# DSE 2256 DESIGN & ANALYSIS OF ALGORITHMS

Lecture 10 & 11

**Brute force Techniques:**
Selection sort, Bubble sort,
Sequential Search,
String Matching

Instructors:

Dr. Savitha G,
Assistant Professor, DSCA, MIT, Manipal

Dr. Abhilash K. Pai,
Assistant Professor, DSCA, MIT, Manipal

# Recap of L8 & L9

- Mathematical analysis of recursive algorithms
  - Recurrence relations
  - Method of backward substitution
  - Algorithm : Factorial of a number
  - Algorithm : Towers of Hanoi

# Brute force

- A straightforward approach, usually based directly on the problem's statement and definitions of the concepts involved.

- Easiest to apply.

- Applicable to a wide variety of problems.

<u>Example:</u>

1. Problem: Cracking a 4-digit PIN.

   What could be the solution using brute force strategy ?

2. Problem: GCD of 2 non-negative integers.

   What could be the solution using brute force strategy ?

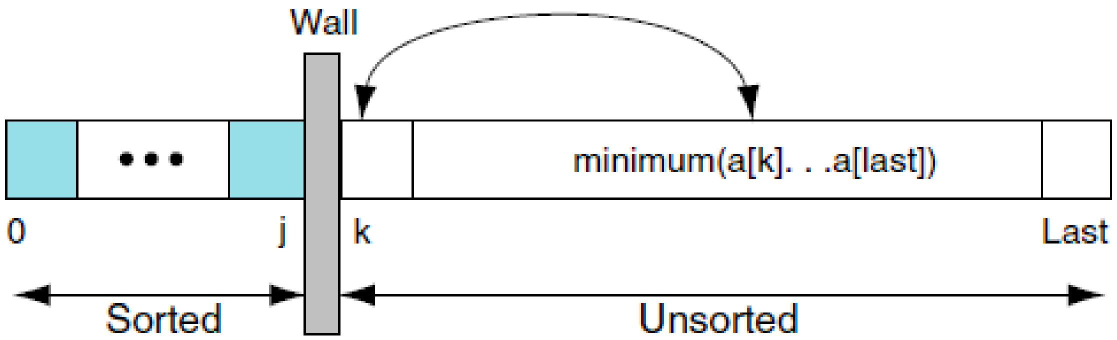
# Brute force Sorting algorithm I

## 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 element.

- Continue this process for $0 \le i \le n\text{-}2$.




iteration 1 – 5 3 4 1 2
min = 1

iteration 2 – 1 3 4 5 2
min = 2

⋮

# Brute force Sorting algorithm I

**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 nondecreasing order
**for** $i \leftarrow 0$ **to** $n-2$ **do**
    $min \leftarrow i$
    **for** $j \leftarrow i+1$ **to** $n-1$ **do**
        **if** $\boxed{A[j] < A[min]}$ $min \leftarrow j$
    swap $A[i]$ and $A[min]$

Basic operation

No. of key swaps $\approx \Theta(n-1)$ (worst case)

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

$$= \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1]$$

$$= \sum_{i=0}^{n-2} (n-1-i)$$

$$\boxed{C(n) = \frac{(n-1)n}{2}} \approx \Theta(n^2)$$

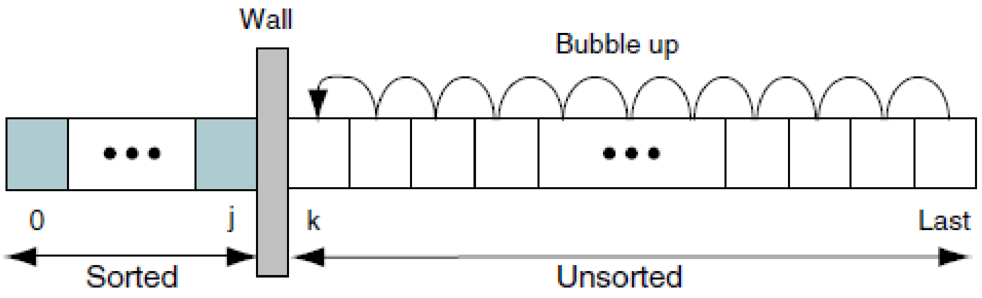
# Brute force Sorting algorithm II

## Bubble **Sort**

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

- By doing it repeatedly, we end up "bubbling up" the largest element to the last position on the list.

- The next pass bubbles up the second largest element, and so on, until after n – 1 passes the list is sorted.




iteration 1 - 8 5 3 1 4 7 9

iteration 2 - 5 8 3 1 4 7 9

iteration 3 - 5 3 8 1 4 7 9

⋮

# Brute force Sorting algorithm II

**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 nondecreasing order
    **for** $i \leftarrow 0$ **to** $n-2$ **do**
        **for** $j \leftarrow 0$ **to** $n-2-i$ **do**
            **if** $\boxed{A[j+1] < A[j]}$ swap $A[j]$ and $A[j+1]$

Basic operation

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 89 | $\overset{?}{\leftrightarrow}$ | 45 | | 68 | | 90 | | 29 | | 34 | | 17 |
| 45 | | 89 | $\overset{?}{\leftrightarrow}$ | 68 | | 90 | | 29 | | 34 | | 17 |
| 45 | | 68 | | 89 | $\overset{?}{\leftrightarrow}$ | 90 | $\overset{?}{\leftrightarrow}$ | 29 | | 34 | | 17 |
| 45 | | 68 | | 89 | | 29 | | 90 | $\overset{?}{\leftrightarrow}$ | 34 | | 17 |
| 45 | | 68 | | 89 | | 29 | | 34 | | 90 | $\overset{?}{\leftrightarrow}$ | 17 |
| 45 | | 68 | | 89 | | 29 | | 34 | | 17 | \| | 90 |
| 45 | $\overset{?}{\leftrightarrow}$ | 68 | $\overset{?}{\leftrightarrow}$ | 89 | $\overset{?}{\leftrightarrow}$ | 29 | | 34 | | 17 | \| | 90 |
| 45 | | 68 | | 29 | | 89 | $\overset{?}{\leftrightarrow}$ | 34 | | 17 | \| | 90 |
| 45 | | 68 | | 29 | | 34 | | 89 | $\overset{?}{\leftrightarrow}$ | 17 | \| | 90 |
| 45 | | 68 | | 29 | | 34 | | 17 | \| | 89 | | 90 |

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

No. of key swaps $\approx \Theta(n^2)$

# Brute force Sequential search

**ALGORITHM** *SequentialSearch*$(A[0..n-1], K)$

//Searches for a given value in a given array by sequential search
//Input: An array $A[0..n-1]$ and a search key $K$
//Output: The index of the first element in $A$ that matches $K$
// or $-1$ if there are no matching elements
$i \leftarrow 0$
**while** $i \cancel{<} n$ **and** $A[i] \neq K$ **do**
  $i \leftarrow i + 1$
**if** $i < n$ **return** $i$
**else return** $-1$

**ALGORITHM** *SequentialSearch2*$(A[0..n], K)$

//Implements sequential search with a search key as a sentinel
//Input: An array $A$ of $n$ elements and a search key $K$
//Output: The index of the first element in $A[0..n-1]$ whose value is
// equal to $K$ or $-1$ if no such element is found
$A[n] \leftarrow K$
$i \leftarrow 0$
**while** $A[i] \neq K$ **do**
  $i \leftarrow i + 1$
**if** $i < n$ **return** $i$
**else return** $-1$

Best case: O (1) , Average case: O (n), Worst case: O (n)

# Brute force String Matching

- Problem: 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.

**Brute-force algorithm**

- 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); or 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

- Problem: find a substring in the text that matches the pattern

**Brute-force algorithm**

- 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); or 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.

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

**Example 1:**

Text: 10010101101001100101111010

Pattern: 001011

**Example 2:**

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

Pattern: happy

# 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
```

Basic operation

$$C_{worst}(n) = m * (n-m+1)$$

$$\approx O(nm)$$

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
          N O T
            N O T
              N O T
```

# Brute force: Strengths and Weaknesses

## Strengths

- Wide applicability
- Simplicity
- Yields reasonable algorithms for some important problems (e.g., matrix multiplication, sorting, searching, string matching)

## Weaknesses

- Rarely yields efficient algorithms
- Some brute-force algorithms are unacceptably slow

# Thank you!

# Any queries?