In [1]:

```python
# setup
from IPython.core.display import display,HTML
display(HTML('<style>.prompt{width: 0px; min-width: 0px; visibility: collapse}</style>'))
display(HTML(open('rise.css').read()))

# imports
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(style="whitegrid", font_scale=1.5, rc={'figure.figsize':(12, 6)})
```

CMPS 2200

Introduction to Algorithms

Quicksort

Today's agenda:

- Quicksort

We saw how the problem of selection could be solved with a randomized algorithm. The key was to choose a random element and then partition the list into two parts.

What if we recursively sorted these two parts?

Let $a = \langle 2, 5, 4, 1, 3, -1, 99 \rangle$. Suppose we chose 4 as the pivot. Then the two parts of the list are $\ell = \langle 2, 1, 3, -1 \rangle$ and $r = \langle 5, 99 \rangle$. In sorted order they are $\ell' = \langle -1, 1, 2, 3 \rangle$ and $r' = \langle 5, 99 \rangle$.

So all we have to do is append l', the pivot and r'!

This suggests a divide-and-conquer algorithm, but with similar characteristics as our algorithm for selection.

This sorting algorithm is actually called *Quicksort* (invented in 1959 by C. A. R. Hoare).

$$
\begin{array}{l}
\mathit{quicksort}\; a = \\
\quad \texttt{if}\; |a| = 0\; \texttt{then}\; a \\
\quad \texttt{else} \\
\quad\quad \texttt{let} \\
\quad\quad\quad p = \texttt{pick a random pivot from}\; a \\
\quad\quad\quad a_1 = \langle\, x \in a \mid x < p \,\rangle \\
\quad\quad\quad a_2 = \langle\, x \in a \mid x = p \,\rangle \\
\quad\quad\quad a_3 = \langle\, x \in a \mid x > p \,\rangle \\
\quad\quad\quad (s_1, s_3) = (\mathit{quicksort}\; a_1) \;\; || \;\; (\mathit{quicksort}\; a_3) \\
\quad\quad \texttt{in} \\
\quad\quad\quad s_1 \texttt{++} a_2 \texttt{++} s_3 \\
\quad\quad \texttt{end}
\end{array}
$$

In [ ]:

```
[1,2,3,4,5,6,7,8]

p = 1
a1 = [], a2 = [1], a3 = [2,3,4,5,6,7,8]

[2,3,4,5,6,7,8]
p = 2
a1 = [], a2 = [2], a3 = [3,4,5,6,7,8]

W(n) = W(n-1) + n

n
n-1
n-2

worst level * n_levels
n * n \in O(n^2)
```

In [ ]:

```

```

In terms of parallelism, we can partition in parallel as before and sort the two parts of the list in parallel.

How should we analyze the work of quicksort?

We'll take a slightly different approach than for selection to estimate the expected work. To account for work, we'll look at the total amount expected work.

Let the random variable $Y(n)$ be the number of comparisons made by Quicksort on an input of size n. Note that the work is $O(Y(n))$, since there is $O(n)$ work done by non-comparisons (i.e., choosing pivots, concatenation of lists).

It'll be useful to consider the sorted version of a, indexed as $z_0, z_1, \ldots, z_{n-1}$. Now, let X_{ij} be an indicator random variable that is 1 if z_i and z_j are ever compared and 0 otherwise. So we have that $Y(n) = \sum_i \sum_j X_{ij}$ and so this means that:

$$\mathbf{E}\left[Y(n)\right] = \mathbf{E}\left[\sum_i \sum_j X_{ij}\right] = \sum_i \sum_j \mathbf{E}\left[X_{ij}\right]$$

We saw that $\mathbf{E}[X_{ij}] = \mathbf{P}[X_{ij} = 1]$. So when are a pair of elements a_i and a_j compared?

e.g.

$$[10, \mathbf{5}, 2, 6, 1, \mathbf{12}, 9]$$

qsort.png

In Quicksort, two elements z_i and z_j are only ever compared if one of them is a pivot. Moreover, if any element *between* z_i and z_j in the sorted order is chosen as a pivot *before* z_i or z_j. Since the pivot is chosen randomly,

$$\mathbf{P}[X_{ij}] = \frac{2}{j - i + 1}.$$

This gives us

$$
\begin{aligned}
\mathbf{E}\left[Y(n)\right] &\le \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \mathbf{E}\left[X_{ij}\right] \\
&= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \frac{2}{j-i+1} \\
&= \sum_{i=0}^{n-1} \sum_{k=2}^{n-i} \frac{2}{k} \\
&\le 2 \sum_{i=0}^{n-1} \ln n \qquad \text{Harmonic series: } \sum_{k=1}^{n} \frac{1}{k} < \ln n \\
&= O(n \lg n).
\end{aligned}
$$

Analyzing the span of Quicksort can be done in the same way as we did for selection. That is, if we have a guarantee that at level d of recursion that the larger of the two lists is $(3/4)^d n$, then we can show that the span at each level is $O(\log n)$ (expected). Using the same approach as for selection we can show that the total span is $O(\log^2 n)$ with high probability.

The Monty Hall Problem

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?"

Is it to your advantage to switch your choice?

Why or why not?

This puzzle is based on the game show *Let's Make a Deal* and named after the host Monty Hall.

So the probability of winning by switching is 2/3, while the probability of keeping yoour choice and winning is 1/3.

Suppose you pick Door 1. With the doors closed, there is a 1/3 probability of winning the car.

With one door open, the odds of that door being winning go to 0 and the odds of Door 2 being winning go to 2/3.

In her column titled "Ask Marilyn", Marilyn vos Savant suggested that it's useful to think of having 1,000,000 doors. You make a choice, and the host opens 999,998 doors. Should you switch?

In this arrangement, we flip a coin after the goat is revealed and have even odds of winning. One way to think about this is that we're just ignoring the information that is given.