LECTURE 05 - BRUTE FORCE

Paul Doliotis (PhD)
Adjunct Assistant Professor
Portland State University

What is Brute Force?

- force of the computer, not of your intellect

What is Brute Force?

- force of the computer, not of your intellect
  - = simple & straightforward
  - just do it!

Why Study Them?

- Simple to implement

Why Study Them?

- Simple to implement
- Often “good enough”, especially when n is small

Why Study Them?

- Simple to implement
- Often “good enough”, especially when n is small
- Widely applicable

Why Study Them?

- Simple to implement
- Often “good enough”, especially when n is small
- Widely applicable
- Actually OK for some problems, e.g., Matrix Multiplication

Why Study Them?

- Simple to implement
- Often “good enough”, especially when n is small
- Widely applicable
- Actually OK for some problems, e.g., Matrix Multiplication
- Can be the starting point for an improved algorithm

Why Study Them?

- Simple to implement
- Often “good enough”, especially when n is small
- Widely applicable
- Actually OK for some problems, e.g., Matrix Multiplication
- Can be the starting point for an improved algorithm
- “Baseline” against which we can compare better algorithms

Why Study Them?

- Simple to implement
- Often “good enough”, especially when n is small
- Widely applicable
- Actually OK for some problems, e.g., Matrix Multiplication
- Can be the starting point for an improved algorithm
- “Baseline” against which we can compare better algorithms
- Can be a “gold standard” of correctness

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

- Run animation: https://visualgo.net/bn/sorting

Runtime Analysis of Selection Sort

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

Runtime Analysis of Selection Sort

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

Runtime Analysis of Selection Sort

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

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

Runtime Analysis of Selection Sort

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

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

- $\frac{(n-1)n}{2} \in ?$

Runtime Analysis of Selection Sort

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

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

Runtime Analysis of Selection Sort

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

  $= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2}$
- $\frac{(n-1)n}{2} \in \Theta(n^2)$
- How many swaps?

Runtime Analysis of Selection Sort

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

  $= \sum_{i=0}^{n-2} (n - 1 - i) = \frac{(n-1)n}{2}$
- $\frac{(n-1)n}{2} \in \Theta(n^2)$
- How many swaps?
  - $\Theta(n)$, or more precisely, *n-1*

Runtime Analysis of Selection Sort

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

  $= \sum_{i=0}^{n-2} (n - 1 - i) = \frac{(n-1)n}{2}$
- $\frac{(n-1)n}{2} \in \Theta(n^2)$
- How many swaps?
  - $\Theta(n)$, or more precisely, *n-1*
  - This is an attractive property that distinguishes Selection Sort from other sorting algorithms

Is Selection Sort Stable?

- As sorting algorithm is called **stable** if it preserves the relative order of any two equal emblements.

Is Selection Sort Stable?

- As sorting algorithm is called **stable** if it preserves the relative order of any two equal emblements.
- original list with position i and j with i < j and A[i] == A[j]
  in the sorted list i’ and j’ with i’ < j’ and A[i’] == A[j’]

Is Selection Sort Stable?

- As sorting algorithm is called **stable** if it preserves the relative order of any two equal emblements.
- original list with position i and j with $i < j$ and $A[i] == A[j]$
  in the sorted list i' and j' with $i' < j'$ and $A[i'] == A[j']$
- Why is this a useful property?

Is Selection Sort Stable?

- As sorting algorithm is called **stable** if it preserves the relative order of any two equal emblements.
- original list with position i and j with i < j and A[i] == A[j]
  in the sorted list i' and j' with i' < j' and A[i'] == A[j']
- Why is this a useful property?
  - For sorting records (e.g., record of student names along with their GPA)

Is Selection Sort Stable?

- As sorting algorithm is called **stable** if it preserves the relative order of any two equal emblements.
- original list with position i and j with i < j and A[i] == A[j]
  in the sorted list i’ and j’ with i’ < j’ and A[i’] == A[j’]
- Why is this a useful property?
  - For sorting records (e.g., record of student names along with their GPA)
  - A stable algorithm will yield a list in which students with the same GPA will be sorted alphabetically

Is Selection Sort Stable?

- Is selection sort stable?

Is Selection Sort Stable

- Is selection sort stable?
    - No, it is not stable

Is Selection Sort Stable

- Is selection sort stable?
  - No, it is not stable
  - Prove it

Selection Sort

- Is it possible to implement selection sort for a linked-list with the same $\Theta(n^2)$ efficiency as for an array?

Selection Sort

- Is it possible to implement selection sort for a linked-list with the same $\Theta(n^2)$ efficiency as for an array?

  - Yes, it is possible

Exercise

4. a. Design a brute-force algorithm for computing the value of a polynomial

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

at a given point x_0 and determine its worst-case efficiency class.

Exercise

4. a. Design a brute-force algorithm for computing the value of a polynomial

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

at a given point x_0 and determine its worst-case efficiency class.

b. If the algorithm you designed is in $\Theta(n^2)$, design a linear algorithm for this problem.

Bubble Sort

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 $A[j+1]$

Bubble Sort

- Run animation: https://visualgo.net/bn/sorting

Runtime Analysis of Bubble Sort

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

Runtime Analysis of Bubble Sort

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

Runtime Analysis of Bubble Sort

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

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

Runtime Analysis of Bubble Sort

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

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

Runtime Analysis of Bubble Sort

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

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

- $C(n) \in ?$

Runtime Analysis of Bubble Sort

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

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

Runtime Analysis of Bubble Sort

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

  $= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2}$
- $C(n) \in \Theta(n^2)$
- How many swaps?

Runtime Analysis of Bubble Sort

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

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

- $C(n) \in \Theta(n^2)$
- How many swaps?
  - In the worst case $\frac{(n-1)n}{2} \in \Theta(n^2)$

Bubble Sort

- Is Bubble Sort stable?

Bubble Sort

- Is Bubble Sort stable?
  - YES, only swaps consecutive elements

Bubble Sort

- Is Bubble Sort stable?
  - YES, only swaps consecutive elements
- Prove that, if BubbleSort makes no exchanges on a pass through the array, then the array is sorted.

Bubble Sort

- Is Bubble Sort stable?
  - YES, only swaps consecutive elements
- Prove that, if BubbleSort makes no exchanges on a pass through the array, then the array is sorted.
- Any obvious improvement to the algorithm?

Bubble Sort

- Is Bubble Sort stable?
  - YES, only swaps consecutive elements
- Prove that, if BubbleSort makes no exchanges on a pass through the array, then the array is sorted.
- Any obvious improvement to the algorithm?
  - If a pass through the list makes no exchanges then the list is sorted and the algorithm can exit

Bubble Sort

- Is Bubble Sort stable?
  - YES, only swaps consecutive elements
- Prove that, if BubbleSort makes no exchanges on a pass through the array, then the array is sorted.
- Any obvious improvement to the algorithm?
  - If a pass through the list makes no exchanges then the list is sorted and the algorithm can exit
  - Runs faster on some inputs but average and worst case is still $\Theta(n^2)$

Brute-Force

- Lesson learned: A first application of the brute-force approach often results in an algorithm that can be improved with a modest amount of effort

String Matching

- Find all occurrences of a particular word in a given text
  - Searching for text in an editor
  - …

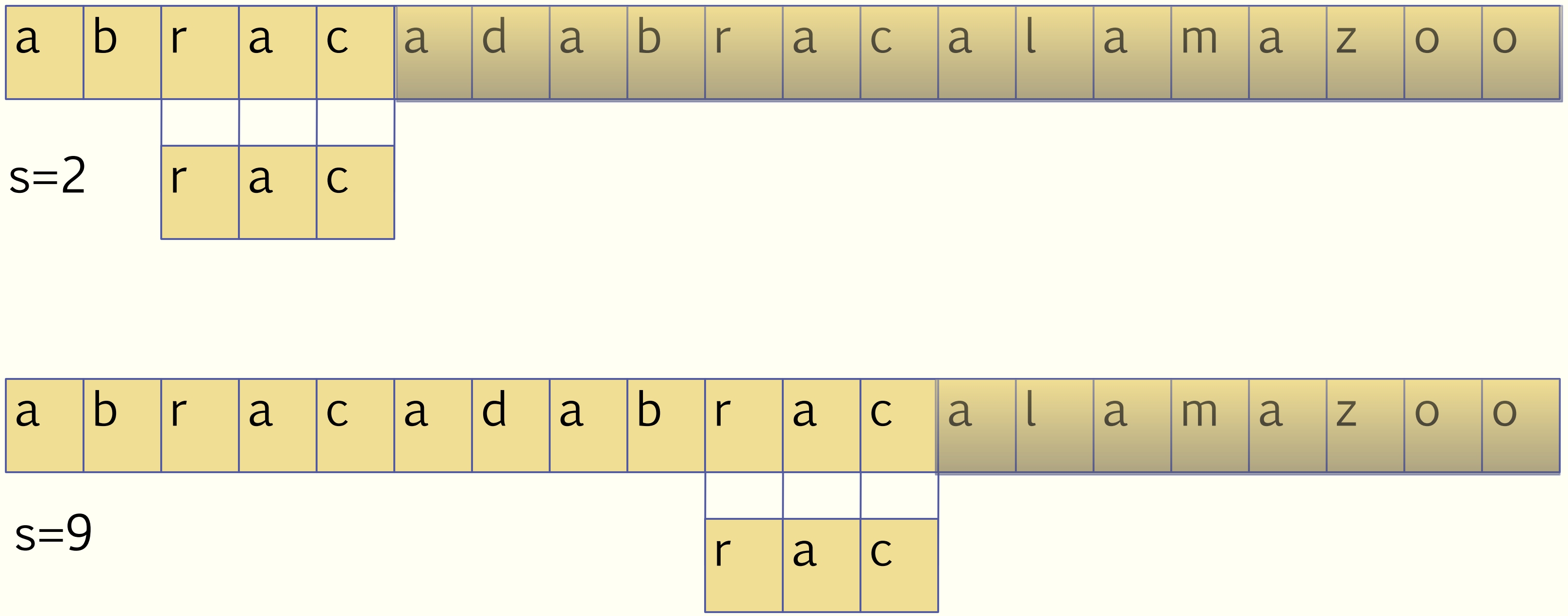
String Matching

- Find all occurrences of a particular word in a given text
  - Searching for text in an editor
  - …

- Compare two strings to see how similar they are to one another …
  - Code diff-ing
  - DNA sequencing
  - …

String Matching, formally

- Given a text string, t, and a pattern string, p, of length m = |p|, find the set of all shifts s such that p = t[s+1..s+m]



Brute-force Matching Algorithm

a	b	r	a	c	a	d	a	b	r	a	c	a	l	a	m	a	z	o	o
r	a	c																	

a	b	r	a	c	a	d	a	b	r	a	c	a	l	a	m	a	z	o	o
	r	a	c																

a	b	r	a	c	a	d	a	b	r	a	c	a	l	a	m	a	z	o	o
		r	a	c															

…

a	b	r	a	c	a	d	a	b	r	a	c	a	l	a	m	a	z	o	o
																	r	a	c

Brute-force Matching Algorithm

a	b	r	a	c	a	d	a	b	r	a	c	a	l	a	m	a	z	o	o
r	a	c																	

a	b	r	a	c	a	d	a	b	r	a	c	a	l	a	m	a	z	o	o
	r	a	c																

What's the asymptotic complexity of brute-force matching?:

A. $\Theta(1)$
B. $\Theta(|t|)$
C. $\Theta(|p|)$
D. $\Theta(|p|(|t|-|p|+1))$
E. None of the above

Brute-force Matching Algorithm

```
match(t, p)
  m ← |p|
  n ← |t|
  results ← {}
  for s ← 0..n-m do
    if p == t[s+1 .. s+m] then
      results ← results ∪ {s}
  return results
```

Asymptotic Complexity:
$\Theta(m(n-m+1))$

Closest-Pair Problem

- Find the two closest points in a set of n points (in the two-dimensional Cartesian plane).

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 and return the indices of the points for which the distance is the smallest.

Closest-Pair Brute-Force Algorithm (cont.)

ALGORITHM *BruteForceClosestPoints*(P)

//Finds two closest points in the plane by brute force
//Input: A list P of n ($n \geq 2$) points $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$
//Output: Indices $index1$ 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**
    $d \leftarrow sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$ //$sqrt$ is the square root function
    if $d < dmin$
      $dmin \leftarrow d$; $index1 \leftarrow i$; $index2 \leftarrow j$
return $index1, index2$

- Efficiency: A: $O(n)$ B: $O(n^2)$ C: $O(\lg n)$ D: $O(n^3)$
- How to make it faster?

Closest-Pair Brute-Force Algorithm (cont.)

ALGORITHM *BruteForceClosestPoints*(P)

```
//Finds two closest points in the plane by brute force
//Input: A list P of n (n ≥ 2) points P1 = (x1, y1), ..., Pn = (xn, yn)
//Output: Indices index1 and index2 of the closest pair of points
dmin ← ∞
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        d ← sqrt((xi − xj)^2 + (yi − yj)^2) //sqrt is the square root function
        if d < dmin
            dmin ← d; index1 ← i; index2 ← j
return index1, index2
```

- Efficiency: A: $O(n)$ B: $O(n^2)$ C: $O(\lg n)$ D: $O(n^3)$
- How to make it faster? Replace sqrt

Closest-Pair Brute-Force Algorithm (cont.)

ALGORITHM *BruteForceClosestPoints*(P)

//Finds two closest points in the plane by brute force
//Input: A list P of n ($n \geq 2$) points $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$
//Output: Indices $index1$ 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**
    $d \leftarrow sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$ //$sqrt$ is the square root function
    if $d < dmin$
      $dmin \leftarrow d$; $index1 \leftarrow i$; $index2 \leftarrow j$
return $index1, index2$

- If sqrt is 10 times slower than +,* how much slower will the algorithm will be?