Reading assignment
- Read sections 3.1-3.2 of *The ALGORITHM Design Manual*

Some Algorithm Design Techniques, I
- General overall idea
  - Reduce solving a problem to a smaller problem or problems of the same type
- Greedy algorithms
  - Used when one needs to build something a piece at a time
  - Repeatedly make the greedy choice - the one that looks the best right away
  - e.g. closest pair in TSP search
  - Usually fast if they work (but often don't)

Some Algorithm Design Techniques, II
- Divide & Conquer
  - Reduce problem to one or more sub-problems of the same type
  - Typically, each sub-problem is at most a constant fraction of the size of the original problem
    - e.g. Mergesort, Binary Search, Strassen’s Algorithm (we’ll see this later), Quicksort (kind of)

Some Algorithm Design Techniques, III
- Dynamic Programming
  - Give a solution of a problem using smaller sub-problems where all the possible sub-problems are determined in advance
  - Useful when the same sub-problems show up again and again in the solution

A simple case: Computing Fibonacci Numbers
- Recall $F_n = F_{n-1} + F_{n-2}$ and $F_0 = 0, F_1 = 1$
- Recursive algorithm:
  - `Fibo(n)`
    - if $n=0$ then return(0)
    - else if $n=1$ then return(1)
    - else return(`Fibo(n-1)` + `Fibo(n-2)`)
Call tree - start

Full call tree

Memo-ization (Caching)
- Remember all values from previous recursive calls
- Before recursive call, test to see if value has already been computed
- Dynamic Programming
  - Convert memo-ized algorithm from a recursive one to an iterative one

Fibonacci - Dynamic Programming
Version
- FiboDP (n):
  - F[0] ← 0
  - F[1] ← 1
  - for i=2 to n do
    - F[i] = F[i-1] + F[i-2]
  - endfor
  - return (F[n])

Dynamic Programming
- Useful when
  - same recursive sub-problems occur repeatedly
  - Can anticipate the parameters of these recursive calls
  - The solution to whole problem can be figured out with knowing the internal details of how the sub-problems are solved
- Principle of optimality
  - "Optimal solutions to the sub-problems suffice for optimal solution to the whole problem"

List partition problem
- Given: a sequence of n positive integers s_1,...,s_n and a positive integer k
- Find: a partition of the list into up to k blocks:
  - s_1,...,s_i | s_{i+1}...s_j | s_{j+1}...s_k | s_{k+1}...s_n
  - so that the sum of the numbers in the largest block is as small as possible.
  - i.e. find spots for up to k-1 dividers
Greedy approach

- Ideal size would be \( P = \sum_{i=1}^{n} \frac{s_i}{k} \)
- Greedy: walk along until what you have so far adds up to \( P \) then insert a divider
- Problem: it may not be exact (or correct)

```
100  200  400  500  900  700  600  800  600
```

sum is 4800 so if \( k=3 \) size must be at least 1600.

Greedy? Best?

Recursive solution

- Try all possible values for the position of the last divider
- For each position of this last divider
  - there are \( k-2 \) other dividers that must divide the list of numbers prior to the last divider as evenly as possible
    
    \[
    s_1, \ldots, s_i, s_{i+1}, \ldots, s_{i+2}, \ldots, s_{i+k-1}, \ldots, s_n
    \]
  - recursive sub-problem of the same type

Recursive idea

- Let \( M[n,k] \) the smallest cost (size of largest block) of any partition of the first \( n \) #'s into \( k \) pieces.
- If best position for last divider lies between the \( i \) and \( i+1 \)st then
  \[
  M[n,k] = \max \left( M[i,k-1], \sum_{j=i+1}^{n} s_j \right) + \text{cost of last block}
  \]
- In general
  \[
  M[n,k] = \min_{\text{pos}} \max \left( M[i,k-1], \sum_{j=i+1}^{n} s_j \right)
  \]
- Base case(s)?

Time-saving - prefix sums

- Computing the costs of the blocks may be expensive and involved repeated work
- Idea: Pre-compute prefix sums
  - Length of block
    
    \[
    s_{i+1} + \ldots + s_{j}
    \]
    is just
    \[
    p[j] - p[i]
    \]
  - Cost: \( n \) additions

Linear Partition Algorithm

Partition(S, k):

```
p[0] <- 0;
for i=1 to n do p[i] <- p[i-1]+s_i;
for i=1 to n do M[i,1] <- p[i];
for j=1 to k do M[1,j] <- s_1;
for i=2 to n do
  for j=2 to k do
    M[i,j] <- minpos(pos, max(M[pos,j-1], p[i]-p[pos]));
    D[i,j] <- value of pos where min is achieved
```

```
Linear Partition Algorithm

Partition(S, k):

```
p[0] <- 0;
for i=1 to n do p[i] <- p[i-1]+s_i;
for i=1 to n do M[i,1] <- p[i];
for j=1 to k do M[1,j] <- s_1;
for i=2 to n do
  for j=2 to k do
    M[i,j] <- minpos(pos, max(M[pos,j-1], p[i]-p[pos]));
    D[i,j] <- value of pos where min is achieved

```

```
for pos=1 to n do
  for i=1 to n do
    s <- max(M[pos-1,j], p[i]-p[pos]);
    if M[i,j] > s then
      M[i,j] <- s; D[i,j] <- pos
```

### Example:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>300</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>400</td>
<td>700</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td>700</td>
<td>500</td>
</tr>
<tr>
<td>900</td>
<td>2100</td>
<td>1200</td>
<td>900</td>
</tr>
<tr>
<td>700</td>
<td>2800</td>
<td>1600</td>
<td>1200</td>
</tr>
<tr>
<td>600</td>
<td>3400</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>4200</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>4800</td>
<td>2700</td>
<td></td>
</tr>
</tbody>
</table>

Partition(S, k):

- \( g(0) = 0 \),
- for \( k \) to \( m \) do \( g(j) = g(j-1) + s_j \)
- for \( k \to m \) do \( M(1,j) = g(j) \)
- for \( j \to k \) do \( \min \{ \max \{ pos \_j \_1 \}, \max \{ pos \_j \_2 \} \} \)
- \( D(1,j) \) = value of gap at \( j \) where min

### Example:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>300</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>400</td>
<td>700</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td>700</td>
<td>500</td>
</tr>
<tr>
<td>900</td>
<td>2100</td>
<td>1200</td>
<td>900</td>
</tr>
<tr>
<td>700</td>
<td>2800</td>
<td>1600</td>
<td>1200</td>
</tr>
<tr>
<td>600</td>
<td>3400</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>4200</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>4800</td>
<td>2700</td>
<td></td>
</tr>
</tbody>
</table>

Partition(S, k):

- \( g(0) = 0 \),
- for \( k \) to \( m \) do \( g(j) = g(j-1) + s_j \)
- for \( k \to m \) do \( M(1,j) = g(j) \)
- for \( j \to k \) do \( \min \{ \max \{ pos \_j \_1 \}, \max \{ pos \_j \_2 \} \} \)
- \( D(1,j) \) = value of gap at \( j \) where min

### Example:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>300</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>400</td>
<td>700</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td>700</td>
<td>500</td>
</tr>
<tr>
<td>900</td>
<td>2100</td>
<td>1200</td>
<td>900</td>
</tr>
<tr>
<td>700</td>
<td>2800</td>
<td>1600</td>
<td>1200</td>
</tr>
<tr>
<td>600</td>
<td>3400</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>4200</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>4800</td>
<td>2700</td>
<td>2000</td>
</tr>
</tbody>
</table>

Partition(S, k):

- \( g(0) = 0 \),
- for \( k \) to \( m \) do \( g(j) = g(j-1) + s_j \)
- for \( k \to m \) do \( M(1,j) = g(j) \)
- for \( j \to k \) do \( \min \{ \max \{ pos \_j \_1 \}, \max \{ pos \_j \_2 \} \} \)
- \( D(1,j) \) = value of gap at \( j \) where min

---

Example:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>300</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>400</td>
<td>700</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td>700</td>
<td>500</td>
</tr>
<tr>
<td>900</td>
<td>2100</td>
<td>1200</td>
<td>900</td>
</tr>
<tr>
<td>700</td>
<td>2800</td>
<td>1600</td>
<td>1200</td>
</tr>
<tr>
<td>600</td>
<td>3400</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>4200</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>4800</td>
<td>2700</td>
<td></td>
</tr>
</tbody>
</table>