Non-comparison Sorting

We have learned that the best worst case for a comparison sort is N log N. No comparison sort can perform better than that.

Assume that we have an array of N unsorted number.
- There are N! (N factorial) different ways of arranging the numbers in the array.

4	1	7

How many different ways can we arrange the array above?

$\underline{3} \cdot \underline{2} \cdot \underline{1} \Rightarrow N!$ where N is the # of elements

4 1 7   4 7 1   1 4 7   1 7 4   7 1 4   7 4 1

There are 2 case, where 4 is the first #, 2 cases where 7 is the first number, 2 cases where 1 is the first number. But only one has the correct arrangement.

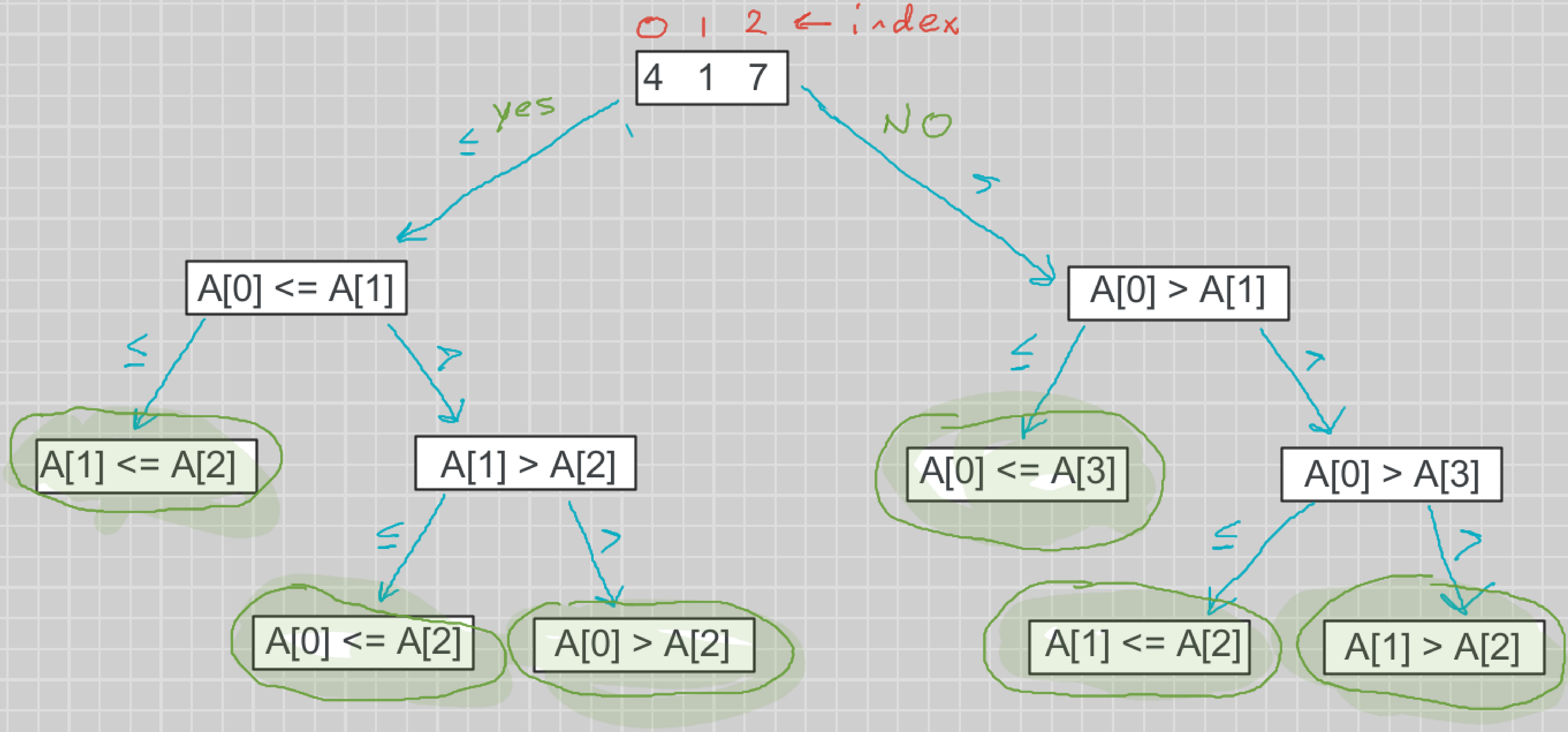
What is the maximum number of comparisons that a comparison algo must do to find the correct arrangement?

We can use a decision tree to illustrate our answer:

A decision tree is a full binary tree that represents comparisons between elements, performed by a particular sorting algo.

In a decision tree we ignore all control, data movement, and all other aspects of the algo. We are only concerned with the comparison taking place.

Given two elements a_i & a_j, we will use the comparison a_i <= a_j.



1. At every step we must make a yes/no decision. It's a tree because every choice branches out.
2. There are exactly N! leaves (the total possible number of arrangements). Each arrangement as at the end of every possible path. A terminal node is called a leaf.
3. The maximum # of comparisons is equal to the length of the longest path in the decision tree, (L).
4. If the decision tree were a complete binary tree, all paths would be the same length, there would be 2^L leaves. This gives us an upper bound to the number of leaves.

We know that the actual number of leaves (N!) cannot exceed the maximum possible number of leaves 2^L. From this we can write:

N! is at most 2^L

$$N! \le 2^L$$

L is the length of the longest path.

$$\log_2 8^k \equiv k \cdot \log_2 8$$

Apply Log to Both sides

$$\log_2(N!) \le \log_2(2^L)$$

Apply Log to Both sides

$$\frac{\log_2(N!)}{\log_2 2} \le \frac{\log_2(2^L)}{\log_2 2}$$

$$\log_2 2^L \rightarrow L \cdot \log_2 2 \rightarrow L \cdot 1 = \underline{\underline{L}}$$

$$\log_2 8 \rightarrow \log_2 2^{\boxed{3}} \rightarrow 3 \cdot \log_2 2 \rightarrow 3 \cdot 1 \rightarrow 3$$

$$\frac{\log(N!)}{\log_2 2} \le L$$

$$\log N! \le L$$

L is Omega(log N!), appliying asymptotic noation.

We know from Stirling's approximation that N! can be asymptically approximate as N^N.

$$N! \approx N^N$$

$$L \text{ is } \Omega(\log N^N)$$

$$L \text{ is } \Omega(N \cdot \log N)$$

Apply log rule

Counting sort

1) Counting sort is a non-comparison sorting algo, with a worst case running time of N (linear).

Imagine you're given a set of numbers sorted in a specific way:

	0	1	2	3	4	5	6	7	8	9	→ index, k
Frequency array C =	1	2	0	0	0	1	3	1	1	0	

frequency of each number, k.

From the frequncy array, we know that there is 1 zero, 2 ones, 1 five, 3 sixes, 1 seven, 1 eight. The frequency tells how many times the number k appears in the set.

What would the sorted array (R) look like, given the frequency array above, in which the number represented by k, is already sorted.

R =	0	1	1	5	6	6	6	7	8

R =

0	1	1	5	6	6	6	7	8

To get the sorted array R, all we had to do was visit every element of C and copy the corresponding number, k, into R as many times as shown.

Steps to complete counting sort:

1. Take an input argument, A, an unsorted array:

0	1	2	3	4	5	6	7	8
8	5	1	6	1	6	0	7	6

2. Create an array C (with the counts), that stores the frequency in which #'s in array A appear:
   a) Find the max value, k, in array A. 8 is the max value. So k = 8.
   b) Create C with the size of (k+1), where k is the max value found in A. Filled with 0s.
   c) Traverse the array A, and update array C with frequencies.

2. Create an array C (with the counts), that stores the frequency in which #'s in array A appear:

a) Find the max value, k, in array A. 8 is the max value. So k = 8.

b) Create C with the size of (k+1), where k is the max value found in A. Filled with 0s.

c) Traverse the array A, and update array C with frequencies.

$$\begin{array}{cccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ A = & 8 & 5 & 1 & 6 & 1 & 6 & 0 & 7 & 6 \end{array}$$

b) C = [0, 0, 0, 0, 0, 0, 0, 0, 0] → array of size k+1

$$\begin{array}{cccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array}$$

c) C = [1, 2, 0, 0, 0, 1, 3, 1, 1] → The frequency that each # in unsorted array A contains.

3. Create another array R (contain the sorted array), using info from array C.

a) Array R, must be of size LENGTH[A], and filled with 0s.

b) Traverse array C, and update the corresponding elements as many times as shown in C.

a) R = [0, 0, 0, 0, 0, 0, 0, 0, 0] → array of size LENGTH[A]

b) R = [0, 1, 1, 5, 6, 6, 6, 7, 8] → our sorted array

Counting sort pseudocode

A: 1D unsorted array of integers (must be integers)

k: maximum value of A

```
function countingSort(A, k)
    C <-- new Array(k+1) of 0s
    R <-- new Array(LENGTH[A]) of 0s
    pos <-- 0
    for 0 <= j < LENGTH[A] do              // populates array C with frequencies
        C[A[j]] <-- C[A[j]] + 1
    end for
    for 0 <= i < k+1 do                    // checks every element of array C
        for pos <= r < pos + C[i] do       // checks number of frequencies of each element
            R[r] <-- i                     // writes the number to the new array a certain # of times
        end for
        pos <-- r                          //updates position with the next element of array R to be fil
    end for
    return R                               // return a sorted array, R.
end function
```

```
function counting-sort(A,k)
   C ← new array(k+1) of zeros
   R ← new array(length(A)) of zeros
   pos ←  0
   for 0 ≤ j < length(A)  do
      C[A[j]] ← C[A[j]] + 1
   end for
   for 0 < i < (k+1)  do
      for pos ≤ r < pos+C[i] do
        R[r]=i
       end for
      pos=r
   end for
   return R
end function
```

$8+1+1$

$$C = \begin{array}{ccc} 0 & 1 & 2 \\ \left[\,1, \right. & 2, & \left. 3\,\right] \end{array}$$

c_0

c_1

c_2

$c_3 N + c_4$

$c_5 N$

$c_6 k + c_7 + c_8 = k+2$

$c_9 N + 8c_{10}$

$c_{11} N$

$c_{12} N + c_{13}, \quad k+1$

c_{14}

$$(c_0 + c_1 + c_2 + c_4 + c_7 + c_8 + 8c_{10} + c_{13} + c_{14}) + (c_3 + c_5 + c_9 + c_{11} + c_{12})N + c_6 K.$$

$$c_{15} + c_{16} N + c_6 K.$$

$A = 5 \qquad [1, 1, 2, 3, 6]$

$$C = \begin{array}{c} \begin{matrix} 0, & 1, & 2, & 3, & 4, & 5, & 6 \end{matrix} \\ \left[\begin{matrix} 0 & 2 & 1 & 1 & 0, & 0, & 1 \end{matrix}\right] \end{array}$$

$i = 0$

$$\text{line} = 8 = \begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 1, & 1, & 1 & 1 & 1, & 1 \end{array}$$

$\text{line} = 4 =$

$C_6 N + 7C_8$