# CST 405
# Algorithm Analysis & Design

**Al Lake**
**Oregon Institute of Technology**
**Chapter 15**
**Dynamic Programming**

# Dynamic Programming

- Dynamic programming is the tabular method of determining the solution to optimization problems.
- A divide-and-conquer method, recursively solves problems by combining the solutions to subproblems.

# Dynamic Programming Algorithm

- **Characterize the structure of an optimal solution**
- **Recursively define the value of an optimal solution**
- **Computer the value of an optimal solution in a bottom-up fashion**
- **Construct an optimal solution from computed information**

- **Steps 1-3 form the basis of a dynamic programming solution to a problem. Step 4 can be omitted if only the value of an optimal solution is required.**

# Dynamic Programming Example

- Assembly-line scheduling
  - The fastest way through the factory
    - Divide into individual steps
  - Recursive solution
    - define the value of an optimal solution recursively in terms of the optimal solutions to subproblems
  - Compute the fastest times
    - Running time of this recursive algorithm is exponential
  - Construct the fastest way through the factory

# Dynamic Programming Elements

- Optimal substructure
  - The first step in solving an optimizing problem is to characterize the structure of an optimal solution.
- Optimal substructure patterns
  - Solution consists of making a choice.
  - Making the choice leaves one or more subproblems to be solved.
  - Assume each choice is optimal
  - Prove optimal by assuming non-optimal and finding a contradiction

# Dynamic Programming Optimization

- **Space of subproblems must be 'small'**
  - **A recursive algorithm for the problem solves the same subproblems over and over, rather than generating new subproblems.**
  - **When a recursive algorithm revisits the same problem over and over again, the optimization problem has overlapping subproblems.**

# Independent and Overlapping

- **Dynamic programming subproblems must be both independent and overlapping.**
- **Two subproblems of the same problem are independent if they do not share resources.**
- **Two subproblems are overlapping if they are really the same subproblem that occurs as a subproblem of different problems, i.e., can be resolved recursively.**

# Memoization

- **An algorithmic technique which saves (memoizes) a computed answer for later reuse, rather than recomputing the answer.**
- **A naive program to compute Fibonacci numbers is:**

```
fib(n) {
        if n is 1 or 2, return 1;
        return fib(n-1) + fib(n-2);
}
```

- **Because fib(n) is recomputed over and over for the same argument, run time is $\Omega(1.6^n)$. If we memoize (save) the value of fib(n) the run time is $\Theta(n)$.**

```
allocate array for memo;
initialize memo[1] and memo[2] to 1;
fib(n) {
        if memo[n] is not zero, return memo[n];
        memo[n] = fib(n-1) + fib(n-2);
        return memo[n];
}
```