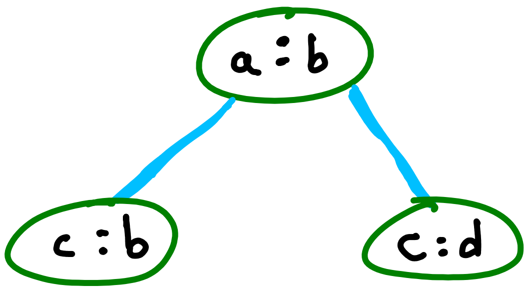
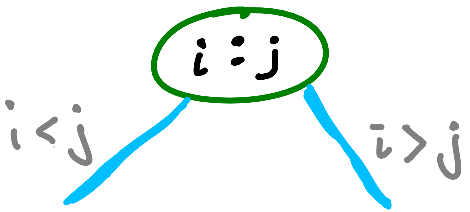
ALGORITHMS REPRESENTED AS DECISION TREES



internal nodes
represent comparison
of two elements

ALGORITHMS REPRESENTED AS DECISION TREES

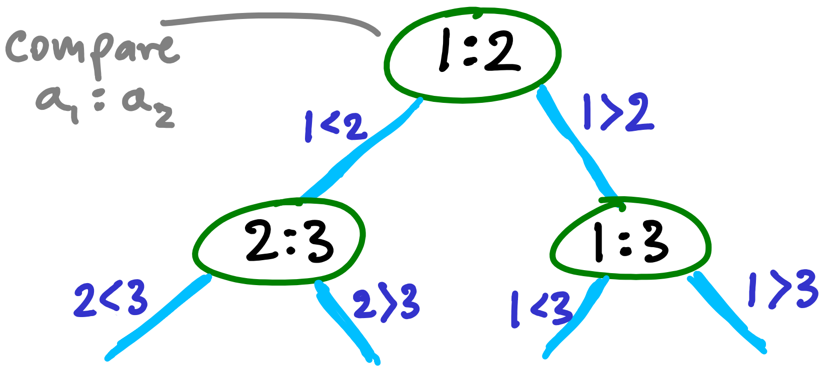


internal nodes represent comparison of two elements i,j

branches represent outcome of comparison

Left : $i<j$
Right $i>j$

Algorithms represented as Decision Trees



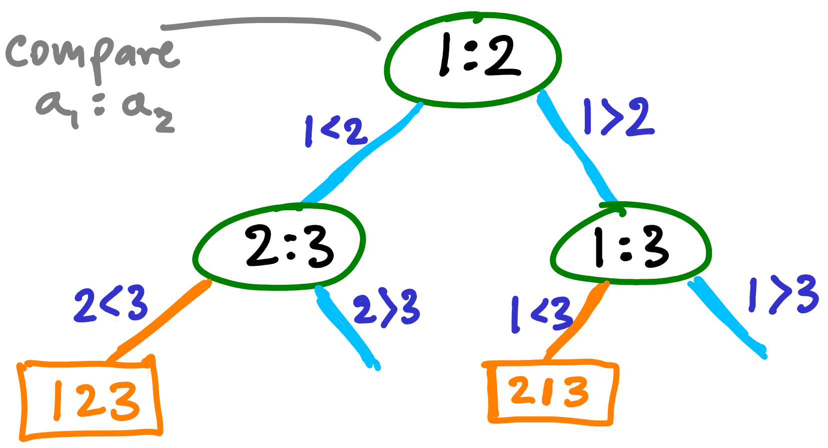
internal nodes represent comparison of two elements i,j

branches represent outcome of comparison

↳ Left : $i<j$
Right $i>j$

Example: sort $a_1 a_2 a_3$

Algorithms represented as Decision Trees



internal nodes represent comparison of two elements i, j

branches represent outcome of comparison

Left : $i < j$
Right $i > j$

Each leaf is a possible output

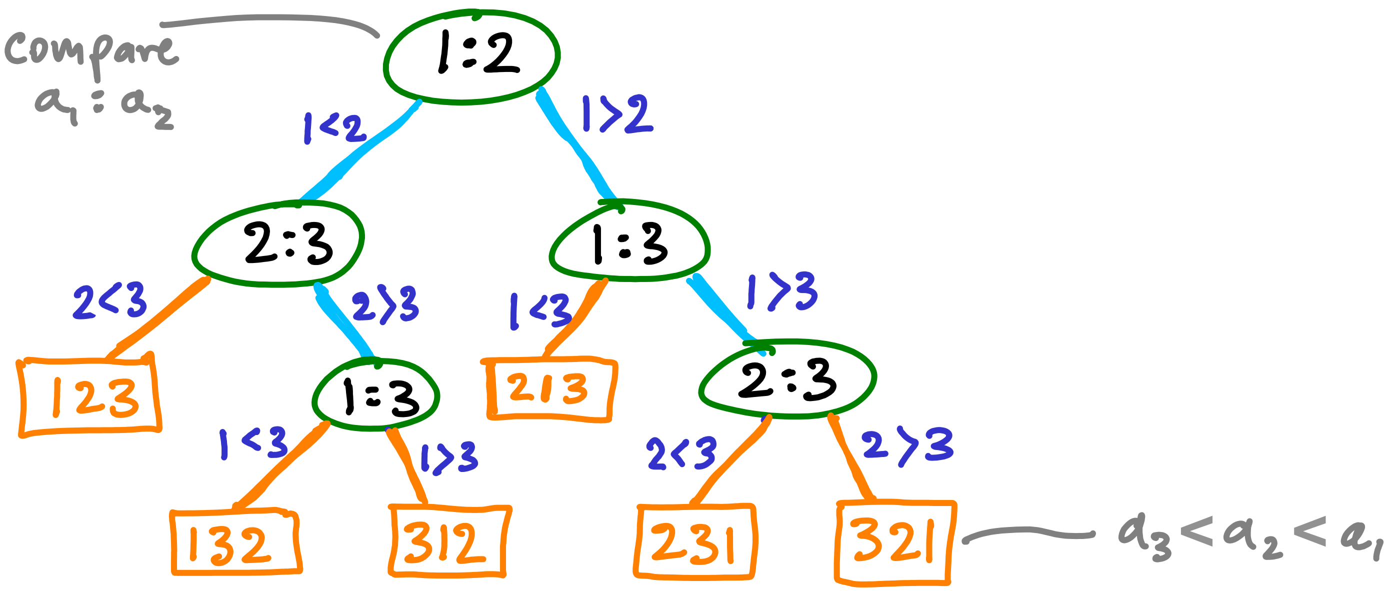
Example: sort $a_1 a_2 a_3$

ALGORITHMS REPRESENTED AS DECISION TREES

internal nodes represent comparison of two elements i,j

branches represent outcome of comparison

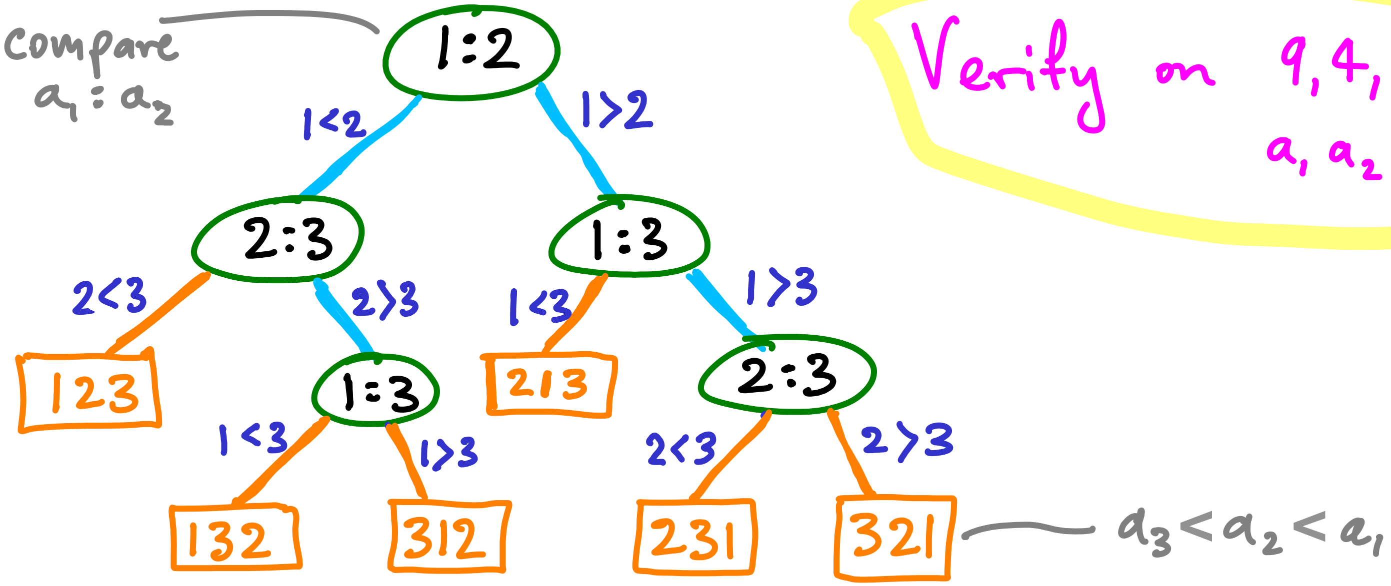
Left : $i<j$
Right $i>j$



Each leaf is a possible output

Example: sort $a_1 a_2 a_3$

ALGORITHMS REPRESENTED AS DECISION TREES



Verify on 9, 4, 6
$a_1\ a_2\ a_3$

internal nodes represent comparison of two elements i, j

branches represent outcome of comparison
Left : $i<j$
Right $i>j$

Each leaf is a possible output

Example: sort $a_1 a_2 a_3$

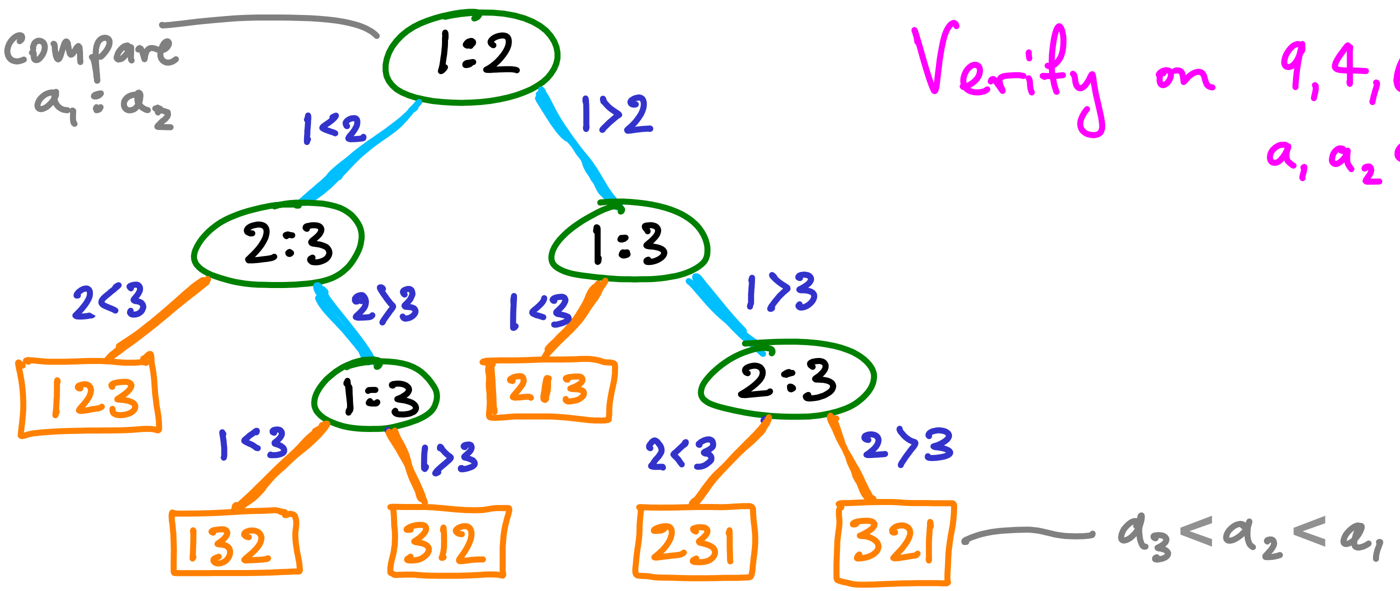
ALGORITHMS REPRESENTED AS DECISION TREES

internal nodes represent comparison of two elements i,j

branches represent outcome of comparison

↳ Left : $i<j$
Right $i>j$

Example: sort $a_1 a_2 a_3$



Verify on 9,4,6
$a_1\ a_2\ a_3$

Each leaf is a possible output

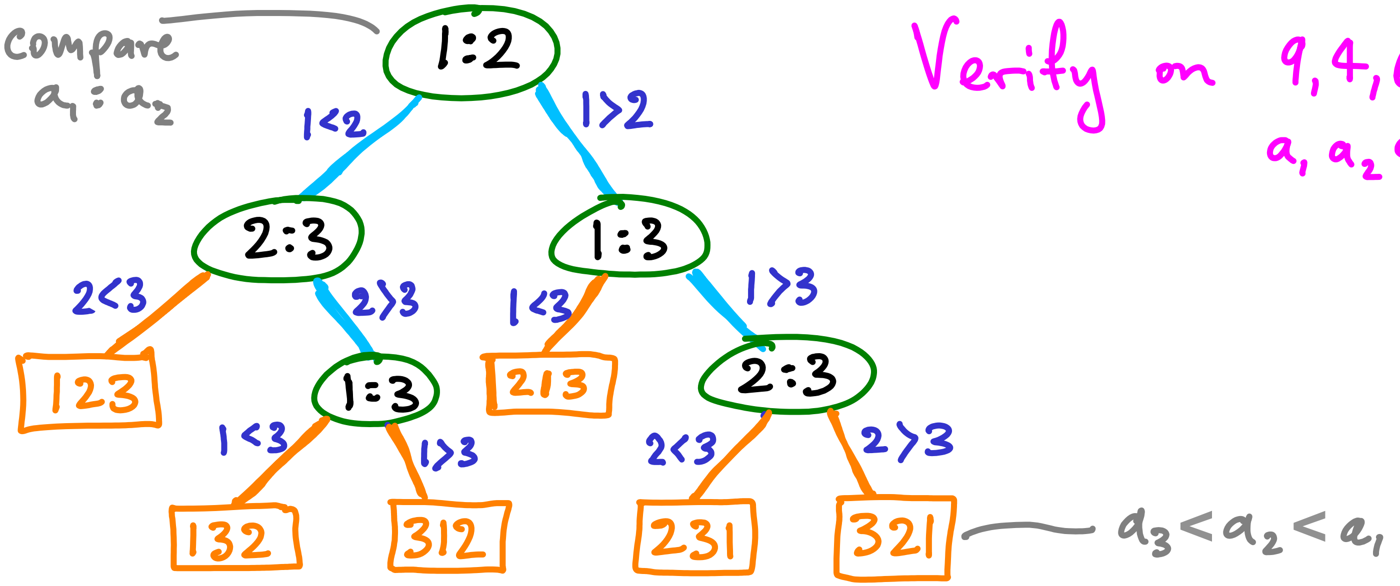
Each root→leaf path represents an execution of algo.

ALGORITHMS REPRESENTED AS DECISION TREES

internal nodes represent comparison of two elements i,j

branches represent outcome of comparison
Left : $i<j$
Right $i>j$

Example: sort $a_1 a_2 a_3$



Verify on 9,4,6
$a_1\ a_2\ a_3$

Each leaf is a possible output

Each root→leaf path represents an execution of algo.

$\boxed{\text{Any decision-based algorithm can be encoded as a decision tree.}}$

If you are designing a decision tree
it's up to you to avoid comparing the same elements many times.

If you are designing a decision tree
it's up to you to avoid comparing the same elements many times.

The worst-case run-time is <u>precisely</u> the longest root-leaf path.

If you are designing a decision tree
it's up to you to avoid comparing the same elements many times.

The worst-case run-time is precisely the longest root-leaf path.

you shouldn't compare $a_i : a_j$ twice on one path.

↳ so max path length $= \binom{n}{2}$

for a good algo

for sorting

If you are designing a decision tree
it's up to you to avoid comparing the same elements many times.

The worst-case run-time is precisely the longest root-leaf path.

you shouldn't compare $a_i : a_j$ twice on one path.
↳ so max path length $= \binom{n}{2}$

Why not write all algorithms this way? (so much prettier than pseudocode)

If you are designing a decision tree
it's up to you to avoid comparing the same elements many times.

The worst-case run-time is precisely the longest root-leaf path.

you shouldn't compare $a_i : a_j$ twice on one path.

↳ so max path length $= \binom{n}{2}$

Why not write all algorithms this way? (so much prettier than pseudocode)

↳ It's huge and repetitive.
It really lists every possible execution of algo.

If you are designing a decision tree
it's up to you to avoid comparing the same elements many times.

The worst-case run-time is precisely the longest root-leaf path.

you shouldn't compare $a_i : a_j$ twice on one path.
↳ so max path length $= \binom{n}{2}$

Why not write all algorithms this way? (so much prettier than pseudocode)
↳ It's huge and repetitive.
It really lists every possible execution of algo.
You actually might need a different tree for each n.

If you are designing a decision tree
it's up to you to avoid comparing the same elements many times.

The worst-case run-time is precisely the longest root-leaf path.

you shouldn't compare $a_i : a_j$ twice on one path.
↳ so max path length $= \binom{n}{2}$

Why not write all algorithms this way? (so much prettier than pseudocode)
↳ It's huge and repetitive.
It really lists every possible execution of algo.
You actually might need a different tree for each n.

What is the shortest possible tree for comparison-sort?

A correct decision tree for sorting must have every possible output represented at a leaf node.

$\hookrightarrow$ # leaves $\geq$?

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.

↳ # leaves $\geq$?

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.

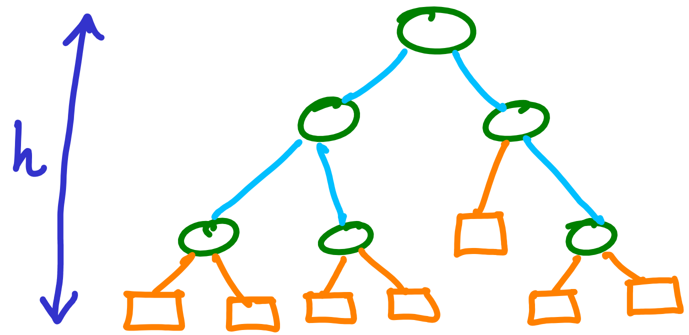
↳ $\#\text{leaves} \geq n!$

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.

$\hookrightarrow$ # leaves $\geq n!$

height of tree = worst case time = h $\rightsquigarrow$ # leaves $\leq$?

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.

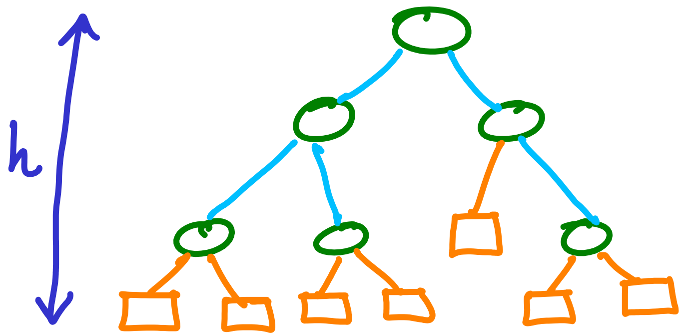


↳ # leaves $\geq n!$

height of tree = worst case time = h ⇝ # leaves $\leq 2^h$

[binary tree; every node has 2 children]

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.



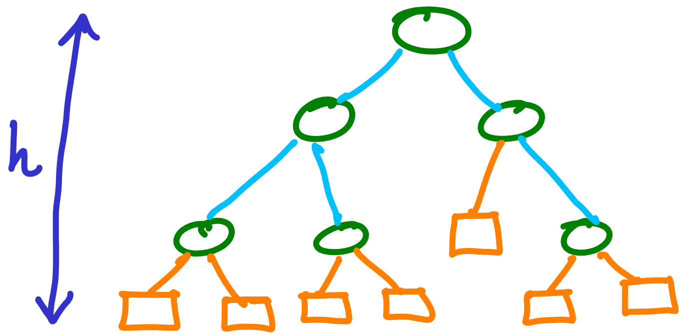
$\hookrightarrow$ # leaves $\geq n!$

height of tree = worst case time = h $\rightsquigarrow$ # leaves $\leq 2^h$

[binary tree; every node has 2 children]

so, $n! \leq$ # leaves $\leq 2^h$

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.



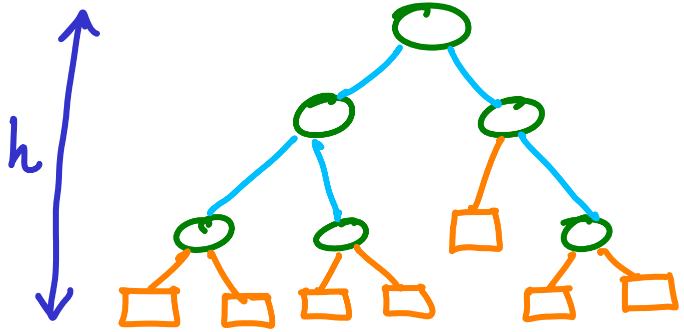
↳ # leaves $\geq n!$

height of tree = worst case time = h $\rightsquigarrow$ # leaves $\leq 2^h$

[binary tree; every node has 2 children]

so, $n! \leq$ # leaves $\leq 2^h \Rightarrow \log n! \leq \log 2^h$

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.



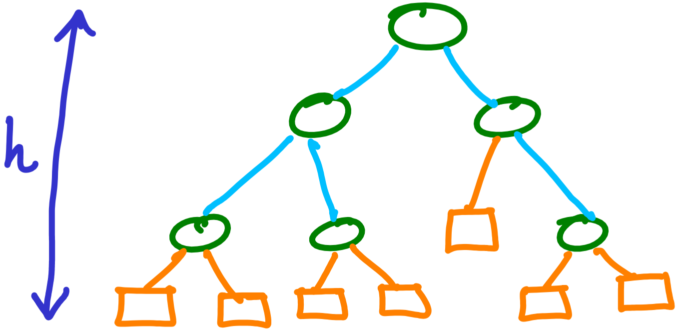
↳ $\#\text{leaves} \geq n!$

height of tree = worst case time = h ⇝ $\#\text{leaves} \leq 2^h$

[binary tree; every node has 2 children]

so, $n! \leq \#\text{leaves} \leq 2^h \Rightarrow \log n! \leq \log 2^h \Rightarrow \underline{h \geq \log n!}$

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.



↳ # leaves $\geq n!$

height of tree = worst case time = h ⇝ # leaves $\leq 2^h$

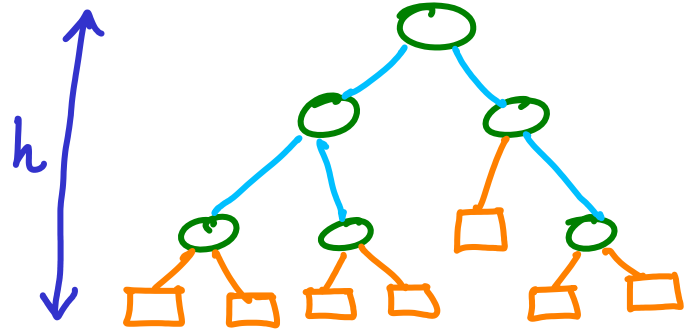
[binary tree; every node has 2 children]

so, $n! \leq \#\text{leaves} \leq 2^h \Rightarrow \log n! \leq \log 2^h \Rightarrow \underline{h \geq \log n!}$

Stirling's formula: $n! \geq \left(\frac{n}{e}\right)^n$

$h \geq \log\left(\frac{n}{e}\right)^n$

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.



↳ # leaves $\geqslant n!$

height of tree = worst case time = h ⇝ # leaves $\leqslant 2^h$

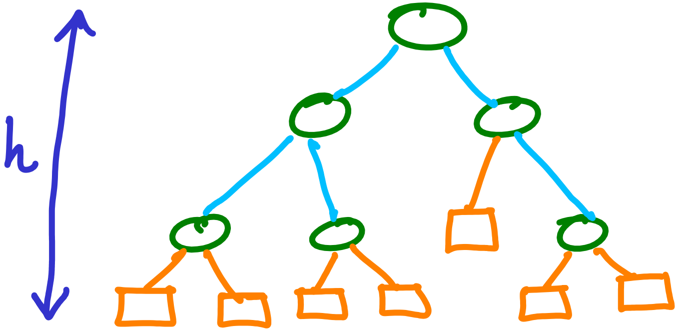
[binary tree; every node has 2 children]

so, $n! \leqslant \#\text{leaves} \leqslant 2^h \Rightarrow \log n! \leqslant \log 2^h \Rightarrow \underline{h \geqslant \log n!}$

Stirling's formula: $n! \geqslant \left(\frac{n}{e}\right)^n$

$h \geqslant \log\left(\frac{n}{e}\right)^n = n \cdot \log \frac{n}{e}$

A correct decision tree for sorting must have every permutation of the input represented at a leaf node.



$\hookrightarrow$ # leaves $\geq n!$

height of tree = worst case time = h $\rightsquigarrow$ # leaves $\leq 2^h$

[binary tree; every node has 2 children]

so, $n! \leq \#\text{leaves} \leq 2^h \Rightarrow \log n! \leq \log 2^h \Rightarrow \underline{h \geq \log n!}$

Stirling's formula: $n! \geq \left(\frac{n}{e}\right)^n$

$$h \geq \log\left(\frac{n}{e}\right)^n = n \cdot \log \frac{n}{e}$$
$$= n\log n - n\log e$$
$$= n\log n - \Theta(n)$$

$$h = \Omega(n\log n)$$

extra analysis of logn! follows

$\log(n!) = O(?)$

$$\log(n!) = O(?)$$

$$\log(n!) \leq \log(n^n)$$

$$\log(n!) = O(?)$$

$$\log(n!) \leq \log(n^n) = n \log n$$

$$\log(n!) = O(n \log n)$$

$$\log(n!) \leq \log(n^n) = n \log n$$

$$\log(n!) = \Omega(?)$$

$$\log(n!) = O(n \log n)$$

$$\log(n!) \leq \log(n^n) = n \log n$$

$$\log(n!) = \Omega(?)$$

$$\log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots \cdots 3 \cdot 2 \cdot 1)$$

$$\log(n!) = O(n \log n)$$

$$\log(n!) \leq \log(n^n) = n \log n$$

$$\log(n!) = \Omega(?)$$

$$\log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots \cdots 3 \cdot 2 \cdot 1)$$

$$= \log(n \cdot 1 \cdot (n-1) \cdot 2 \cdot (n-2) \cdot 3 \cdot (n-3) \cdot 4 \cdots \cdots \sim (n-\tfrac{n}{2}) \cdot (n-\tfrac{n}{2}))$$

exactly if n: even

$$\log(n!) = O(n \log n)$$

$$\log(n!) \leq \log(n^n) = n \log n$$

$$\log(n!) = \Omega(?)$$

$$\log(n!) = \log\left(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots \cdots 3 \cdot 2 \cdot 1\right)$$

$$= \log\left(\underbrace{n \cdot 1} \cdot \underbrace{(n-1) \cdot 2} \cdot \underbrace{(n-2) \cdot 3} \cdot \underbrace{(n-3) \cdot 4} \cdots \cdots \sim \underbrace{\left(n - \tfrac{n}{2}\right) \cdot \left(n - \tfrac{n}{2}\right)}\right)$$

$$\geq \log\left(n \cdot n \cdot n \cdot n \cdot \cdots n\right)$$

$$\log(n!) = O(n \log n)$$

$$\log(n!) \leq \log(n^n) = n \log n$$

$$\log(n!) = \Omega(?)$$

$$\log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots \cdots 3 \cdot 2 \cdot 1)$$

$$= \log(\underbrace{n \cdot 1} \cdot \underbrace{(n-1) \cdot 2} \cdot \underbrace{(n-2) \cdot 3} \cdot \underbrace{(n-3) \cdot 4} \cdots \cdots \underbrace{(n-\tfrac{n}{2}) \cdot (n-\tfrac{n}{2})})$$

$$\geq \log(n \cdot n \cdot n \cdot n \cdot \cdots n)$$

$$= \log(n^{n/2})$$ (assume n: even otherwise $\lfloor \frac{n}{2} \rfloor$)

$$\log(n!) = O(n\log n)$$

$$\log(n!) \leqslant \log(n^n) = n\log n$$

$$\log(n!) = \Omega(n\log n)$$

$$\log(n!) = \log(n\cdot(n-1)\cdot(n-2)\cdot(n-3)\cdots\cdots 3\cdot 2\cdot 1)$$

$$= \log(\underbrace{n\cdot 1}\cdot\underbrace{(n-1)\cdot 2}\cdot\underbrace{(n-2)\cdot 3}\cdot\underbrace{(n-3)\cdot 4}\cdots\cdots\underbrace{(n-\tfrac{n}{2})\cdot(n-\tfrac{n}{2})})$$

$$\geqslant \log(n\cdot n\cdot n\cdot n\cdot\ldots n)$$

$$= \log(n^{n/2})$$ (assume n: even
otherwise $\lfloor \frac{n}{2} \rfloor$)

$$\Rightarrow \log(n!) \geqslant \frac{n}{2}\log n$$

$\log(n!) = O(n\log n)$

$\log(n!) \leq \log(n^n) = n\log n$

$\log(n!) = \Omega(n\log n)$

$$\log(n!) = \log(n\cdot(n-1)\cdot(n-2)\cdot(n-3)\cdots\cdots 3\cdot 2\cdot 1)$$

$$= \log(\underbrace{n\cdot 1}\cdot\underbrace{(n-1)\cdot 2}\cdot\underbrace{(n-2)\cdot 3}\cdot\underbrace{(n-3)\cdot 4}\cdots\cdots\sim\underbrace{(n-\tfrac{n}{2})\cdot(n-\tfrac{n}{2})})$$

$$\geq \log(n\cdot n\cdot n\cdot n\cdot\ldots n)$$

$$= \log(n^{n/2})$$ (assume n: even otherwise $\lfloor\frac{n}{2}\rfloor$) $\Rightarrow \log(n!) \geq \frac{n}{2}\log n$

---

so $\frac{1}{2}n\log n \leq \log(n!) \leq n\log n$