Algorithms represented as Decision Trees

Internal nodes represent comparison of two elements

\[ \text{a : b} \]

\[ \text{c : b} \]

\[ \text{c : d} \]
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: $i > j$

Example: sort $a_1, a_2, a_3$

Each leaf is a possible output
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 $i,j$
Brances represent outcome of comparison
\( 6 \text{ Left: } i < j \)
\( 6 \text{ Right: } i > j \)

Example: sort $a_1, a_2, a_3$

Verify on 9, 4, 6

$\text{a}_3 < \text{a}_2 < \text{a}_1$

Each leaf is a possible output
Algorithms represented as Decision Trees

Internal nodes represent comparison of two elements \( i,j \)
Branches represent outcome of comparison
\( \leq \) Left: \( i < j \)
\( > \) Right: \( i > j \)

Example: sort \( a_1, a_2, a_3 \)

Each leaf is a possible output
Each root-\( \rightarrow \)leaf path represents an execution of algo.

Verify on 9, 4, 6
\( a_1, a_2, a_3 \)
Algorithms represented as Decision Trees

Internal nodes represent comparison of two elements \( i,j \)
Brances represent outcome of comparison
\( \leq \text{Left: } i < j, \text{Right: } i > j \)

Example: sort \( a_1, a_2, a_3 \)

Each leaf is a possible output
Each root->leaf path represents an execution of algo.

Verify on 9, 4, 6
\( a_1, a_2, a_3 \)

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 precisely 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 = $(n)$ for sorting for a good 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. Therefore, the max path length is $n$. 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.

\[ \Rightarrow \text{so max path length} = (n) \]

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

\[ \Rightarrow \text{It's huge and repetitive.} \]

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.

\[ \implies \text{so max path length} = (n) \]

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

\[ \implies \text{It's huge and repetitive.} \]

It really lists every possible execution of algo.

\[ \implies \text{You actually might need a different tree for each n.} \]
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. 

\[ \text{so max path length} = \binom{n}{2} \]

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

\[ \text{It's huge and repetitive.} \]
\[ \text{It really lists every possible execution of algo.} \]
\[ \text{You actually might need a different tree for each n.} \]

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.

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

#leaves?
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.

\[ \text{#leaves} \geq n! \]

height of tree = worst case time = h  \[ \Rightarrow \text{#leaves} \leq ? \]
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 = \( h \) \( \Rightarrow \) \( \#\text{leaves} \leq 2^h \)

[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.

\[ \#\text{leaves} \geq n! \]

Height of tree = worst case time = \( h \)  \( \Rightarrow \) \#leaves \( \leq 2^h \)

[binary tree; every node has 2 children]

so, \( n! \leq \#\text{leaves} \leq 2^h \)
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 = \( h \) \( \Rightarrow \#\text{leaves} \leq 2^h \) [binary tree; every node has 2 children]

so, \( n! \leq \#\text{leaves} \leq 2^h \) \( \Rightarrow \log n! \leq \log 2^h \)
A correct decision tree for sorting must have every permutation of the input represented at a leaf node.

\[ \text{\#leaves} \geq n! \]

\[ \text{height of tree} = \text{worst case time} = h \implies \text{\#leaves} \leq 2^h \]

[binary tree; every node has 2 children]

so, \[ n! \leq \text{\#leaves} \leq 2^h \] \[ \implies \log n! \leq \log 2^h \] \[ \implies h \geq \log n! \]
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 = \( h \) \( \rightarrow \) \( \#\text{leaves} \leq 2^h \)

[Binary tree; every node has 2 children]

So, \( n! \leq \#\text{leaves} \leq 2^h \) \( \Rightarrow \) \( \log n! \leq \log 2^h \) \( \Rightarrow \) \( h \geq \log n! \)

Stirling's formula: \( n! \geq (\frac{n}{e})^n \) \( h \geq \log (\frac{n}{e})^n \)
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 = \( h \) \( \Rightarrow \#\text{leaves} \leq 2^h \)
[binary tree; every node has 2 children]

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

Stirling's formula: \( n! \approx (\frac{n}{e})^n \sqrt{2\pi n} \)

\[ h \geq \log(\frac{n}{e})^n = n \cdot \log \frac{n}{e} \]
A correct decision tree for sorting must have every permutation of the input represented at a leaf node. 

\[ \#\text{leaves} \geq n! \]

Height of tree = worst case time = \( h \) \( \Rightarrow \) \( \#\text{leaves} \leq 2^h \)

[Binary tree; every node has 2 children]

So, \( n! \leq \#\text{leaves} \leq 2^h \) \( \Rightarrow \) \( \log n! \leq \log 2^h \) \( \Rightarrow \) \( h \geq \log n! \)

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

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

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

Extra analysis of \( \log n! \) follows
\log(n!) = O(\mathbb{?})
\log(n!) = O(?)

\log(n!) \leq \log(n^n)
\log(n!) = O(?)

\log(n!) \leq \log(n^n) = n \log n
\[
\log(n!) = O(n \log n)
\]
\[
\log(n!) \leq \log(n^n) = n \log n
\]
\[
\log(n!) = \Omega(?)
\]
\[ \log(n!) = O(n \log n) \]

\[ \log(n!) \leq \log(n^n) = n \log n \]

\[ \log(n!) = \Omega(?) \]

\[ \log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots 3 \cdot 2 \cdot 1) \]
\[
\log(n!) = O(n \log n)
\]
\[
\log(n!) \leq \log(n^n) = n \log n
\]

\[
\log(n!) = \Omega(?)
\]

\[
\log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \ldots \ldots 3 \cdot 2 \cdot 1)
\]

\[
= \log(n \cdot 1 \cdot (n-1) \cdot 2 \cdot (n-2) \cdot 3 \cdot (n-3) \cdot 4 \ldots \ldots \cdot n \cdot \frac{n}{2} \cdot \frac{n}{2})
\]

\[
\Rightarrow \text{exactly if } n: \text{even}
\]
\[ \log(n!) = \Theta(n \log n) \]

\[ \log(n!) \leq \log(n^n) = n \log n \]

\[ \log(n!) = \Omega(?) \]

\[ \log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \ldots \cdot 3 \cdot 2 \cdot 1) \]

\[ = \log \left( n \cdot 1 \cdot (n-1) \cdot 2 \cdot (n-2) \cdot 3 \cdot (n-3) \cdot 4 \cdot \ldots \cdot n \cdot \left( n - \frac{n}{2} \right) \cdot \left( n - \frac{n}{2} \right) \right) \]

\[ \geq \log \left( n \cdot n \cdot n \cdot n \cdot \ldots \cdot n \right) \]
\log(n!) = O(n \log n)
\log(n!) \leq \log(n^n) = n \log n

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

\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 n \cdot \frac{n}{2} \cdot \frac{n}{2})
\geq \log(n \cdot n \cdot n \cdot n \cdot n \cdot n \cdots n)
= \log(n^{n/2}) \quad \text{(assume } n \text{ even)}
\quad \text{otherwise } \frac{n}{2^k}
\[ \log(n!) = O(n \log n) \]

\[ \log(n!) \leq \log(n^n) = n \log n \]

\[ \log(n!) = \Omega(n \log n) \]

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

\[ = \log(n \cdot 1 \cdot (n-1) \cdot 2 \cdot (n-2) \cdot 3 \cdot (n-3) \cdot 4 \cdots \cdots \cdot n \cdot \frac{n}{2} \cdot \frac{n}{2}) \]

\[ \geq \log(n \cdot n \cdot n \cdot n \cdot \cdots \cdots \cdot n) \]

\[ = \log\left(n^{\frac{n}{2}}\right) \quad \text{(assume } n: \text{even)} \quad \Rightarrow \quad \log(n!) \geq \frac{n}{2} \log n \]

\[ \text{otherwise } \frac{n}{2} \]
\[ \log(n!) = O(n \log n) \]
\[ \log(n!) \leq \log(n^n) = n \log n \]
\[ \log(n!) = \Omega(n \log n) \]

\[ \log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots 3 \cdot 2 \cdot 1) \]
\[ = \log(n \cdot 1 \cdot (n-1) \cdot 2 \cdot (n-2) \cdot 3 \cdot (n-3) \cdot 4 \cdots n \cdot \frac{n}{2} \cdot \frac{n}{2}) \]
\[ \geq \log(n \cdot n \cdot n \cdot n \cdot n \cdot n \cdot \cdots n) \]
\[ = \log(n^{n/2}) \quad \text{(assume } n \text{: even)} \Rightarrow \log(n!) \geq \frac{n}{2} \log n \]

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