# CIS 575. Introduction to Algorithm Analysis
# Lower Bound for Comparison-Based Sorting
# Material for March 6, 2024



The topic of this note is covered in *Cormen*'s Section 8.1.

## 1 Lower Bounds from Information Theory

Consider a given problem, like **sorting**. Until now in this course, we have analyzed numerous *specific* algorithms so as to find upper and lower bounds for their running times. But what about giving a **lower** bound[1] that will hold for **all** algorithms one may devise for a given problem?

To address this question, let us first consider an apparently quite different task:

> if I think of a number between 1 and 16, how many yes/no questions do you need to ask in order to find out which?

While a naive explorer may need 15 questions (is it 1? is it 2? is it 3? etc), it is not hard to see that 4 questions suffice, for example

1. is it at least 9? *NO*
2. is it at least 5? *YES*
3. is it at least 7? *NO*
4. is it 6? *NO*

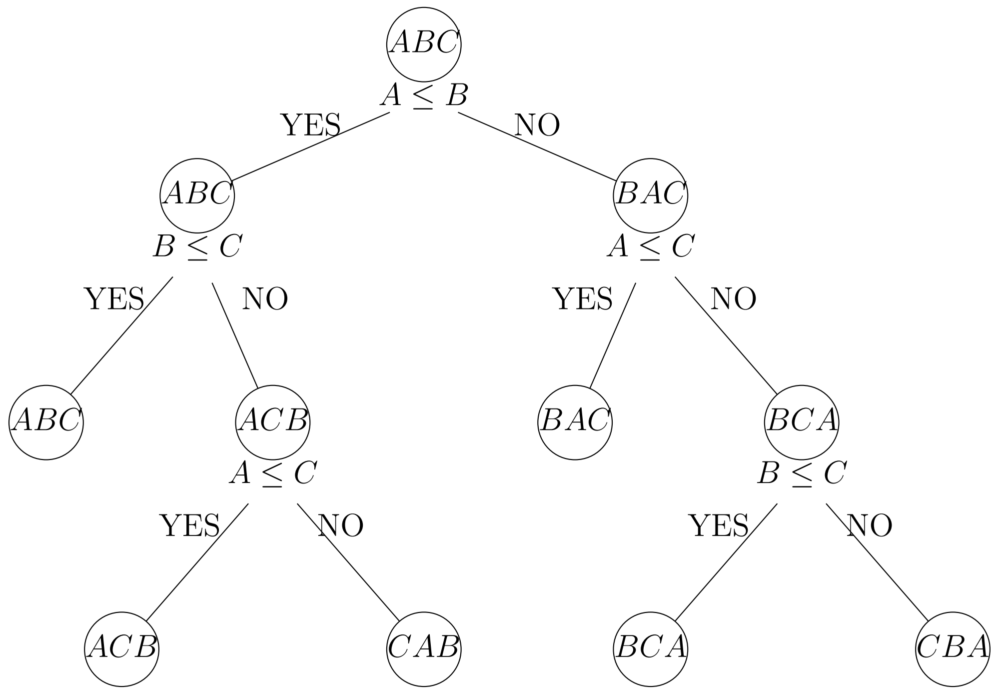
in which case we can deduce that the number must be **5**. But on the other hand, 3 questions will not always suffice. This is an instance of a general result:

> To find the correct answer among $n$ options, asking only yes/no questions, there is no strategy that always asks less than $\lg(n)$ questions.

## 2 Decision Trees

*Certain* algorithms can be viewed as exploring binary (or ternary) **decision trees**. For example, applying InsertionSort to a 3-element sequence $ABC$ will explore the decision tree

[1] There cannot be any upper bound since an algorithm may do an arbitrary amount of unnecessary work before solving the problem.




Observe that

- the **height** of such a decision tree measures worst-case running time
- if the algorithm must distinguish between $k$ eventual **outcomes** then the decision tree must have at least $k$ leaves and hence its height must be at least $\lg(\mathbf{k})$.

In general, a decision tree for InsertionSort will not contain any comparison that is *redundant* in the sense that it can be decided by recalling the result of previous comparisons. Still, for all but very small values of $n$, the tree is quite *unbalanced* in that it contains comparisons *not likely* to gain much information: when the $n$'th element is inserted into the result of sorting the $n-1$ first elements, it is first compared to the largest of those elements, and is very likely to be found smaller. As a consequence, many branches have length quadratic in the number of elements.

On the other hand, the decision trees explored by HeapSort or MergeSort are much more balanced, as reflected in their $\Theta(n \lg(n))$ running time. (But HeapSort does perform redundant comparisons, partly explaining its reputation for being slower than QuickSort.)

# 3 Comparison-Based Sorting

All the sorting algorithms we have encountered work by asking yes/no questions, of the form: is $A[i]$ less than $A[j]$? Any sorting algorithm needs to be able to distinguish between different permutations of input: it we say run it first on $3, 7, 5$ and next on $7, 3, 5$ then different actions must be taken and hence some comparison must have given different results. Since there are $n!$ permutations of input (assuming all elements are different), we see that a sorting algorithm must be able to distinguish between $n!$ different outcomes, and thus:

> To find the correct course of action, asking only yes/no questions, there is no sorting algorithm that always makes less than $\lg(n!)$ comparisons.

It is not hard to estimate $\lg(n!)$; we get (using *Howell*'s Theorem 3.28)

$$\lg(n!) = \lg(\prod_{i=1}^{n} i) = \sum_{i=1}^{n} \lg(i) \in \Theta(n\lg(n))$$

which shows that

> **All comparison-based sorting algoritms must have worst-case running time in** $\boldsymbol{\Omega}(\mathbf{n}\lg(\mathbf{n}))$.

This is the content of *Cormen*'s Theorem 8.1. Observe the qualifier "comparison-based"; there exists a sorting algorithm, COUNTING SORT, which is *not* comparison-based and which runs in time $\Theta(n+k)$ where $k$ is the highest possible key.