DESIGN OF ALGORITHM
1.
Brute Force Algorithms Techniques

CS456 - Algorithm Design & Analysis,

Ashesi University

Warm-up Groups Activity – [3 – 5 minutes]

- For this group discussion, do not use the internet/open any book. Brainstorm and come out with a solution. Pseudocode ok
- Write an algorithm to compute x^n
- Write another (better?) algorithm to compute x^n

Brute Force Algorithm Design

- Is a straightforward approach to solving a problem, usually based directly on the problem's statement and definitions of the concepts involved
- Examples:
  - Computing a^n ($a > 0$, n a nonnegative integer)
  - Computing $n!$ **[** n ! = n*(n-1)*(n-2)…2*1**]**
  - Multiplying two matrices **C = A * B**
  - Searching for a(n) key/element of a given value in a list or array

Brute Force Algorithms

- Sequential Search
- Bubble Sort
- Selection Sort

Brute Force Sequential Algorithm

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

$$T(n) = \sum_{0}^{n} 1$$

$$= \underline{\underline{\Theta(n)}}$$

Hence the worst-case scenario for Sequential search is O(n)

Brute force – BubbleSort Algorithm

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

What is the time complexity?

First two iterations of a brute force algorithm - Bubble Sort

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

etc.

Adapted from: A. Levitin "Introduction to the Design & Analysis of Algorithms," 3rd ed., Ch. 2 ©2012 Pearson Education, Inc. Upper Saddle River, NJ.

Analysis of Brute force – BubbleSort Algorithm

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

$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2} \in \Theta(n^2).$$

Hence the time complexity of Bubble sort is $\Theta(n^2)$

Brute Force 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]
```

7 iterations trace of Selection Sort Algorithm

| 89 45 68 90 29 34 **17**

17 | 45 68 90 **29** 34 89

17 29 | 68 90 45 **34** 89

17 29 34 | 90 **45** 68 89

17 29 34 45 | 90 **68** 89

17 29 34 45 68 | 90 **89**

17 29 34 45 68 89 | 90

Adapted from: A. Levitin "Introduction to the Design & Analysis of Algorithms," 3rd ed., Ch. 2 ©2012 Pearson Education, Inc. Upper Saddle River, NJ.

Analysis of Brute force – Selection Sort

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

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

Hence the time complexity of Brute force – Selection Sort is $O(n^2)$

Brute Force - String Matching Algorithm

- *pattern*: a string of *m* characters to search for
- *text*: a (longer) string of *n* characters to search in
- **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

Video on brute force -Pattern Matching

- https://www.youtube.com/watch?v=FL5VXD6BWAU

Examples of Brute-Force String Matching .. 1/4

Example 1 : Search for the pattern **001011** inside the text **10010101101001100101111010**

Pattern: **001011**

Text: **10010101101001100101111010**

1. **Example 2**: Search for the pattern "happy" inside the text "It is never too late to have a happy childhood"

Pattern: happy

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

Brute Force String Matching Example .. 2/4

Text: 100101011010011001011111010
Pattern: 001011

- Iteration 0: 100101011010011001011111010
  001011
- Iteration 1: 100101011010011001011111010
  001011
- Iteration 2: 100101011010011001011111010
  001011
- Iteration 3: 100101011010011001011111010
  001011
- Iteration 4: 100101011010011001011111010
  001011

Brute Force String Matching Example … 3/4

- Iteration 5: 100101011010011001011110100
  001011
- Iteration 6: 100101011010011001011110100
  001011
- Iteration 7: 100101011010011001011110100
  001011
- Iteration 8: 100101011010011001011110100
  001011
- Iteration 9: 100101011010011001011110100
  001011
- Iteration 10: 100101011010011001011110100
  001011

Brute Force String Matching Example … 4/4

- Iteration 11: 10010101101001100101111010
  001011
- Iteration 12: 10010101101001100101111010
  001011
- Iteration 13: 10010101101001100101111010
  001011
- Iteration 14: 10010101101001100101111010
  001011
- Iteration 15: 10010101101001100101111010
  001011

Pseudocode and Time Complexity

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 Complexity is O(nm), why?

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
  - not as constructive as some other design techniques