**Divide & Conquer**

- Solve big problem by solving subproblems
- Recursively calls same function to solve subproblems
- Often divides problem into 2 halves
- Use if subproblems don't repeat.

**Dynamic**

- Stores subproblems in array
- Often divides into n-1, 1 but not always
- Use if subproblem repeats

---

**Sequence Alignment**

- $n, m$
  - $n-1, m + P_{gap}$
  - $n-2, m + P_{gap}$
  - $n-1, m-1 + P_{gap}$
  - $n-2, m-1 + \Delta P_{mm}$

- $\checkmark$ Subproblem repeats!
New Paradigm: Greedy Algorithm: does the thing that looks best right now

Knapsack: Capacity 10,

Items

\begin{align*}
  v_1 &= 6, \quad w_1 = 5 \\
  v_2 &= 7, \quad w_2 = 5 \\
  v_3 &= 10, \quad w_3 = 10
\end{align*}

Greedy algorithm would put the largest value item (\#3) that fits knapsack. Not optimal!

---

Divide & Conquer,
Dynamic Programming

- Solves subproblems
- Difficult to describe algorithm
- Relatively easy to prove correctness (induction)

Greedy Algorithm

- Doesn't solve subproblems
- Easy to describe algorithm (usually)
- Hard to prove correctness (induction, replacement, others...)

Scheduling Greedy Page 2
Scheduling

Suppose you are trying to run many applications on a processor

Java    Python
skype    itunes

What order is best?

Each job $i$, $i \in \{1, \ldots, n\}$
- weight (importance) $w_i$
- time $t_i$ to complete

def: Completion time $C_j$ of job $j$ is sum of times required
to complete all jobs run before $j$, plus $t_j$.

Q: Suppose there are 3 jobs, with $t_1 = 1$, $t_2 = 2$, $t_3 = 3$,
and that they are run in reverse order. What is
$C_1, C_2, C_3$

A) 3, 2, 1   B) 1, 2, 3   C) 3, 6, 7
   D) 3, 5, 6
Scheduling Goal:

\[
\text{Minimize } \sum_{j=1}^{n} w_j c_j \iff \text{"Objective Function"}
\]

Q: Why is this a good goal?
A: If important jobs left until end, \( w_j c_j \rightarrow \text{large} \)

\[
\begin{array}{c|c|c}
\text{job} & \text{time} & \text{weight} \\
1 & 1 & 3 \\
2 & 2 & 2 \\
3 & 3 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{Order} & A \\
123 & \uparrow \\
132 & \uparrow \text{large} \\
213 & \uparrow \text{large} \\
231 & \uparrow \text{large} \\
312 & \uparrow \text{large} \\
321 & \uparrow \text{large} \\
\end{array}
\]
Schedulin

Greedy algorithm: picks best job to schedule first

What is best? Create function \( f(w_i, t_i) \), find job with largest f-value. Put first. Repeat!

Q: Create a good f:
A: Good jobs have large weight, short time

\[
\begin{align*}
&f_1 = \frac{w_i}{t_i} \\
&f_2 = w_i - t_i \\
&f = \frac{w_i}{t_i} + w_i - t_i
\end{align*}
\]

Which to choose? 

Try to find simple examples where behave differently!

<table>
<thead>
<tr>
<th>job</th>
<th>weight</th>
<th>time</th>
<th>( f_1 )</th>
<th>( f_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1/3 ≈ 0.3</td>
<td>-2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2/5</td>
<td>-3</td>
</tr>
</tbody>
</table>

If follow \( f_1 \) \( \rightarrow \) order \((2, 1)\)
If follow \( f_2 \) \( \rightarrow \) order \((1, 2)\)

\[\text{Order} \quad A \]
\[
\begin{array}{c|c}
1 & 2 & 1 \cdot 3 + 2 \cdot 8 = 19 \\
2 & 1 & 2 \cdot 5 + 1 \cdot 8 = 18 \\
\end{array}
\]

\( f_1 \) is better!

In fact, \( f_1 \) is optimal!