Today

- Big-O
- Sorting intro

Assignments

- HW6 out (due after Spring Break)
- Quiz on Thurs
In Big-O notation we don’t include constant terms

e.g., instead of writing $3n + 2$, we just write $O(n)$

and instead of writing $6n^2 + n - 8$, we just write $O(n^2)$

- We are interested in algorithm’s general rate of growth (orders of magnitude)
- Which makes it easier to broadly compare them

Order-of-magnitude analysis ... Why can we remove the constants?

- Focus on input sizes large enough to make only the growth relevant
  ... thus, as the input size increases without bound (in the limit)

- Sometimes called the "asymptotic" efficiency of algorithms
One definition of the Order of an algorithm:

Algorithm $A$ is $O(f(n))$ if constants $k$ and $n_0$ exist that s.t. $A$ requires no more than $k \cdot f(n)$ time units to solve a problem of size $n \geq n_0$.

Alternative definition of the Order of an algorithm:

For a given function $f(n)$, $O(f(n))$ is the set of functions:

$$O(f(n)) = \{g(n) \mid \text{constants } k \text{ and } n_0 \text{ exist s.t. } 0 \leq g(n) \leq k \cdot f(n) \text{ for all } n \geq n_0\}$$

Big-O gives an upper bound on a function $f(n)$ to within a constant factor
Some Common Algorithm Growth Rates
Asymptotic Upper Bounds

Big-O notation can sometimes be misleading:

- With upper bounds, e.g., any algorithm that is $O(n)$ is also $O(n^2)$ (etc.)

- Are we analyzing the algorithm or the problem?
  - Algorithm is a particular approach to the problem (bubble sort v sorting)
  - Problem itself may be bound (e.g., what is sorting’s smallest upper bound)
Removing the constants ...

**Example 1:** Assume $T(n) = 27$, which is $O(1)$ (constant time).
- In this case, pick $n_0$ to be 0 (any value will work in this case)
- Pick $k$ to be 27, giving:
  
  $$0 \leq 27 \leq 27 \cdot 1$$
  
  for $k = 27$ and all $n \geq 0$

**Example 2:** Assume $T(n) = 3n + 2$, which is $O(n)$ (linear time).

Q: What values of $n_0$ and $k$ do we pick to solve this?
- There are lots, but let's pick $k = 5$ and $n_0 = 1$, giving:
- $3n + 2 \leq 5n$ for all $n \geq 1$
- e.g., $3 \cdot 1 + 2 \leq 5$, $3 \cdot 2 + 2 \leq 10$, $3 \cdot 3 + 2 \leq 15$, etc.
Example 3: Assume $T(n) = n^2 - 5n + 10$, which is $O(n^2)$ (quadratic time).

Q: What values of $n_0$ and $k$ do we pick to solve this?

- There are lots, but let’s pick $n_0 = 2$ and $k = 4$, giving:
- $n^2 - 5n + 10 \leq 4n^2$ for all $n \geq 1$
- Note: $T(2) = 4 - 10 + 10 = 4 \leq 4 \cdot 2$
## Sorting Algorithms

<table>
<thead>
<tr>
<th>Sorting Alg</th>
<th>Best Case</th>
<th>Avg Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bubble Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insertion Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merge Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quick Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tree Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We’ll go algorithm by algorithm, but save the last two until later

In HW-6 you’ll implement Selection, Merge, and Quick Sort over Linked Lists!
Selection Sort

The basic idea

- Select (i.e., “find”) the largest item
- Swap it with the last item in the list
- Repeat with items 0 to $n - 2$
- Stop when only one item left

Pass 1:

\[
\begin{array}{cccc}
29 & 10 & 14 & 13 \\
\end{array}
\quad \Rightarrow \quad
\begin{array}{cccc}
13 & 10 & 14 & 29 \\
\end{array}
\]

Pass 2:

\[
\begin{array}{cccc}
13 & 10 & 14 & 29 \\
\end{array}
\quad \Rightarrow \quad
\begin{array}{cccc}
13 & 10 & 14 & 29 \\
\end{array}
\]

Pass 3:

\[
\begin{array}{cccc}
13 & 10 & 14 & 29 \\
\end{array}
\quad \Rightarrow \quad
\begin{array}{cccc}
10 & 13 & 14 & 29 \\
\end{array}
\]