Analyzing runtime of recursive algorithms.

ex: Merge Sort

Quick Review
"Continuously split list in ½ until lists of size 1, merge these sublists together."

```
MergeSort(A) // A is list

// sublist has > 2 items
if (|A| > 1)
    leftA = getLeftHalf(A)
    rightA = getRightHalf(A)
    MergeSort(leftA)
    MergeSort(rightA)
    Merge(leftA, rightA)
```

Divide-and-Conquer-(and-combine)
- Divide: break problem into smaller subproblems
- Conquer: recursively solve each subproblem
- Combine: combine subsolutions

"""For Merge Sort?""
recurrence - describes runtime of a recursive call in terms of input size \( n \).

\[ T(n) = \text{runtime of MergeSort} \]

"Let's write as a recurrence. Since the function has 2 parts: base case + recursive, the recurrence has 2 parts."

\[
T(n) = \begin{cases} 
  c & \text{if } n \leq 1 \\
  \frac{n}{2} T\left(\frac{n}{2}\right) + cn + c & \text{if } n > 1 
\end{cases}
\]

"What if we can write conquer time in terms of \( T(n) \)?"

"\( T(n) \) doesn't tell us much about runtime since it's in terms of \( T(n) \). Want it in terms of \( n \) only. To do so, we will solve the recurrence."

To solve a recurrence: (3 methods)

1. Recurrence Tree (visual) < today
2. Master's Theorem
3. Substitution Method

Recurrence Tree
Assume \( n \) is a power of 2: \( n = 2^k \).
Recall tree for memoized Coin-Change

\[ \text{Change}(n) \quad \text{— non-recursive part took } O(k) \text{ and we made 3 recursive calls} \]

\[ \text{Change}(n-12), \text{Change}(n-5), \text{Change}(n-1) \]

- Each node represents a recursive call.
- Each call takes time \( O(k) \) (to find min) and makes 3 recursive calls.
- \( n \) total calls.
- Total time \( O(nk) \).

Similarly for MergeSort.

- Each call takes \( \log n \) and makes 2 recursive calls

\[
\begin{align*}
\frac{cn}{2} & \quad \Rightarrow \quad \frac{cn}{2} \\
? & \quad \frac{T(n)}{2} \quad \text{"unravel"} \quad \frac{cn}{2} \quad \frac{cn}{2} \quad \Rightarrow \quad \frac{cn}{2} \quad \frac{cn}{2} \\
& \quad \text{write in terms of } T(n)? \quad T(\frac{n}{4}) \quad T(\frac{n}{4}) \quad T(\frac{n}{4}) \quad T(\frac{n}{4}) \quad \frac{cn}{4} \quad \frac{cn}{4} \\
& \quad = T(\frac{n}{2}) \\
& \quad T(\frac{n}{8}) \quad T(\frac{n}{8})
\end{align*}
\]

Eventually? Last level of tree?

All \( c \)'s \( \Rightarrow \) Recurse down to lists of size 1.

\[ \text{Time} = c \]
"Let's find time at each level and multiply by number of levels."

Time at each level = \( cn \)

# levels = \( O(\log n) \). (Start from \( n \), halve until 1)

Total time = \( O(n \log n) \).

Note: Actually \( \Theta(n \log n) \) not just \( O(n \log n) \).
Ex.(2)

\[ T(n) = \begin{cases} 
  c & \text{if } n=1 \\
  2T(\frac{n}{2}) + cn^2 & \text{if } n > 1 \Rightarrow \text{function takes } n^2 \\
  \text{time and makes } 2 \text{ recursive calls.}
\end{cases} \]

\[ \frac{cn^2}{T(\frac{n}{2})} = \frac{n^2}{c(n)\frac{n^2}{2}} \Rightarrow \frac{T(n)}{T(\frac{n}{4})} \]

Time:

\[ \frac{cn^2}{c(n)} = \frac{c(n)}{\frac{4}{n}} \cdot 2 = \frac{cn^2}{2} \]

\[ \frac{c(n)}{\frac{4}{n}} \cdot \frac{c(n)}{\frac{4}{n}} = \frac{cn^2}{4} \cdot 4 = \frac{cn^2}{4} \]

\[ \frac{cn^2}{\frac{16}{8}} = \frac{cn^2}{8} \]

(Doesn’t really matter what the last level looks like)

Can’t just multiply time at each level by # levels since each level has different time, so let’s take sum.
\[ T(n) = \frac{cn^2}{2} + \frac{cn^2}{4} + \frac{cn^2}{8} + \ldots = cn^2 \left[ 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots \right] \]

\[ = cn^2 \sum_{i=0}^{\log_2 n} \frac{1}{2^i} \]

Somewhat complicated summation.

What if we just want Big-Oh, so an upper bound to \( T(n) \) (something more than \( T(n) \)).

Easier summation?

\[ T(n) = cn^2 \sum_{i=0}^{\log_2 n} \frac{1}{2^i} \]

Geometric Series: \( \sum_{i=0}^{\infty} r^i = \frac{1}{1-r} \) for \( |r| < 1 \)

Our equation:

\[ T(n) = cn^2 \sum_{i=0}^{\log_2 n} \frac{1}{2^i} = cn^2 \left( \frac{1}{1 - \frac{1}{2}} \right) = 2cn^2 \]

\[ \therefore T(n) = O(cn^2). \]
From recurrence, can see $T(n) = \Omega(n^2)$.

$\therefore \ T(n) = \Theta(n^2)$.

Ex (3): Group work.

$T(n) = \begin{cases} c & \text{for } n=1 \\ 4T(n/2) + cn & \text{for } n>1 \end{cases}$

\[ T(n) = \frac{cn}{2} \Rightarrow \frac{cn}{2} \Rightarrow \frac{cn}{2} \Rightarrow \frac{cn}{2} \]

\[ 4\left(\frac{cn}{2}\right) = 2cn \]

\[ 16\left(\frac{cn}{4}\right) = 4cn \]

\[ 64\left(\frac{cn}{8}\right) = 8cn \]

\[ T(n) = cn + 2cn + 4cn + 8cn + \ldots \]

$= cn \ (1 + 2 + 4 + 8 + \ldots)$.

How many levels? $O(\log n)$. (n halved until 1).

Write as sum:

$T(n) = cn \leq \sum_{i=0}^{\log n} 2^i$

Can we take sum to $\infty$ here as in previous example? No, then would get $T(n) = O(\infty)$. 
Again apply geometric series:

\[ \sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r} \quad \text{for } |r| < 1 \]

Our equation:

\[ T(n) = cn \sum_{i=0}^{\log(n)} 2^i = cn \left[ \frac{1-2^{\log(n)+1}}{1-2} \right] \]

\[ = cn \left[ \frac{1-2 \cdot 2^{\log(n)}}{-1} \right] = 2n^2 \quad \text{if } n \geq 2 \]

\[ = cn \left[ \frac{1-2 \cdot n}{-1} \right] = cn (2n-1) \]

\[ = \Theta(n^2) \]