Brute Force and Exhaustive Search

Murat Osmanoglu

Brute Force Approach

- a straightforward approach to solving a given problem
(solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved

Brute Force Approach

- a straightforward approach to solving a given problem
(solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem

Brute Force Approach

- a straightforward approach to solving a given problem
(solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem
- may perform more operations than necessary
- typically not considered efficient method of solving a problem

Brute Force Approach

- a straightforward approach to solving a given problem (solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem
- may perform more operations than necessary
- typically not considered efficient method of solving a problem

Exponentiation Problem

- Given a nonzero number A and a nonnegative integer n, compute A^n

Brute Force Approach

- a straightforward approach to solving a given problem
(solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem
- may perform more operations than necessary
- typically not considered efficient method of solving a problem

Exponentiation Problem

- Given a nonzero number A and a nonnegative integer n, compute A^n
  - $A^n = \underbrace{A.A.A....A}_{\text{n times}}$, thus multiply 1 by A n times to compute the output

Brute Force Approach

- a straightforward approach to solving a given problem (solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem
- may perform more operations than necessary
- typically not considered efficient method of solving a problem

Pluses and Minuses

- Given n consecutive integer from 1 to n, devise an algorithm that puts signs "+" and "-" between them so that the expression obtained is equal 0, or if no such expression exists, returns the message "no solution"

Brute Force Approach

- a straightforward approach to solving a given problem
(solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem
- may perform more operations than necessary
- typically not considered efficient method of solving a problem

Pluses and Minuses

- Given n consecutive integer from 1 to n, devise an algorithm that puts signs "+" and "-" between them so that the expression obtained is equal 0, or if no such expression exists, returns the message "no solution"
  - write down all possible combinations to verify whether there is such expression

Brute Force Approach

- a straightforward approach to solving a given problem
(solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem
- may perform more operations than necessary
- typically not considered efficient method of solving a problem

Pluses and Minuses

- Given n consecutive integer from 1 to n, devise an algorithm that puts signs "+" and "-" between them so that the expression obtained is equal 0, or if no such expression exists, returns the message "no solution"
  - write down all possible combinations to verify whether there is such expression ($\approx O(2^n)$)

Brute Force Approach

- a straightforward approach to solving a given problem
(solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem
- may perform more operations than necessary
- typically not considered efficient method of solving a problem

Pluses and Minuses

- *Given n consecutive integer from 1 to n, devise an algorithm that puts signs "+" and "-"* between them so that the expression obtained is equal 0, or if no such expression exists, returns the message "no solution"
  - write down all possible combinations to verify whether there is such expression ($\approx O(2^n)$)
  - for the input 3, 7, 8, 2, 1, 5;

Brute Force Approach

- a straightforward approach to solving a given problem (solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem
- may perform more operations than necessary
- typically not considered efficient method of solving a problem

Pluses and Minuses

- Given n consecutive integer from 1 to n, devise an algorithm that puts signs "+" and "-" between them so that the expression obtained is equal 0, or if no such expression exists, returns the message "no solution"
  - write down all possible combinations to verify whether there is such expression ($\approx O(2^n)$)
  - for the input 3, 7, 8, 2, 1, 5;

3-7-8-2-1-5 = -20, 3-7-8-2-1+5 = -10, 3-7-8-2+1-5 = -18, 3-7-8-2+1+5 = -8, 3-7-8+2-1-5 = -16
3-7-8+2-1+5 = -6, 3-7-8+2+1-5 = -14, 3-7-8+2+1+5 = -4, 3-7+8-2-1-5 = -4, 3-7+8-2-1+5 = 6,
…

Brute Force Approach

- a straightforward approach to solving a given problem (solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does not take advantage of structure or pattern in the problem
- may perform more operations than necessary
- typically not considered efficient method of solving a problem

Pluses and Minuses

- *Given n consecutive integer from 1 to n, devise an algorithm that puts signs "+" and "-" between them so that the expression obtained is equal 0, or if no such expression exists, returns the message "no solution"*
  - write down all possible combinations to verify whether there is such expression ($\approx O(2^n)$)
  - for the input 3, 7, 8, 2, 1, 5;

3-7-8-2-1-5 = -20, 3-7-8-2-1+5 = -10, 3-7-8-2+1-5 = -18, 3-7-8-2+1+5 = -8, 3-7-8+2-1-5 = -16
3-7-8+2-1+5 = -6, 3-7-8+2+1-5 = -14, 3-7-8+2+1+5 = -4, 3-7+8-2-1-5 = -4, 3-7+8-2-1+5 = 6,
…
3+7-8+2+1-5 = 0

Brute Force Approach

- a straightforward approach to solving a given problem (solving a given problem in the most simple, direct, or obvious way)
- directly based on the problem statement and definitions of the concepts involved
- does
- may
- typic

- unlike other design techniques, brute-force can be applied to a very wide variety of problems
- if only a few instances or small-size instances of a problem need to be solved, brute-force can lift the burden of designing more efficient algorithms
- brute-force can serve as a reference point when judging the efficiency of other alternatives

Pluses and

- Given n … d "-" between … exists, returns the message "no solution"
  - write down all possible combinations to verify whether there is such expression ($\approx O(2^n)$)
  - for the input 3, 7, 8, 2, 1, 5;

3-7-8-2-1-5 = -20, 3-7-8-2-1+5 = -10, 3-7-8-2+1-5 = -18, 3-7-8-2+1+5 = -8, 3-7-8+2-1-5 = -16
3-7-8+2-1+5 = -6, 3-7-8+2+1-5 = -14, 3-7-8+2+1+5 = -4, 3-7+8-2-1-5 = -4, 3-7+8-2-1+5 = 6,
…
3+7-8+2+1-5 = 0

Brute Force Approach (Sorting Problem)

- given a sequence of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \le a_2' \le \ldots \le a_n'$

Brute Force Approach (Sorting Problem)

- given a sequence of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \leq a_2' \leq \ldots \leq a_n'$
- Why is sorting worth so much attention ?

Brute Force Approach (Sorting Problem)

- given a sequence of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \le a_2' \le \ldots \le a_n'$
- Why is sorting worth so much attention ?
  - many important techniques have been developed for solving this problem

Brute Force Approach (Sorting Problem)

- given a sequence of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \leq a_2' \leq \ldots \leq a_n'$
- Why is sorting worth so much attention ?
  - many important techniques have been developed for solving this problem
  - the most studied problem in CS

Brute Force Approach (Sorting Problem)

- given a sequence of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \le a_2' \le \ldots \le a_n'$
- Why is sorting worth so much attention ?
  - many important techniques have been developed for solving this problem
  - the most studied problem in CS
  - the algorithms often use sorting as a key subroutine

Brute Force Approach (Sorting Problem)

- given a sequence of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \le a_2' \le \ldots \le a_n'$
- Why is sorting worth so much attention ?
  - many important techniques have been developed for solving this problem
  - the most studied problem in CS
  - the algorithms often use sorting as a key subroutine

  searching; given a set of n items and a separate item, search whether the set contains the given item

Brute Force Approach (Sorting Problem)

- given a sequence of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \le a_2' \le \ldots \le a_n'$
- Why is sorting worth so much attention ?
  - many important techniques have been developed for solving this problem
  - the most studied problem in CS
  - the algorithms often use sorting as a key subroutine

  searching; given a set of n items and a separate item, search whether the set contains the given item

  closest pair; given a set of n numbers, find the pair of numbers that have the smallest possible difference

Brute Force Approach (Sorting Problem)

- given a sequence of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \leq a_2' \leq \ldots \leq a_n'$
- Why is sorting worth so much attention ?
  - many important techniques have been developed for solving this problem
  - the most studied problem in CS
  - the algorithms often use sorting as a key subroutine

  searching; given a set of n items and a separate item, search whether the set contains the given item

  closest pair; given a set of n numbers, find the pair of numbers that have the smallest possible difference

  frequency distribution; given a set of n items, find the frequencies of the items

Brute Force Approach
(Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

Brute Force Approach (Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

Brute Force Approach (Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

  moving the larger elements to the last positions in the sequence (Bubble Sort)

Brute Force Approach
(Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

  moving the larger elements to the last positions in the sequence (Bubble Sort)

Selection Sort

Brute Force Approach
(Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

  moving the larger elements to the last positions in the sequence (Bubble Sort)

Selection Sort

- find the smallest element by scanning the sequence from the first to the last, and exchange it with the first element of the sequence

Brute Force Approach (Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

  moving the larger elements to the last positions in the sequence (Bubble Sort)

Selection Sort

- find the smallest element by scanning the sequence from the first to the last, and exchange it with the first element of the sequence
- find the second smallest element by scanning the sequence from the second to the last, and exchange it with the second element of the sequence

Brute Force Approach
(Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

  moving the larger elements to the last positions in the sequence (Bubble Sort)

Selection Sort

- find the smallest element by scanning the sequence from the first to the last, and exchange it with the first element of the sequence

- find the second smallest element by scanning the sequence from the second to the last, and exchange it with the second element of the sequence

  ⋮ ⋮ ⋮

- find the i-th smallest element by scanning the sequence from the (i+1)-th to the last, and exchange it with the i-th element of the sequence

Brute Force Approach (Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

```
for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if a_j < a_min
            min ← j
    swap a_i and a_min
```

Brute Force Approach (Sorting Problem)

SelectionSort($[a_1, a_2, \dots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
    min ← i
    for $j = i + 1$ to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

S E L E C T I O N

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 1

min ← 1

S E L E C T I O N

j = 2

Brute Force Approach
(Sorting Problem)

SelectionSort($[a_1, a_2, \dots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
    min $\leftarrow$ i
    for $j = i + 1$ to n
        if $a_j < a_{min}$
            min $\leftarrow$ j
    swap a_i and a_{min}

i = 1

min $\leftarrow$ 5

S E L E C T I O N

j = 5

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 1

min ← 5

C	E	L	E	S	T	I	O	N

j = 5

Brute Force Approach
(Sorting Problem)

SelectionSort$([a_1, a_2, \ldots, a_n])$

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 2

min ← 2

C E L E S T I O N

j = 3

Brute Force Approach
(Sorting Problem)

SelectionSort($[a_1, a_2, \ldots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
    min $\leftarrow$ i
    for $j = i + 1$ to n
        if $a_j < a_{min}$
            min $\leftarrow$ j
    swap a_i and a_{min}

$i = 2$

min $\leftarrow$ 2

C E L E S T I O N

$j = 9$

Brute Force Approach
(Sorting Problem)

SelectionSort($[a_1, a_2, \dots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
    min ← i
    for $j = i + 1$ to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

$i = 3$

min ← 3

C E L E S T I O N

$j = 4$

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
  min ← i
  for j = i + 1 to n
    if $a_j < a_{min}$
      min ← j
  swap a_i and a_{min}

i = 3

C E L E S T I O N

min ← 4

j = 4

Brute Force Approach
(Sorting Problem)

SelectionSort($[a_1, a_2, \ldots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 3

min ← 4

C E E L S T I O N

j = 4

Brute Force Approach
(Sorting Problem)

SelectionSort($[a_1, a_2, \ldots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
    $min \leftarrow i$
    for $j = i + 1$ to n
        if $a_j < a_{min}$
            $min \leftarrow j$
    swap a_i and a_{min}

i = 4

C E E L S T I O N

min ← 4

j = 5

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 4

min ← 7

C E E L S T I O N

j = 7

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 4

min ← 7

C E E I S T L O N

j = 7

Brute Force Approach
(Sorting Problem)

SelectionSort([$a_1, a_2, \ldots, a_n$])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 8

min ← 8

C E E I L N O S N

j = 9

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 8

min ← 8

C E E I L N O S N

j = 9

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

Brute Force Approach (Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 8

min ← 8

C E E I L N O S N

j = 9

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

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

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
  min ← i
  for j = i + 1 to n
    if a_j < a_{min}
      min ← j
  swap a_i and a_{min}

i = 8

min ← 8

C E E I L N O S N

j = 9

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

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

Brute Force Approach
(Sorting Problem)

SelectionSort($[a_1, a_2, \dots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n - 1
    min ← i
    for j = i + 1 to n
        if $a_j < a_{min}$
            min ← j
    swap a_i and a_{min}

i = 8

min ← 8

C E E I L N O S N

j = 9

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

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

$$T(n) = n(n-1)/2 \in O(n^2)$$

Brute Force Approach
(Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

  moving the larger elements to the last positions in the sequence (Bubble Sort)

Bubble Sort

Brute Force Approach
(Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

  moving the larger elements to the last positions in the sequence (Bubble Sort)

Bubble Sort

- compare the adjacent elements of the sequence from the first to the last, and exchange them if they are out of order in order to bubble up the largest to the last position

Brute Force Approach (Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

  moving the larger elements to the last positions in the sequence (Bubble Sort)

Bubble Sort

- compare the adjacent elements of the sequence from the first to the last, and exchange them if they are out of order in order to bubble up the largest to the last position
- compare the adjacent elements of the sequence from the first to the second last, and exchange them if they are out of order in order to bubble up the second largest to the second last position

Brute Force Approach (Sorting Problem)

- What would be the most straightforward way of solving sorting problem ?

  moving the smaller elements to the first positions in the sequence (Selection Sort)

  moving the larger elements to the last positions in the sequence (Bubble Sort)

Bubble Sort

- compare the adjacent elements of the sequence from the first to the last, and exchange them if they are out of order in order to bubble up the largest to the last position
- compare the adjacent elements of the sequence from the first to the second last, and exchange them if they are out of order in order to bubble up the second largest to the second last position

$\vdots \qquad \vdots \qquad \vdots$

Brute Force Approach (Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

Brute Force Approach (Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

SELECTION

Brute Force Approach (Sorting Problem)

SelectionSort($[a_1, a_2, \ldots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
  for $j = 1$ to $n - i$
    if $a_{j+1} < a_j$
      swap a_{j+1} and a_j

$\underline{i = 1}$

S E L E C T I O N

$j = 1$

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 1

S	E	L	E	C	T	I	O	N

j = 1

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 1

E	S	L	E	C	T	I	O	N

j = 2

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
    for $j = 1$ to $n - i$
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

$i = 1$

E	L	S	E	C	T	I	O	N

$j = 3$

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 1

E	L	E	S	C	T	I	O	N	

j = 4

Brute Force Approach
(Sorting Problem)

SelectionSort($[a_1, a_2, \ldots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
    for $j = 1$ to $n - i$
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

$i = 1$

E L E C S T I O N

$j = 5$

Brute Force Approach
(Sorting Problem)

SelectionSort($[a_1, a_2, \ldots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
  for $j = 1$ to $n - i$
    if $a_{j+1} < a_j$
      swap a_{j+1} and a_j

$i = 1$

E L E C S T I O N

$j = 6$

Brute Force Approach (Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
  for j = 1 to n – i
    if $a_{j+1} < a_j$
      swap a_{j+1} and a_j

i = 1

E L E C S I T O N

j = 7

Brute Force Approach (Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 1

E	L	E	C	S	I	O	T	N

j = 8

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 1

E	L	E	C	S	I	O	N	**T**

j = 8

Brute Force Approach
(Sorting Problem)

<u>SelectionSort</u><u>([a_1, a_2,…, a_n])</u>

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

<u>i = 2</u>

E L E C S I O N <u>T</u>

j = 1

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
    for $j = 1$ to $n - i$
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

$i = 2$

E	L	E	C	S	I	O	N	T

$j = 1$

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 2

E L E C S I O N T

j = 2

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 2

E E L C S I O N T

j = 3

Brute Force Approach (Sorting Problem)

SelectionSort($[a_1, a_2, \ldots, a_n]$)

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for $i = 1$ to $n - 1$
  for $j = 1$ to $n - i$
    if $a_{j+1} < a_j$
      swap a_{j+1} and a_j

$i = 2$

E E C L S I O N T

$j = 4$

Brute Force Approach (Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 2

E E C L S I O N T

j = 5

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 2

E E C L I S O N T

j = 6

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 2

E E C L I O S N T

j = 7

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 2

E E C L I O N S T

j = 7

Brute Force Approach
(Sorting Problem)

SelectionSort([a_1, a_2,…, a_n])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 8

C	E	E	I	L	O	N	S	T

j = 1

Brute Force Approach
(Sorting Problem)

SelectionSort([$a_1, a_2, \dots, a_n$])

input : a sequence of orderable items
output : sorted sequence in nondecreasing order

for i = 1 to n – 1
    for j = 1 to n – i
        if $a_{j+1} < a_j$
            swap a_{j+1} and a_j

i = 8

C E E I L O N S T

j = 1

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} 1 = \sum_{i=1}^{n-1} (n-i) = n(n-1)/2 \in O(n^2)$$

Brute Force Approach (Searching Problem)

- given a sequence of n items $[a_1, a_2, \ldots, a_n]$ and a search key K, determine whether the sequence contains the search key K

Brute Force Approach
(Searching Problem)

- given a sequence of n items [a_1, a_2,…, a_n] and a search key K, determine whether the sequence contains the search key K

Linear Search

Brute Force Approach (Searching Problem)

- given a sequence of n items [a_1, a_2,…, a_n] and a search key K, determine whether the sequence contains the search key K

Linear Search

- simply compares the items of the sequence with the search key one by one, or terminates without finding a match

Brute Force Approach (Searching Problem)

- given a sequence of n items $[a_1, a_2,\ldots, a_n]$ and a search key K, determine whether the sequence contains the search key K

Linear Search

- simply compares the items of the sequence with the search key one by one, or terminates without finding a match

LinearSearch($[a_1, a_2,\ldots, a_n]$;K)

input : a sequence of n items
output : the index of the element that is equal to K, or 0 if no such element is found

```
for i = 1 to n
     if a_i = K
        return i
return 0
```

Brute Force Approach (Searching Problem)

- given a sequence of n items [a_1, a_2,…, a_n] and a search key K, determine whether the sequence contains the search key K

Linear Search

- simply compares the items of the sequence with the search key one by one, or terminates without finding a match

LinearSearch([a_1, a_2,…, a_n];K)

input : a sequence of n items
output : the index of the element that is equal to K, or 0 if no such element is found

for i = 1 to n
    if a_i = K
        return i
return 0

$$T(n) = \sum_{i=1}^{n} 1 = n \in O(n)$$

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern

  (your algorithm outputs the index of the leftmost character of the first matching the substring if there is any, or 0 if there is no such matching)

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern

(your algorithm outputs the index of the leftmost character of the first matching the substring if there is any, or 0 if there is no such matching)

(the characters in the matching substring must be successive)

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern

 (your algorithm outputs the index of the leftmost character of the first matching the substring if there is any, or 0 if there is no such matching)

 (the characters in the matching substring must be successive)

text   : FEDERICO_FELLINI
pattern : FEL

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern

  (your algorithm outputs the index of the leftmost character of the first matching the substring if there is any, or 0 if there is no such matching)

  (the characters in the matching substring must be successive)

text : FEDERICO_FELLINI
pattern : FEL

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern

  (your algorithm outputs the index of the leftmost character of the first matching the substring if there is any, or 0 if there is no such matching)

  (the characters in the matching substring must be successive)

- find all the substrings of length m in the text, and check whether any of them matches the pattern

  text : FEDERICO_FELLINI
  pattern : FEL

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern

  (your algorithm outputs the index of the leftmost character of the first matching the substring if there is any, or 0 if there is no such matching)

  (the characters in the matching substring must be successive)

- find all the substrings of length m in the text, and check whether any of them matches the pattern

  text : FEDERICO_FELLINI
  pattern : FEL

  FED, EDE, DER, ERI, RIC, ICO, CO_, O_F, _FE, FEL, ELL, LLI, LIN, INI

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern

  (your algorithm outputs the index of the leftmost character of the first matching the substring if there is any, or 0 if there is no such matching)

  (the characters in the matching substring must be successive)

- find all the substrings of length m in the text, and check whether any of them matches the pattern

  text : FEDERICO_FELLINI
  pattern : FEL

  FED, EDE, DER, ERI, RIC, ICO, CO_, O_F, _FE, FEL, ELL, LLI, LIN, INI

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern

  (your algorithm outputs the index of the leftmost character of the first matching the substring if there is any, or 0 if there is no such matching)

  (the characters in the matching substring must be successive)

- find all the substrings of length m in the text, and check whether any of them matches the pattern

  text : FEDERICO_FELLINI
  pattern : FEL

  FED, EDE, DER, ERI, RIC, ICO, CO_, O_F, _FE, FEL, ELL, LLI, LIN, INI

  output : 10

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2\ldots t_n$, $P = p_1p_2\ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

for $i = 1$ to $n - m + 1$
    $j \leftarrow 1$
    while $j < m + 1$ and $p_j = t_{i+j-1}$
        $j \leftarrow j+1$
    if $j = m + 1$
        return i
return 0

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2 \ldots t_n$, $P = p_1p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

```
for i = 1 to n – m + 1
    j ← 1
    while j < m + 1 and p_j = t_{i+j-1}
        j ← j+1
    if j = m + 1
        return i
return 0
```

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

$i = 1$

$j = 1$

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2 \ldots t_n$, $P = p_1p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

```
for i = 1 to n – m + 1
    j ← 1
    while j < m + 1 and p_j = t_(i+j-1)
        j ← j+1
    if j = m + 1
        return i
return 0
```

FEDERICO_FELLINI n = 16

FEL m = 3

i = 1

j = 1

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2 \ldots t_n$, $P = p_1p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

for $i = 1$ to $n - m + 1$
  $j \leftarrow 1$
  while $j < m + 1$ and $p_j = t_{i+j-1}$
    $j \leftarrow j+1$
  if $j = m + 1$
    return i
return 0

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

$i = 1$

$j = 2$

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2\ldots t_n$, $P = p_1p_2\ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

```
for i = 1 to n – m + 1
    j ← 1
    while j < m + 1 and p_j = t_{i+j-1}
        j ← j+1
    if j = m + 1
        return i
return 0
```

FEDERICO_FELLINI   n = 16

FEL   m = 3

i = 1

j = 2

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1 t_2 \ldots t_n$, $P = p_1 p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

for $i = 1$ to $n - m + 1$
    $j \leftarrow 1$
    while $j < m + 1$ and $p_j = t_{i+j-1}$
        $j \leftarrow j+1$
    if $j = m + 1$
        return i
return 0

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

$i = 1$

$j = 3$

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2 \ldots t_n$, $P = p_1p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

```
for i = 1 to n – m + 1
    j ← 1
    while j < m + 1 and p_j = t_(i+j-1)
        j ← j+1
    if j = m + 1
       return i
return 0
```

i = 1

j = 3

FEDERICO_FELLINI n = 16

FEL m = 3

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2...t_n$, $P = p_1p_2...p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

```
for i = 1 to n – m + 1
    j ← 1
    while j < m + 1 and p_j = t_{i+j-1}
        j ← j+1
    if j = m + 1
        return i
return 0
```

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

$i = 2$

$j = 1$

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2 \ldots t_n$, $P = p_1p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

```
for i = 1 to n – m + 1
    j ← 1
    while j < m + 1 and p_j = t_(i+j-1)
        j ← j+1
    if j = m + 1
        return i
return 0
```

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

$i = 2$

$j = 1$

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2 \ldots t_n$, $P = p_1p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

for $i = 1$ to $n - m + 1$
    $j \leftarrow 1$
    while $j < m + 1$ and $p_j = t_{i+j-1}$
        $j \leftarrow j+1$
    if $j = m + 1$
        return i
return 0

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

$i = 3$

$j = 1$

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2 \ldots t_n$, $P = p_1p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

for $i = 1$ to $n - m + 1$
    $j \leftarrow 1$
    while $j < m + 1$ and $p_j = t_{i+j-1}$
        $j \leftarrow j+1$
    if $j = m + 1$
        return i
return 0

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

$i = 10$

$j = 1$

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2...t_n$, $P = p_1p_2...p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

for $i = 1$ to $n - m + 1$
    $j \leftarrow 1$
    while $j < m + 1$ and $p_j = t_{i+j-1}$
        $j \leftarrow j+1$
    if $j = m + 1$
        return i
return 0

FEDERICO_FELLINI   n = 16

FEL   m = 3

i = 10

j = 2

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2 \ldots t_n$, $P = p_1p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

for $i = 1$ to $n - m + 1$
    $j \leftarrow 1$
    while $j < m + 1$ and $p_j = t_{i+j-1}$
        $j \leftarrow j+1$
    if $j = m + 1$
        return i
return 0

FEDERICO_FELLINI   n = 16

FEL   m = 3

i = 10

j = 3

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2\ldots t_n$, $P = p_1p_2\ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

for $i = 1$ to $n - m + 1$
    $j \leftarrow 1$
    while $j < m + 1$ and $p_j = t_{i+j-1}$
        $j \leftarrow j+1$
    if $j = m + 1$
        return i
return 0

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

$i = 10$

$j = 4$

Brute Force Approach
(Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2\ldots t_n$, $P = p_1p_2\ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

for $i = 1$ to $n - m + 1$
    $j \leftarrow 1$
    while $j < m + 1$ and $p_j = t_{i+j-1}$
        $j \leftarrow j+1$
    if $j = m + 1$
        return i
return 0

$i = 10$

$j = 4$

FEDERICO_FELLINI $n = 16$

FEL $m = 3$

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2 \ldots t_n$, $P = p_1p_2 \ldots p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

```
for i = 1 to n – m + 1
    j ← 1
    while j < m + 1 and p_j = t_{i+j-1}
        j ← j+1
    if j = m + 1
        return i
return 0
```

Brute Force Approach (Searching Problem)

String Matching

- Given a string of n characters (text) and a string of m characters (pattern), determine whether the text has a substring that matches the pattern
- find all the substrings of length m in the text, and check whether any of them matches the pattern

StringMatching($T = t_1t_2...t_n$, $P = p_1p_2...p_m$)

input : a text with n characters and a pattern with m characters

output : outputs the index of the leftmost character of the first matching substring, or 0 if no such substring exists

```
for i = 1 to n – m + 1
    j ← 1
    while j < m + 1 and p_j = t_{i+j-1}
        j ← j+1
    if j = m + 1
        return i
return 0
```

$$T(n) = (n - m + 1).m = O(nm)$$

Exhaustive Search

- a brute force approach involving search for a solution with a special property or constraint by exploring every possible combination from a set of choices or values

Exhaustive Search

- a brute force approach involving search for a solution with a special property or constraint by exploring every possible combination from a set of choices or values
  - create all possible solutions in a systematic manner
  - evaluate potential solutions one by one, remove infeasible ones and keep track of the best one found so far
  - when there is no more to evaluate, output the solution found

Exhaustive Search

- a brute force approach involving search for a solution with a special property or constraint by exploring every possible combination from a set of choices or values
  - create all possible solutions in a systematic manner
  - evaluate potential solutions one by one, remove infeasible ones and keep track of the best one found so far
  - when there is no more to evaluate, output the solution found

Knapsack Problem

- given a knapsack of capacity W, and n items so that each of them has a weight and value pair (w_i, p_i), find the most valuable subset of the items that fit into knapsack

Exhaustive Search

- a brute force approach involving search for a solution with a special property or constraint by exploring every possible combination from a set of choices or values
  - create all possible solutions in a systematic manner
  - evaluate potential solutions one by one, remove infeasible ones and keep track of the best one found so far
  - when there is no more to evaluate, output the solution found

Knapsack Problem

- given a knapsack of capacity W, and n items so that each of them has a weight and value pair (w_i, p_i), find the most valuable subset of the items that fit into knapsack

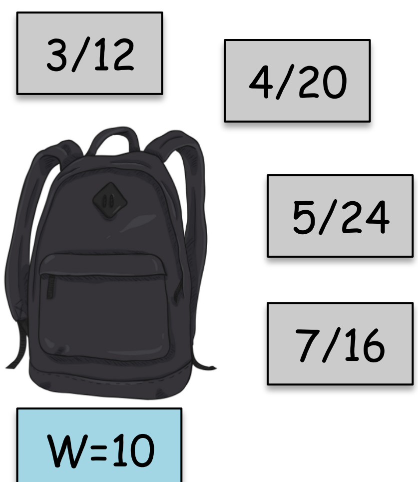
W=10

Exhaustive Search

- a brute force approach involving search for a solution with a special property or constraint by exploring every possible combination from a set of choices or values
  - create all possible solutions in a systematic manner
  - evaluate potential solutions one by one, remove infeasible ones and keep track of the best one found so far
  - when there is no more to evaluate, output the solution found

Knapsack Problem

- given a knapsack of capacity W, and n items so that each of them has a weight and value pair (w_i, p_i), find the most valuable subset of the items that fit into knapsack

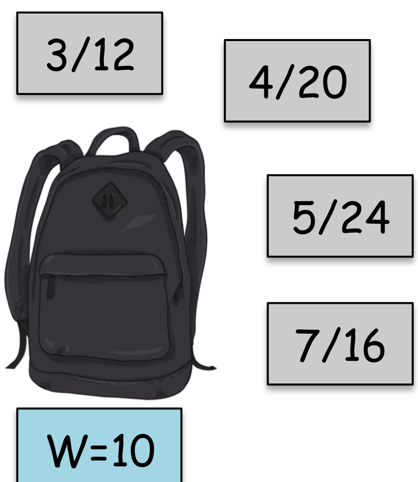


Exhaustive Search

- a brute force approach involving search for a solution with a special property or constraint by exploring every possible combination from a set of choices or values
  - create all possible solutions in a systematic manner
  - evaluate potential solutions one by one, remove infeasible ones and keep track of the best one found so far
  - when there is no more to evaluate, output the solution found

Knapsack Problem

- given a knapsack of capacity W, and n items so that each of them has a weight and value pair (w_i, p_i), find the most valuable subset of the items that fit into knapsack



subset	total weight	total value
{}	0	0
{1}	3	12
{2}	4	20
{3}	5	24
{4}	7	16
{1,2}	7	32
{1,3}	8	36
{1,4}	10	28

Exhaustive Search

- a brute force approach involving search for a solution with a special property or constraint by exploring every possible combination from a set of choices or values
  - create all possible solutions in a systematic manner
  - evaluate potential solutions one by one, remove infeasible ones and keep track of the best one found so far
  - when there is no more to evaluate, output the solution found

Knapsack Problem

- given a knapsack of capacity W, and n items so that each of them has a weight and value pair (w_i, p_i), find the most valuable subset of the items that fit into knapsack

3/12

4/20

5/24

7/16

W=10

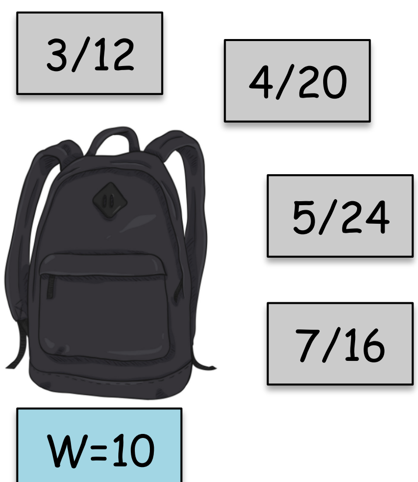
subset	total weight	total value	subset	total weight	total value
{}	0	0	{2,3}	9	44
{1}	3	12	{2,4}	11	no
{2}	4	20	{3,4}	12	no
{3}	5	24	{1,2,3}	12	no
{4}	7	16	{1,2,4}	14	no
{1,2}	7	32	{1,3,4}	15	no
{1,3}	8	36	{2,3,4}	16	no
{1,4}	10	28	{1,2,3,4}	19	no

Exhaustive Search

- a brute force approach involving search for a solution with a special property or constraint by exploring every possible combination from a set of choices or values
  - create all possible solutions in a systematic manner
  - evaluate potential solutions one by one, remove infeasible ones and keep track of the best one found so far
  - when there is no more to evaluate, output the solution found

Knapsack Problem

- *given a knapsack of capacity W, and n items so that each of them has a weight and value pair (w_i, p_i), find the most valuable subset of the items that fit into knapsack*



subset	total weight	total value	subset	total weight	total value
{}	0	0	{2,3}	9	44
{1}	3	12	{2,4}	11	no
{2}	4	20	{3,4}	12	no
{3}	5	24	{1,2,3}	12	no
{4}	7	16	{1,2,4}	14	no
{1,2}	7	32	{1,3,4}	15	no
{1,3}	8	36	{2,3,4}	16	no
{1,4}	10	28	{1,2,3,4}	19	no

Exhaustive Search
(Graph Traversal)

- can be applied to graph traversal algorithms that exhaustively search a graph

Exhaustive Search (Graph Traversal)

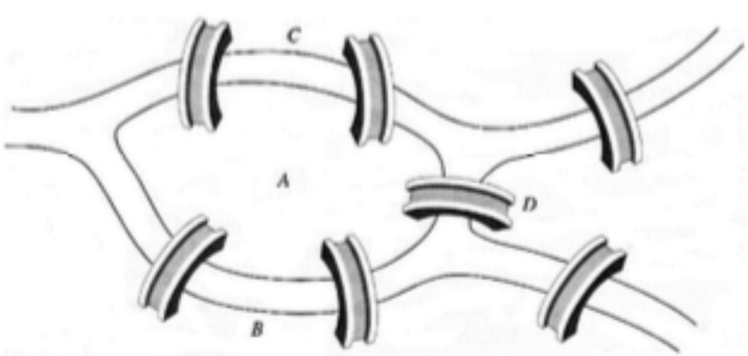
- can be applied to graph traversal algorithms that exhaustively search a graph
  - graph traversal algorithms systematically explore every vertex and every edge of a graph

Exhaustive Search
(Graph Traversal)

- can be applied to graph traversal algorithms that exhaustively search a graph
  - graph traversal algorithms systematically explore every vertex and every edge of a graph

Graph Theory

- Königsberg was a city in Germany in 18th century. There was a river Pregel that divide the city into four distinct regions



Is it possible to take a walk around the city that passes each bridge exactly once ?