Divide and Conquer

Recurrence Relations
Divide-and-Conquer

**Strategy:**
- Break up problem into parts.
- Solve each part recursively.
- Combine solutions to sub-problems into the overall solution.
Merge Sort

Mergesort.

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make a sorted whole.

Jon von Neumann (1945)

<table>
<thead>
<tr>
<th>A</th>
<th>L</th>
<th>G</th>
<th>O</th>
<th>R</th>
<th>I</th>
<th>T</th>
<th>H</th>
<th>M</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>L</td>
<td>G</td>
<td>O</td>
<td>R</td>
<td>I</td>
<td>T</td>
<td>H</td>
<td>M</td>
<td>S</td>
</tr>
<tr>
<td>A</td>
<td>G</td>
<td>L</td>
<td>O</td>
<td>R</td>
<td>H</td>
<td>I</td>
<td>M</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
</tbody>
</table>

\[ T(n) \]

- divide \( O(1) \)
- sort \( 2T(n/2) \)
- merge \( O(n) \)
Complexity of merge

**time**

$O(n)$, WHY?

**space**

$O(n)$

Often done using a new size $n$ array stepwise placing next smallest of left and right into the new array

It takes constant time (compare, write, increment indices) to move each of the $n$ elements in its place.

Can you do it in less than $2n$ space?

1.5$n$ still $O(n)$ space
A Recurrence Relation for Merge Sort

\( T(n) = \text{number of comparisons required to mergesort an input of size } n. \)

Mergesort recurrence.

\[
T(n) \leq \begin{cases} 
  c & \text{if } n = 1 \\
  T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + cn & \text{otherwise}
\end{cases}
\]
A recurrence relation for the sequence \( \{a_n\} \) is an equation that expresses \( a_n \) in terms of one of more of the previous terms of the sequence, namely, \( a_0, a_1, \ldots a_{n-1} \), for all integers \( n \) with \( n \geq n_0 \) where \( n_0 \) is a nonnegative integer.

A sequence is defined by a recurrence relation + initial conditions ("base cases")

Example, Merge sort: \[
\begin{align*}
a_n &= 2a_{n/2} + n \\
a_1 &= 1
\end{align*}
\]
A Recurrence Relation for Merge Sort

T(n) = number of comparisons required to mergesort an input of size n.

Mergesort recurrence.

\[
T(n) = \begin{cases} 
  c & \text{if } n = 1 \\
  T(n/2) + T(n/2) + cn & \text{otherwise}
\end{cases}
\]

Solution. \( T(n) = O(n \log_2 n). \)

We assume \( n \) is a power of 2 and replace \( \leq \) with = (we only care about the order of magnitude)
Unrolling the recursion

\[ T(n) = \begin{cases} 
  c & \text{if } n = 1 \\
  2T(n/2) + cn & \text{otherwise} 
\end{cases} \]

by definition of log
Divide and Conquer

```python
def f(A):
    # |A| = n
    if n==1: return base(A)
    else:
        A_1,A_2,...,A_a = split(A)  # |A_i| = n/b
        C_1 = f(A_1)
        C_2 = f(A_2)
        ...
        C_a = f(A_a)
        return combine(C_1,C_2,...,C_a)
```

Time assumptions
- base() takes constant time
- split() + combine() takes \( n^d \) time

Time recurrence
\[
T(n) = a \cdot T(n/b) + n^d
\]
\[
= a \cdot (a \cdot T(n/b^2) + (n/b)^d) + n^d
\]
\[
= a^2 \cdot T(n/b^2) + a \cdot n^d/b^d + n^d
\]
\[
= a^2(aT(n/b^3) + (n/b^2)^d)) + a \cdot n^d/b^d + n^d
\]
\[
= a^3T(n/b^3) + a^2 \cdot n^d/b^{2d} + a \cdot n^d/b^d + n^d
\]
\[...
= a^k(T(n/b^k) + a^{k-1}n^d/b^{(k-1)d} + ... + a \cdot n^d/b^d + n^d
\]

This series ends when \( k = \log_b n \)  \( n == b^{\log_b n} \)

This inductive process is called repeated substitution
Analyzing the process in a stepwise fashion

Each non-leaf node with $T(x)$ represents a call with $a$ children with parameter size $x/b$ (initially $x=n$) and time contribution $x^d$

- The number of nodes at level 0 is 1, at level 1: $a$
  the number of nodes at level $i$ is? $a^i$
- The parameter size of at level 0 is $n$, at level 1: $n/b$
  the parameter size of nodes at level $i$ is? $n/b^i$
- The time contribution at level 0 is $n^d$, at level 1: $n^d/b^d$
  the time contribution of one node at level $i$ is? $n^d/b^{di}$
  the time contribution of all nodes at level $i$ is? $a^i n^d/b^{di} = (a/b^d)^i n^d$
  the time contribution of the whole tree is? $n^d \sum (a/b^d)^i$

remember geometric series: $\sum_{i=0}^{k} r^i = \frac{r^{k+1}-1}{r-1}$ \hspace{1cm} r$\neq 1$

$= k+1 \hspace{1cm} r=1$

Here $r = (a/b^d)$ The bounds of the sum are 0 and the depth of the tree: base case when $n/b^i = 1 \rightarrow b^i=n \rightarrow i = \log_b n$
\[ f(n) = af(n/b) + n^d \]
\[ f(1) = c \quad \leftarrow \text{does not play a role, as we only care about } O \]

Level \( i \): \( a^i \) calls of \( f(n/d^i) \)

Stops when \( n/b^i = 1 \) i.e. \( i = \log_b n \)

\[ n^d \sum_{i=0}^{\log_b n} (a/b^d)^i \]
Three Cases for $r = (a/b^d)$

*Geometric series:* $\sum_{i=0}^{k} r^i = \frac{r^{k+1} - 1}{r - 1}$  Here $r = (a/b^d)$

1. $r < 1$  e.g.  $r = \frac{1}{2}$  $1 + 1/2 + 1/4 + ... < 2$ for any $k$
2. $r = 1$  $\sum_{i=0}^{k} 1^i = k + 1 = O(k)$
3. $r > 1$  e.g.  $r = 2$  $1 + 2 + 4 + 2^k = 2^{k+1} - 1 = O(2^k)$
The three cases in practice

\[ T(n) = 2T(n/2) + n \]  \hspace{1em} // mergesort

\[ r = 1 \quad a=2, \ b=2, \ d=1 \quad r = a/b^d = 1 \quad n^1 \sum_{i=0}^{\log n} 1^i = n (\log n + 1) \]

\[ T(n) = O(n \log n) \]

\[ T(n) = 2T(n/2) + 1 \]  \hspace{1em} // e.g. recursive max in array size \( n \):

if \( n=1 \), then the element is the max.

\[ r > 1 \quad \text{else divide array in 2 halves, find max of each and choose max of the two} \]

\[ a=2, \ b=2, \ d=0 \quad r = a/b^d = 2 \quad n^0 \sum_{i=0}^{\log n} 2^i = (2^{\log n + 1} - 1)/(2-1) = (2n-1)/1 \]

\[ T(n) = O(n) \]

\[ T(n) = 2T(n/2) + n^2 \]

\[ r < 1 \quad a=2, \ b=2, \ d=2 \quad r = - a/b^d = 1/2 \quad n^2 \sum_{i=0}^{\log n} \left(\frac{1}{2}\right)^i = n^2 \left(1+1/2+1/4+\ldots\right) < 2 \ n^2 \]

\[ T(n) = O(n^2) \]

Draw trees for these and do the analysis, \hspace{1em} as in slides 9, 10, 11
The Master Theorem

Let $f$ be an increasing function that satisfies

$$f(n) = a \cdot f(n/b) + c \cdot n^d$$

whenever $n = b^k$, where $k$ is a positive integer, $a \geq 1$, $b$ is an integer $>1$, and $c$ and $d$ are real numbers with $c$ positive and $d$ nonnegative. Then

$$f(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

The Master theorem specifies the big $O$ solution of the geometric series in Slide 11 for $r<1$, $r=1$, $r>1$ $(r = a/b^d)$

Proof: See e.g. Rosen,

Cormen et.al. has a more general version of the Master Theorem
Master Theorem

\[ f(n) = a \cdot f(n/b) + cn^d \]

**Merge Sort:**

\[ f(n) = \begin{cases} 
  O(n^d) & \text{if } a < b^d \\
  O(n^d \log n) & \text{if } a = b^d \\
  O(n^{\log_b a}) & \text{if } a > b^d 
\end{cases} \]

- \( a = 2 \)
- \( b = 2 \)
- \( d = 1 \)
- \( O(n \log n) \)