Decision Trees, Sorting Lower-Bound

Ira A. Fulton Schools of Engineering
Arizona State University

Comparison-based sorting algorithms

- Rearrange elements by element-wise comparisons
- Insertion Sort: A[j] > A[i]?
- Quicksort: A[j] ≤ A[r]?
  - Within partition, Key has the same value as A[r]
- Mergesort: A[i] ≤ A[j]?
  - Within merge
- All of them are comparison-based sorting

- Non-comparison-based sorting: radix sort

Decision Tree

- A decision tree is a binary tree that represents the comparisons between elements that are performed by a particular sorting algorithm, operating on an input of a given size.
- We annotate each internal node by the comparison made at the corresponding step of the algorithm.
- We annotate each leaf node by a permutation $<\pi(1), \pi(2), \ldots, \pi(n)>$.

Decision Tree

- A decision tree corresponds to a sorting algorithm for a given length of the input sequence.

- Therefore, we have a decision tree for insertion sort on 3 elements. We have a decision tree for quicksort on 10 elements, etc.

- We do not have a decision tree for insertion sort.
- We do not have a decision tree for quicksort on the input sequence 3, 5, 8.

Decision Tree

- **Each path from the root node to a leaf node represents the execution of the algorithm on a particular input.**

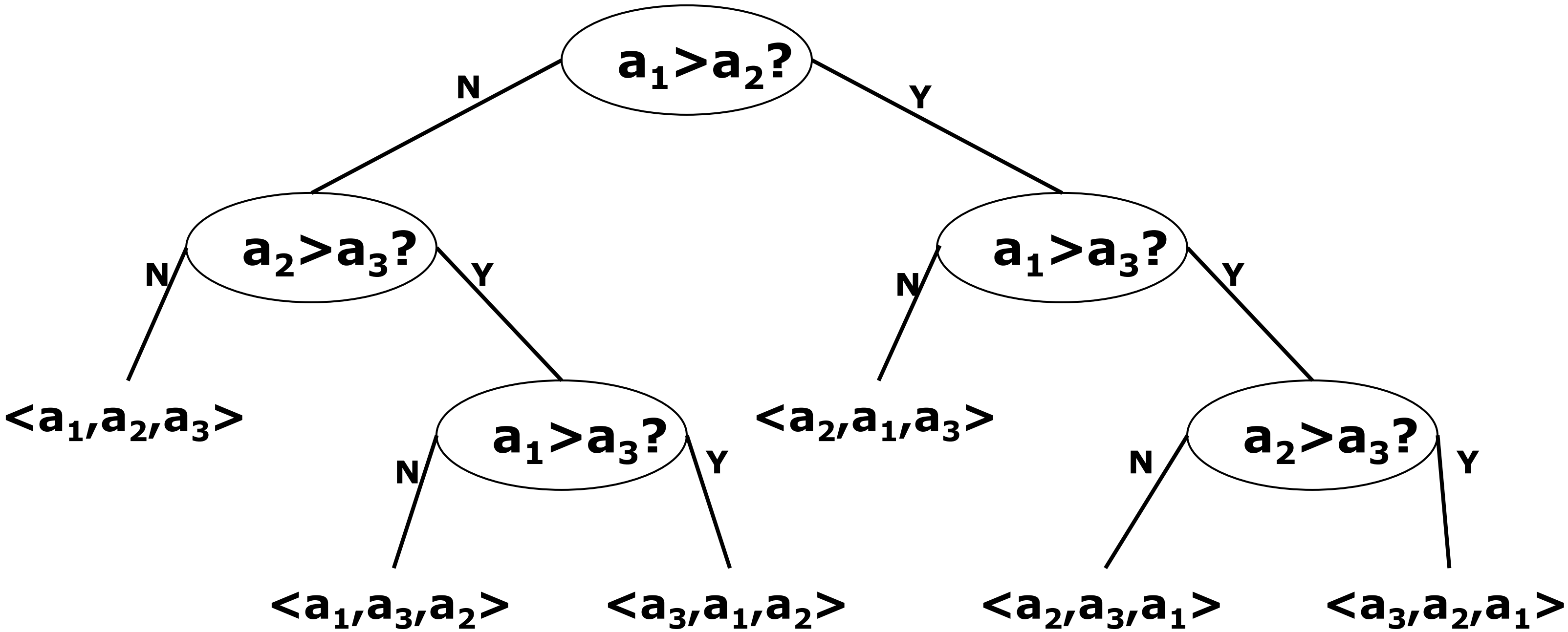
- **Since there are $n!$ possible results for sorting an n-elements array A[1], A[2], …, A[n], there are $n!$ leaf nodes in a decision tree for a sorting algorithm on n elements.**

Insertion Sort

```
Insertion-Sort(A, n)
for i := 2 to n do
    key := A[i]
    // insert A[i] into sorted A[1..i-1]
    j := i -1
    while j > 0 and A[j] > key do
        A[j+1] := A[j]
        j := j - 1
    A[j+1] := key
```

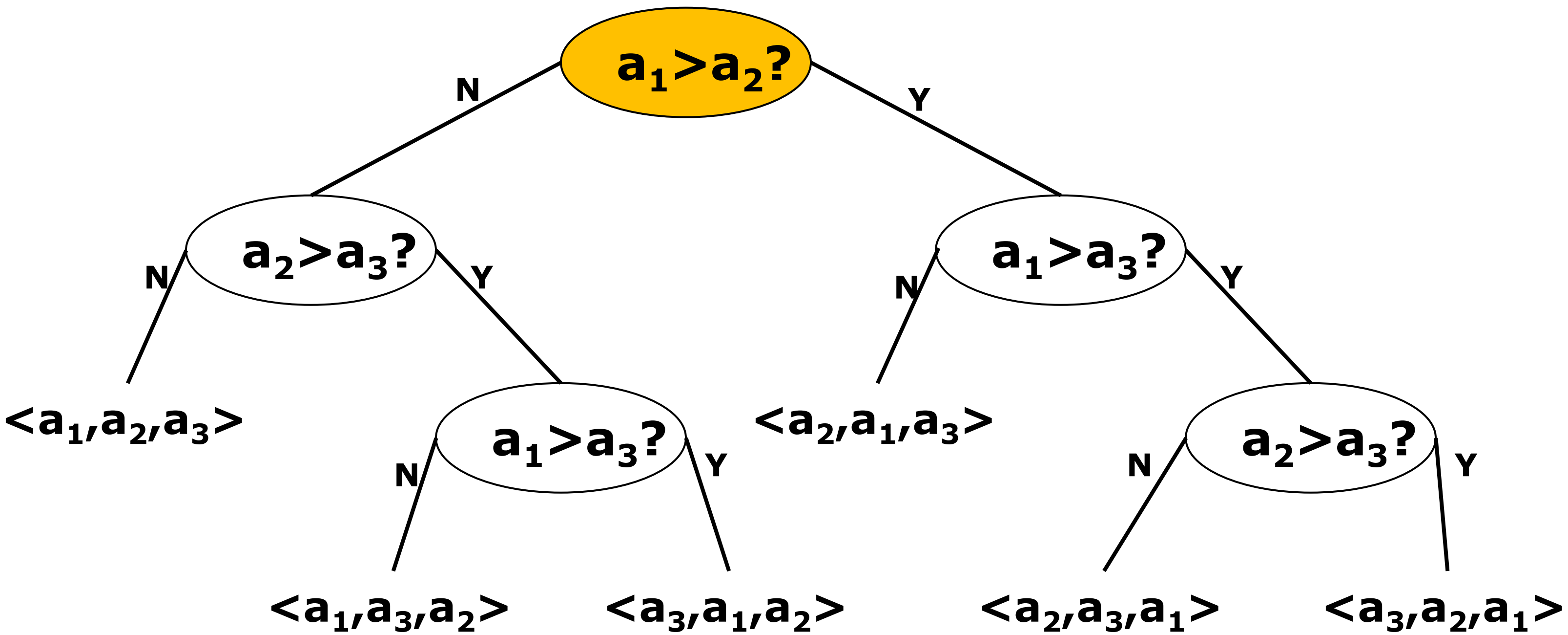
Trace the algorithm using inputs: 3 5 2 8 3

DT for Insertion Sort on <a_1, a_2, a_3>



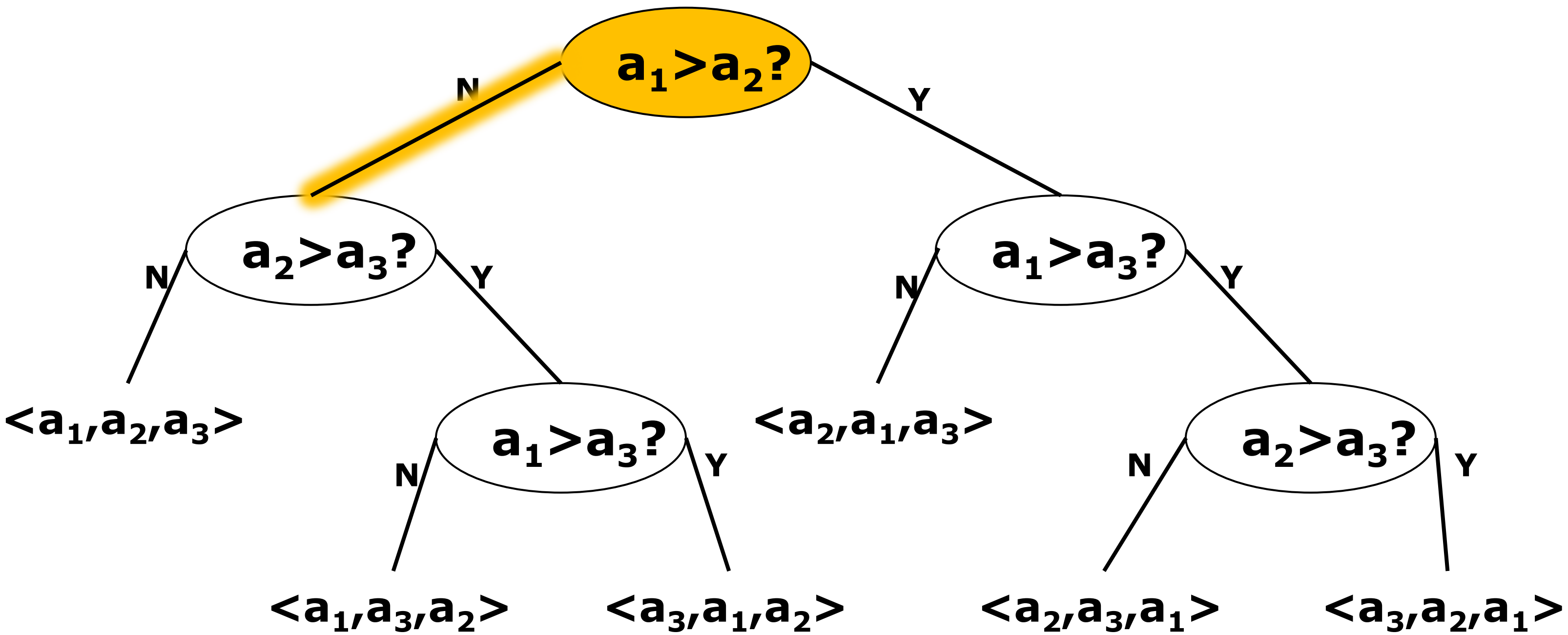
- Array content is a_1 a_2 a_3. Insert a_2.

DT for Insertion Sort on $<a_1, a_2, a_3>$



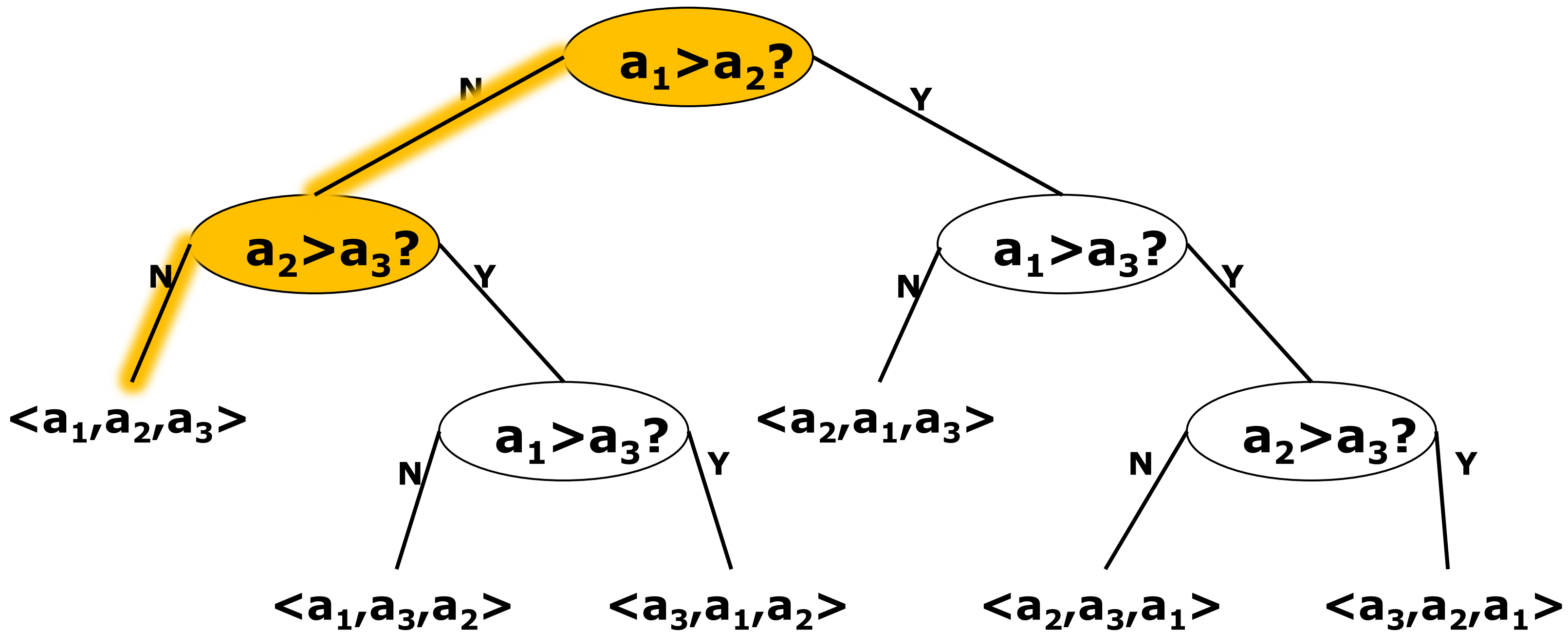
- Array content is a_1 a_2 a_3. Insert a_2.
- Is $a_1>a_2$?

DT for Insertion Sort on $<a_1, a_2, a_3>$



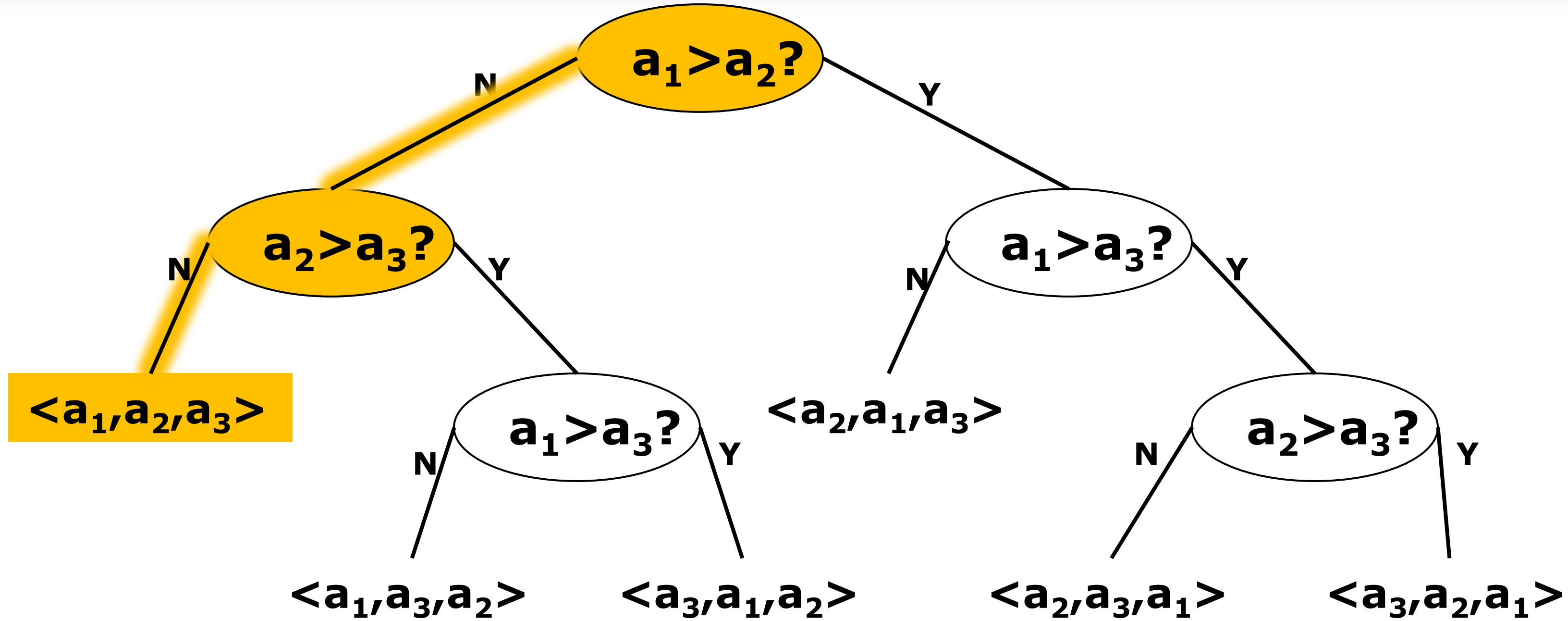
- Array content is a_1 a_2 a_3. Insert a_2.
- Is $a_1>a_2$? No
- Array content is a_1 a_2 a_3. Insert a_3.

DT for Insertion Sort on <a_1, a_2, a_3>



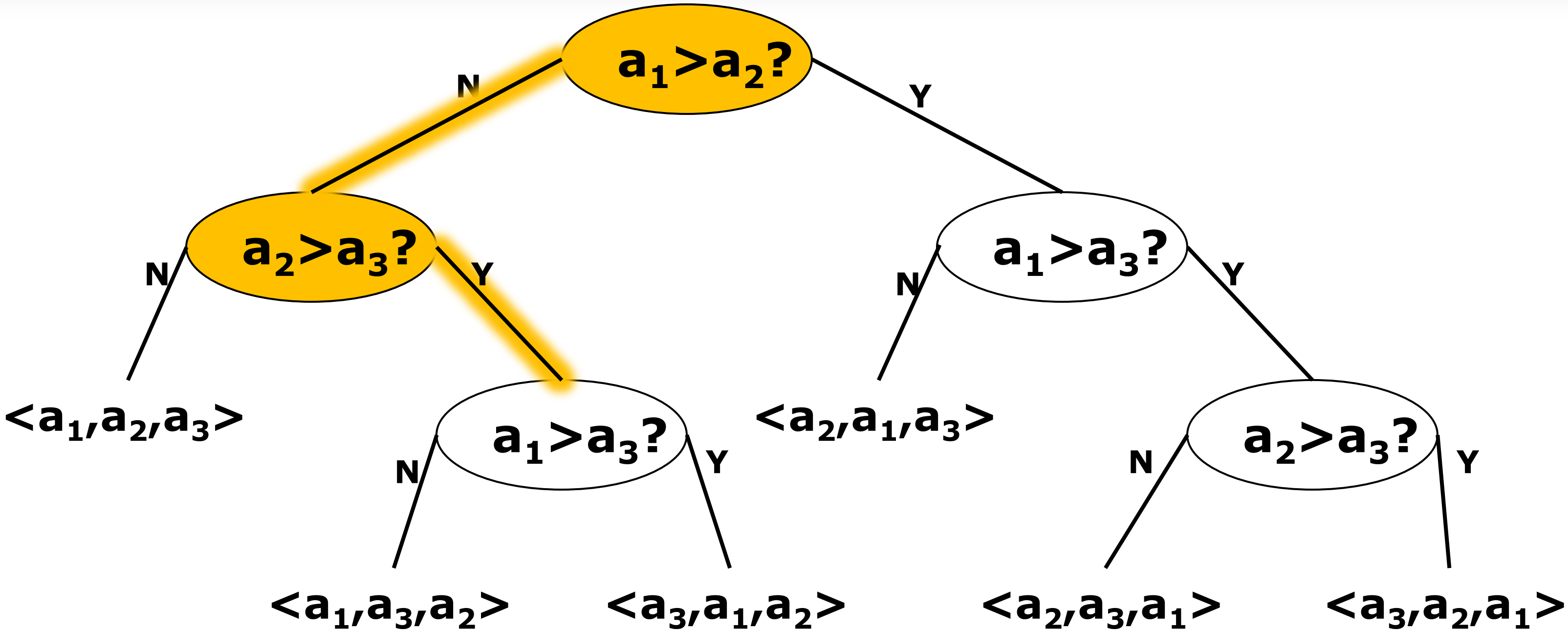
- Array content is a_1 a_2 a_3. Insert a_2.
- Is $a_1>a_2$? No
- Array content is a_1 a_2 a_3. Insert a_3.
- Is $a_2>a_3$? No

DT for Insertion Sort on <a_1, a_2, a_3>



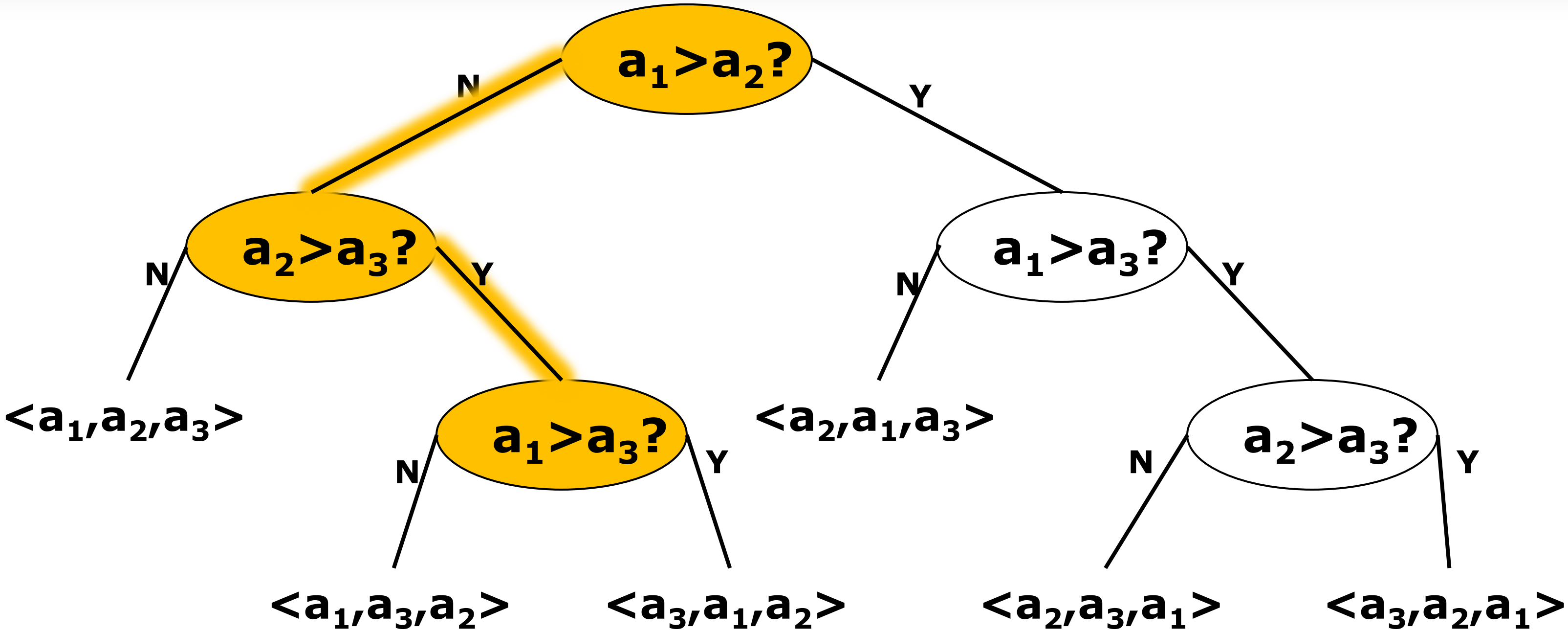
- Array content is a_1 a_2 a_3. Insert a_2.
- Is $a_1>a_2$? No
- Array content is a_1 a_2 a_3. Insert a_3.
- Is $a_2>a_3$? No
- STOP with a_1 a_2 a_3.

DT for Insertion Sort on <a_1, a_2, a_3>



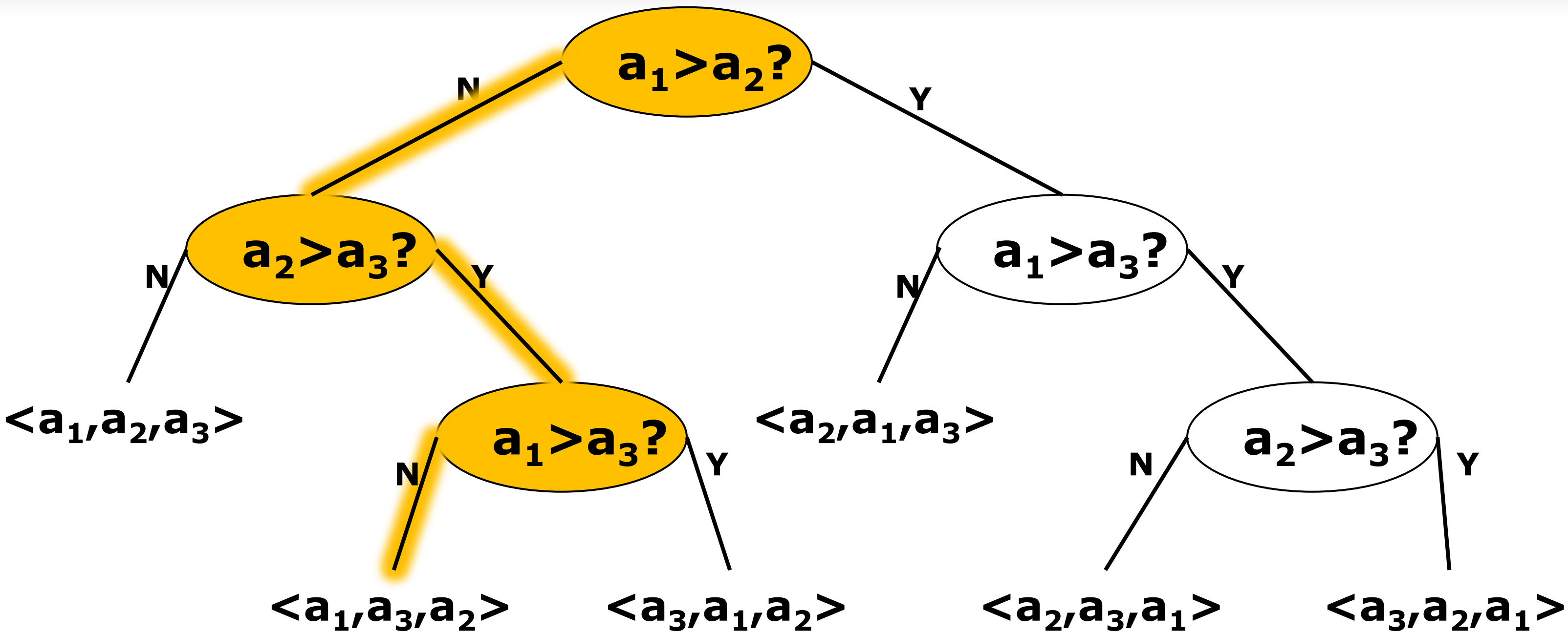
- Array content is a_1 a_2 a_3. Insert a_2. Is $a_1>a_2$? No
- Array content is a_1 a_2 a_3. Insert a_3. Is $a_2>a_3$? Yes

DT for Insertion Sort on $\langle a_1, a_2, a_3\rangle$



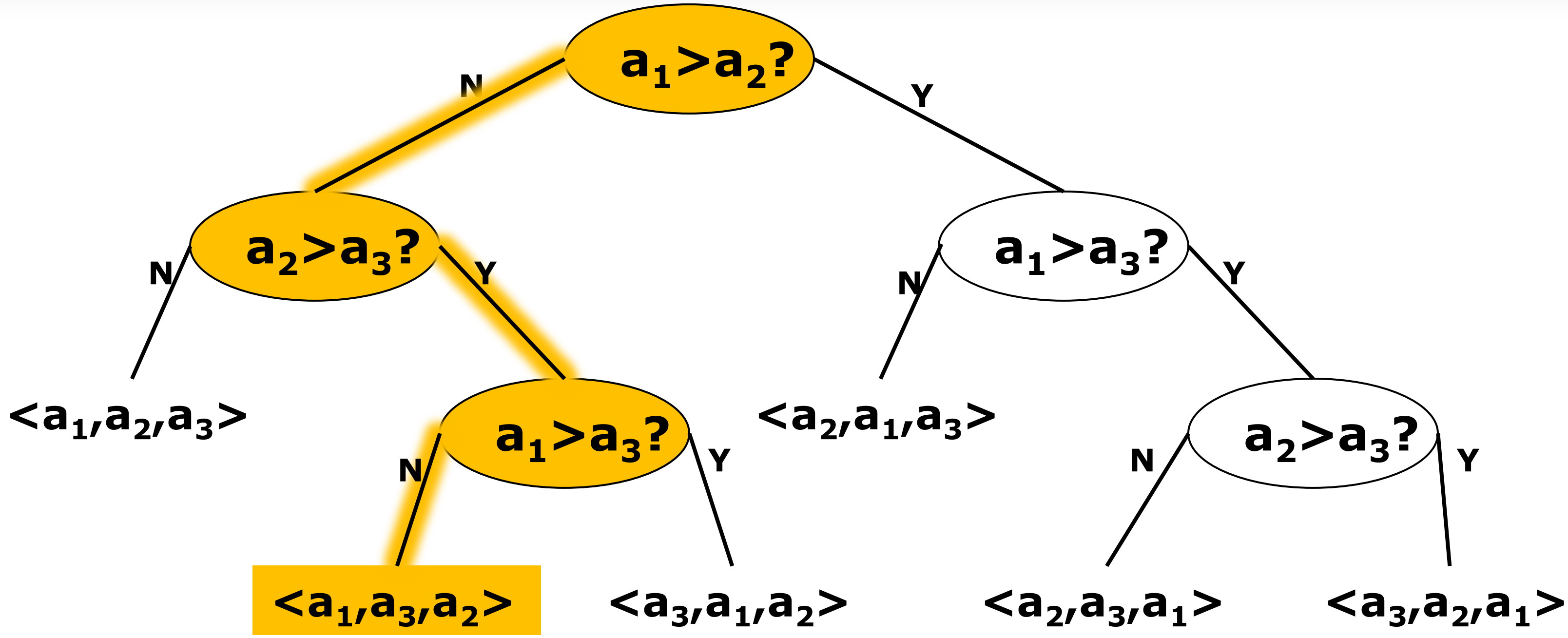
- Array content is a_1 a_2 a_3. Insert a_2. Is $a_1>a_2$? No
- Array content is a_1 a_2 a_3. Insert a_3. Is $a_2>a_3$? Yes
- Array content is a_1 a_2 a_2. Is $a_1>a_3$?

DT for Insertion Sort on <a_1, a_2, a_3>



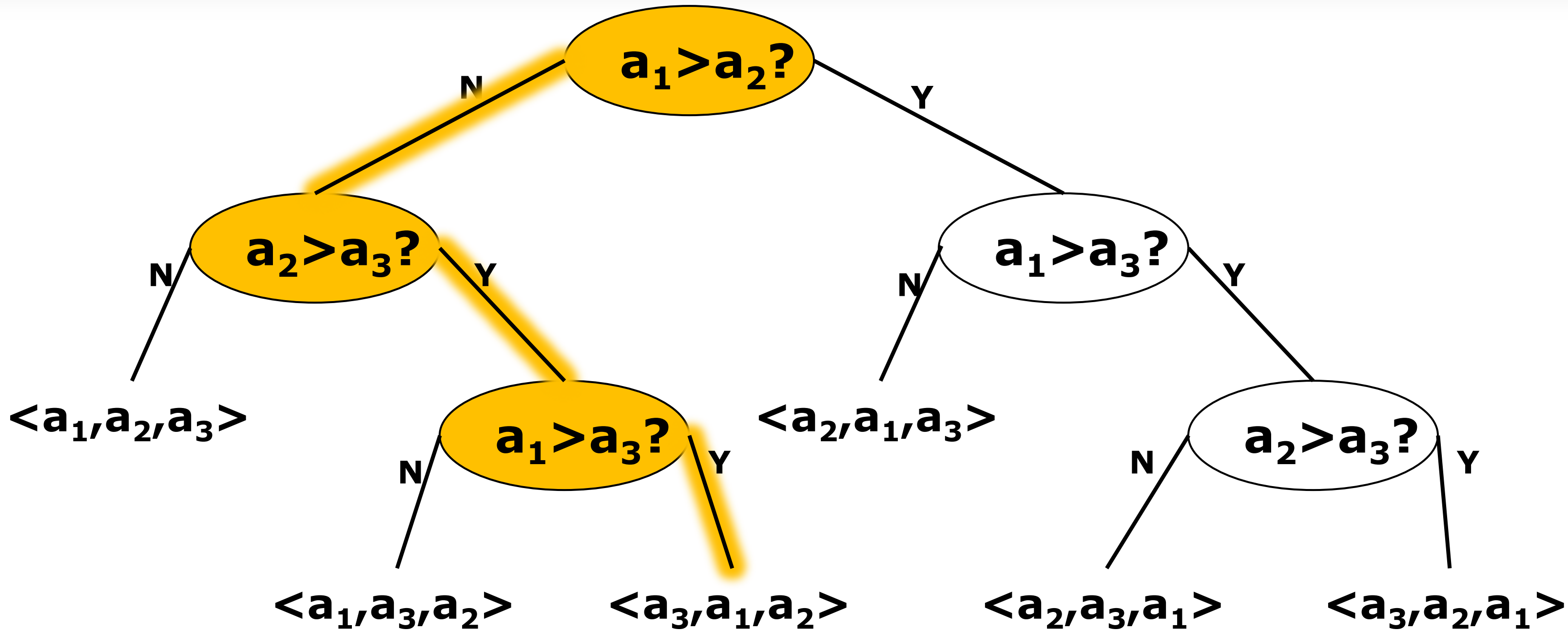
- Array content is a_1 a_2 a_3. Insert a_2. Is $a_1>a_2$? No
- Array content is a_1 a_2 a_3. Insert a_3. Is $a_2>a_3$? Yes
- Array content is a_1 a_2 a_2. Is $a_1>a_3$? No

DT for Insertion Sort on $<a_1, a_2, a_3>$



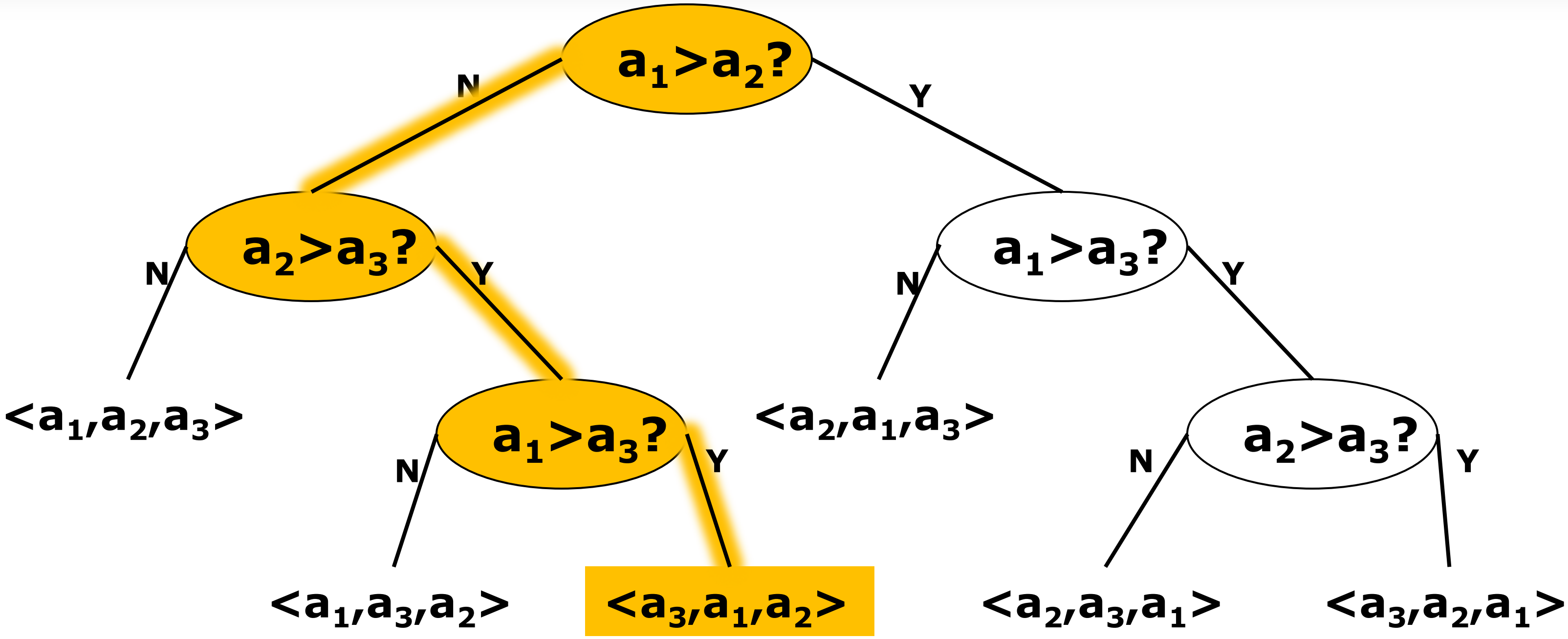
- Array content is $a_1\ a_2\ a_3$. Insert a_2. Is $a_1>a_2$? No
- Array content is $a_1\ a_2\ a_3$. Insert a_3. Is $a_2>a_3$? Yes
- Array content is $a_1\ a_2\ a_2$. Is $a_1>a_3$? No
- STOP with $a_1\ a_3\ a_2$.

DT for Insertion Sort on $<a_1, a_2, a_3>$



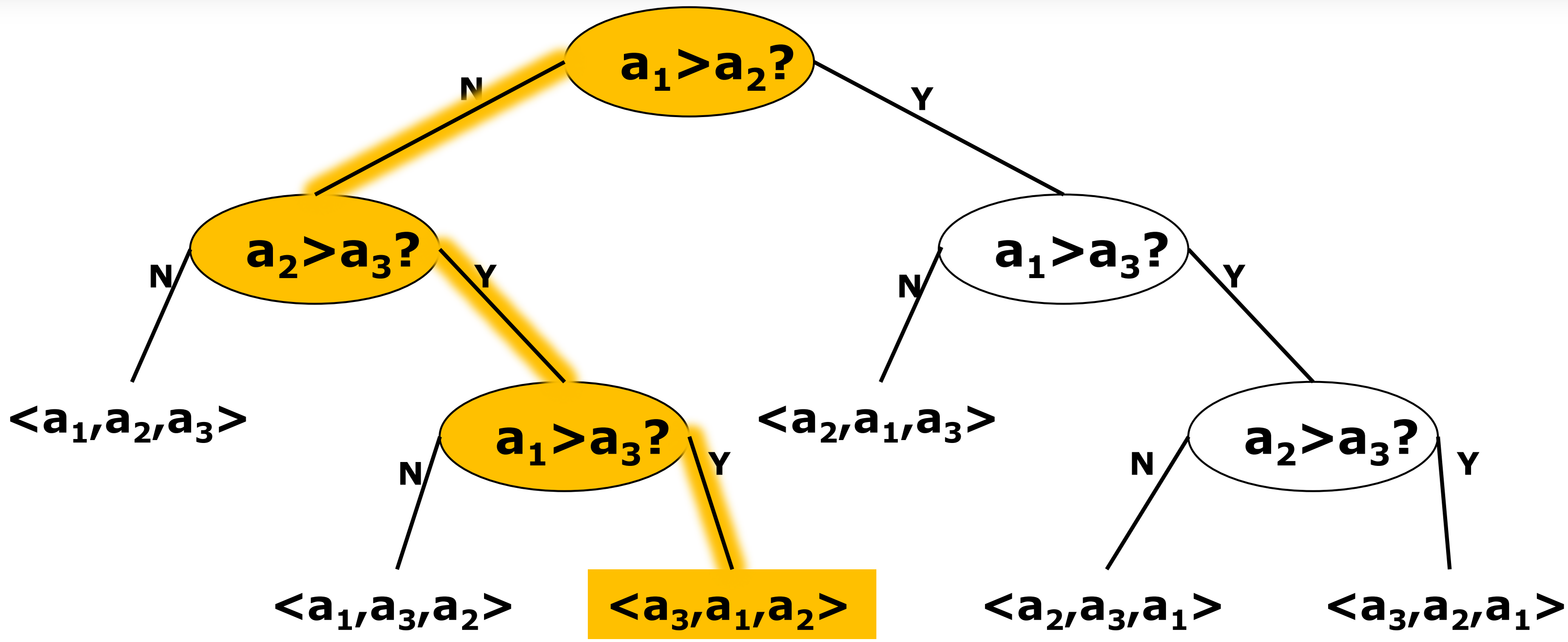
- Array content is $a_1\ a_2\ a_3$. Insert a_2. Is $a_1>a_2$? No
- Array content is $a_1\ a_2\ a_3$. Insert a_3. Is $a_2>a_3$? Yes
- Array content is $a_1\ a_2\ a_2$. Is $a_1>a_3$? Yes
- Array content is $a_1\ a_1\ a_2$.

DT for Insertion Sort on $\langle a_1, a_2, a_3 \rangle$



- Array content is $a_1\ a_2\ a_3$. Insert a_2. Is $a_1>a_2$? No
- Array content is $a_1\ a_2\ a_3$. Insert a_3. Is $a_2>a_3$? Yes
- Array content is $a_1\ a_2\ a_2$. Is $a_1>a_3$? Yes
- Array content is $a_1\ a_1\ a_2$. Overwrite A[1] with a_3.
- STOP with $a_3\ a_1\ a_2$.

DT for Insertion Sort on <a_1, a_2, a_3>



- Continue this process, we can draw the complete decision tree for Insertion sort on three elements.

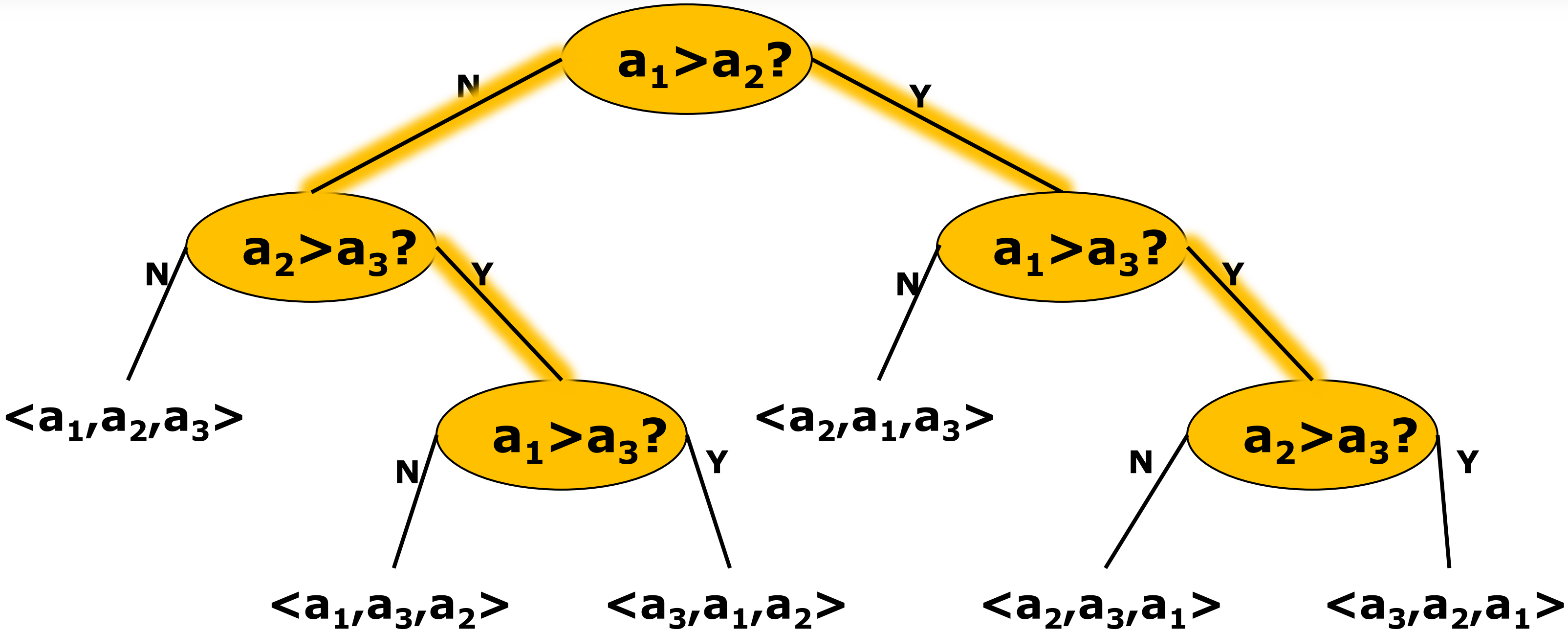
DT for Insertion Sort on <a_1, a_2, a_3>

- **There are 3!=6 leaf nodes in the decision tree for Insertion Sort on three elements.**
- **Each root-to-leaf path corresponds to the execution of the algorithm for the corresponding input.**
- **Example: If the input is 18, 10, 15, the sorted order is a_2, a_3, a_1. Therefore, the execution of the algorithm corresponds to the path from the root node to the leaf node <a_2, a_3, a_1>.**

Ira A. Fulton Schools of Engineering

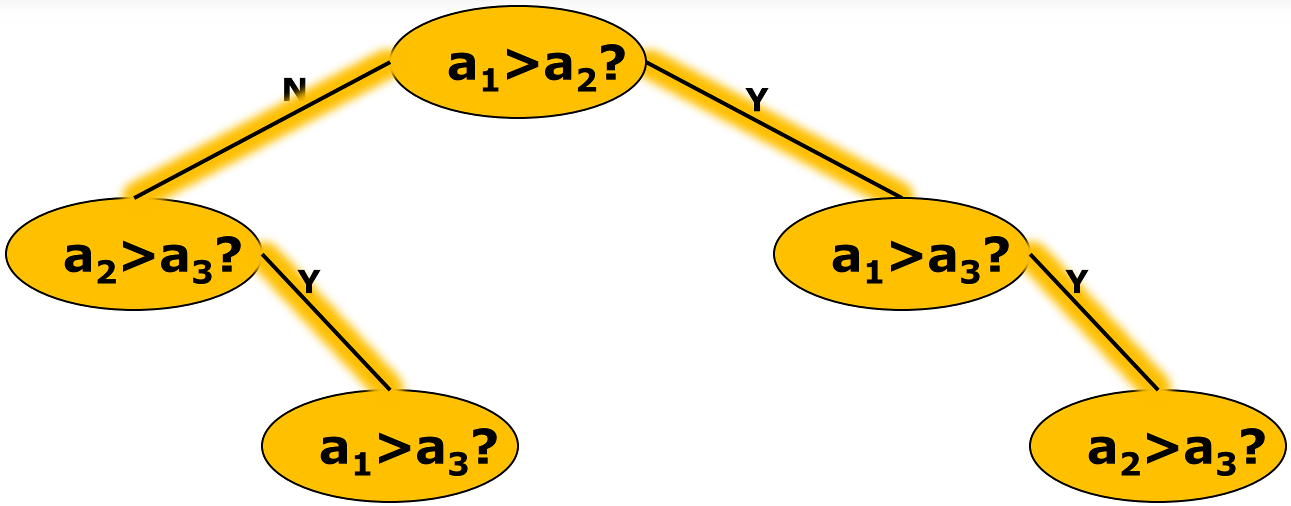
Arizona State University

DT for Insertion Sort on <a_1, a_2, a_3, a_4>

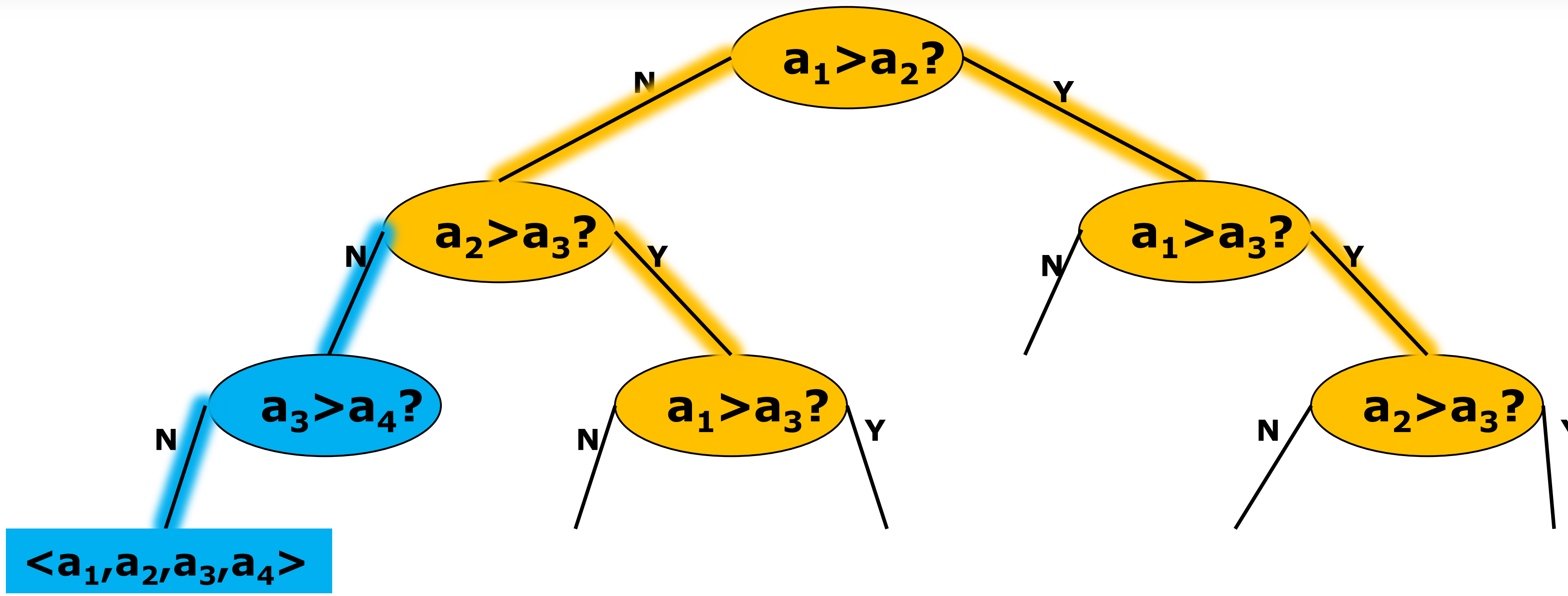


- For Insertion sort, we can grow the DT for 3 elements to the DT for 4 elements.
- The internal nodes will stay. The leaf nodes need to be changed.
- This growing process does not work for Mergesort or Quicksort.

DT for Insertion Sort on $<a_1, a_2, a_3, a_4>$

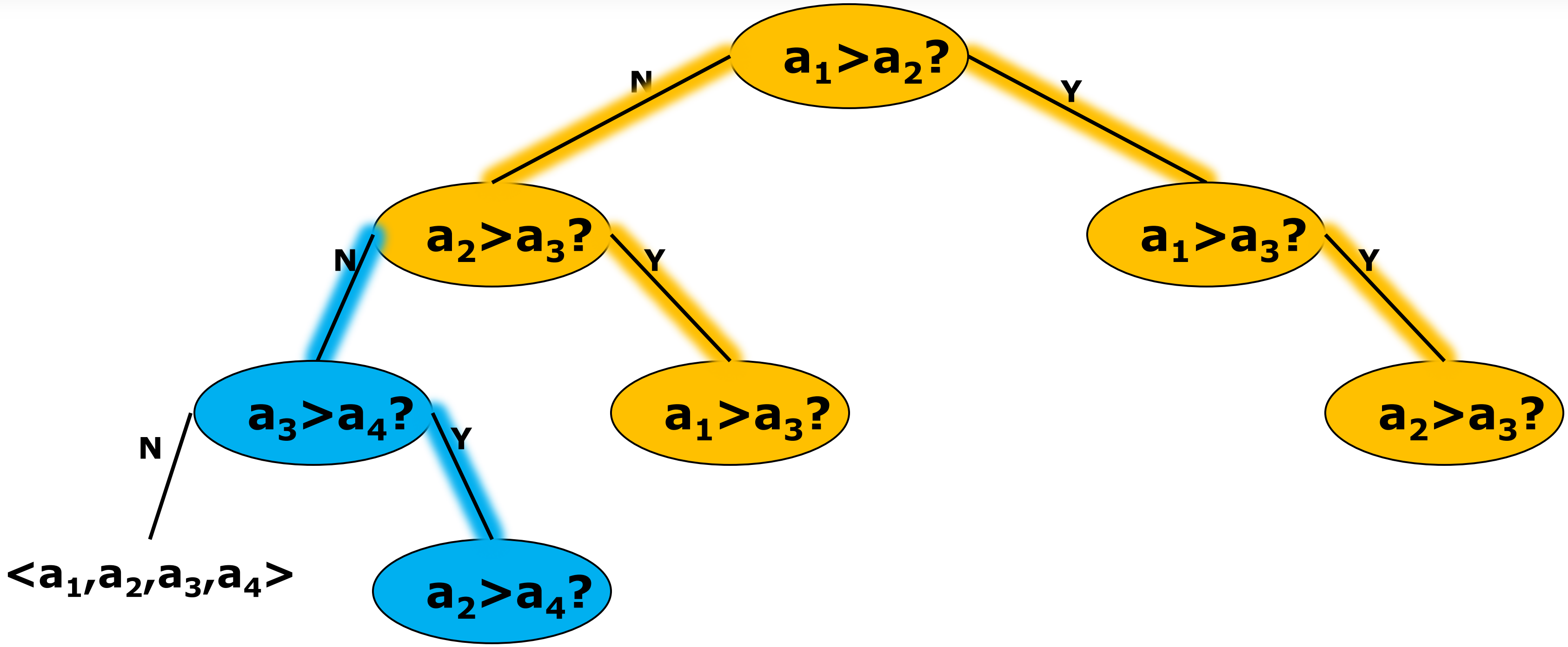


DT for Insertion Sort on <a_1, a_2, a_3, a_4>



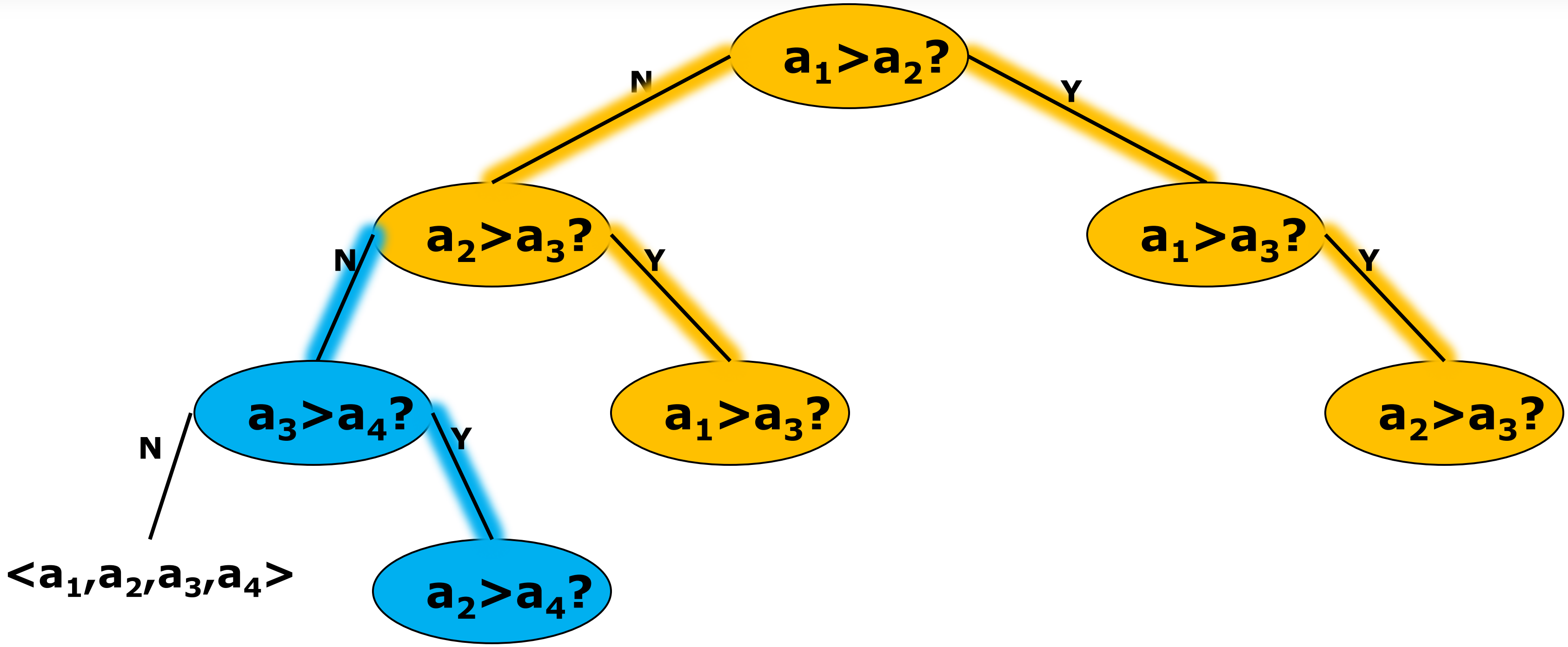
- Array content is a_1 a_2 a_3 a_4. Insert a_2. Is $a_1>a_2$? No
- Array content is a_1 a_2 a_3 a_4. Insert a_3. Is $a_2>a_3$? No
- Array content is a_1 a_2 a_3 a_4. Insert a_4. Is $a_3>a_4$? No
- STOP with a_1 a_2 a_3 a_4.

DT for Insertion Sort on <a_1, a_2, a_3, a_4>

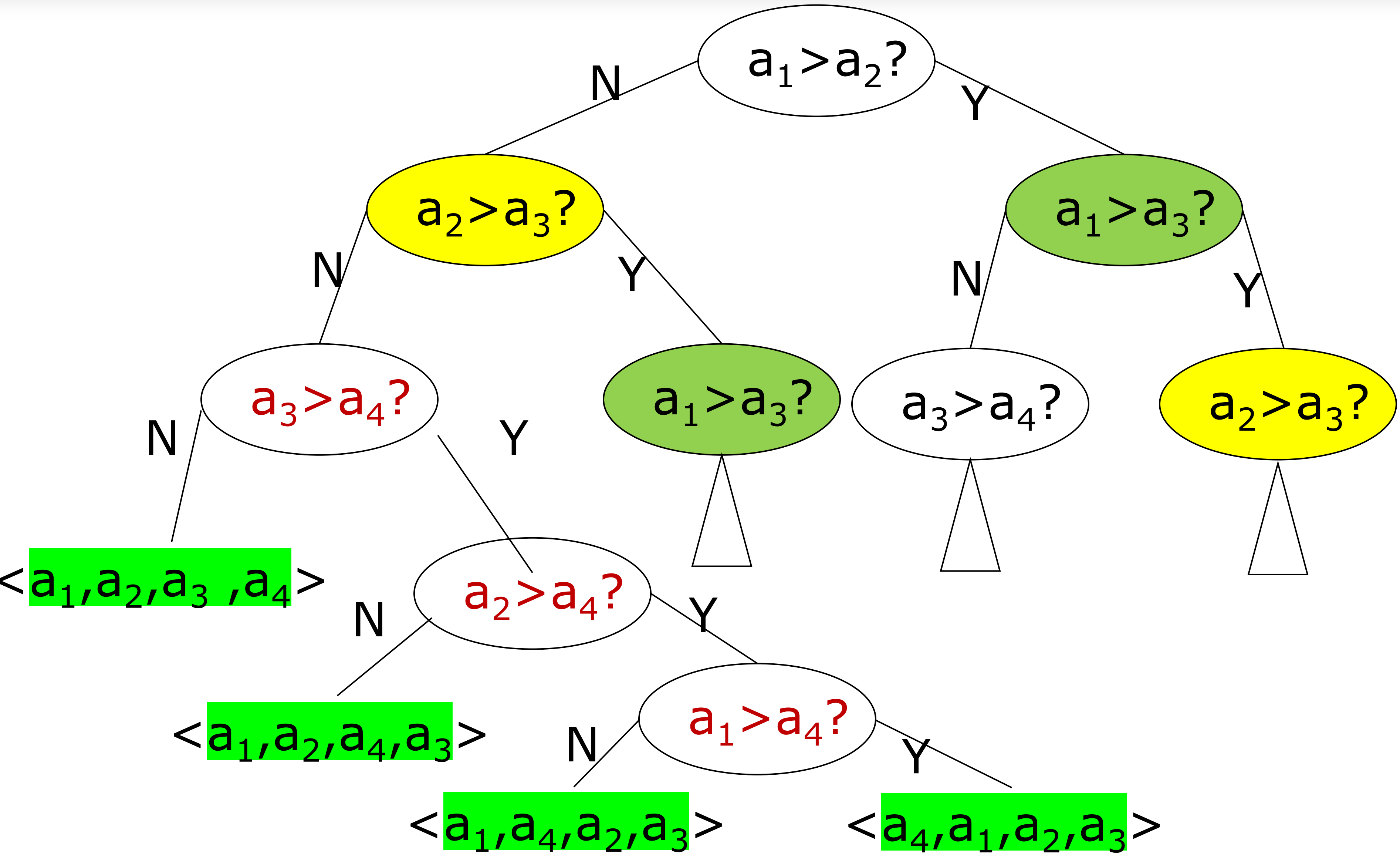


- Array content is $a_1\ a_2\ a_3\ a_4$. Insert a_2. Is $a_1>a_2$? No
- Array content is $a_1\ a_2\ a_3\ a_4$. Insert a_3. Is $a_2>a_3$? No
- Array content is $a_1\ a_2\ a_3\ a_4$. Insert a_4. Is $a_3>a_4$? Yes
- Array content is $a_1\ a_2\ a_3\ a_3$. Is $a_2>a_4$?
- The process would continue until we have the complete DT for Insertion sort on 4 elements

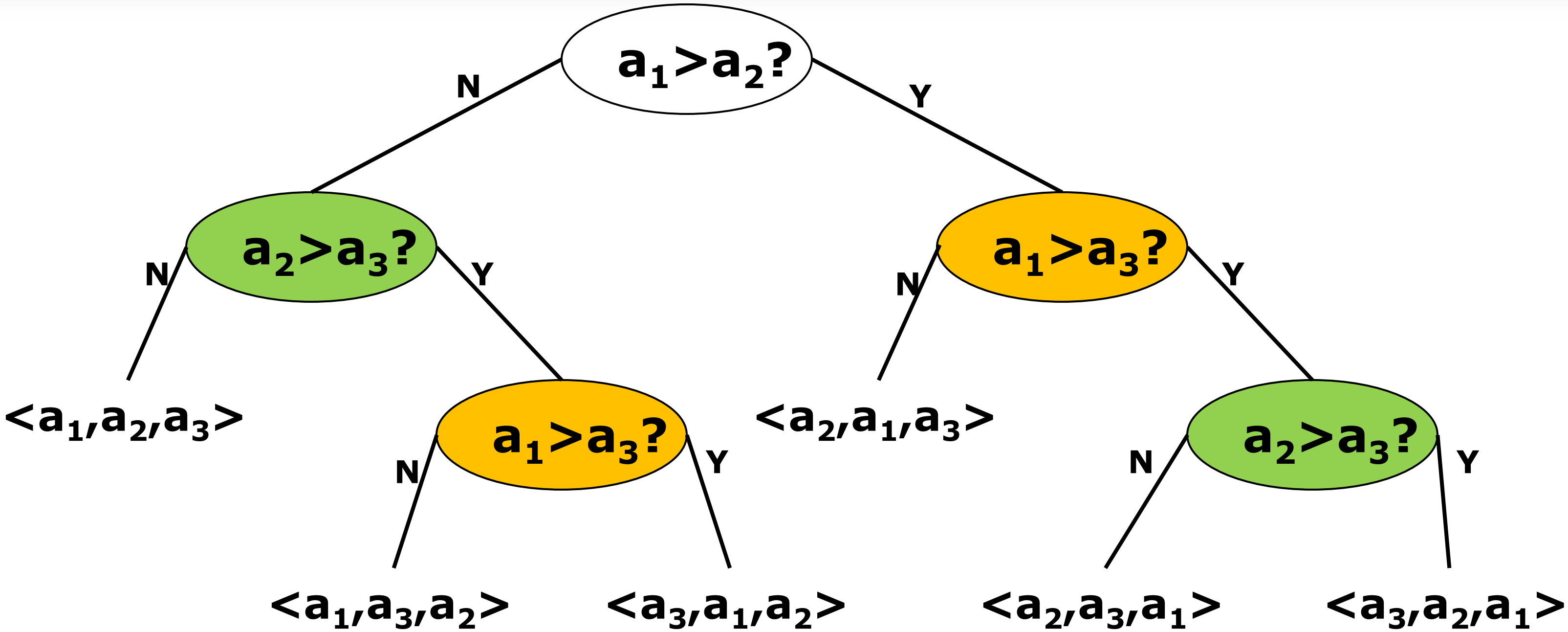
DT for Insertion Sort on <a_1, a_2, a_3, a_4>



There are n!-1 internal nodes



There are n!-1 internal nodes



- There are $n!$ internal nodes.
- There are $O(n^2)$ possible comparisons.
- The same comparison may appear in multiple internal nodes.

DT for Insertion Sort on <a_1, a_2, a_3, a_4>

- **There are 4!=24 leaf nodes in the decision tree for Insertion Sort on four elements.**
- **Each root-to-leaf path corresponds to the execution of the algorithm for the corresponding input.**
- **Example: If the input is <2, 4, 6, 1>, the sorted order is a_4, a_1, a_2, a_3. Therefore, the execution of the algorithm corresponds to the path from the root node to the leaf node <a_4, a_1, a_2, a_3>.**

Ira A. Fulton Schools of Engineering

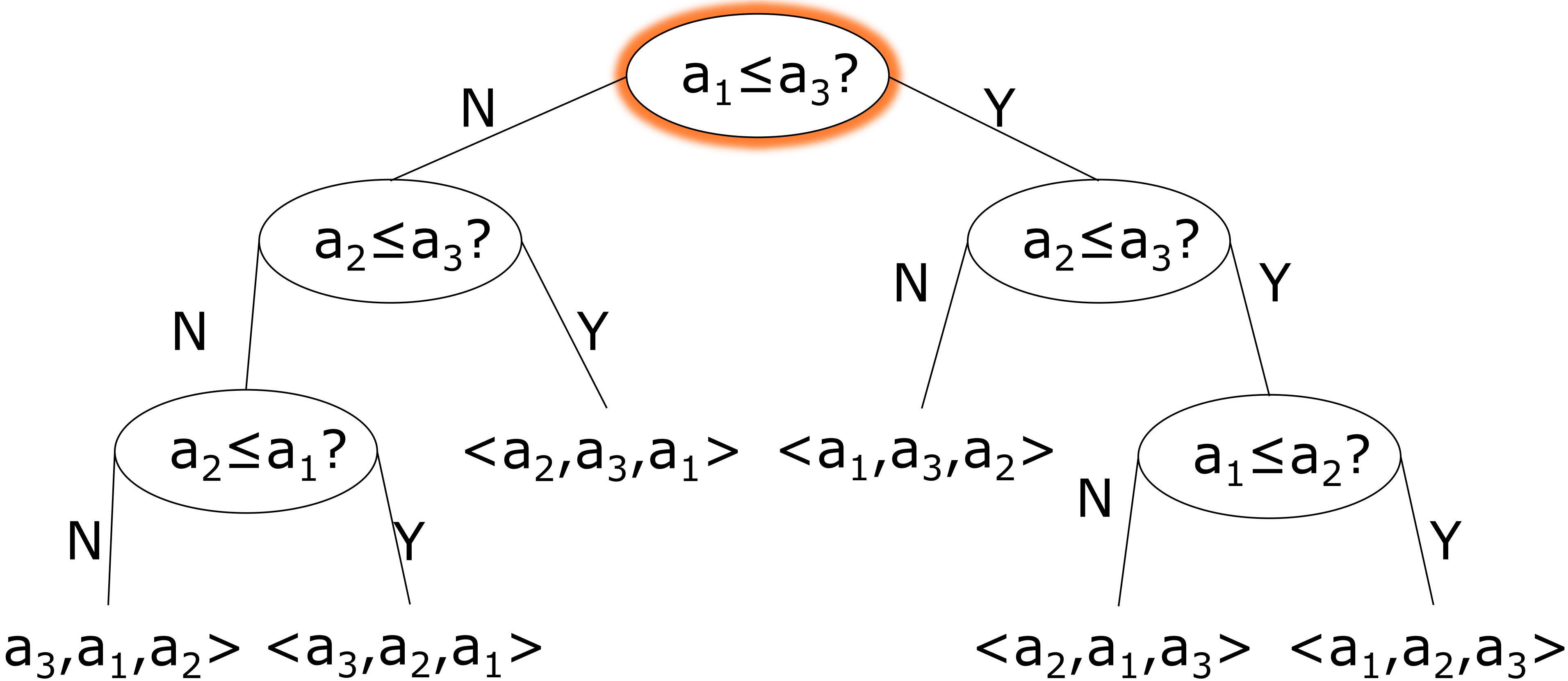
Arizona State University

The Quicksort Algorithm

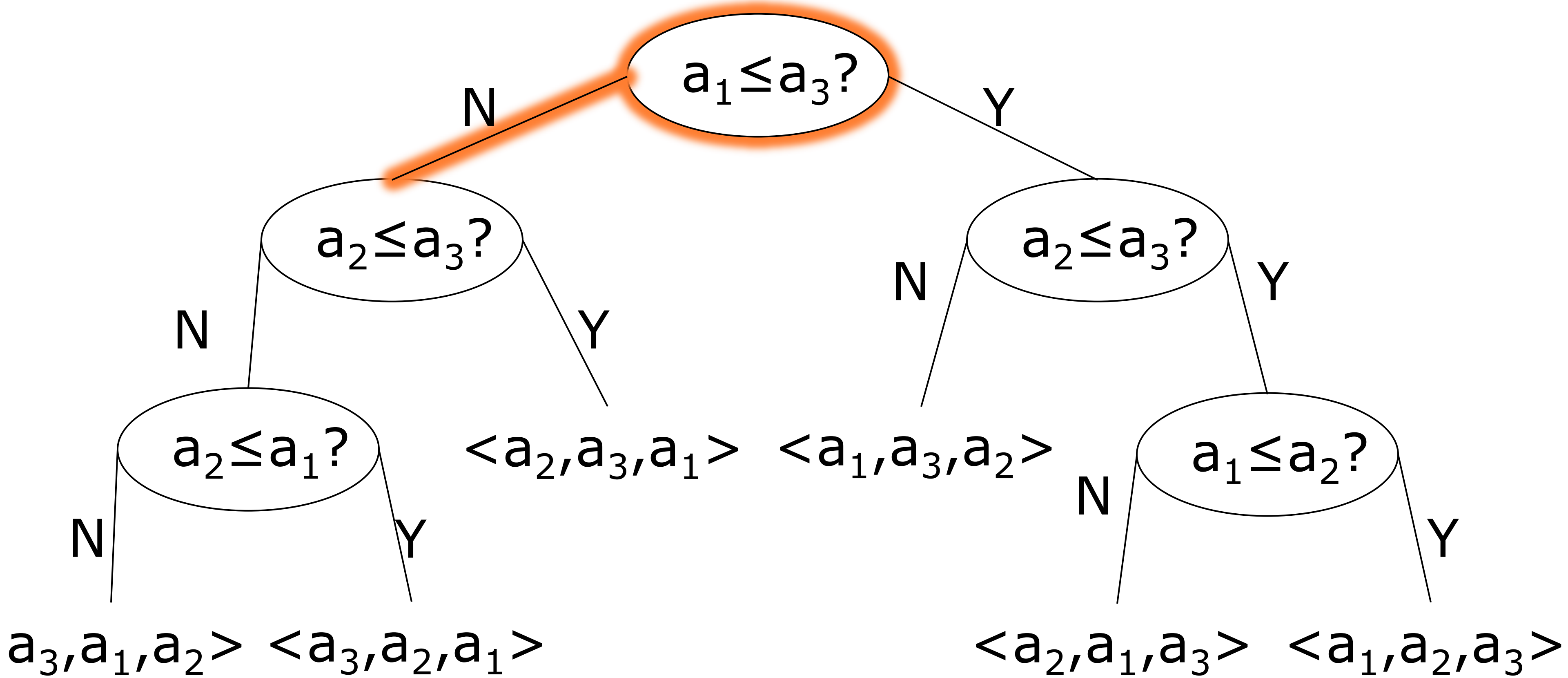
```
QuickSort(A, p, r)
if p < r
   q = Partition(A, p, r)
    QuickSort(A, p, q-1)
    QuickSort(A, q+1, r)
```

```
Partition(A, p, r)
x := A[r]
i := p-1
for j:=p to r-1 do
    If A[j] ≤ x then
        i := i+1
        temp = A[i]
        A[i] = A[j]
        A[j] = temp
temp = A[i+1]
A[i+1] = A[r]
A[r] = temp
return i+1
```

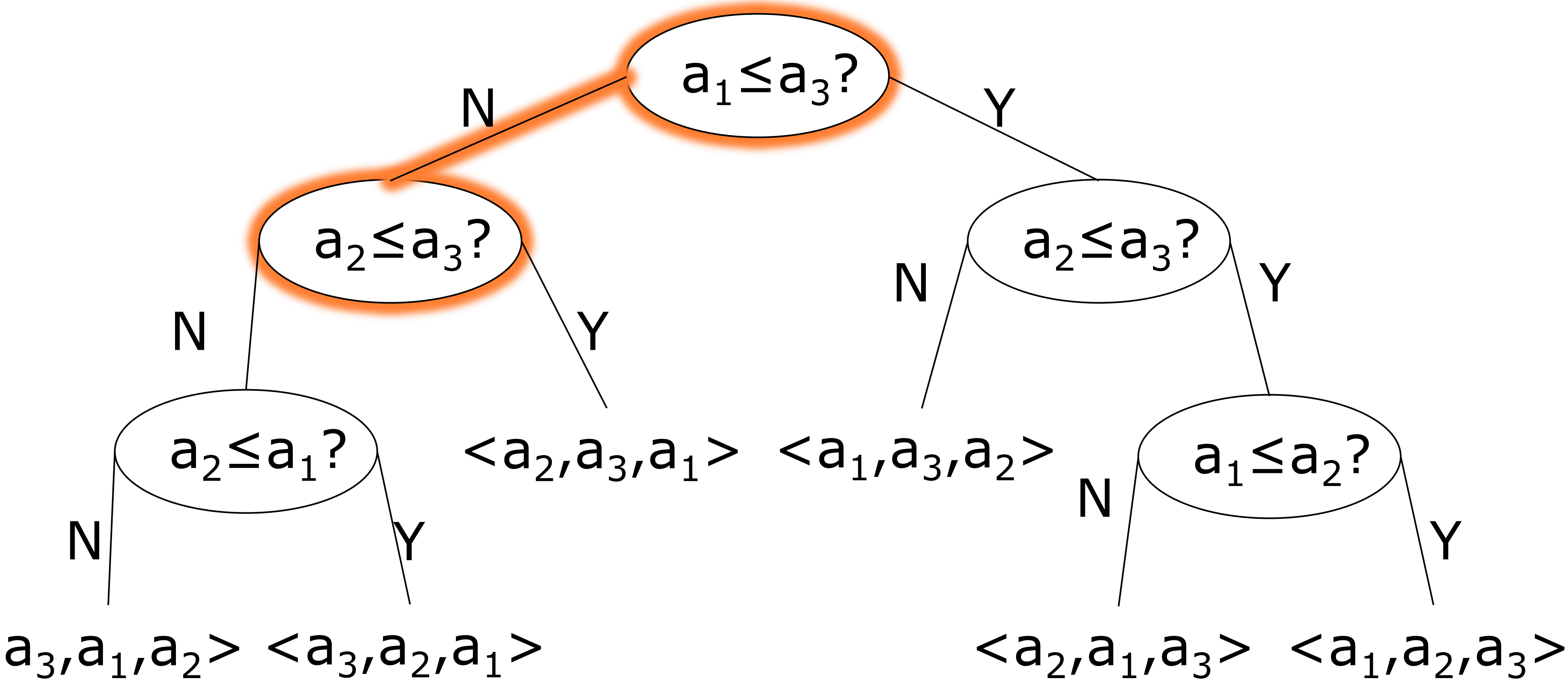
DT for Quicksort on <a_1, a_2, a_3>



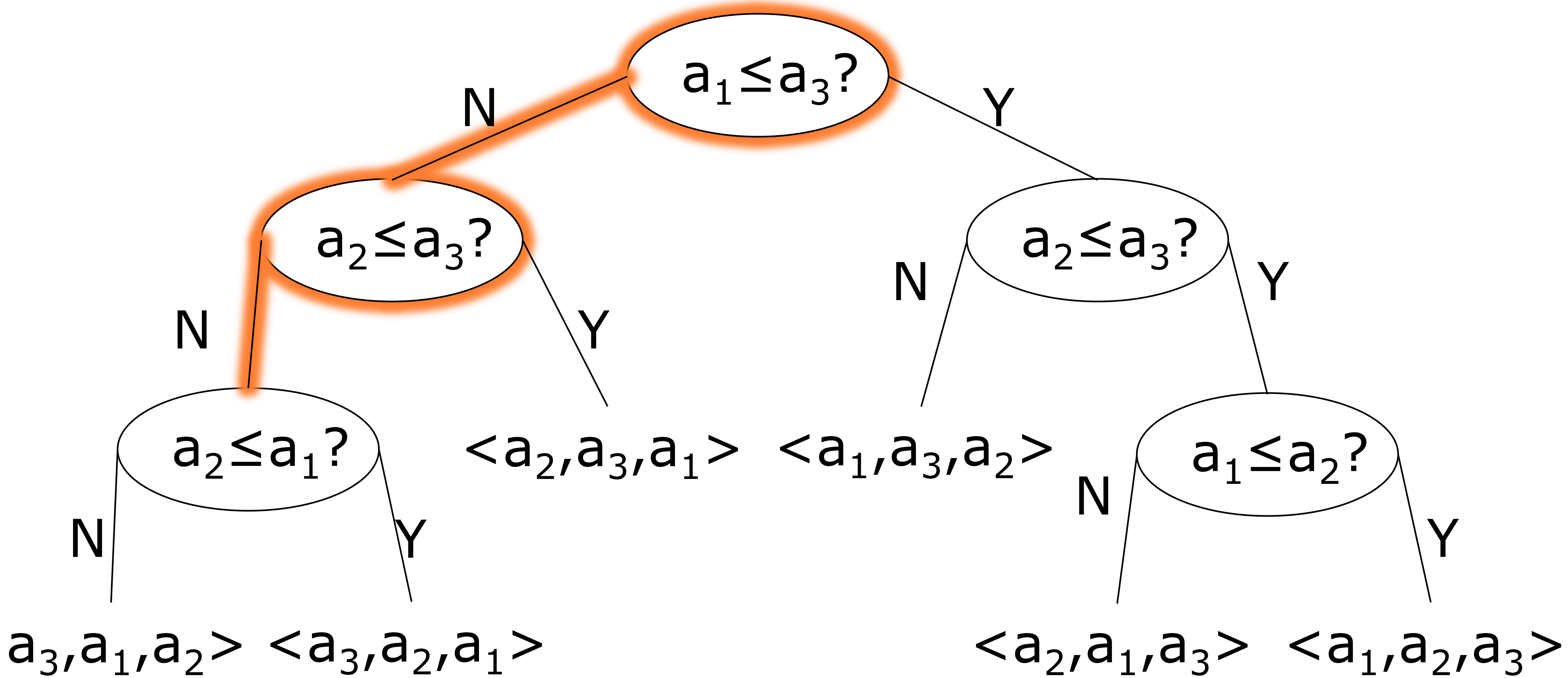
DT for Quicksort on <a_1, a_2, a_3>



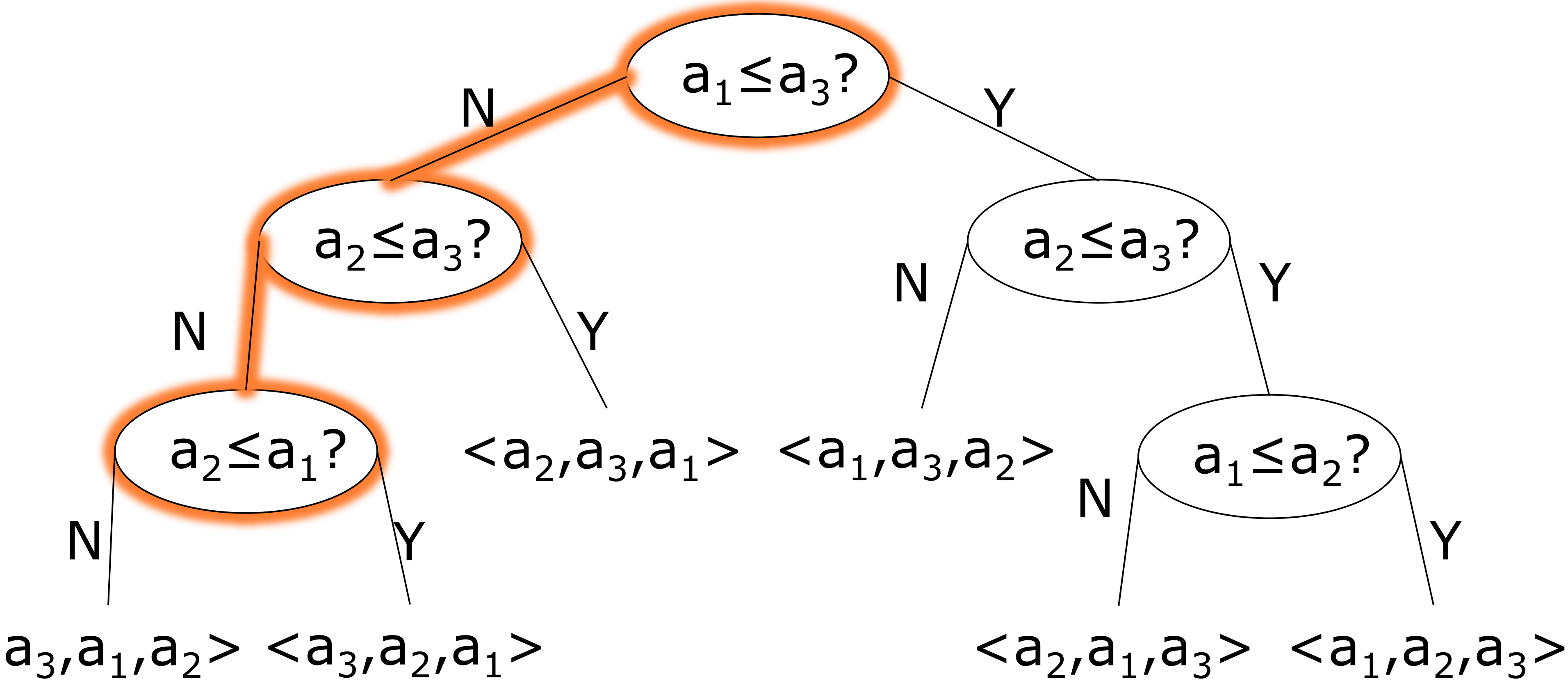
DT for Quicksort on <a_1, a_2, a_3>



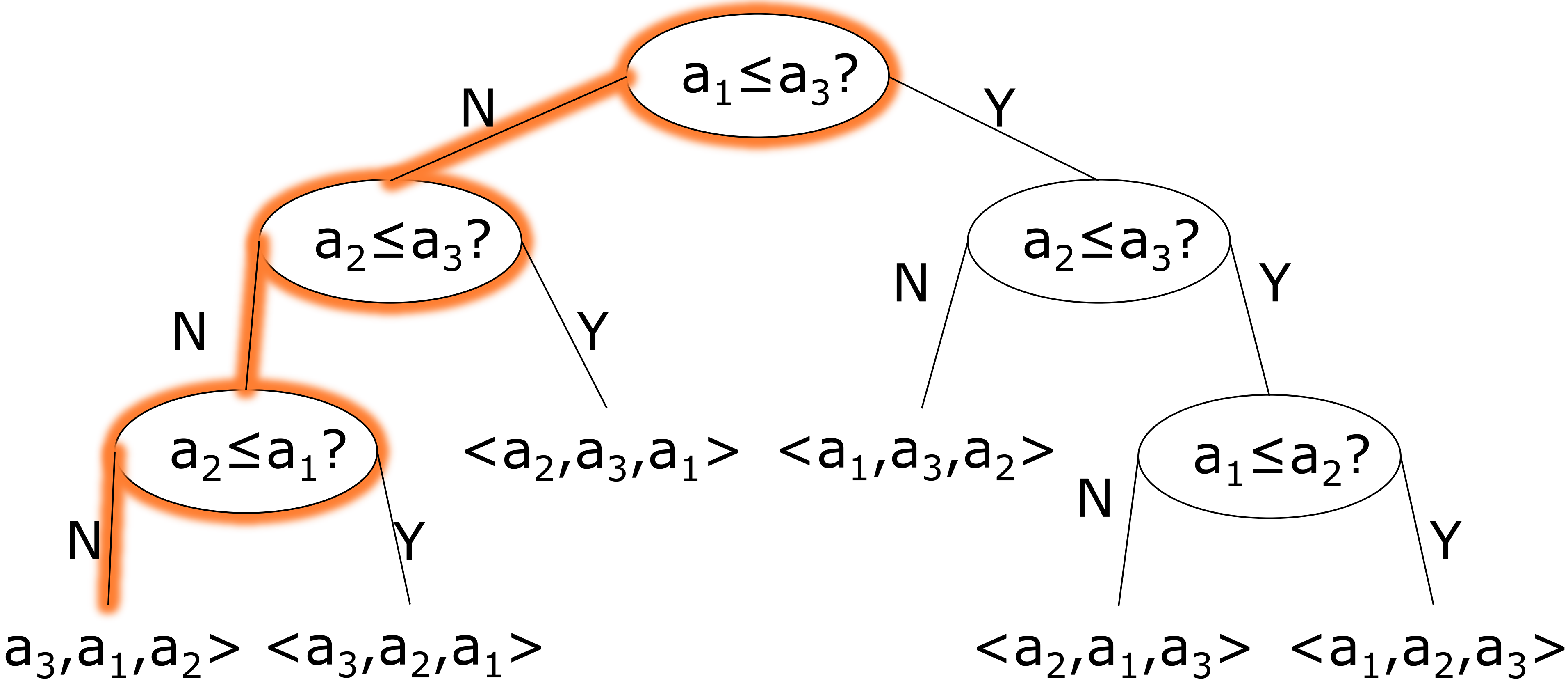
DT for Quicksort on $<a_1, a_2, a_3>$



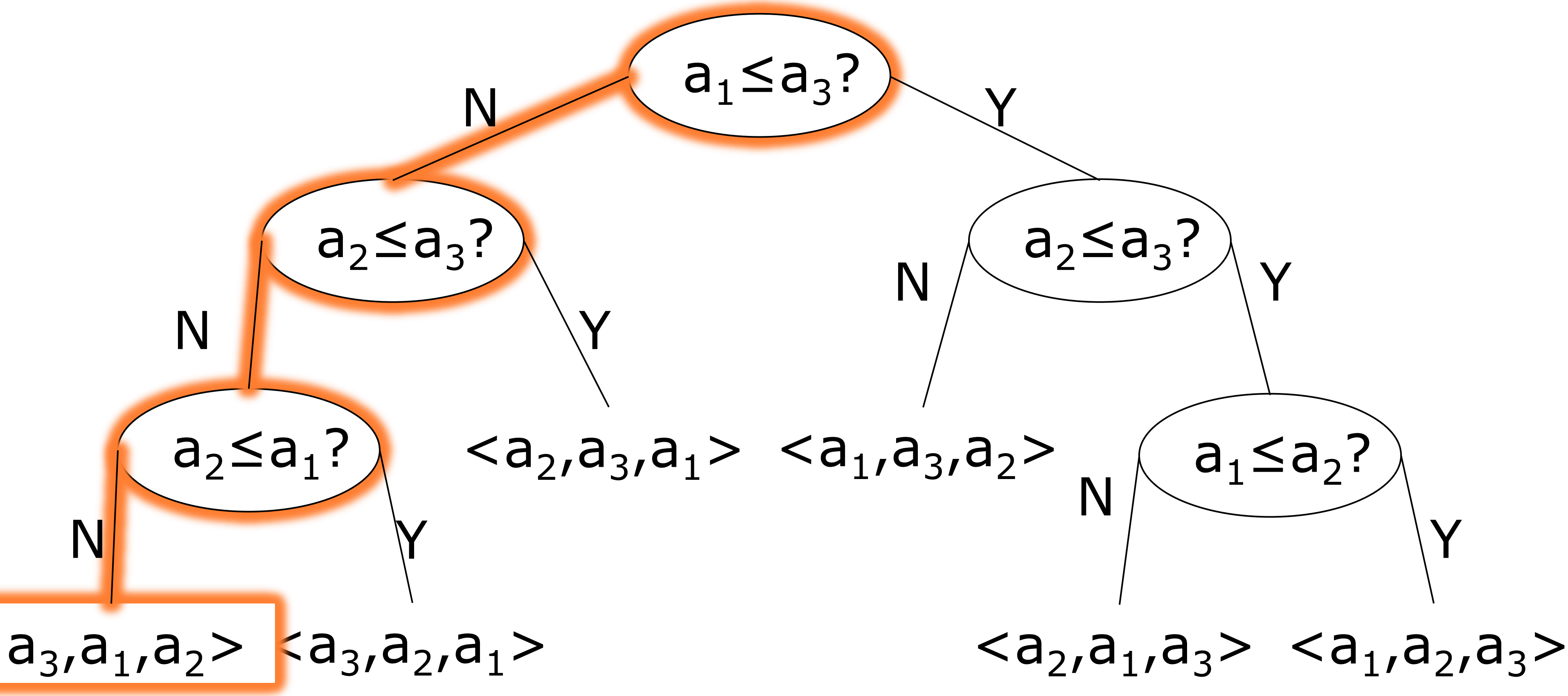
DT for Quicksort on <a_1, a_2, a_3>



DT for Quicksort on $<a_1, a_2, a_3>$



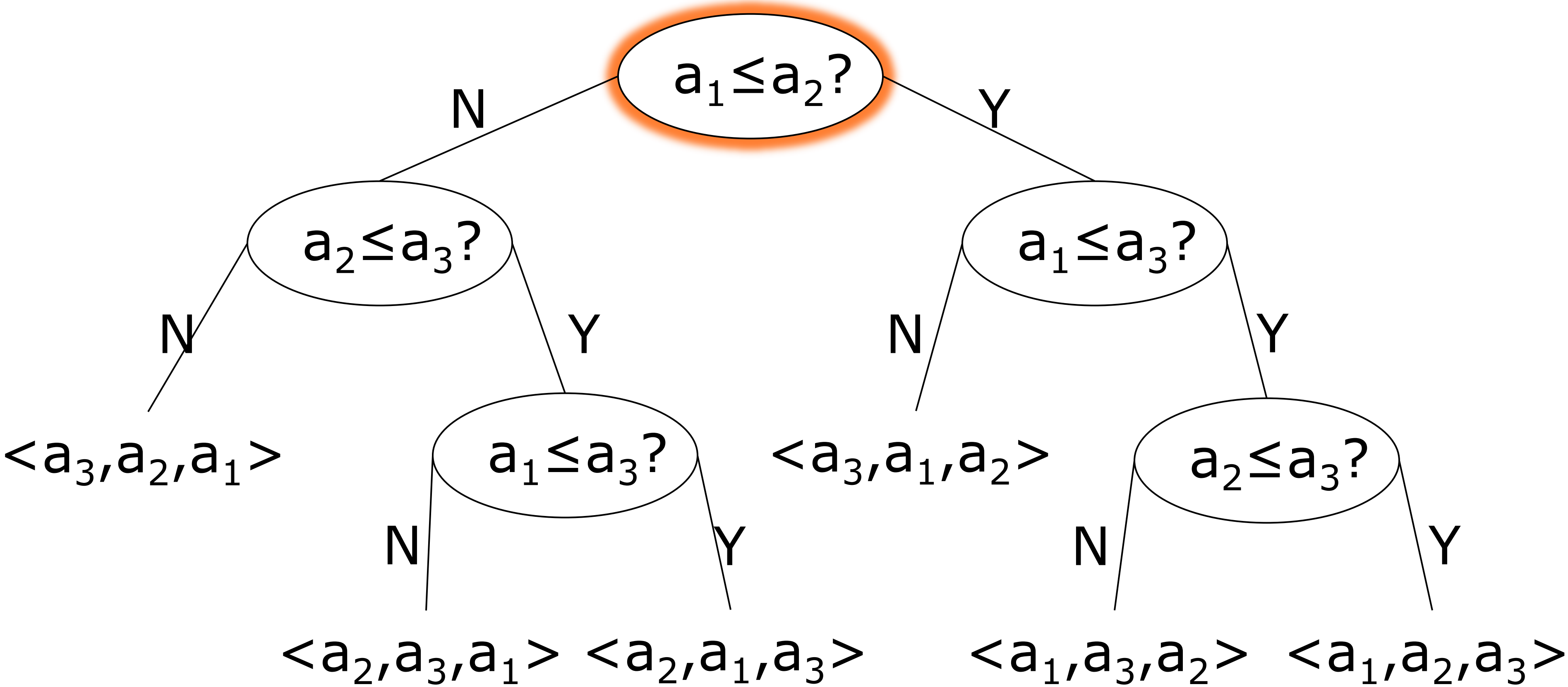
DT for Quicksort on $\langle a_1, a_2, a_3\rangle$



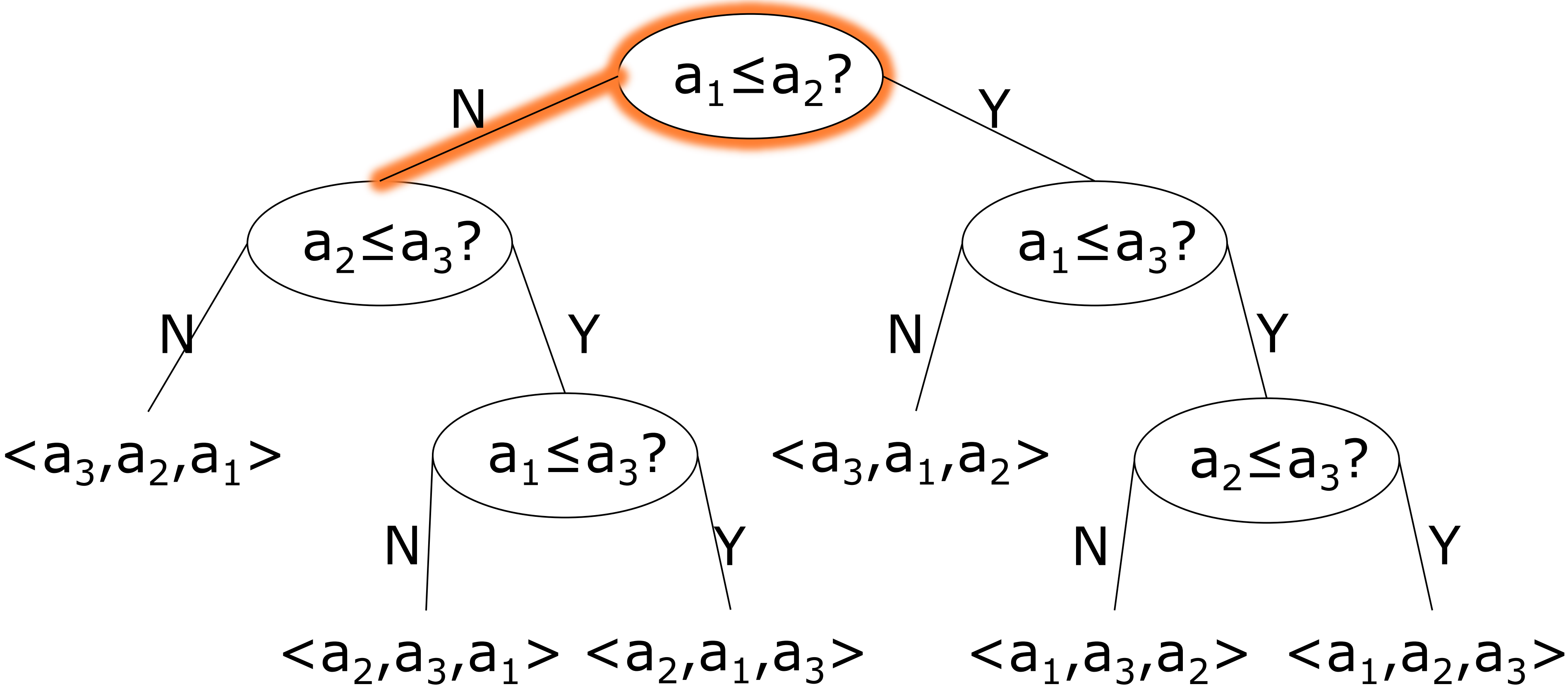
Merge Algorithm

```
Merge(A, p, q, r)                   // A[p:q]and A[q+1:r] are already sorted
01: nL = q – p + 1;                 // nL is the length of A[p:q]
02: nR = r – q;                     // nR is the length of A[q+1:r]
03: new L[0:nL-1]; new R[0:nR-1];   // two new arrays
04: for i=0 to nL-1                 //
05:   L[i] = A[p+i];                // L[0:nL-1] = A[p:q]
06: for j=0 to nR-1                 //
07:   R[j] = A[q+j+1];              // R[0:nR-1] = A[q+1:r]
08: i = 0;                          // i points to the start of L[]
09: j = 0;                          // j points to the start of R[]
10: k = p;                          // k points to the start of A[]
12: while i < nL and j < nR         //
13:   if L[i] <= R[j];              //
14:     A[k] = L[i];                // L[i] is k-th smallest
15:     i = i+1;                    //
16:   else A[k] = R[j];             // R[j] is k-th smallest
17:     j = j+1;                    //
18:   k = k+1;                      //
20: while i < nL                    // 20-23 and 24-27 are mutually exclusive
21:   A[k] = L[i];                  // copy rest of L[] to A[]
22:   i = i+1;                      //
23:   k = k+1;                      //
24: while j < nR                    //
25:   A[k] = R[j];                  // copy rest of R[] to A[]
26:   j = j+1;                      //
27:   k = k+1;                      //
```

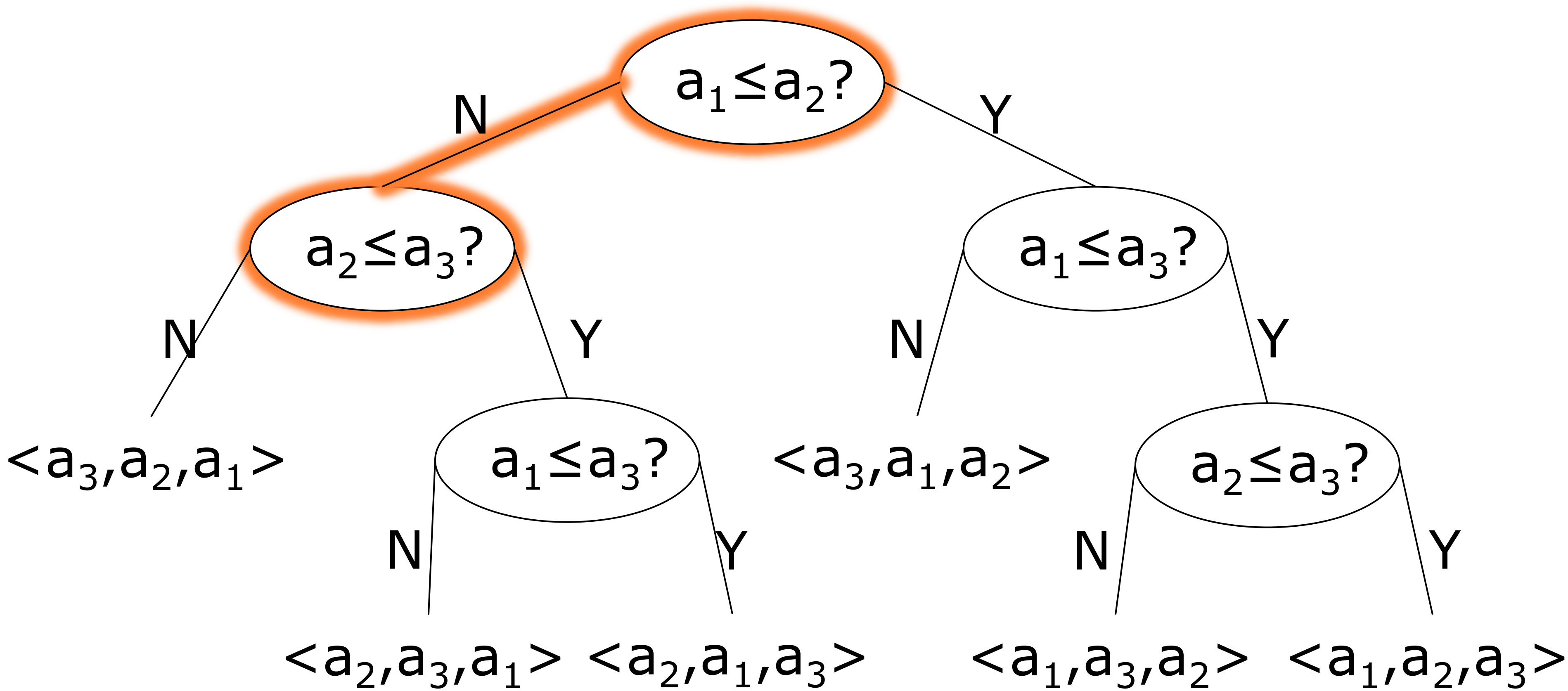
DT for Mergesort on <a_1, a_2, a_3>



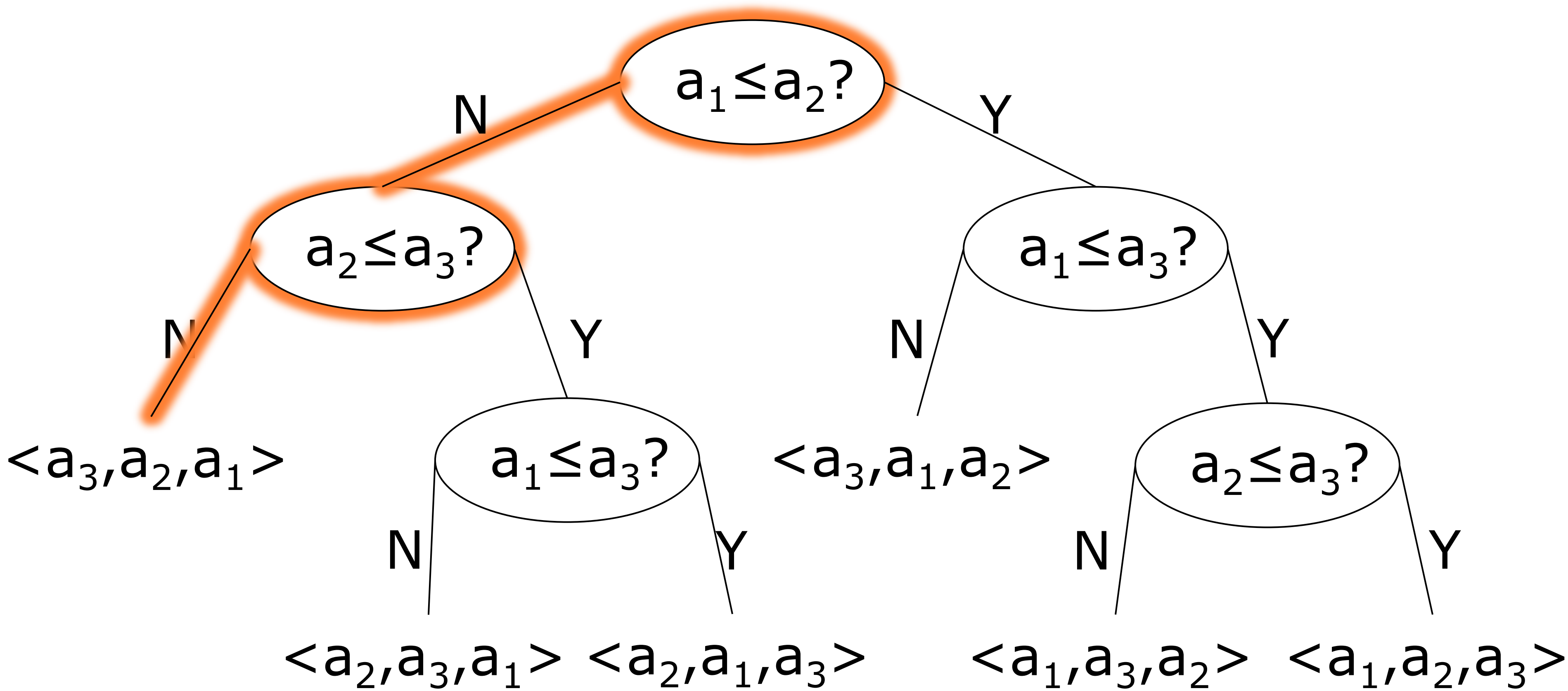
DT for Mergesort on <a_1, a_2, a_3>



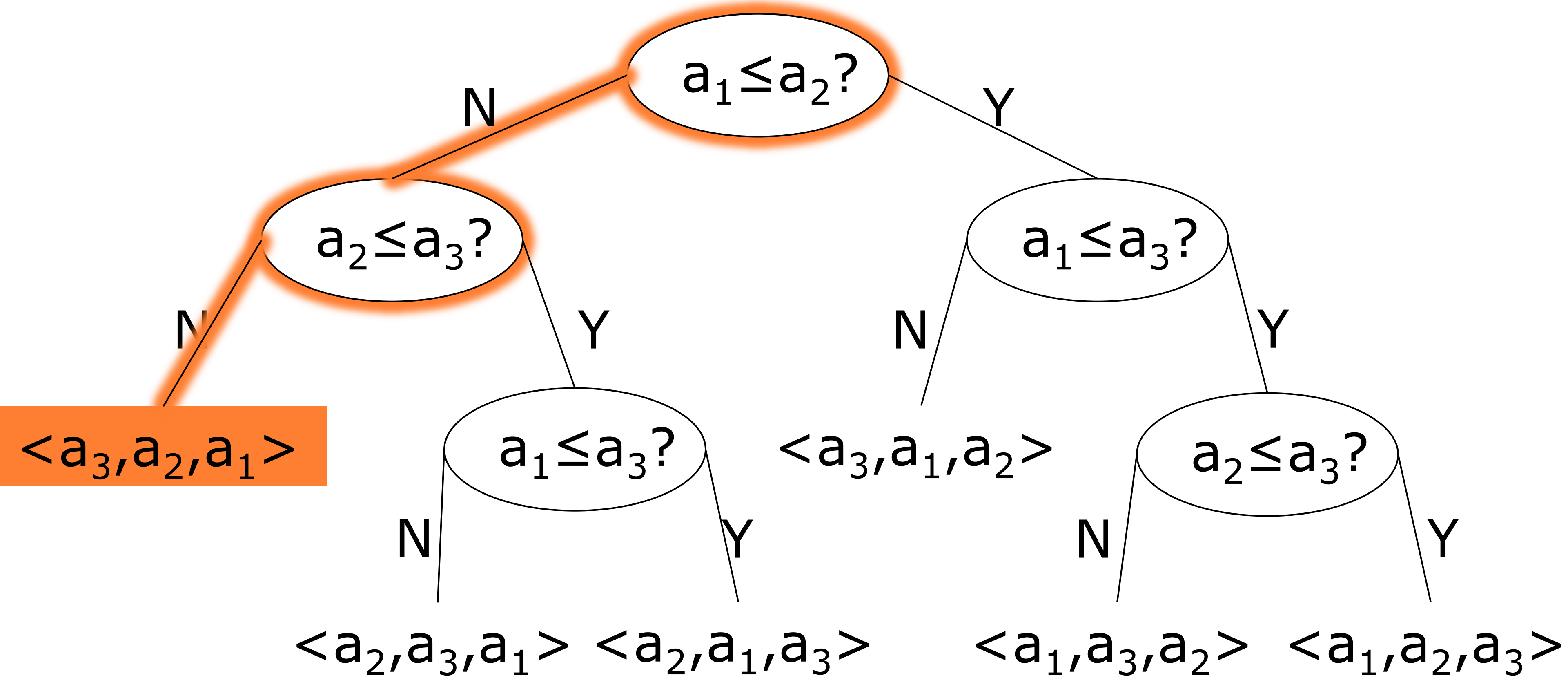
DT for Mergesort on <a_1, a_2, a_3>



DT for Mergesort on $<a_1, a_2, a_3>$



DT for Mergesort on $\langle a_1, a_2, a_3 \rangle$



Ira A. Fulton Schools of
Engineering
Arizona State University

Sorting lower-bound

- The length of the longest path from the root to a leaf in the decision tree represents the worst-case number of comparisons the sorting algorithm performs.
- Worst-case number of comparisons corresponds to height of the decision tree of the algorithm.

Height of a Binary Tree

- Minimum height of a binary tree→A binary tree of height k has at most $2^{k+1}-1$ nodes.
  - At most 2^{k-1} nodes at level k-1
  - At most $2^{k+1}-1$ nodes total.
- As a result, the minimum height of a binary tree with N nodes is floor(log(N)).
- In a decision tree for using a comparison-based sorting algorithm to sort n elements, there are n! leaf nodes. Hence the height is $\Omega(\log(n!))$.

Sorting lower-bound

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1 \tag{1}$$

$$\geq n \times (n-1) \times (n-2) \times \cdots \times \lceil \frac{n}{2} \rceil \tag{2}$$

$$\geq \left(\frac{n}{2}\right)^{\frac{n}{2}} \tag{3}$$

Therefore

$$\log(n!) \geq \frac{n}{2} \log(\frac{n}{2}) \tag{4}$$

Hence $\log(n!) \in \Omega(n \log n)$.

Sorting lower-bound

Summary

- The DT for a comparison-based sorting algorithm on n elements has n! leaf nodes.
- The height of a decision tree corresponds to the worst-case time complexity of the corresponding sorting algorithm, when sorting the given number of elements.
- The height of the DT is lower bounded by $\Omega(\log(n!))$, which is $\Omega(n*\log(n))$.
- Comparison based sorting has $\Omega(n*\log(n))$ as an asymptotic lower bound.

Ira A. Fulton Schools of Engineering
Arizona State University