Brute Force

Brute Force

- A straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved

  - Compute a^n, $a > 0$, n a nonnegative integer
  - Consecutive integer checking algorithm for $\gcd(m, n)$
  - Definition-based algorithm for matrix multiplication

Brute-force Sorting Algorithms

- The problem of sorting:
  - Given a list of n items, rearrange them in non-decreasing order.

- Two brute-force solutions:
  - Selection sort
  - Bubble sort

Selection Sort

- Scan the array to find its smallest element and swap it with the first element.
- Then, starting with the second element, scan the elements to the right of it to find the smallest among them and swap it with the second elements.
- Generally, on pass i $(0 \leq i \leq n-2)$, find the smallest element in $A[i..n-1]$ and swap it with $A[i]$:

$A[0] \leq \ . \ . \ . \ \leq A[i\text{-}1]$ | $A[i], \ . \ . \ . \ . \ , A[min], \ . \ . \ ., A[n\text{-}1]$

in their final positions | *the last n-i elements*

Selection Sort

ALGORITHM *SelectionSort*$(A[0..n-1])$

//Sorts a given array by selection sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in ascending order
for $i \leftarrow 0$ **to** $n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[j] < A[min]$ $min \leftarrow j$
    swap $A[i]$ and $A[min]$

Selection Sort

ALGORITHM *SelectionSort*$(A[0..n-1])$

```
//Sorts a given array by selection sort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in ascending order
for i ← 0 to n − 2 do
    min ← i
    for j ← i + 1 to n − 1 do
        if A[j] < A[min]   min ← j
    swap A[i] and A[min]
```

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \frac{(n-1)n}{2} \in \theta(n^2)$$

- Time efficiency: $C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \frac{(n-1)n}{2} \in \Theta(n^2)$
- Space efficiency: n

Bubble Sort

- Compare adjacent elements of the list and exchange them if they are out of order

- Pass i $(0 \leq i \leq n-2)$:

$$A[0], \ldots, A[j] \leftrightarrow A[j+1], \ldots, A[n-i-1] \mid A[n-i] \leq \ldots \leq A[n-1]$$

in their final positions

Bubble Sort

$\boldsymbol{ALGORITHM}\ \ BubbleSort(A[0..n-1])$

$// \, Sorts\ a\ given\ array\ by\ bubble\ sort$

$// \, Input\text{:}\ An\ array\ A[0..n-1]\ of\ orderable\ elements$

$// \, Output\text{:}\ Array\ A[0..n-1]\ sorted\ in\ ascending\ order$

$\boldsymbol{for}\ i\ =\ 0\ \boldsymbol{to}\ n-2\ \boldsymbol{do}$

$\quad \boldsymbol{for}\ j\ =\ 0\ \boldsymbol{to}\ n-2-i\ \boldsymbol{do}$

$\qquad \boldsymbol{if}\ A[j+1] < A[j]\ swap\ A[j]\ and\ A[j+1]$

Bubble Sort

$\boldsymbol{ALGORITHM}\ \ BubbleSort(A[0..n-1])$

$// Sorts\ a\ given\ array\ by\ bubble\ sort$

$// Input{:}\ An\ array\ A[0..n-1]\ of\ orderable\ elements$

$// Output{:}\ Array\ A[0..n-1]\ sorted\ in\ ascending\ order$

$\boldsymbol{for}\ i\ =\ 0\ \boldsymbol{to}\ n-2\ \boldsymbol{do}$

$\quad \boldsymbol{for}\ j\ =\ 0\ \boldsymbol{to}\ n-2-i\ \boldsymbol{do}$

$\qquad \boldsymbol{if}\ A[j+1] < A[j]\ swap\ A[j]\ and\ A[j+1]$

- Time efficiency: $C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \frac{(n-1)n}{2} \in \Theta(n^2)$
- Space efficiency: n

Brute-force Search Algorithms

- **Sequential search**:
  Compares successive elements of a given list with a given search key until either a match is encountered or the list is exhausted without finding a match

Brute-force Search Algorithms

- **String Matching**: find a substring in the text that matches the pattern
  - **pattern:** a string of *m* characters to search for
  - **text:** a (longer) string of *n* characters to search in

Step 1 Align pattern at beginning of text

Step 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)
- a mismatch is detected

Step 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

```
ALGORITHM  BruteForceStringMatch(T[0..n − 1], P[0..m − 1])
    //Implements brute-force string matching
    //Input: An array T[0..n − 1] of n characters representing a text and
    //       an array P[0..m − 1] of m characters representing a pattern
    //Output: The index of the first character in the text that starts a
    //        matching substring or −1 if the search is unsuccessful
    for i ← 0 to n − m do
        j ← 0
        while j < m and P[j] = T[i + j] do
            j ← j + 1
        if j = m return i
    return −1
```

Brute-force String Matching

ALGORITHM *BruteForceStringMatch*($T[0..n-1]$, $P[0..m-1]$)

//Implements brute-force string matching
//Input: An array $T[0..n-1]$ of n characters representing a text and
// an array $P[0..m-1]$ of m characters representing a pattern
//Output: The index of the first character in the text that starts a
// matching substring or -1 if the search is unsuccessful

for $i \leftarrow 0$ **to** $n - m$ **do**
 $j \leftarrow 0$
 while $j < m$ **and** $P[j] = T[i + j]$ **do**
  $j \leftarrow j + 1$
 if $j = m$ **return** i
return -1

Time efficiency: $C_{worst}(n, m) = m(n - m + 1) \in (nm)$

Brute-Force Polynomial Evaluation

Find the value of polynomial

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

Brute-Force Polynomial Evaluation

Find the value of polynomial

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

<u>Brute-force algorithm</u>

```
p ← 0.0
for i = n downto 0 do            //compute terms
    power = 1
    for j = 1 to i do            //compute x^i
        power = power × x
    p = p + a[i] × power
return p
```

Brute-Force Polynomial Evaluation

Find the value of polynomial

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

<u>Brute-force algorithm</u>

$p \leftarrow 0.0$
for $i = n$ **downto** 0 **do** **//compute terms**
  $power = 1$
  for $j = 1$ **to** i **do** **//compute x^i**
    $power = power \times x$
  $p = p + a[i] \times power$
return p

Time efficiency: $C(n) = \sum_{i=0}^{n}(\sum_{j=1}^{i} 1 + 1) = \frac{(n+1)(n+2)}{2} \in \Theta(n^2)$

Closest-Pair Problem

- Find the two closest points in a set of n points (in the two-dimensional Cartesian plane).
- Brute-force algorithm:
  - Compute the distance between every pair of distinct points
  - Return the indexes of the points for which the distance is the smallest.

Closest-Pair Brute-Force Algorithm

ALGORITHM BruteForceClosesestPoints(P)

//Input: A list P of $n(n \geq 2)$ points $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_x)$

//Output: Indicates index1 and index2 of the closest pair of points

$dmin \leftarrow \infty$

for $i \leftarrow 1$ **to** $n - 1$ **do**

  for $j \leftarrow i + 1$ **to** n **do**

    $d \leftarrow sqrt\left(\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2\right)$

    if $d < dmin$

      $dmin \leftarrow d$

      $index1 \leftarrow i$

      $index2 \leftarrow j$

return $index1, index2$

Closest-Pair Brute-Force Algorithm

ALGORITHM BruteForceClosesestPoints(P)

//Input: A list P of $n(n \geq 2)$ points $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_x)$

//Output: Indicates index1 and index2 of the closest pair of points

$dmin \leftarrow \infty$

for $i \leftarrow 1$ **to** $n - 1$ **do**

  for $j \leftarrow i + 1$ **to** n **do**

    $d \leftarrow sqrt\left(\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2\right)$

    if $d < dmin$

      $dmin \leftarrow d$

      $index1 \leftarrow i$

      $index2 \leftarrow j$

return $index1, index2$

Time efficiency: $C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \frac{n(n-1)}{2} \in \Theta(n^2)$

Brute-Force Strengths and Weaknesses

- **Strengths**
  - Wide applicability
  - Simplicity
  - Yields reasonable algorithms for some important problems
  - multiplication, sorting, searching, string matching

- **Weaknesses**
  - Rarely yields efficient algorithms
  - Some brute-force algorithms are unacceptably slow
  - Not as constructive as some other design techniques