What is Brute Force?

force of the computer, not of your intellect

= simple & stupid
just do it!
Why study them?

✦ Simple to implement
  suppose you need to solve only one instance?
✦ 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
  use as oracle in unit tests
Sequential Search

**searchFor:** needle
"sequential search for needle. Returns true if found."

```
self do: [:each |
  (each == needle) ifTrue: [ ^true ].
].
^false
```
Sequential Search

**searchFor**: needle

"sequential search for needle. Returns true if found."

```plaintext
self do: [:each |
    (each == needle) ifTrue: [ ^ true ].
].
^ false
```

**searchUsingSentinal**: needle

"sequential search for needle. Returns true if found."

```plaintext
| i |
i ← 1.
[((self at: i) == needle ] whileFalse: [ i ← i + 1 ].
^ (i ~< self size)
```
Sequential Search

`searchFor: needle`
"sequential search for needle. Returns true if found."

```
self do: [:each |
    (each == needle) ifTrue: [ ^true ].
].
^false
```

`searchUsingAt: needle`
"sequential search for needle. Returns true if found."

```
| i sz |
sz ← self size.
i ← 1.
[((self at: i) == needle) | (i = sz) ] whileFalse: [ i ← i + 1 ].
^ (i ~< sz)
```
Sequential Search

**searchUsingAt:** *needle*

"sequential search for needle. Returns true if found."

```plaintext
| i  sz |
sz ← self size.
i ← 1.
[((self at: i) == needle) | (i = sz) ] whileFalse: [ i ← i + 1 ].
↑ (i ~= sz)
```

**searchUsingSentinal:** *needle*

"sequential search for needle. Returns true if found."

```plaintext
| i  |
i ← 1.
[((self at: i) == needle ] whileFalse: [ i ← i + 1 ].
↑ (i ~= self size)
```
Timing Sequential Search

testSequentialSearch

| A B N M res t1 t2 t3 |
N ← 100000.
M ← 500000. "bigger than the array to be searched, and any value in it"
A ← self randomArrayOfSize: N.
t1 ← Time millisecondsToRun: [1000 timesRepeat: [res ← A searchFor: M]].
self deny: res.
B ← A copyWith: M.
t2 ← Time millisecondsToRun: [1000 timesRepeat: [res ← B searchUsingSentinel: M]].
self deny: res.
t3 ← Time millisecondsToRun: [1000 timesRepeat: [res ← A searchUsingAt: M]].
self deny: res.
Transcript show: 'Sequential search, size: '; show: N; cr;
    show: 'sequential, for each: '; show: t1; show: 'µs'; cr;
    show: 'with sentinel: '; show: t2; show: 'µs'; cr;
    show: 'without sentinel, at: '; show: t3; show: 'µs'; cr; cr.
Timing Results

Sequential search, size: 100000
  sequential, for each: 1430μs
  with sentinel: 850μs
  without sentinel, at: 1287μs

Sequential search, size: 100000
  sequential, for each: 1396μs
  with sentinel: 788μs
  without sentinel, at: 1280μs
Timing Results

Sequential search, size: 100000
- sequential, for each: 1430\mu s
- with sentinel: 850\mu s
- without sentinel, at: 1287\mu s

Sequential search, size: 100000
- sequential, for each: 1396\mu s
- with sentinel: 788\mu s
- without sentinel, at: 1280\mu s

Coding details *can* make a difference!
Timing Results

Sequential search, size: 100000
sequential, for each: 1430µs
with sentinel: 850µs
without sentinel, at: 1287µs

Sequential search, size: 100000
sequential, for each: 1396µs
with sentinel: 788µs
without sentinel, at: 1280µs

Coding details *can* make a difference!

But *not* to the asymptotic complexity.
**ALGORITHM**  \( \text{SelectionSort}(A[0..n-1]) \)

//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
  \( \text{min} \leftarrow i \)
  for \( j \leftarrow i + 1 \) to \( n-1 \) do
    if \( A[j] < A[\text{min}] \)  \( \text{min} \leftarrow j \)
  swap \( A[i] \) and \( A[\text{min}] \)
selectionSort

"Sort me using selection sort. Levitin §3.1"

indexOfMin n A |
A ← self.
n ← self size.
1 to: n - 1 do: [ i |
    indexOfMin ← i.
    i + 1 to: n do: [ :j |
        (A at: j) < (A at: indexOfMin) ifTrue: [ |
            indexOfMin ← j]. |
    A swap: i with: indexOfMin ] |

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
            min ← j
    swap A[i] and A[min]
Ex 3.1, Problem 4

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

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

at a given point \( x_0 \) and determine its worst-case efficiency class.
Ex 3.1, Problem 4

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

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

at a given point \( x_0 \) and determine its worst-case efficiency class.
Assume that exponentiation is not built-in.
Ex 3.1, Problem 4

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

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

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

Assume that exponentiation is \textit{not} built-in.

Write it down clearly, so I can project it with the document camera.
Ex 3.1, Problem 4

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

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

at a given point \( x_0 \) and determine its worst-case efficiency class.
Ex 3.1, Problem 4

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

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

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.
Solution to Problem 4

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

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

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

**Algorithm** \textit{BruteForcePolynomialEvaluation}(\( P[0..n] \), \( x \))

//The algorithm computes the value of polynomial \( P \) at a given point \( x \)
//by the “highest-to-lowest term” brute-force algorithm
//Input: Array \( P[0..n] \) of the coefficients of a polynomial of degree \( n \),
//stored from the lowest to the highest and a number \( x \)
//Output: The value of the polynomial at the point \( x \)
\( p \leftarrow 0.0 \)
\( \text{for } i \leftarrow n \text{ downto } 0 \text{ do} \)
\( \quad \text{power} \leftarrow 1 \)
\( \quad \text{for } j \leftarrow 1 \text{ to } i \text{ do} \)
\( \quad \quad \text{power} \leftarrow \text{power} \ast x \)
\( \quad \quad p \leftarrow p + P[i] \ast \text{power} \)
\( \text{return } p \)
Solution to Problem 4

Algorithm \textit{BruteForcePolynomialEvaluation}(P[0..n], x)

// The algorithm computes the value of polynomial P at a given point x
// by the “highest-to-lowest term” brute-force algorithm
// Input: Array P[0..n] of the coefficients of a polynomial of degree n,
// stored from the lowest to the highest and a number x
// Output: The value of the polynomial at the point x

\textit{p} \leftarrow 0.0
for \( i \leftarrow n \) downto 0 do
    \textit{power} \leftarrow 1
    for \( j \leftarrow 1 \) to \( i \) do
        \textit{power} \leftarrow \textit{power} \ast x
        \( p \leftarrow p + P[i] \ast \textit{power} \)
    return \( p \)

- size of input is degree of polynomial, \( n \)
- number of multiplications depends only on \( n \)
- number of multiplications, \( M(n) \in \) ?

A. \( \Theta(n) \)
B. \( \Theta(n^2) \)
C. \( \Theta(n \log n) \)
D. \( \Theta(n^3) \)
Ex 3.1, Problem 4

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

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

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.
Solution to Problem 4

**Algorithm** BetterBruteForcePolynomialEvaluation($P[0..n], x$)

// The algorithm computes the value of polynomial $P$ at a given point $x$
// by the “lowest-to-highest term” algorithm

// Input: Array $P[0..n]$ of the coefficients of a polynomial of degree $n$,
// from the lowest to the highest, and a number $x$

// Output: The value of the polynomial at the point $x$

$p ← P[0]; \quad power ← 1$

for $i ← 1$ to $n$ do
    $power ← power \ast x$
    $p ← p + P[i] \ast power$

return $p$
True or False?

It is possible to design an algorithm with better-than-linear efficiency to calculate the value of a polynomial.

A. True
B. False
Is selection sort stable?

- The definition of a stable sort was given in Levitin §1.3

A. Yes, it is stable
B. No, it is not stable
Ex 3.1, Problem 10

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

A. Yes, it is possible

B. No, it is not possible
**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 ← 0 to n − 2 do
    for j ← 0 to n − 2 − i do
ALGORITHM \texttt{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

• Is \texttt{BubbleSort} stable?
BubbleSort

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 ← 0 to n − 2 do
    for j ← 0 to n − 2 − i do

• Is BubbleSort stable?

A: Yes, it is stable    B: No, it is not stable
BubbleSort

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 ← 0 to n − 2 do
        for j ← 0 to n − 2 − i do

• Is BubbleSort stable?
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 ← 0 to n - 2 do
    for j ← 0 to n - 2 - i do

• Is BubbleSort stable?

• Prove that, if BubbleSort makes no swaps on a pass through the array, then the array is sorted.
String Matching
Applications:

- 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
  - ...

...
Notation

Let A be a set of characters (the alphabet)

The set of strings that consist of finite sequences of characters in A is written $A^*$ (the Kleene Star)

For a string $s$, we’ll write:

- $s[j]$ for the $j^{th}$ character in $s$
- $|s|$ for the length of $s$
- $s[i..j]$ for the substring of $s$ from $s[i]$ to $s[j]$
- $s[..n]$ for the prefix $s[1..n]$, and $s[m..]$ for $s[m..|s|]$
- $\varepsilon$ for the empty string (example: $s[1..0] = \varepsilon$)
- $st$ for the concatenation of $s$ with another string $t$
Simple Complexities:

Assume that string is represented by an array of consecutive characters

What’s the worst case running time for brute-force testing to determine:

- whether \( s = t \)
Simple Complexities:

Assume that string is represented by an array of consecutive characters

What’s the worst case running time for brute-force testing to determine:

- whether $s = t$

A. $O(1)$  
B. $O(|s|)$  
C. $O(|\min(s, t)|)$  
D. $O(|s|^2)$  
E. None of the above
Simple Complexities:

Assume that string s is represented by an array of consecutive characters

- Worst case running time for computing s[i]?
Simple Complexities:

Assume that string s is represented by an array of consecutive characters.

Worst case running time for computing s[i]?

A. $\Theta(1)$
B. $\Theta(|s|)$
C. $\Theta(|\min(|s|, i)|)$
D. $\Theta(i)$
E. None of the above
Simple Complexities:

Assume that strings are represented by arrays of consecutive characters

- Worst case running time for computing \( st \)?
Simple Complexities:

Assume that strings are represented by arrays of consecutive characters.

Worst case running time for computing $st$?

A. $\Theta(1)$
B. $\Theta(|s|)$
C. $\Theta(|\min(s, t)|)$
D. $\Theta(|\min(s, t)|^2)$
E. None of the above
Simple Complexities:

Assume that string is represented by an array of consecutive characters

- Worst case running times for computing $s[i..j]$
Simple Complexities:

Assume that string is represented by an array of consecutive characters

Worst case running times for computing s[i..j]

A. $\Theta(1)$
B. $\Theta(|s[i..j]|)$
C. $\Theta(j-i)$
D. $\Theta((j-i)^2)$
E. None of the above
String Matching

Find all occurrences of a pattern string $p$ in a text string $t$

For example:

```
  a b r a c a d a b r a c a l a m a z o o
  r a c            r a c
```
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
\[
p = t[s+1..s+m]
\]

\[
\begin{array}{cccccccccccccccc}
abracadaba & & & & & & & & & & & & & & & & \\
s=2 & & & & & & & & & & rac & & & & & & \\
abracadaba & & & & & & & & & & & & & & & & \\
s=9 & & & & & & & & & & rac & & & & & & \\
\end{array}
\]
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 |
Brute-force Matching Algorithm

abracadabra calamazoo

rac
Brute-force Matching Algorithm

abracadabraclamazoo
rac

abracadabraclamazoo
rac

abracadabraclamazoo
rac
Brute-force Matching Algorithm

abracadabra calamazoo
rac

abracadabra calamazoo
rac

abracadabra calamazoo
rac

...
Brute-force Matching Algorithm

abraca\ndabra\ncalama\nzoorac

abraca\ndabra\ncalama\nzoorac

abraca\ndabra\ncalama\nzoorac

abraca\ndabra\ncalama\nzoorac

...
Brute-force Matching Algorithm

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

p = 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

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

:\[ t = \text{abraca}d\text{aabrac}alama\text{azo}o \]

\[ p = \text{r} \text{a} \text{c} \]

\[ a\text{braca}\text{daba}bracalama\text{azo}o \]

\[ \text{r} \text{a} \text{c} \]
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

\[
\text{match}(t, p) \\
\text{ } m \leftarrow |p| \\
\text{ } n \leftarrow |t| \\
\text{results} \leftarrow \{\} \\
\text{for } s \leftarrow 0..n-m \text{ do} \\
\text{ } \text{if } p == t[s+1 .. s+m] \text{ then} \\
\text{ } \text{results} \leftarrow \text{results} \cup \{s\} \\
\text{return results}
\]
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: Θ(m(n-m+1))
Brute-force Matching Algorithm

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

Asymptotic Complexity:
\( \Theta(m(n-m+1)) \)
Can we do better?

- Perhaps surprisingly: yes!
- Key insight: when a match fails, we learned something
  - Better algorithms in Chapter 7