Med = 38.5
Hi = 50
Lo = 4

Lower bounds for comparison-based sorting.

Comparison-based: only information about arrangement is compare two keys to see which one is less (comes first)

**Theorem**: Any algorithm that correctly sorts a list of \( n \) items making only comparisons must take \( \Omega(n \log n) \) time in the worst case.

[must make \( \Omega(n \log n) \) many comparisons]

Mergesort, Heapsort are asymptotically optimal (can’t do better with comparison based sort).

**Proof**: Information-based.
Less than \( n \ln n \) yes-no questions does not provide enough information to sort.

\[
\text{arrangement 1} \neq \text{arrangement 2}
\]

[assume no duplicate keys]

Let \( A \) be any sort comparison-based sorting algorithm. \( A \) must behave differently on \( \text{arr. 1} \) than on \( \text{arr. 2} \), so there must be some comparison answered differently from the two arrangements.

Must gather enough bits to distinguish among all possible initial arrangements.

There are \( n! \) many initial arrangement (permutations) of \( n \) items.

\[
\begin{array}{c}
\left[a_1, a_2, a_3, a_4, \ldots \right] \\
\text{choices} \\
\text{choices} \\
\text{choices} \\
\end{array}
\]

\( k \) many comparisons in the worst case distinguish between \( 2^k \) many possibilities, so \( 2^k \geq n! \)
(If $2^k < n!$, then there are two different initial arrangements of the input that both lead to the same sequence of answers to comparisons. So $A$ moves keys around same for both, so the final arrangements will be different also.

\[ \therefore \text{At most one of the two final arrangements will be sorted (the other won't be)} \]

\[ \therefore A \text{ is not a correct sorting algo.} \]

\[ \text{So } 2^k \geq n! \text{ on some path.} \]

\[ \iff k \geq \lg(n!) \]

\[ = \lg \left( \frac{n!}{\prod_{i=1}^{n} i} \right) \]

\[ = \sum_{i=1}^{n} \lg i \]

\[ = \sum_{i=2}^{n} \lg i \]

\[ = \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \lg \left( \frac{n}{i} \right) \]

\[ = \sum_{i=\lceil \frac{n}{2} \rceil}^{n} (\lg n - 1) \]

\[ = \frac{n}{2} \lg n - \frac{n}{2} = \Omega(n \lg n) \]
Decision Tree Approach:

\[
\begin{align*}
&\text{abc?} \quad \text{yes} \\
&\text{bcd?} \quad \text{no} \\
&\text{def?} \quad \text{yes} \\
&\text{ghi?} \quad \text{no} \\
&\text{jkl?} \quad \text{no} \\
&\text{mno?} \quad \text{no} \\
&\text{pqr?} \quad \text{no} \\
&\text{stu?} \quad \text{no} \\
&\text{vwx?} \quad \text{no} \\
&\text{yz?} \quad \text{no} \\
&\text{a?} \quad \text{no} \\
&\text{b?} \quad \text{no} \\
&\text{c?} \quad \text{no} \\
\end{align*}
\]

at least \( n! \) many leaves

Fact: (pigeon-hole principle)

Any binary tree with depth \( k \) has \( \leq 2^k \) many leaves. Again \( 2^k \geq n! \).

The tree has depth \( \mathcal{O}(n \log n) \).

A has worst-case time \( \mathcal{O}(n \log n) \).

Order Statistics

Selection Problem

Given a list of \( n \) values (numbers). The \( i^{th} \) order statistic \( (1 \leq i \leq n) \) is the \( i^{th} \) smallest value in the list (counting duplicates).

If list is sorted, stored in \( A[l..n] \), then \( i^{th} \) order statistic is \( A[i] \).

Selection Problem: given \( A[l..n] \) and \( i \), find \( i^{th} \) order stat.
E.g., Median is the \( \left\lfloor \frac{n}{2} \right\rfloor \)th order stat
\( \left\lceil \frac{n}{2} \right\rceil \)
\[ \] is sorted. Selection takes constant time \( O(1) \)
[linked list: \( O(n) \) time]
sequential file
Selection Alg: Sort A first, then find \( i \)th element.
\( O(n \log n) \) time.
Optimal? no
(Randomized selection in expected time \( @n \))
Deterministic selection in time \( @n \) worst case.
(optimal.)

Randomized Selection:
\[
\begin{array}{c}
\text{Random Partition} \\
\hline
\leq x \quad |x| \quad \geq x \\
\hline
k \uparrow \quad \quad n-k-1
\end{array}
\]
If \( i = k+1 \), then return pivot
If \( i \leq k \), recurse on left sublist (same \( i \))
If \( i > k+1 \), recurse on right sublist
\( (i-k-1 \) order stat)
Randomized Select \( (A, p, q, i) \)
// Find the \( i \)th order
// stat in \( A[p \ldots q] \)
// Precondition: \( 1 \leq i \leq q-p+1 \)
if \( p = q \) then return \( A[p] \)
\( r \leftarrow \text{Random Partition} (A, p, q) \)
// \( r = \text{index of pivot} \)
if \( (i = r-p+1) \)
    return \( A[r] \)
else if \( (i < r-p+1) \)
    return \( \text{Randomized Select} (A, p, r-1, i) \)
else
    return \( \text{Randomized Select} (A, r+1, q, i-r+p-1) \).