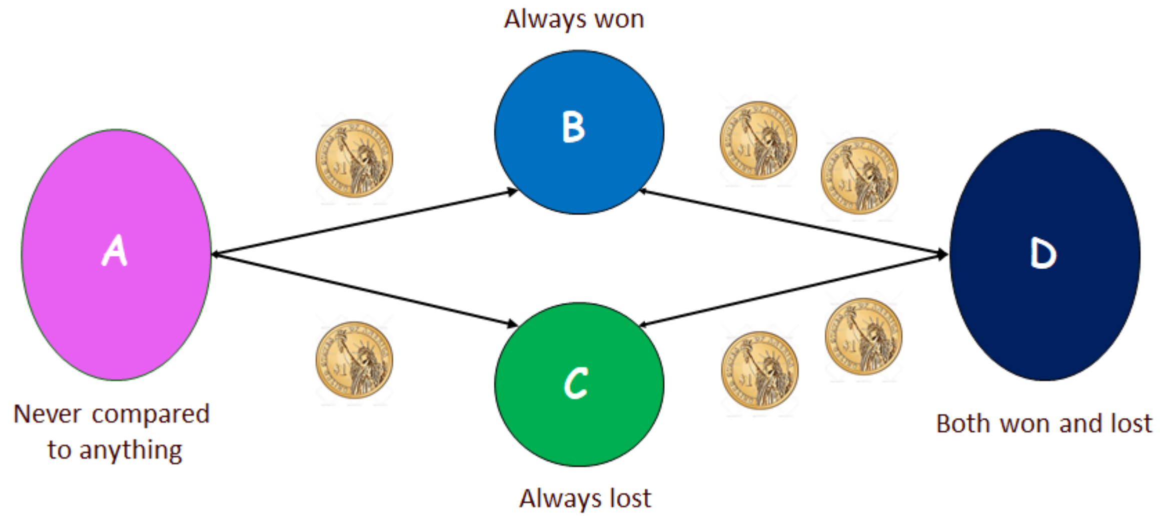
Cases:

- $\boldsymbol{x, y \in A}$: Let x move to B and y to C ➔ gain 2 points
- $\boldsymbol{x, y \in B}$: Make x win (it stays in B), y moves to D ➔ How many points are gained?



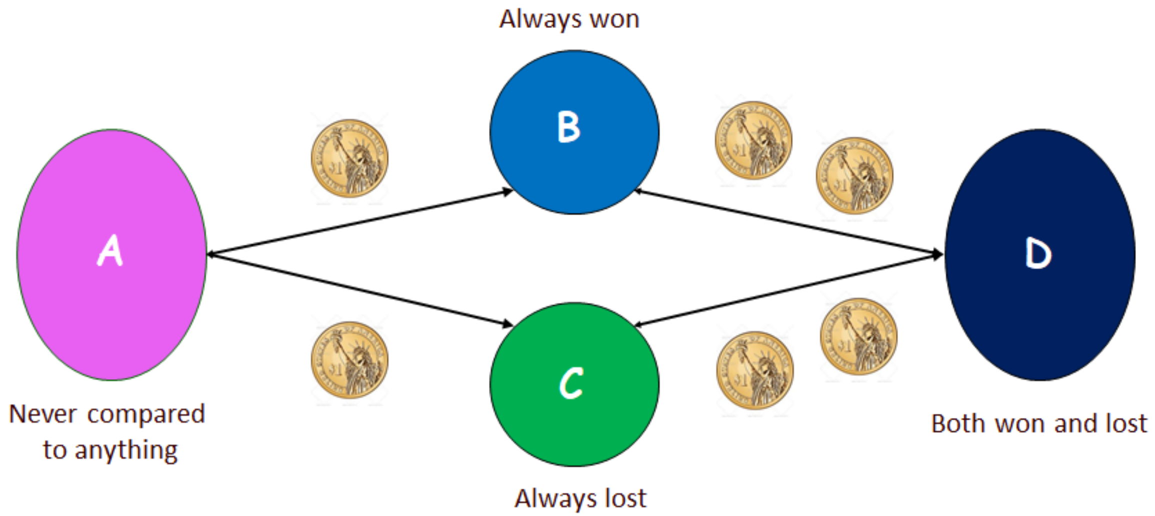
Cases:

- $\boldsymbol{x, y \in A}$: Let x move to B and y to C ➔ gain 2 points
- $\boldsymbol{x, y \in B}$: Make x win (it stays in B), y moves to D ➔ gain 2 points for y. Similar if $x, y \in C$.



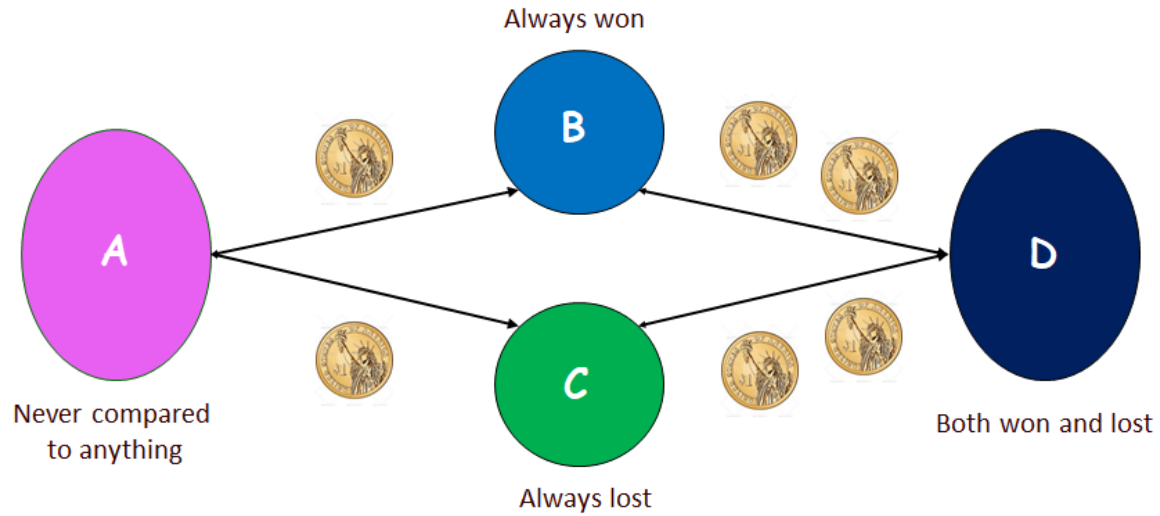
Cases:

- $x, y \in A$: Let x move to B and y to C ➔ gain 2 points
- $x, y \in B$: Make x win (it stays in B), y moves to D ➔ gain 2 points for y. Similar if $x, y \in C$.
- $x \in B, y \in C$: Make x win ➔ How many points are gained?



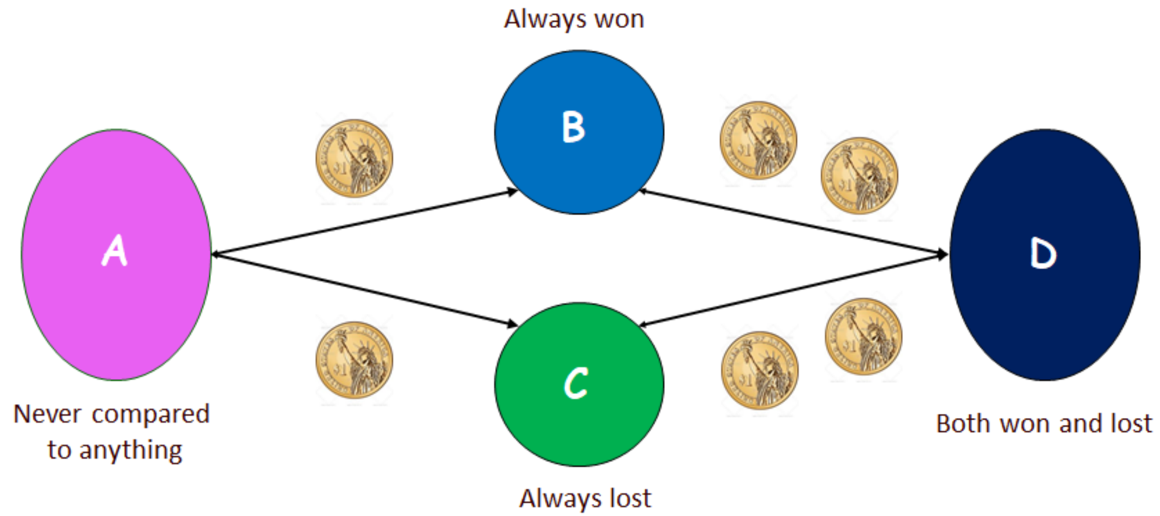
Cases:

- $\boldsymbol{x, y \in A}$: Let x move to B and y to C ➔ gain 2 points
- $\boldsymbol{x, y \in B}$: Make x win (it stays in B), y moves to D ➔ gain 2 points for y. Similar if $x, y \in C$.
- $\boldsymbol{x \in B, y \in C}$: Make x win ➔ both stay put (no points).



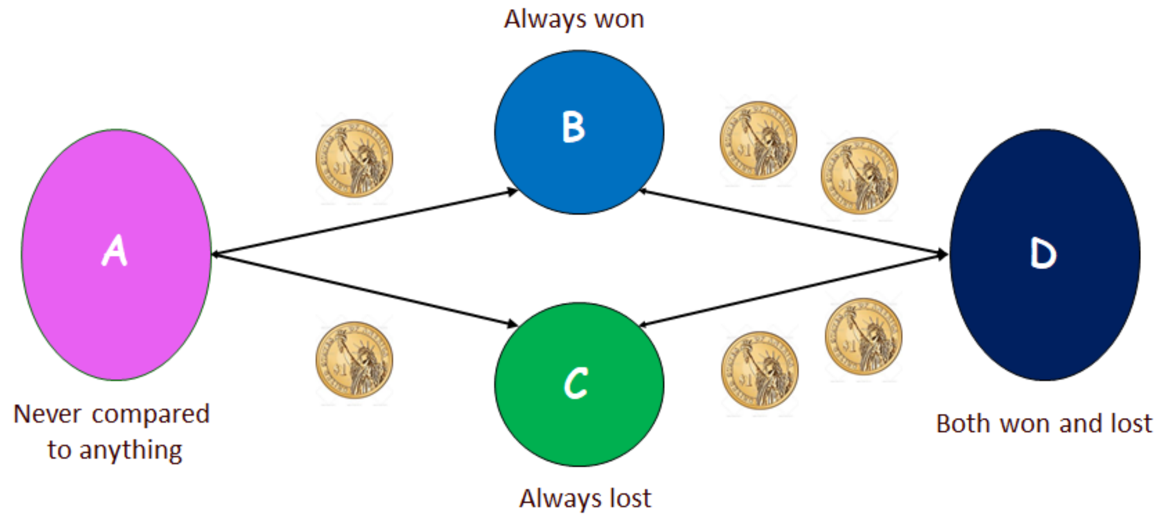
Cases:

- $\boldsymbol{x, y \in A}$: Let x move to B and y to C ➔ gain 2 points
- $\boldsymbol{x, y \in B}$: Make x win (it stays in B), y moves to D ➔ gain 2 points for y. Similar if $x, y \in C$.
- $\boldsymbol{x \in B, y \in C}$: Make x win ➔ both stay put (no points).
- $\boldsymbol{x \in (A \cup B \cup C \cup D), y \in D}$: Answer in a way that is compatible with at least one existing permutation that is still feasible ➔ How many points are gained?



Cases:

- $\boldsymbol{x, y \in A}$: Let x move to B and y to C ➔ gain 2 points
- $\boldsymbol{x, y \in B}$: Make x win (it stays in B), y moves to D ➔ gain 2 points for y. Similar if $x, y \in C$.
- $\boldsymbol{x \in B, y \in C}$: Make x win ➔ both stay put (no points).
- $\boldsymbol{x \in (A \cup B \cup C \cup D), y \in D}$: Answer in a way that is compatible with at least one existing permutation that is still feasible ➔ gain at most 2 points.



Cases:

- $\boldsymbol{x, y \in A}$: Let x move to B and y to C ➔ gain 2 points
- $\boldsymbol{x, y \in B}$: Make x win (it stays in B), y moves to D ➔ gain 2 points for y. Similar if $x, y \in C$.
- $\boldsymbol{x \in B, y \in C}$: Make x win ➔ both stay put (no points).
- $\boldsymbol{x \in (A \cup B \cup C \cup D), y \in D}$: Answer in a way that is compatible with at least one existing permutation that is still feasible ➔ gain at most 2 points. → **Can we test this quickly?**

Lemma: Let S be a set of elements. Given a set X of comparison queries about elements from S with their Yes/No answers, construct a directed graph with vertices S and for each (x, y) involved in a query, an edge $x \to y$ if the answer of that query was $x > y$.

Then there is a permutation of S compatible with the queries in X if and only if the graph has no cycle (is a DAG).

Lemma: Let S be a set of elements. Given a set X of comparison queries about elements from S with their Yes/No answers, construct a directed graph with vertices S and for each (x, y) involved in a query, an edge $x \to y$ if the answer of that query was $x > y$.

Then there is a permutation of S compatible with the queries in X if and only if the graph has no cycle (is a DAG).

Proof: **If no cycle:** then there is an element x with no incoming edge; make x the first (largest) in the permutation. Delete x and its outgoing edges, then recurse on the remaining set of vertices.

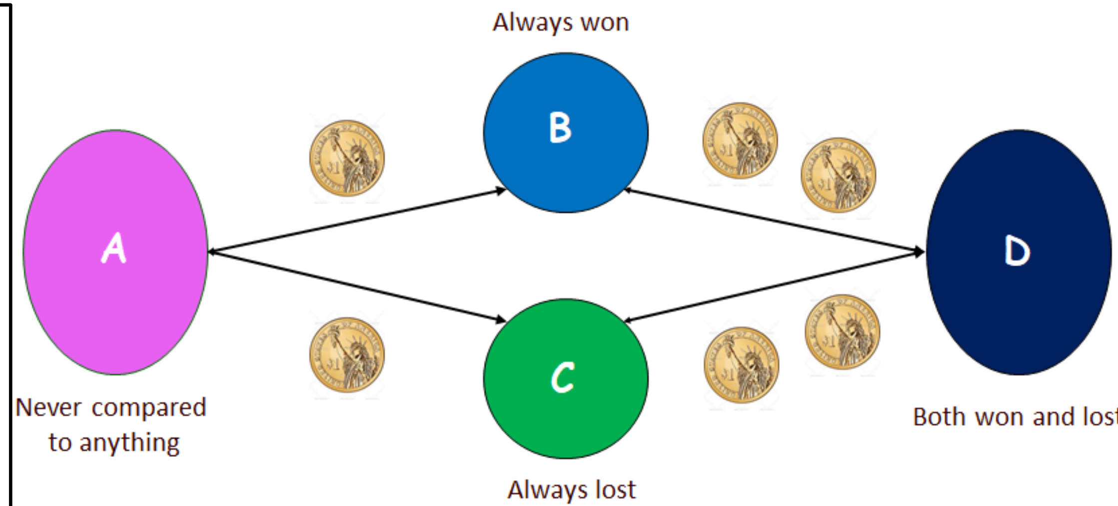
Lemma: Let S be a set of elements. Given a set X of comparison queries about elements from S with their Yes/No answers, construct a directed graph with vertices S and for each (x, y) involved in a query, an edge $x \to y$ if the answer of that query was $x > y$.

Then there is a permutation of S compatible with the queries in X if and only if the graph has no cycle (is a DAG).

Proof: **If there is a permutation compatible with the queries:** then for any subset of elements, there can be no cycle (can't have an element be the min and at the same time beat the max).

Summary:

Devise an adversary strategy for answering queries posed by M, so that each query does not gather more than 2 points. Say the current query asked by M is “x < y”?.

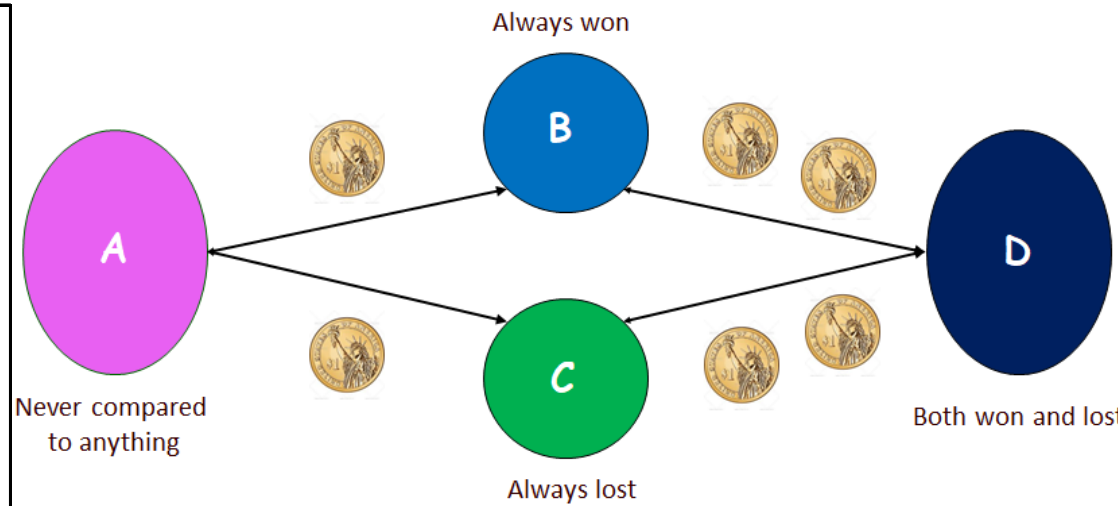


Cases:

- $x, y \in A$: Let x move to B and y to C ➔ gain 2 points
- $x, y \in B$: Make x win (it stays in B), y moves to D ➔ gain 2 points for y. Similar if $x, y \in C$.
- $x \in B, y \in C$: Make x win ➔ both stay put (no points).
- $x \in (A \cup B \cup C \cup D), y \in D$: Answer in a way that is compatible with at least one existing permutation that is still feasible ➔ gain at most 2 points. (Can test all cases quickly)

Summary:

Devise an adversary strategy for answering queries posed by M, so that each query does not gather more than 2 points. Say the current query asked by M is "x < y"?.

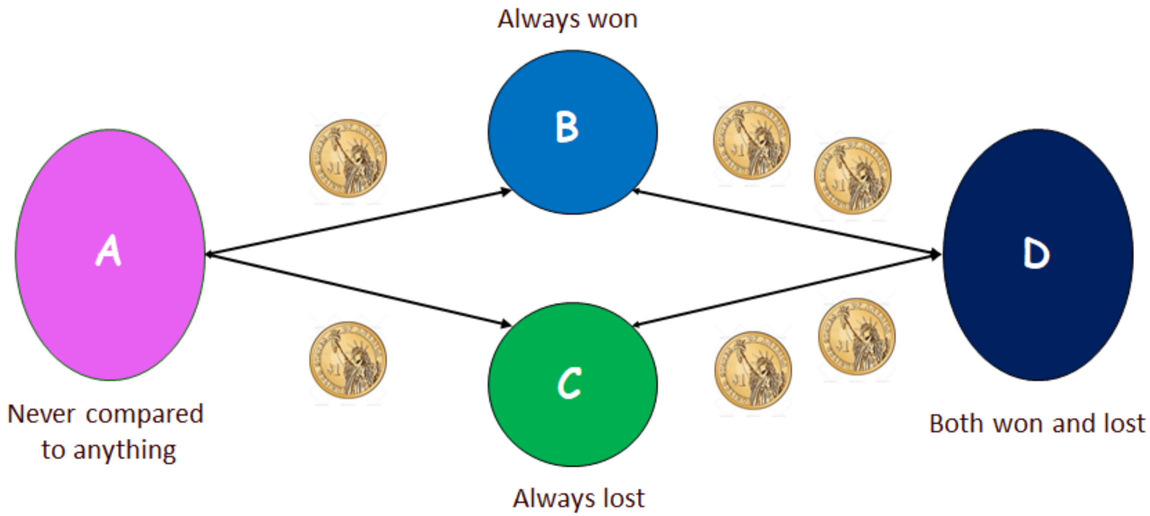


Cases:

- $x, y \in A$: Let x move to B and y to C ➔ gain 2 points
- $x, y \in B$: Make x win (it stays in B), y moves to D ➔ gain 2 points for y. Similar if $x, y \in C$.
- $x \in B, y \in C$: Make x win ➔ both stay put (no points).
- $x \in (A \cup B \cup C \cup D), y \in D$: Answer in a way that is compatible with at least one existing permutation that is still feasible ➔ gain at most 2 points. (Can test all cases quickly)

Question: How can we use this construction to derive a lower bound on the number of comparisons of the algorithm?

Proof of lower bound (cont.)

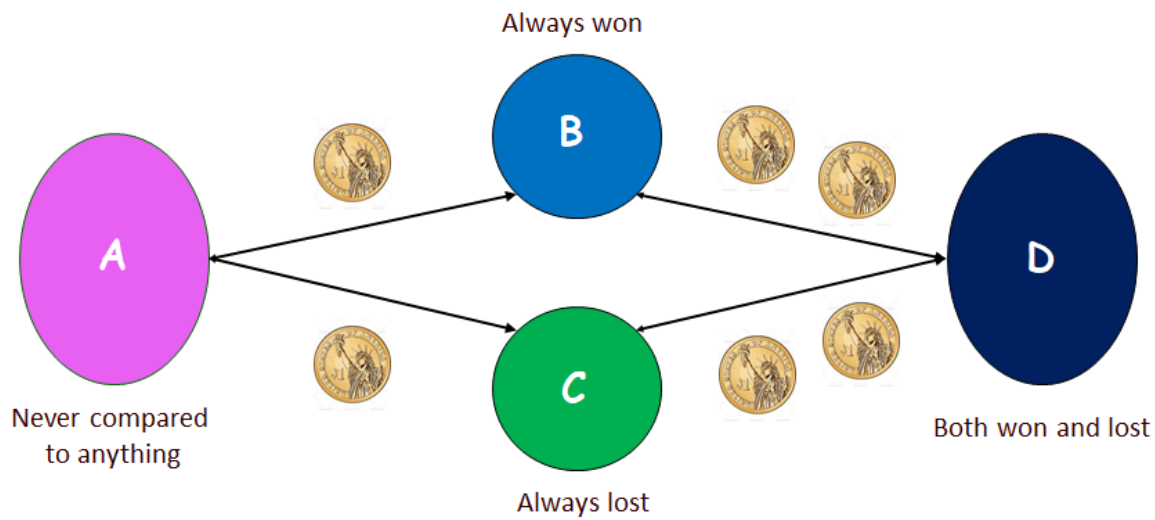


In each of the four cases we progress with ≤ 2 points in each step. We need

- $3 \cdot (\mathrm{n} - 2)$ points from elements that are not min or max (an element must win some comparison and lose some other comparison to be discarded as a candidate for max and min)
- 2 more points, one for the min and one for the max.

Then can we say we need at least [TBD] number of steps?

Proof of lower bound (cont.)



In each of the four cases we progress with ≤ 2 points in each step. We need

- $3 \cdot (n-2)$ points from elements that are not min or max (an element must win some comparison and lose some other comparison to be discarded as a candidate for max and min)
- 2 more points, one for the min and one for the max.

The number of steps is $\geq \frac{\text{total points needed}}{\text{max points per step}} \geq \frac{3\cdot(n-2)+2}{2} = \frac{3n}{2} - 2$.