How fast can we sort?

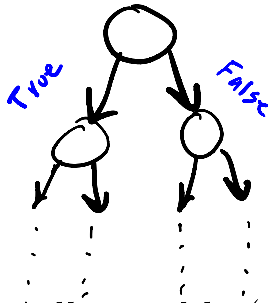
Insertion sort: $O(n^2)$
Merge sort: $\Theta(n \log n)$
Quicksort: $\Theta(n \log n)$
. (expected runtime, with $O(n^2)$ worst case)
Heapsort: $\Theta(n \log n)$

Can we do better than $\Theta(n \log n)$?

All of these sorts are based on comparing pairs of elements in the array so we can model them using decision trees. = Visual representation of comparisons for algorithm

↳ Tall trees

- We have a different tree for each input size n (i.e., the number of elements in the array we are sorting).



$x < y$

- The tree contains all possible (= if-branches) that could be executed for any array of n elements.

- For one particular array, there is a unique path down to a leaf which is executed.

- Running time = the length of the path taken

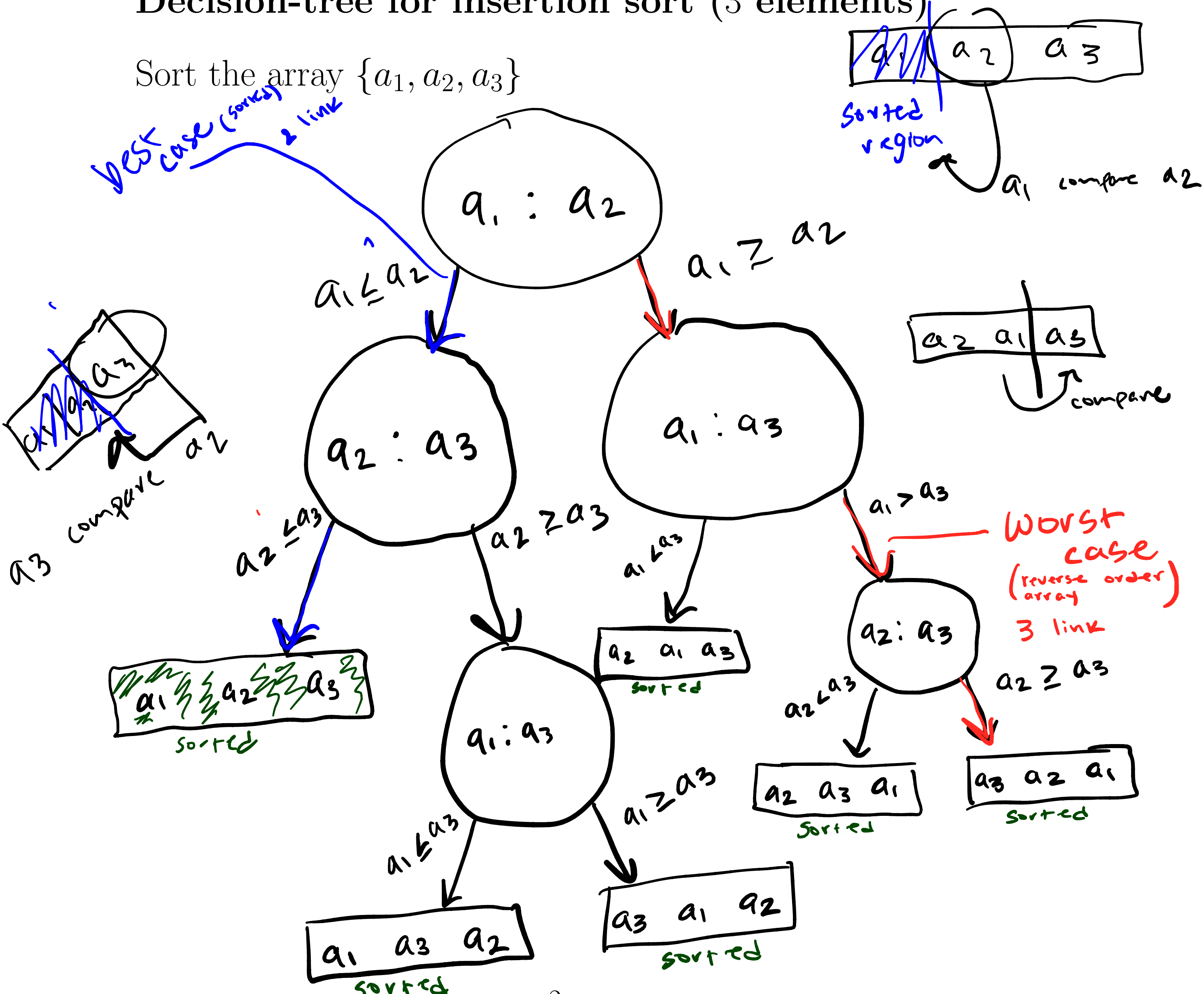
- The Worst case running time = height of the tree

Algorithm 1 int[] insertionSort(int $A[1 \dots n]$)

```
j = 2;
while j <= n do
    key = A[j];
    i = j - 1;
    while (i > 0) and (A[i] > key) do
        A[i + 1] = A[i];
        i--;
    end while
    A[i + 1] = key;
    j++;
end while
return A;
```

Decision-tree for insertion sort (3 elements)

Sort the array $\{a_1, a_2, a_3\}$



$n = 3$

$3! = 6$ permutations

↳ # of leaf nodes

Lower-bound for comparison sorting

Theorem Any decision tree that can sort n elements must have height $\Omega(n \log n)$.

(thus our sort has a worst case runtime of $\overset{\text{lower bound}}{\Omega}(n \log n)$)

Proof:

Suppose our decision tree has height h. The tree must contain $\geq n!$ leaves, since there are $n!$ possible permutations of the elements in our array.

relationship between # of leaves & height of tree

A height h binary tree has $\leq 2^h$ leaves. Thus, $2^h \geq n!$

$\Rightarrow$ This tells us...

$$\Rightarrow \cancel{\log_2}(\cancel{2}^h) \geq \log_2(n!)$$

$$\Rightarrow h \geq \log_2(n!)$$ // h must at least equal $\log_2(n!)$

// $\log_2(n!) \in \Theta(n \log n)$ // from lec. 7

$$\Rightarrow h \in \Omega(n \log n)$$

---

another way to solve

$$n! = \underbrace{1 \cdot 2 \cdot 3 \cdot 4}_{\text{replace w/ 1: } 1\cdot 1\cdot 1\cdot 1} \cdot \ldots \cdot (n/2 - 1) \cdot (n/2) \cdot \underbrace{(n/2+1) \cdot \ldots \cdot (n-1) \cdot n}_{\text{replace w/ } n/2} \rightarrow n/2 \cdot n/2 \cdot n/2$$

$$= (n/2)^{n/2} \Rightarrow \log_2(n/2)^{n/2} = n/2 \log_2(n/2) = n \log n$$

// replacing because doing so makes problem smaller equal aka lower bound (Ω)

Theorem In a decision tree that can sort n elements, the average level of a leaf is $\Omega(n \log n)$.

(thus our sort has an average case runtime of $\Omega(n \log n)$)

Corollary Heapsort and merge sort are asymptotically optimal comparison sorting algorithms.

So we can't do better using comparison-based sorting.

Can we sort without comparing pairs of elements in our array? — Is this even possible???

— yes

- Counting Sort: range of #s ; $O(n+k)$ k is range of # (1 – 100) k = 100
- Radix Sort: # of bits of e/a number

The runtimes of these sorts both depend on the number of elements in the array being sorted, n, (obviously) and the size of the largest element in the array.

For both sorts, if we have a constant bound on the size of the elements in the array (e.g., all elements in the array are 1000 or less) then their asymptotic runtimes are $\Theta(n)$.