Today: Linear-Time Sorting
- comparison model
- lower bounds:
  - searching: \( \Omega(lg n) \)
  - sorting: \( \Omega(n lg n) \)
- \( O(n) \) sorting algorithms
  - counting sort
  - radix sort

Lower bounds: claim
- searching among \( n \) preprocessed items requires \( \Omega(lg n) \) time
  \( \Rightarrow \) binary search, AVL tree search optimal
- sorting \( n \) items requires \( \Omega(n lg n) \)
  \( \Rightarrow \) mergesort, heap sort, AVL sort optimal
  ... in the comparison model

Comparison model of computation:
- input items are black boxes (ADTs)
- only support comparisons (\(<, >, \leq, \geq\) etc.)
- time cost = \# comparisons
Decision tree: any comparison algorithm can be viewed/specified as a tree of all possible comparison outcomes & resulting output, for a particular n:

- e.g. binary search for n=3:

```
A[1] < x?
  NO
  A[0] < x?
    NO
    x ≤ A[0]
    YES
    A[0] < x ≤ A[1]
  YES
    NO
    YES
    YES
```

- internal node = binary decision
- leaf = output (algorithm is done)
- root-to-leaf path = algorithm execution
- path length (depth) = running time
- height of tree = worst-case running time

In fact, binary decision tree model is more powerful than comparison model, and lower bounds extend to it.
Search lower bound:
- \# leaves ≥ \# possible answers
- \# leaves ≥ \( n \) (at least 1 per \( A[i] \))
- decision tree is binary
  \( \Rightarrow \) height ≥ \( \lg \Theta(n) = \lg n + \Theta(1) \)

Sorting lower bound:
- leaf specifies answer as permutation:
- all \( n! \) are possible answers
  \( \Rightarrow \) \# leaves ≥ \( n! \)
  \( \Rightarrow \) height ≥ \( \lg n! \)
  \[ = \lg (1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n) \]
  \[ = \sum_{i=1}^{n} \lg i \]
  \[ ≥ \sum_{i=\lceil \log_2 n \rceil}^{n} \lg i \]
  \[ ≥ \sum_{i=\lceil \log_2 n \rceil}^{n} \log \frac{n}{i} \]
  \[ = \frac{n}{2} \lg n - \frac{n}{\log n} = \Omega(n \lg n) \]
- in fact \( \lg n! = n \lg n - O(n) \) via:

  Sterling's formula:
  \[ n! \sim \sqrt{2\pi n} \left( \frac{n}{e} \right)^n \]
  \[ \Rightarrow \lg n! \sim n \lg n - (\lg e) n + \frac{1}{2} \lg n + \frac{1}{2} \lg (2\pi) \]
Linear-time sorting: if \( n \) keys are integers \( \in \{0, 1, \ldots, k-1\} \), can do more than compare them.

\( \Rightarrow \) lower bounds don't apply.
- if \( k = n^{O(1)} \), can sort in \( O(n) \) time.

\( \text{OPEN: } O(n) \) time possible for all \( k \)?

Counting sort:
- \( L = \) array of \( k \) empty lists \( \text{?} \) \( O(k) \)
- \( \text{for } j \text{ in range}(n): \)
  \( \text{L[key(A[j])].append(A[j])} \) \( O(1) \) \( \checkmark \) random access using integer key
- output = []
- \( \text{for } i \text{ in range}(k): \)
  \( \text{output.extend(L[i])} \) \( O(\sum_{i=1}^{k} L[i]) \) \( \sum = O(k+n) \)

Time: \( \Theta(n+k) \)
- also \( \Theta(n+k) \) space

Intuition: count key occurrences using RAM output <count> copies of each key in order
- but item is more than just a key

CLRS has cooler implementation of counting sort with counters, no lists ~
- but time bound is the same
Radix sort:
- imagine each integer in base b
  \[ d = \log_b k \] digits \( \in \{0, 1, \ldots, b-1\} \)
- sort by least significant digit
- \( \ldots \Rightarrow \) all \( n \) items
- sort by most significant digit
  \( \Rightarrow \) sort must be stable:
  preserve relative order of items
  with the same key
  \( \Rightarrow \) don’t mess up previous sorting

\[ \begin{array}{c|c|c|c|c}
3 & 2 & 9 & 7 & 2 \\
4 & 5 & 7 & 3 & 5 \\
6 & 5 & 7 & 4 & 3 \\
8 & 3 & 9 & 4 & 5 \\
7 & 2 & 0 & 6 & 5 \\
3 & 5 & 5 & 8 & 3 \\
\end{array} \]

\[ \begin{array}{c|c|c|c|c}
7 & 2 & 0 & 3 & 2 \\
3 & 5 & 5 & 4 & 3 \\
4 & 3 & 6 & 8 & 3 \\
6 & 5 & 7 & 4 & 5 \\
7 & 2 & 0 & 6 & 5 \\
8 & 3 & 9 & 3 & 2 \\
\end{array} \]

\[ \begin{array}{c|c|c|c|c}
3 & 2 & 9 & 7 & 2 \\
3 & 5 & 5 & 4 & 3 \\
4 & 3 & 6 & 8 & 3 \\
6 & 5 & 7 & 4 & 5 \\
7 & 2 & 0 & 6 & 5 \\
8 & 3 & 9 & 3 & 2 \\
\end{array} \]

- use counting sort for digit sort
  \( \Rightarrow \Theta(n+b) \) per digit
  \( \Rightarrow \Theta((n+b)d) = \Theta((n+b) \log_b k) \) total time
- minimized when \( b = n \)
  \( \Rightarrow \Theta(n \log n k) \)
  \( = O(n^c) \) if \( k \leq n^c \)