Adversary Arguments

A method for obtaining lower bounds
What is an Adversary?

- A Second Algorithm Which Intercepts Access to Data Structures
- Constructs the input data only as needed
- Attempts to make original algorithm work as hard as possible
- Analyze Adversary to obtain lower bound
Important Restriction

- Although data is created dynamically, it must return consistent results.

Max and Min

- Keep values and status codes for all keys
- Codes: N-never used
  - W-won once but never lost
  - L-lost once but never won
  - WL-won and lost at least once
- Key values will be arranged to make answers to come out right
When comparing $x$ and $y$

<table>
<thead>
<tr>
<th>Status</th>
<th>Response</th>
<th>NewStat</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>N,N</td>
<td>$x&gt;y$</td>
<td>W,L</td>
<td>2</td>
</tr>
<tr>
<td>W,N</td>
<td>$x&gt;y$</td>
<td>W,L</td>
<td>1</td>
</tr>
<tr>
<td>WL,N</td>
<td>$x&gt;y$</td>
<td>WL,L</td>
<td>1</td>
</tr>
<tr>
<td>L,N</td>
<td>$x&lt;y$</td>
<td>L,W</td>
<td>1</td>
</tr>
<tr>
<td>W,W</td>
<td>$x&gt;y$</td>
<td>W,WL</td>
<td>1</td>
</tr>
<tr>
<td>L,L</td>
<td>$x&gt;y$</td>
<td>WL,L</td>
<td>1</td>
</tr>
<tr>
<td>W,L; WL,L; W,WL</td>
<td>$x&gt;y$</td>
<td>N/C</td>
<td>0</td>
</tr>
<tr>
<td>L,W; L,WL; WL,W</td>
<td>$x&lt;y$</td>
<td>N/C</td>
<td>0</td>
</tr>
<tr>
<td>WL,WL</td>
<td>Consistent</td>
<td>N/C</td>
<td>0</td>
</tr>
</tbody>
</table>
Accumulating Information

- 2n-2 bits of information are required to solve the problem
- All keys except one must lose, all keys except one must win
- Comparing N,N pairs gives n/2 comparisons and n bits of info
- n-2 additional bits are required
- One comparison each is needed
Results

- 3n/2 - 2 comparisons are needed
  (This is a lower bound.)

- Upper bound is given by the following
  - Compare elements pairwise, put losers in one pile, winners in another pile
  - Find max of winners, min of losers
  - This gives 3n/2 - 2 comparisons

- The algorithm is optimal
Largest and Second Largest

- Second Largest must have lost to largest
- Second Largest is Max of those compared to largest
- Tournament method gives $n-1+\lg n$ comparisons for finding largest and second largest
Second Largest: Adversary

- All keys are assigned weights $w[i]$
- Weights are all initialized to 1
- Adversary replies are based on weights
When x is compared to y

<table>
<thead>
<tr>
<th>Weights</th>
<th>Reply</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w[x] &gt; w[y]$</td>
<td>$x &gt; y$</td>
<td>$w[x] := w[x] + w[y]; w[y] := 0;$</td>
</tr>
<tr>
<td>$w[x] = w[y] &gt; 0$</td>
<td>$x &gt; y$</td>
<td>$w[x] := w[x] + w[y]; w[y] := 0;$</td>
</tr>
<tr>
<td>$w[y] &gt; w[x]$</td>
<td>$y &gt; x$</td>
<td>$w[y] := w[y] + w[x]; w[x] := 0;$</td>
</tr>
<tr>
<td>$w[x] = w[y] = 0$</td>
<td>Consistent</td>
<td>None</td>
</tr>
</tbody>
</table>
Accumulation of Weight

- Solution of the problem requires all weight to be accumulated with one key.
- All other keys must have weight zero.
- Since weight accumulates to highest weight, weight can at most double with each comparison.
- $\lg n$ comparisons are required to accumulate all weight.
The largest key must be compared with $\lg n$ other keys.

Finding the second largest requires at least $\lg n$ comparisons after finding the largest.

This is a lower bound.

The tournament algorithm is optimal.
Adversary Arguments

A method for obtaining lower bounds
What is an Adversary?

- A Second Algorithm Which Intercepts Access to Data Structures
- Constructs the input data only as needed
- Attempts to make original algorithm work as hard as possible
- Analyze Adversary to obtain lower bound
Important Restriction

- Although data is created dynamically, it must return consistent results.

Max and Min

- Keep values and status codes for all keys
- Codes: N-never used
  W-won once but never lost
  L-lost once but never won
  WL-won and lost at least once
- Key values will be arranged to make answers to come out right
When comparing x and y

<table>
<thead>
<tr>
<th>Status</th>
<th>Response</th>
<th>NewStat</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>N,N</td>
<td>x&gt;y</td>
<td>W,L</td>
<td>2</td>
</tr>
<tr>
<td>W,N</td>
<td>x&gt;y</td>
<td>W,L</td>
<td>1</td>
</tr>
<tr>
<td>WL,N</td>
<td>x&gt;y</td>
<td>WL,L</td>
<td>1</td>
</tr>
<tr>
<td>L,N</td>
<td>x&lt;y</td>
<td>L,W</td>
<td>1</td>
</tr>
<tr>
<td>W,W</td>
<td>x&gt;y</td>
<td>W, WL</td>
<td>1</td>
</tr>
<tr>
<td>L,L</td>
<td>x&gt;y</td>
<td>WL,L</td>
<td>1</td>
</tr>
<tr>
<td>W,L; WL,L; W, WL</td>
<td>x&gt;y</td>
<td>N/C</td>
<td>0</td>
</tr>
<tr>
<td>L,W; L, WL; WL, W</td>
<td>x&lt;y</td>
<td>N/C</td>
<td>0</td>
</tr>
<tr>
<td>WL, WL</td>
<td>Consistent</td>
<td>N/C</td>
<td>0</td>
</tr>
</tbody>
</table>
Accumulating Information

- 2n-2 bits of information are required to solve the problem
- All keys except one must lose, all keys except one must win
- Comparing N,N pairs gives n/2 comparisons and n bits of info
- n-2 additional bits are required
- One comparison each is needed
Results

- $3n/2-2$ comparisons are needed (This is a lower bound.)
- Upper bound is given by the following
  - Compare elements pairwise, put losers in one pile, winners in another pile
  - Find max of winners, min of losers
  - This gives $3n/2-2$ comparisons
- The algorithm is optimal
Largest and Second Largest

- Second Largest must have lost to largest
- Second Largest is Max of those compared to largest
- Tournament method gives $n-1+\lg n$ comparisons for finding largest and second largest
Second Largest: Adversary

- All keys are assigned weights $w[i]$
- Weights are all initialized to 1
- Adversary replies are based on weights
<table>
<thead>
<tr>
<th>Weights</th>
<th>Reply</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w[x] &gt; w[y] )</td>
<td>( x &gt; y )</td>
<td>( w[x] := w[x] + w[y]; \ w[y] := 0; )</td>
</tr>
<tr>
<td>( w[x] = w[y] &gt; 0 )</td>
<td>( x &gt; y )</td>
<td>( w[x] := w[x] + w[y]; \ w[y] := 0; )</td>
</tr>
<tr>
<td>( w[y] &gt; w[x] )</td>
<td>( y &gt; x )</td>
<td>( w[y] := w[y] + w[x]; \ w[x] := 0; )</td>
</tr>
<tr>
<td>( w[x] = w[y] = 0 )</td>
<td>Consistent</td>
<td>None</td>
</tr>
</tbody>
</table>
Accumulation of Weight

- Solution of the problem requires all weight to be accumulated with one key.
- All other keys must have weight zero.
- Since weight accumulates to highest weight, weight can at most double with each comparison.
- \( \lg n \) comparisons are required to accumulate all weight.
Results

- The largest key must be compared with \( \lg n \) other keys.
- Finding the second largest requires at least \( \lg n \) comparisons after finding the largest.
- This is a lower bound.
- The tournament algorithm is optimal.