Lower Bound of Comparison-based Sorting O(nlogn)

sorting/selecting algorithms—comparison based

time cost → # comparisons

Finding Maximum Algorithm

1. cur.max = bigger(1, 2)
2. for i = 3 to n
3.   cur.max = bigger(cur_max, i)
4. retrun cur.max

comparision = n-1

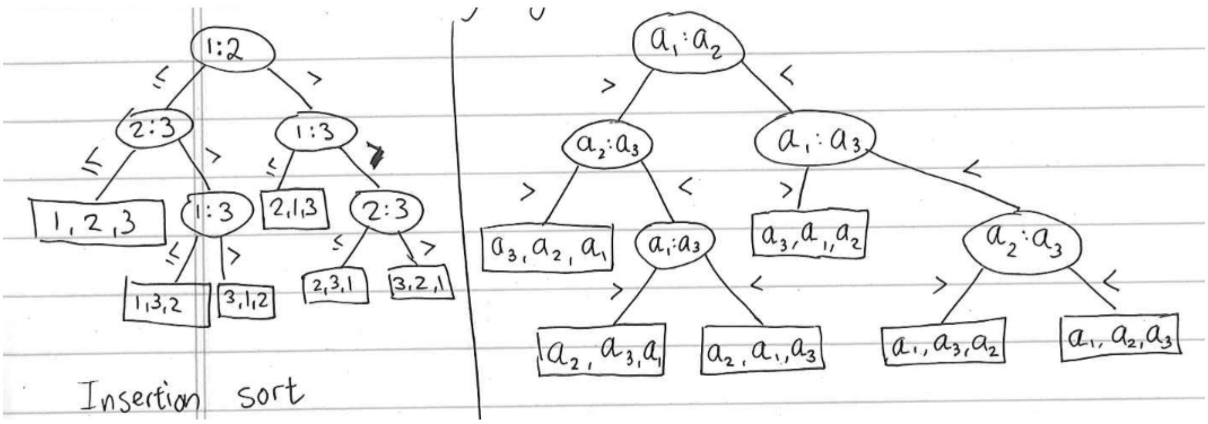
Claim: Computing maximum of n elements requires ≥ n-1 comparisons

Proof: There can be at most one element that has never lost a comparison. Otherwise, each of the two elements can potentially be the maximum and the algorithm has no way of telling.

Since there is one loser in each comparison, we need n-1 comparisons.

Sorting

We can model any comparison-based algorithm as a decision tree.



We have 6 leaves correspoinding to all 3! = 6 permutations of the input.

In general, when sorting n elements, any correct algorithm must have at least n! leaves, since all permutations are possible.

What is number of comparisons in the worst case? It is the depth of the tree.

$k = log(n!) = \Omega(nlogn)$ since $n! > (\frac{n}{2})^{\frac{n}{2}} \rightarrow log(n!) > \frac{n}{2}log(\frac{n}{2})$

Therefore, any comparison-based sorting algorithm has at least $\Omega(nlogn)$ comparisons. In particular, MergeSort and QuickSort (with median pivot) are optimal.