Brute Force

A straightforward approach usually based on problem statement and definitions
Examples:
1. Computing \( a^n (a > 0, n \) a nonnegative integer)
2. Computing \( n! \)
3. Multiply two \( n \) by \( n \) matrices
4. Selection sort
5. Sequential search

String matching

- **Pattern**: a string of \( m \) characters to search for
- **Text**: a (long) string of \( n \) characters to search in

**Brute force algorithm**:
1. Align pattern at beginning of text
2. moving from left to right, compare each character of pattern to the corresponding character in text until
   - all characters are found to match (successful search); or
   - a mismatch is detected
3. while pattern is not found and the text is not yet exhausted, realign pattern one position to the right and repeat step 2.

Brute force string matching – Examples:

1. Pattern: 001011
   Text: 1001011010100101111010

2. Pattern: happy
   Text: It is never too late to have a happy childhood.

Number of comparisons:
Efficiency:

Brute force polynomial evaluation

**Problem**: Find the value of polynomial

\[ p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \]

at a point \( x = x_0 \)

**Algorithm**:

\[
p := 0.0 \\
\text{for } i := n \text{ down to } 0 \text{ do} \\
\quad \text{power} := 1 \\
\quad \text{for } j := 1 \text{ to } i \text{ do} \\
\quad \quad \text{power} := \text{power} \times x \\
\quad \quad p := p + a[i] \times \text{power} \\
\text{return } p
\]

**Efficiency**:
Polynomial evaluation: improvement

We can do better by evaluating from right to left:

Algorithm:

\[
p := a[0] \\
power := 1 \\
for i := 1 to n do \\
power := power * x \\
p := p + a[i] * power \\
return p
\]

Efficiency:

More brute force algorithm examples:

- Closest pair
  - Problem: find closest among \(n\) points in \(k\)-dimensional space
  - Algorithm: Compute distance between each pair of points
  - Efficiency:

- Convex hull
  - Problem: find smallest convex polygon enclosing \(n\) points on the plane
  - Algorithm: For each pair of points \(p_i\) and \(p_j\), determine whether all other points lie to the same side of the straight line through \(p_i\) and \(p_j\)
  - Efficiency:

Brute force strengths and weaknesses

- Strengths:
  - wide applicability
  - simplicity
  - yields reasonable algorithms for some important problems
    - searching
    - string matching
    - matrix multiplication
  - yields standard algorithms for simple computational tasks
    - sum/product of \(n\) numbers
    - finding max/min in a list

- Weaknesses:
  - rarely yields efficient algorithms
  - some brute force algorithms unacceptably slow
  - not as constructive/creative as some other design techniques

Exhaustive search

- A brute force solution to a problem involving search for an element with a special property, usually among combinatorial objects such as permutations, combinations, or subsets of a set.

- Method:
  - construct a way of listing all potential solutions to the problem in a systematic manner
    - all solutions are eventually listed
    - no solution is repeated
  - Evaluate solutions one by one, perhaps disqualifying infeasible ones and keeping track of the best one found so far
  - When search ends, announce the winner

Example 1: Traveling salesman problem

- Given \(n\) cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city.

- Alternatively: Find shortest Hamiltonian circuit in a weighted connected graph.

Example:

```
\begin{align*}
\text{Tour} & \quad \text{Cost} \\
a \rightarrow b \rightarrow c \rightarrow d \rightarrow a & \quad 2+3+7+5 = 17 \\
a \rightarrow b \rightarrow d \rightarrow c \rightarrow a & \quad 2+4+7+8 = 21 \\
a \rightarrow c \rightarrow b \rightarrow d \rightarrow a & \quad 8+3+4+5 = 20 \\
a \rightarrow c \rightarrow d \rightarrow b \rightarrow a & \quad 8+7+4+2 = 21 \\
a \rightarrow d \rightarrow b \rightarrow c \rightarrow a & \quad 5+4+3+8 = 20 \\
a \rightarrow d \rightarrow c \rightarrow b \rightarrow a & \quad 5+7+3+2 = 17
\end{align*}
```

Efficiency:
Given a knapsack with maximum capacity $W$, and a set $S$ consisting of $n$ items

each item $i$ has some weight $w_i$ and benefit value $v_i$ (all $w_i$, $v_i$ and $W$ are integer values)

Problem: How to pack the knapsack to achieve maximum total value of packed items?

Given $n$ items:
- weights: $w_1, w_2, \ldots, w_n$
- values: $v_1, v_2, \ldots, v_n$
- a knapsack of capacity $W$

Find the most valuable subset of the items that fit into the knapsack

Example: $W = 16$

<table>
<thead>
<tr>
<th>item</th>
<th>weight</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>$20</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>$30</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>$50</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>$10</td>
</tr>
</tbody>
</table>

This is a knapsack

Max weight: $W = 20$

<table>
<thead>
<tr>
<th>Items</th>
<th>$W_i$</th>
<th>$V_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Let's first solve this problem with a straightforward algorithm

We go through all combinations and find the one with maximum value and with total weight less or equal to $W$

Example 2: Knapsack Problem

Knapsack by exhaustive search

<table>
<thead>
<tr>
<th>Subset</th>
<th>Total weight</th>
<th>Total value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>2</td>
<td>$20</td>
</tr>
<tr>
<td>(2)</td>
<td>5</td>
<td>$30</td>
</tr>
<tr>
<td>(3)</td>
<td>10</td>
<td>$50</td>
</tr>
<tr>
<td>(4)</td>
<td>5</td>
<td>$10</td>
</tr>
<tr>
<td>(1,2)</td>
<td>7</td>
<td>$50</td>
</tr>
<tr>
<td>(1,3)</td>
<td>12</td>
<td>$70</td>
</tr>
<tr>
<td>(1,4)</td>
<td>7</td>
<td>$30</td>
</tr>
<tr>
<td>(2,3)</td>
<td>15</td>
<td>$80</td>
</tr>
<tr>
<td>(2,4)</td>
<td>10</td>
<td>$40</td>
</tr>
<tr>
<td>(3,4)</td>
<td>15</td>
<td>$60</td>
</tr>
<tr>
<td>(1,2,3)</td>
<td>17</td>
<td>$60</td>
</tr>
<tr>
<td>(1,2,4)</td>
<td>12</td>
<td>$60</td>
</tr>
<tr>
<td>(1,3,4)</td>
<td>17</td>
<td>not feasible</td>
</tr>
<tr>
<td>(2,3,4)</td>
<td>20</td>
<td>not feasible</td>
</tr>
<tr>
<td>(1,2,3,4)</td>
<td>22</td>
<td>not feasible</td>
</tr>
</tbody>
</table>

Efficiency:
Algorithm:
- We go through all combinations and find the one with maximum value and with total weight less or equal to $W$.

Efficiency:
- Since there are $n$ items, there are $2^n$ possible combinations of items.
- Thus, the running time will be $O(2^n)$.

Final comments:
- Exhaustive search algorithms run in a realistic amount of time only on very small instances.
- In many cases there are much better alternatives!
  - Euler circuits
  - Shortest paths
  - Minimum spanning tree
  - Assignment problem
- In some cases exhaustive search (or variation) is the only known solution.