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.
- If it replies that $x[1]<x[2]$, it can never say later that $x[2]<x[1]$.

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

Status	Response	NewStat	Info
N,N	x>y	W,L	2
W,N	x>y	W,L	1
WL,N	x>y	WL,L	1
L,N	x<y	L,W	1
W,W	x>y	W,WL	1
L,L	x>y	WL,L	1
W,L; WL,L; W,WL	x>y	N/C	0
L,W; L,WL; WL,W	x<y	N/C	0
WL,WL	Consistent	N/C	0

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

Weights	Reply	Changes
w[x]>w[y]	x>y	w[x]:=w[x]+w[y]; w[y]:=0;
w[x]=w[y]>0	x>y	w[x]:=w[x]+w[y]; w[y]:=0;
w[y]>w[x]	y>x	w[y]:=w[y]+w[x]; w[x]:=0;
w[x]=w[y]=0	Consistent	None

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