What is Algorithm?

Definition

- A process that performs a sequence of operations (wikipedia)
- A series of well-defined instructions to perform a specific task

```plaintext
BottlesOfBeer (x)
for i ← x down to 1
    sing "i bottles of beer on the wall, i bottles of beer,"
    sing "take one down, pass it around, (i-1) bottles of beer on the wall."
    sing "No bottles of beer on the wall, no bottles of beer,"
    sing "Go to the store, buy some more, x bottles of beer on the wall."
```
How to Express Algorithm?

- **Natural Language** (English)
- **Programming Language** (Code)
- **Flow Chart, Diagram**
- **Pseudocode**
  - compact and informal high-level description of algorithms
  - ignoring details in codes, but unambiguous for the task

```plaintext
FIBONACCI(n)
F_1 ← 1
F_2 ← 1
for i ← 3 to n
    F_i ← F_{i-1} + F_{i-2}
return F_n
```

How to Evaluate Algorithm?

- **Correctness**
  - It should work correct on all possible inputs

- **Efficiency**
  - It should run in a reasonable time
  - *Big-O* notation is used
Overview

- Exhaustive Search
- Divide-and-Conquer Algorithm
- Dynamic Programming
- Greedy Algorithm
- Randomized Algorithm
- Graph Algorithm

Exhaustive Search

- **Process**
  - Examine all possible cases to find a solution
  - Also, called brute force search

- **Features**
  - Simple
  - Sometimes, very inefficient because of combinatorial explosion

- **Example**
  - Selection sort

- **Alternatives**
  - Random Sampling
  - Branch and bound algorithm
  - Anti-monotonic property
Selection Sort

- **Algorithm**
  - Iteratively search the smallest one

  ```
  SelectionSort(a, n)
  for i ← 1 to n − 1
    a_j ← smallest one between a_i and a_n
    swap a_i and a_j
  return a
  ```

- **Runtime ?**

Anti-monotonic Property

- **Definition**
  - If a case satisfies a condition, then more general cases always satisfy it
  - If a case violates a condition, then more specific cases always violate it

- **Example**
  - Find maximal sized sets of genes that occur together at least twice

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A, B, C</td>
</tr>
<tr>
<td>20</td>
<td>C, D, F</td>
</tr>
<tr>
<td>30</td>
<td>A, C, E</td>
</tr>
<tr>
<td>40</td>
<td>A, B, C, E</td>
</tr>
</tbody>
</table>
Divide-and-Conquer Algorithm

- **Process**
  1. Recursively splitting the problem into smaller sub-problems
  2. Solve the smallest sub-problem independently
  3. Recursively merging the solutions of sub-problems until having a solution of the original problem

- **Purpose**
  - Improve efficiency

- **Examples**
  - Merge sort
  - Quick sort

---

**Merge Sort**

- **Algorithm**
  1. Recursively divide the array
  2. Recursively combine two arrays in a sorted order

\[
\text{MergeSort}(A[1..n])
\]
\[
\text{if } (n > 1)
\]
\[
m \leftarrow \lfloor n/2 \rfloor
\]
\[
\text{MergeSort}(A[1..m])
\]
\[
\text{MergeSort}(A[m+1..n])
\]
\[
\text{Merge}(A[1..n], m)
\]
\[
i \leftarrow 1; \ j \leftarrow m + 1
\]
\[
\text{for } k \leftarrow 1 \text{ to } n
\]
\[
\text{if } i > m
\]
\[
B[k] \leftarrow A[j]; \ j \leftarrow j + 1
\]
\[
\text{else if } j > n
\]
\[
B[k] \leftarrow A[i]; \ i \leftarrow i + 1
\]
\[
\text{else if } A[i] > A[j]
\]
\[
B[k] \leftarrow A[j]; \ j \leftarrow j + 1
\]
\[
\text{else if } A[i] < A[j]
\]
\[
B[k] \leftarrow A[i]; \ i \leftarrow i + 1
\]
\[
\text{for } k \leftarrow 1 \text{ to } n
\]
\[
A[k] \leftarrow B[k]
\]

---

Runtime ?
Quick Sort

- **Algorithm**
  1. Recursively divide the array based on the pivot
  2. Recursively combine two arrays

  ```
  QuickSort(A[1..n])
  if (n > 1)
      k ← Partition(A, p)
      QuickSort(A[1..k-1])
      QuickSort(A[k+1..n])
  ```

- **Runtime**

Dynamic Programming

- **Process**
  1. Formulate the problem recursively by breaking it down into sub-problems
  2. Build solutions in a linear fashion
     (Repeatedly use the result of a sub-problem to solve the next sub-problem)

- **Features**
  - Optimization (finding an optimal solution)
  - Memoization (storing results of intermediate sub-problems)

- **Examples**
  - Sequence alignment
  - Binary search tree
'Rocks' Game (1)

- **Rule**
  - 2 piles of rocks
  - A player may take either 1 rock (from either pile) or 2 rocks (1 from each)
  - The player who takes the last rock wins

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
</table>
| W | W | W | W | W | W | W | W | W | W | W | W
| L | W | L | W | L | W | L | W | L | W | L | W
| W | W | W | W | W | W | W | W | W | W | W | W
| L | W | L | W | L | W | L | W | L | W | L | L

'W' indicates a win, 'L' indicates a loss.

'Rocks' Game (2)

- **Recursive Formula**

\[
\begin{align*}
R_{i,0} &= W \\
R_{0,j} &= W \\
R_{i,1} &= W \\
R_{i,j} &= L & \text{if } R_{i-1,j} = W \text{ (where } i \geq 1 \text{) and } R_{i,j-1} = W \text{ (where } j \geq 1 \text{) and } R_{i-1,j-1} = W \text{ (where } i \geq 1 \text{ and } j \geq 1 \\
R_{i,j} &= W & \text{Otherwise}
\end{align*}
\]

To solve a problem by dynamic programming, find the *recursive formula* at first!
Binary Search Tree

- **Search Time**
  - Data search time in array?
  - Data search time in binary search tree?

- **Formula**
  \[ S(T) = \sum_{i=1}^{n} (|\text{depth}(T, i)| + 1) \cdot f[i] \]
  \[ |\text{depth}(T, i)| = \begin{cases} 
    \text{depth}(\text{left}(T), i) + 1 & \text{if } i < r \\
    0 & \text{if } i = r \\
    \text{depth}(\text{right}(T), i) + 1 & \text{if } i > r 
  \end{cases} \]

Greedy Algorithm

- **Process**
  1. Determine the optimal structure of a problem
  2. Find the local optimal solution at each step

- **Features**
  - Local optimization

- **Examples**
  - Huffman codes
  - Minimum spanning tree
Money Counting

- **Problem**
  - Count a certain amount of money using the fewest bills and coins

- **Local Optimum Solution**
  - Take the largest bill or coin at each step

- **Examples**
  - $14.27

To solve a problem by a greedy algorithm, find the *local optimum solution* at first!

Scheduling

- **Problem**
  - Assign $m$ jobs into $n$ processors to finish all the jobs as early as possible
  - Suppose $m > n$

- **Local Optimum Solution?**

- **Examples**
  - 9 jobs on 3 processors
    - (runtimes of 9 jobs are 3, 5, 6, 10, 11, 14, 15, 18, and 20 min.)
Huffman Coding (1)

- **Binary Codes**
  - Encoding method
  - Use binary representation (0 and 1)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed-length binary codes</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>variable-length binary codes</td>
<td>0</td>
<td>10</td>
<td>11</td>
<td>100</td>
<td>110</td>
<td>111</td>
</tr>
</tbody>
</table>

- Advantages
  - Fixed-length codes: straightforward
  - Variable-length codes: efficient (encode with lower bits)
  - e.g., ace
  - How to make correct variable-length codes? Make prefix codes!

Huffman Coding (2)

- **Prefix Codes**
  - No codeword is a prefix of any other codeword
  - How to make? Build a binary tree!

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed-length binary codes</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>variable-length binary codes</td>
<td>0</td>
<td>100</td>
<td>101</td>
<td>111</td>
<td>1100</td>
<td>1101</td>
</tr>
</tbody>
</table>

Build a binary tree!
Huffman Coding (3)

- **Problem**
  - Find optimal prefix codes for the most efficient compression
  - Suppose each letter has different frequency

- **Local Optimum Solution?**

- **Examples**
  - a: 45, b: 13, c: 12, d: 16, e: 9, f: 5

Randomized Algorithm

- **Process**
  - Examine random samples to find a solution

- **Features**
  - Simple
  - Probabilistic
  - Sometimes, non-deterministic

  - **Deterministic algorithm:**
    always produce the same solution given a particular input
  - **Non-deterministic algorithm:**
    allows multiple solutions based on an input or random choices
Bolts and Nuts

➢ Problem
   ▪ Among \( n \) nuts, find the nut that matches a given bolt

➢ Expected Number of Comparison?
   ▪ \( T(n) \): number of comparison to find a match for a single bolt out of \( n \) nuts

\[
E[T(n)] = \sum_{k=1}^{n-1} k \cdot Pr[T(n) = k]
\]

\[
Pr[T(n) = k] = \begin{cases} 
1/n & \text{if } k < n - 1 \\
2/n & \text{if } k = n - 1 
\end{cases}
\]

\[
E[T(n)] = \frac{n+1}{2} - \frac{1}{n}
\]

Graph Algorithms (1)

➢ Graph \( G \)
   ▪ An ordered pair \( G(V,E) \) with a set of vertices \( V \) and a set of edges \( E \)

➢ Degree of a Vertex \( v_i \)
   ▪ The number of links from \( v_i \) to other vertices
   ▪ Incoming degree and outgoing degree for directed networks

➢ Adjacent Neighbors \( N(v_i) \) of a Vertex \( v_i \)
   ▪ A set of vertices linked from \( v_i \)

➢ Degree Distribution \( P(k) \)
   ▪ Probability that a vertex has exactly \( k \) links
   ▪ The number of vertices with degree of \( k \) over the total number of vertices
Graph Algorithms (2)

- **Walk**
  - A sequence of vertices such that each vertex is linked to its succeeding one

- **Path**
  - A walk such that each vertex in the walk is distinct

- **Path Length** $|p|$ 
  - The number of edges in the path $p$

- **Shortest Path between $v_i$ and $v_j$, $p_s(v_i,v_j)$**
  - A path with the smallest length among all paths from $v_i$ to $v_j$

- **Characteristic Path Length of a Graph $G$**
  - Average length of the shortest paths between all possible pairs of vertices

- **Diameter of a Graph $G$**
  - Largest length of the shortest paths between all possible pairs of vertices

Graph Algorithms (3)

- **Density of a Graph $G(V,E)$**
  - The number of actual edges over the number of all possible edges
  - $D(G) = 2|E| / |V|(|V|-1)$

- **Clique**
  - A fully connected graph (complete graph) such that $D(G) = 1$

- **Graph Representations**
  - Adjacency list
  - Adjacency matrix

- **Graph Search Algorithms**
  - Breadth-First Search
  - Depth-First Search
Breadth-First Search

- **Features**
  - Exhaustive search of a specific data
  - FIFO queue
  - Computation of shortest path length between two data objects

- **Algorithm**

- **Time Complexity?**

Depth-First Search

- **Features**
  - Search of a specific data in a tree structure
  - Backtracking

- **Algorithm**

- **Time Complexity?**
Questions?

- Lecture Slides are found on the Course Website, web ecs.baylor.edu/faculty/cho/3360