PREPROCESSING, LOWER BOUNDS

THEOREM: Every pairwise-comparison based sorting algorithm must do $\Omega(n \lg n)$ swaps in the worst-case.

PROOF: Can be modeled by binary decision tree, which must have $\geq n!$ leaves. Any binary tree of *m* leaves must have height $\lg m$, so there must be a path (sequence of comparisons) of length at least

$$\lg n! = \lg n(n-1)\ldots 1 \geq \lg n(n-1)\ldots(n/2) \geq \lg(n/2)^{n/2} = \frac{n}{2}(\lg n - 1) \in \Omega(n \lg n).$$

THEOREM: Every pairwise-comparison based sorting algorithm must do $\Omega(n \lg n)$ swaps in the average-case. (**Problem *8-1***)

SELECTIONSORT (EXERCISE 2.2-2)

for $i \leftarrow n$ **downto** 2 **do**
  swap($A[i]$,MAX of $A[1..i]$)

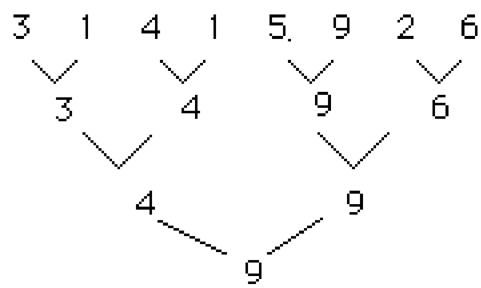
<u>Analysis</u>: To find MAX of $A[1..i]$, $\geq i-1$ pairwise comparisons (worst(best,average)-case). Why?

So: $T_{a.c.(w.c.)(b.c.)}(n) = \sum_{2 \leq i \leq n} (i-1) = \sum_{1 \leq i \leq n-1} i \in \Theta(n^2)$

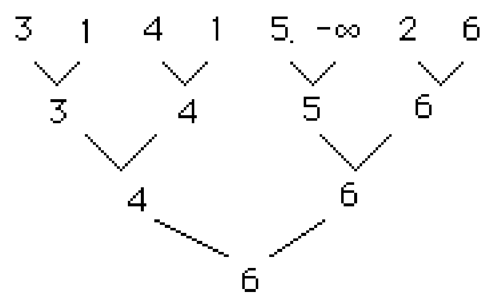
<u>TreeSort:</u> Having found MAX, we have (essentially) no help in finding $2^{\underline{nd}}$MAX. Arrange tests for MAX to facilitate test for $2^{\underline{nd}}$MAX.

<u>SUBPROBLEM</u>: Find MAX and 2^{nd}MAX using pairwise comparisons.

Set up a tournament: To find MAX of 3 1 4 1 5 9 2 6



We know that $2^{\underline{nd}}$MAX lost to MAX. Only need check lg*n* losers (trace MAX to its leaf, (lg*n*), replace it by -∞, & re-run the lg*n* competitions involving MAX).



So we get 6 is $2^{\underline{nd}}$ MAX. Repeat...

Can find $k^{\underline{th}}$ largest in $n+k\lg n$ comparisons.

<u>Lower bound: **Adversary Arguments**</u>: What is worst-case complexity of *Find*ing $2^{\underline{nd}}$ MAX elements of a set, that is, for any (binary) decision tree to *Find* $2^{\underline{nd}}$ MAX, what is height? We know there is some tree with a leaf (<u>best-case</u>) at distance *n*-1.

if $A[1]<A[2]$
then {*Big*←*A*[2]; *Second*←*A*[1];}

else {*Big*←*A*[1]; *Second*←*A*[2];}
for *i*←3 **to** *n* **do**
if *Second*<*A*[*i*] **then if** *Big*<*A*[*i*] **then** {*Second*←*Big*;*Big*← *A*[*i*];}
else *Second*←*A*[*i*]

An adversary makes an algorithm look as bad as possible. What is worst-case for the above? **Note** that adversary (and reality) makes candidates known to be big win.

‣ Prove that 2nd MAX must "know" MAX ⇒If not, adversary could flip input to make 2nd MAX (which never lost) equal ∞. We know we need n-1 comparisons for MAX. We want to maximize ignorance for algorithm after it knows MAX, i.e., maximize # of losers to *Big*.

At any point in computation, for each $A[i]$, we define

$w(i)$ = **if** $A[i]$ can't be max **then** 0 **else** $|\{j \mid \text{we know } A[j] \le A[i]\}|$

Initially, $(\forall i) w(i)=1$; finally $w(\text{MAX})=n$.

Adversary's Algorithm:

Algorithm says "compare $A[i]$ to $A[j]$"

We say: **if** $w(i) \ge w(j)$ **then** $A[i]>A[j]$ $\{w(i) \leftarrow w(j)+w(i), w(j) \leftarrow 0\}$

else $A[j]>A[i]$ $\{w(j) \leftarrow w(j)+w(i), w(i) \leftarrow 0\}$

Analysis of any algorithm's performance:

We know that 2nd MAX has lost a comparison to MAX.

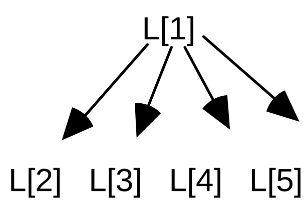
Worst-case - what is smallest number of keys which lost to MAX?

$w(\text{MAX})$ after comparison $\le 2*w(\text{MAX})$ before comparison

fewest # losers to MAX occurs when MAX doubles each time - lg$n$

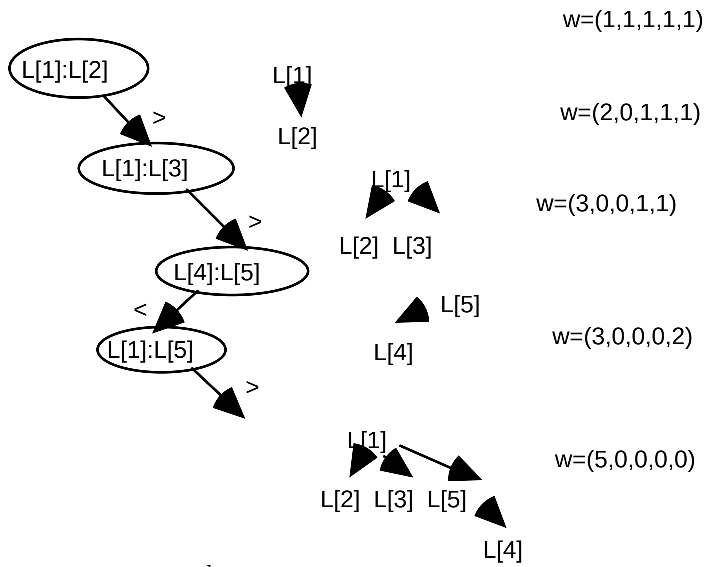
Adversary in action: n=5, sequential search

(A) $L[1]$ wins all. Initially w=(1,1,1,1,1); finally w=(5,0,0,0,0). We know



and we could assign weights accordingly.

(B)



& there are 3 candidates for 2nd MAX. Note that values are assignable consistent with the ranking.

<u>4 Finding *MAX* **and** *MIN*</u>

An algorithm to find *MAX*-*MIN* using n-1 comparisons (in the best-case):

if $A[1]<A[2]$ **then** $Big \leftarrow A[2]$
$\quad Little \leftarrow A[1]$
else $Big \leftarrow A[1]$
$\quad Little \leftarrow A[2]$
for $i \leftarrow 3$ **to** n **do**
if $Big < A[i]$ **then** $Big \leftarrow A[i]$
else if $Little > A[i]$ **then** $Little \leftarrow A[i]$

Average case, assuming all elements distinct & all permutations of A equally likely.

$2n - H_n - \frac{3}{2}$ pairwise comparisons.

<u>But</u>, worst-case - $2n$-3.

Assume n even

$Winners \leftarrow \emptyset$
$Losers \leftarrow \emptyset$
for $k \leftarrow 2$ **to** n **by** 2 **do**
compare $L[k]$: $L[k-1]$, put larger in *Winners*
smaller in *Losers* $\qquad n/2$
return (max(*Winners*), min(*Losers*)) $\qquad 2\left(\frac{n}{2}-1\right)$

<u>Lower-bound on *MAX*-*MIN*</u> State-space approach.

Divide L into 4 categories:	cardinality
*B*eginners - neither known to be *MAX* nor *MIN*	b
*W*inners - may be *MAX* , definitely not *MIN*	w
*L*osers - may be *MIN*, definitely not *MAX*	l
*O*thers - definitely neither *MAX* nor *MIN*	o

Initially, $(b,w,l,o)=(n,0,0,0)$; finally $(b,w,l,o)=(0,1,1,n-2)$

Comparisons $(b,w,l,o) \rightarrow$

$B{:}B$	$(b\text{-}2,w+1,l+1,o)$	
$B{:}W$	$(b\text{-}1,w,l+1,o) \mid \cancel{(b\text{-}1,w,l,o+1)}$	adversary eliminates 2nd
$B{:}L$	$(b\text{-}1,w+1,l,o) \mid \cancel{(b\text{-}1,w,l,o+1)}$	adversary eliminates 2nd
$B{:}O$	$(b\text{-}1,w+1,l,o) \mid \cancel{(b\text{-}1,w,l}+1\cancel{,o)}$	adversary eliminates 2nd
WW	$(b,w\text{-}1,l,o+1)$	
$W{:}L$	$(b,w,l,o) \mid \cancel{(b,w\text{-}1,l}\text{-}1\cancel{,o+2)}$	adversary eliminates 2nd
$W{:}O$	$(b,w,l,o) \mid \cancel{(b,w\text{-}1,l,o}+\cancel{1)}$	adversary eliminates 2nd
$L{:}L$	$(b,w,l\text{-}1,o+1)$	
$L{:}O$	$(b,w,l,o) \mid \cancel{(b,w,l\text{-}1,o+1)}$	adversary eliminates 2nd
$O{:}O$	(b,w,l,o)	

It takes $\geq \left\lceil \frac{n}{2} \right\rceil$ comparisons to empty *B*. Adversary can assure that nothing goes from $B \to O$. It then takes n-2 to empty *W* and *L*. Note that the lower bound yields an algorithm.