AI VIETNAM
All-in-One Course

Algorithm design and analysis

Brute Force – Exhaustive Search

Nguyen Quoc Thai

Year 2023

CONTENT

(1) – Brute Force

Searching Problem: Linear Search
Sorting Problem: Selection Sort, Bubble Sort

(2) – Exhaustive Search

Travelling Salesman Problem

1 – Brute Force

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
- Not considered efficient method of solving a problem

1 – Brute Force

Brute Force Approach

- **Example: Exponentiation Problem**

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

$$A^n = \underbrace{A.A\ldots A.A}_{\text{n times}}$$

=> Multiply 1 by A *n* times to compute the output

- **Example: Computing n! by repeated multiplication**

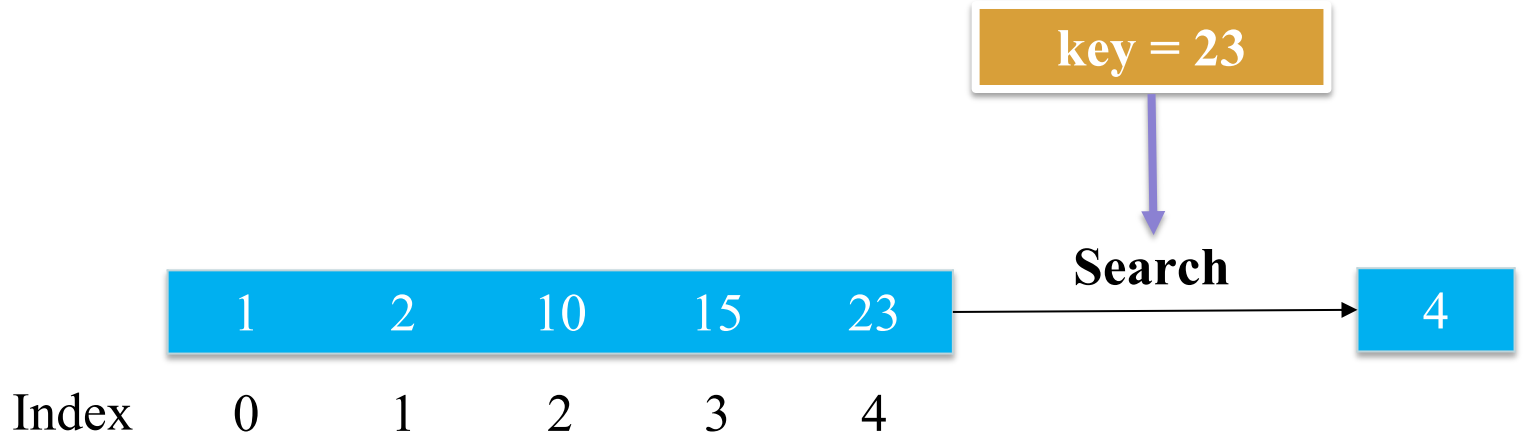
$$n! = \underbrace{(n).(n-1)\ldots 1}_{\text{n times}}$$

=> Repeated multiplication *n* times

1 – Brute Force

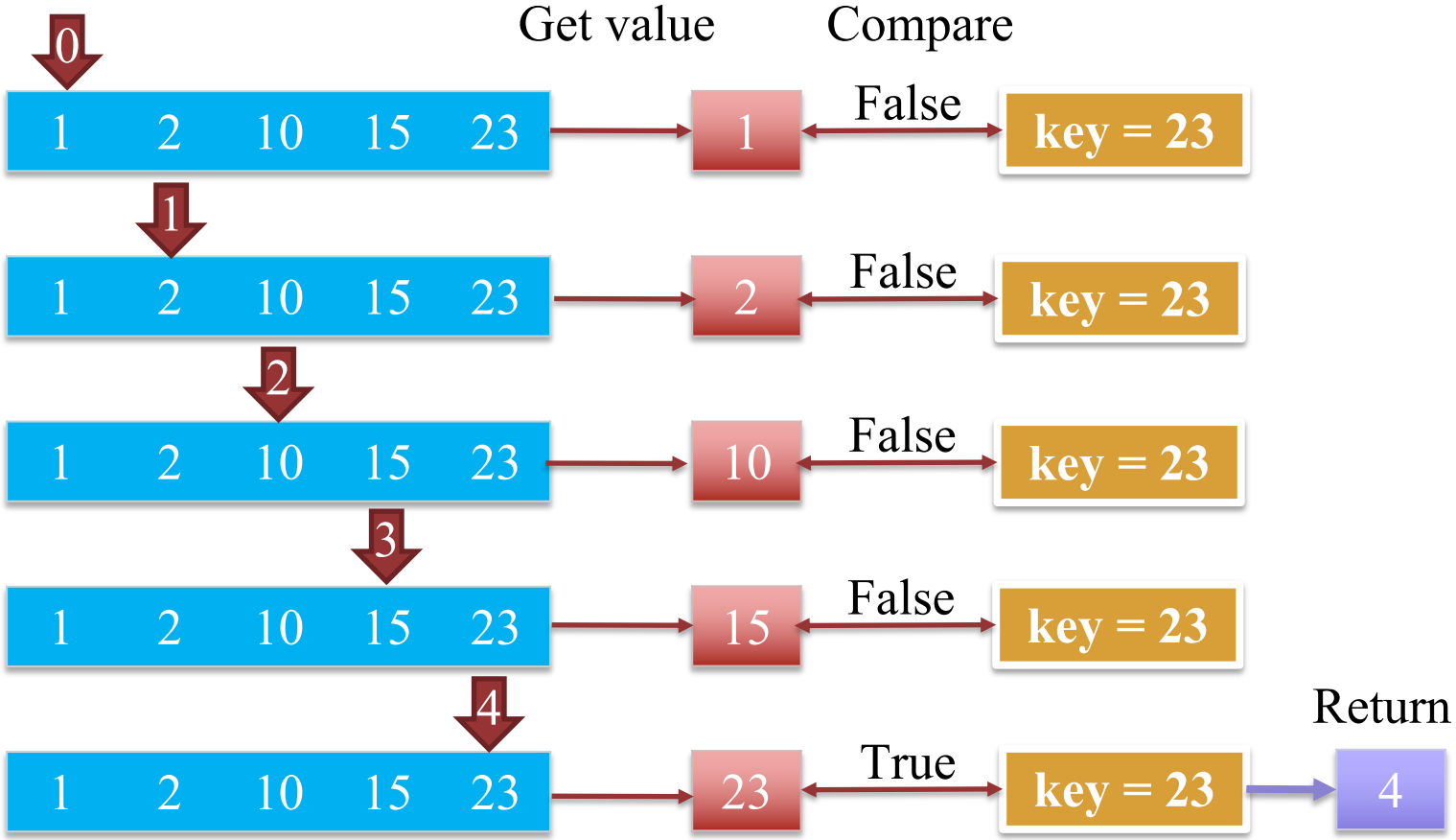
1.1. Searching problem

- Input: a **sorted sequence** of n integer number $<a_1, a_2, \ldots, a_n>$ and *key*
- Output: index of *key* in the sequence if exist, -1 if not exist



1 – Brute Force

1.1. Searching problem



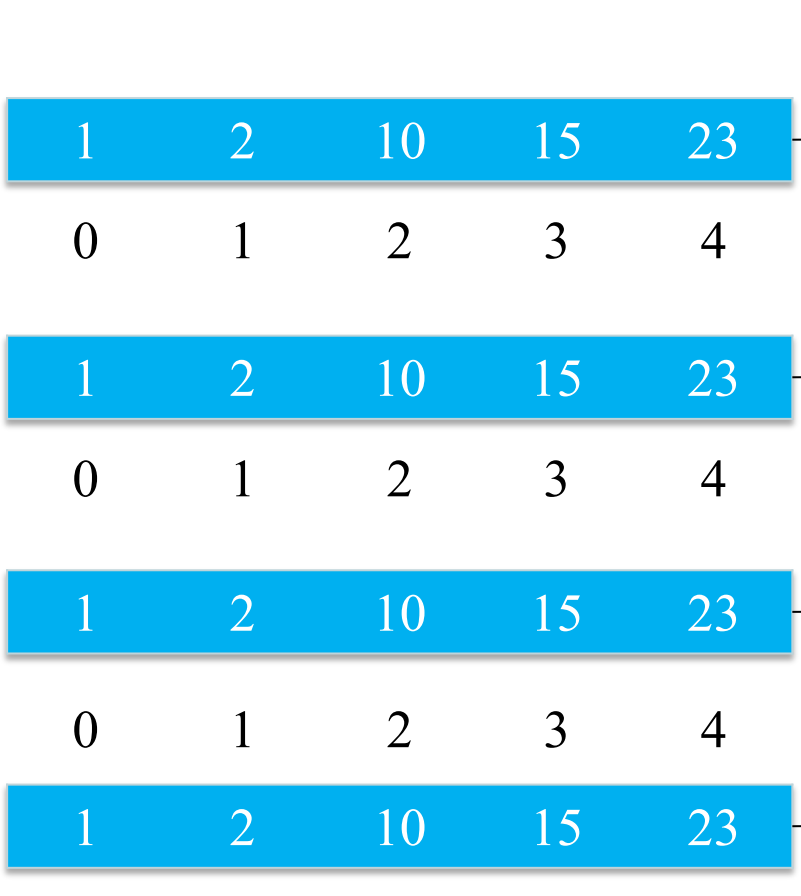
Python Source

```
[15] def linear_search(arr, key):

        n = len(arr)
        for idx in range(n):

            element = arr[idx]
            if element == key:
                return idx

        return -1

    arr = [1, 2, 10, 15, 23]
    key = 23
    linear_search(arr, key)
```

4

1 – Brute Force

1.1. Searching problem



Python Source

```
[15] def linear_search(arr, key):

         n = len(arr)
         for idx in range(n):

             element = arr[idx]
             if element == key:
                 return idx

         return -1

arr = [1, 2, 10, 15, 23]
key = 23
linear_search(arr, key)
```

4

1 – Brute Force

1.2. Sorting problem

- Input: a **sequence** of n integer number $<a_1, a_2, \ldots, a_n>$
- Output: a permutation (reordering)) $<a'_1, a'_2, \ldots, a'_n>$ such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

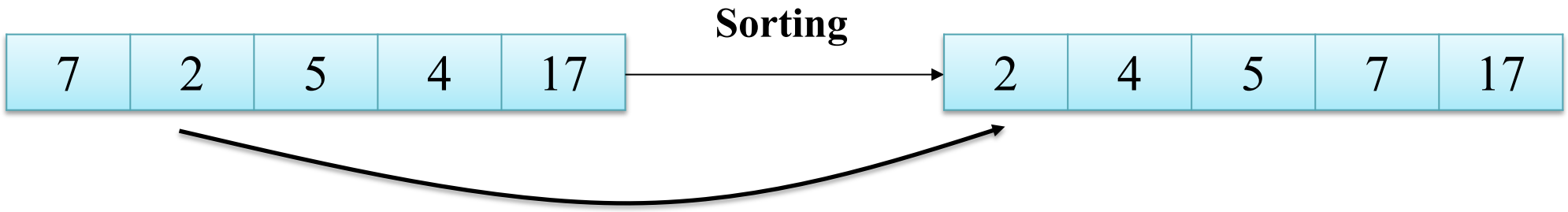
1 – Brute Force

1.2. Sorting problem

- Input: a **sequence** of n integer number $<a_1, a_2, \ldots, a_n>$
- Output: a permutation (reordering)) $<a'_1, a'_2, \ldots, a'_n>$ such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Straightforward way of solving sorting problem

- Moving the smaller elements to the first positional in the sequence (Selection Sort)



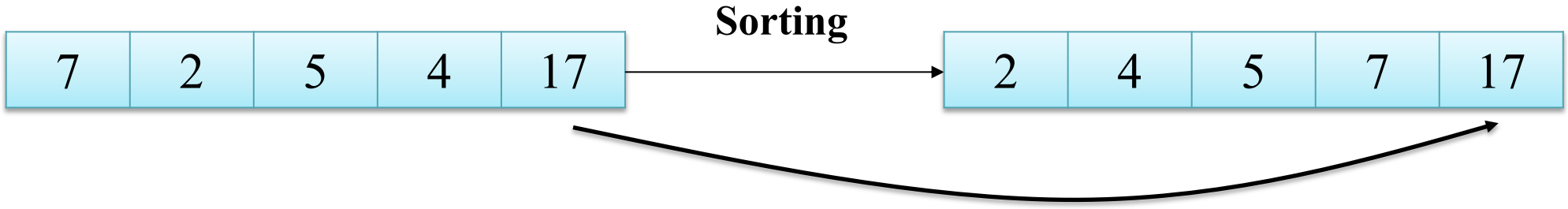
1 – Brute Force

1.2. Sorting problem

- Input: a **sequence** of n integer number $<a_1, a_2, \ldots, a_n>$
- Output: a permutation (reordering)) $<a'_1, a'_2, \ldots, a'_n>$ such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Straightforward way of solving sorting problem

- Moving the larger elements to the last positional in the sequence (Bubble Sort)



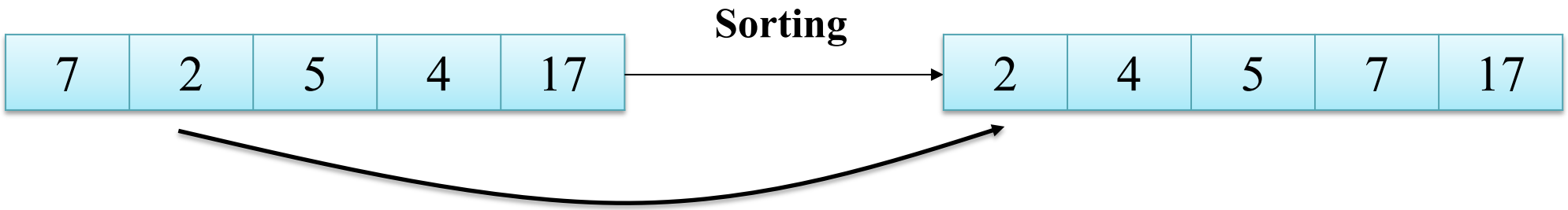
1 – Brute Force

1.2. Sorting problem: SELECTION SORT

- Input: a **sequence** of n integer number $<a_1, a_2, \ldots, a_n>$
- Output: a permutation (reordering)) $<a'_1, a'_2, \ldots, a'_n>$ such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

SELECTION SORT

- Select the smallest element from unsorted list in each iteration
- Place that element at the beginning of the unsorted list



1 – Brute Force

1.2. Sorting problem: SELECTION SORT

- Set the first element as minimum

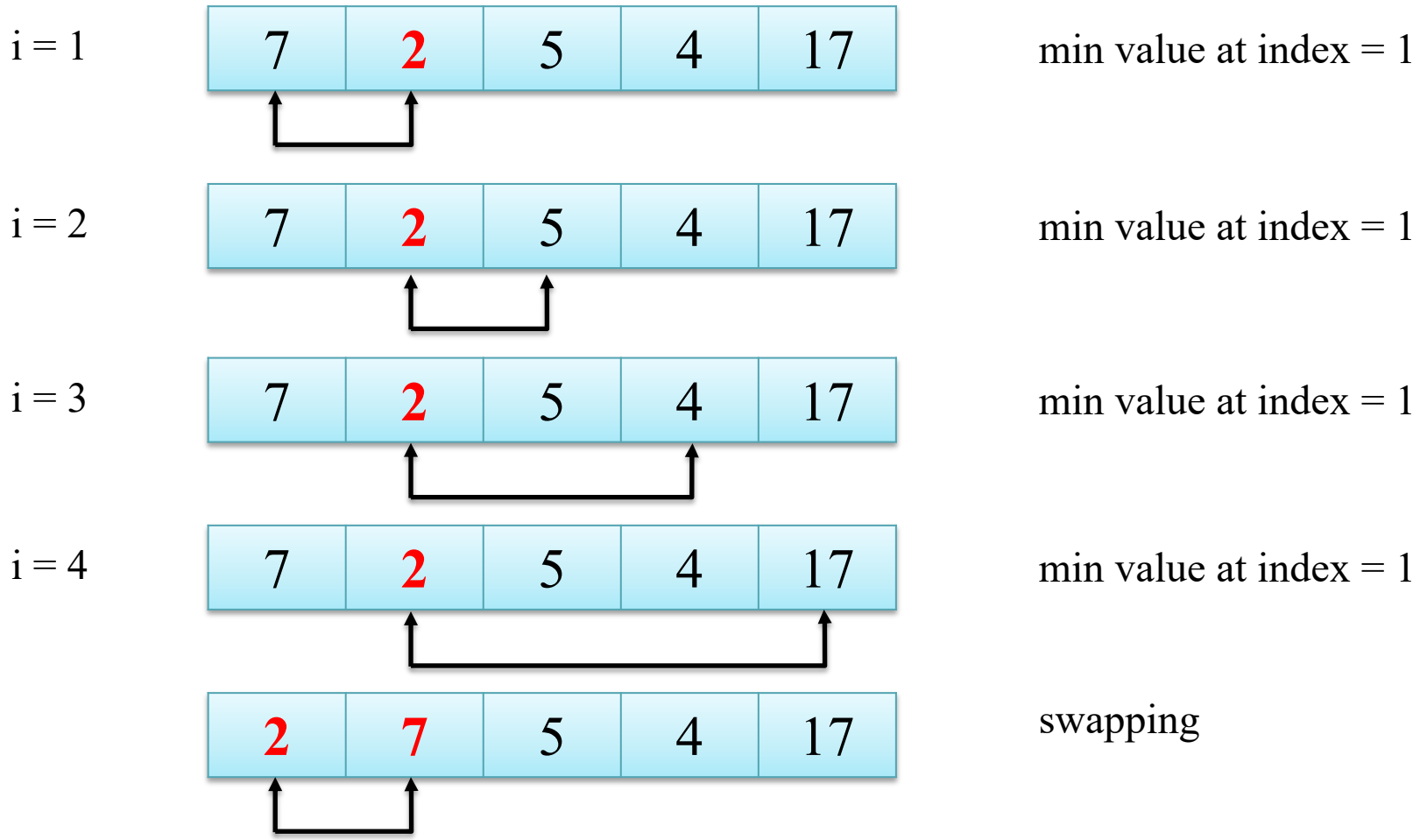
- Compare minimum with the other element. If the other element is smaller than minimum, assign the second element as minimum.

- After each iteration, minimum is placed in the front of the unsorted list

1 – Brute Force

1.2. Sorting problem: SELECTION SORT

Step 0

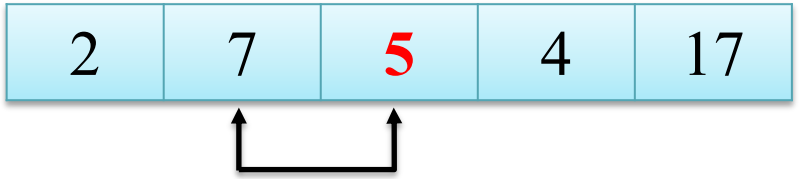


1 – Brute Force

1.2. Sorting problem: SELECTION SORT

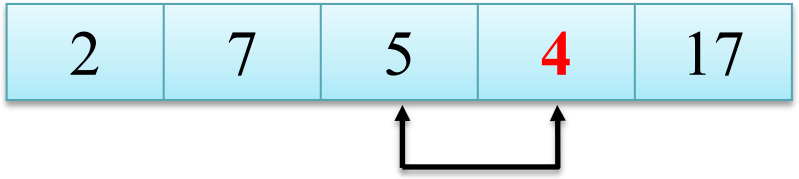
Step 1

i = 2



min value at index = 2

i = 3



min value at index = 3

i = 4

min value at index = 3

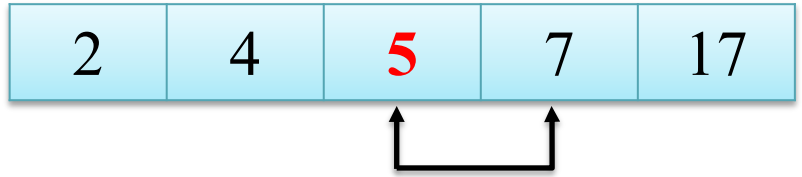
swapping

1 – Brute Force

1.2. Sorting problem: SELECTION SORT

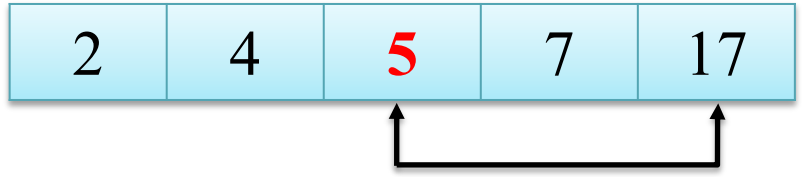
Step 2

i = 3



min value at index = 2

i = 4



min value at index = 2

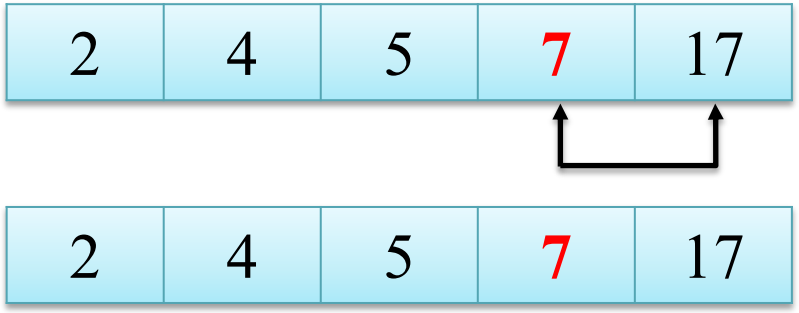
already in place

1 – Brute Force

1.2. Sorting problem: SELECTION SORT

Step 3

i = 4



min value at index = 3

already in place

1 – Brute Force

1.2. Sorting problem: SELECTION SORT

- Number of comparisons:

Cycle	Number of comparisions
1st	(n-1)
2nd	(n-2)
3rd	(n-3)
…	…
last	1

$\Rightarrow (n-1) + (n-2) + (n-3) + \ldots + 1$
$= n(n-2)/2$

```python
# aivietnam

def selection_sort(array):
    n = len(array)

    for step in range(n):
        min_idx = step

        for i in range(step + 1, n):

            if array[i] < array[min_idx]:
                min_idx = i

        array[step], array[min_idx] = array[min_idx], array[step]

    return array

arr = [7, 2, 5, 4, 17]
selection_sort(arr)
```

```
[2, 4, 5, 7, 17]
```

1 – Brute Force

1.2. Sorting problem: SELECTION SORT

Time Complexities:

- Best case: $O(n^2)$

The array is already sorted

- Worst case: $O(n^2)$

The array is sorted

- Average case: $O(n^2)$

```python
# aivietnam

def selection_sort(array):
    n = len(array)

    for step in range(n):
        min_idx = step

        for i in range(step + 1, n):

            if array[i] < array[min_idx]:
                min_idx = i

        array[step], array[min_idx] = array[min_idx], array[step]

    return array

arr = [7, 2, 5, 4, 17]
selection_sort(arr)
```

```
[2, 4, 5, 7, 17]
```

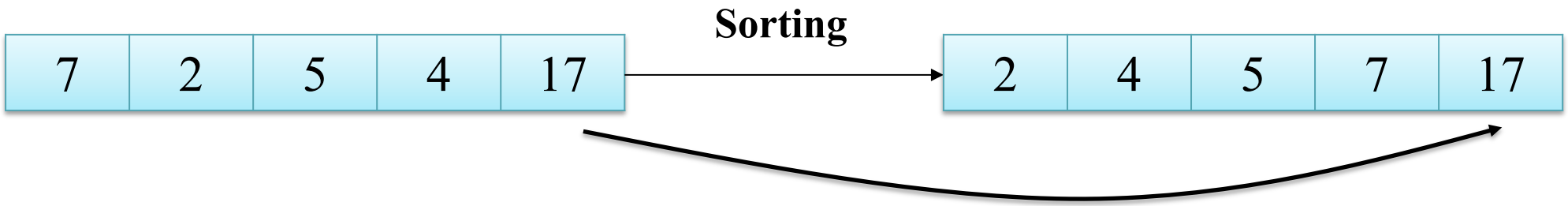
1 – Brute Force

1.2. Sorting problem: BUBBLE SORT

- Input: a **sequence** of n integer number $<a_1, a_2, \ldots, a_n>$
- Output: a permutation (reordering)) $<a'_1, a'_2, \ldots, a'_n>$ such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

BUBBLE SORT

- Select the largest element from unsorted list in each iteration
- Place that element at the end of the unsorted list

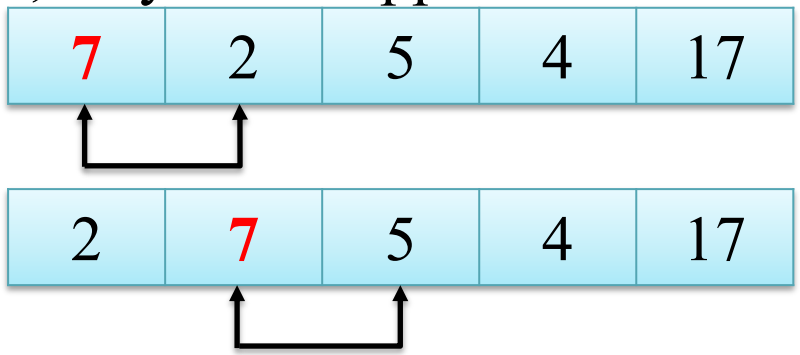


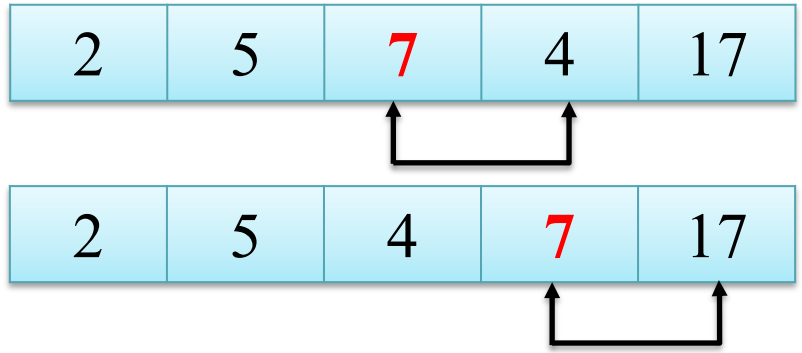
1 – Brute Force

1.2. Sorting problem: BUBBLE SORT

7	2	5	4	17

- Compare the first element with the other element. If the first element is greater than other, they are swapped.





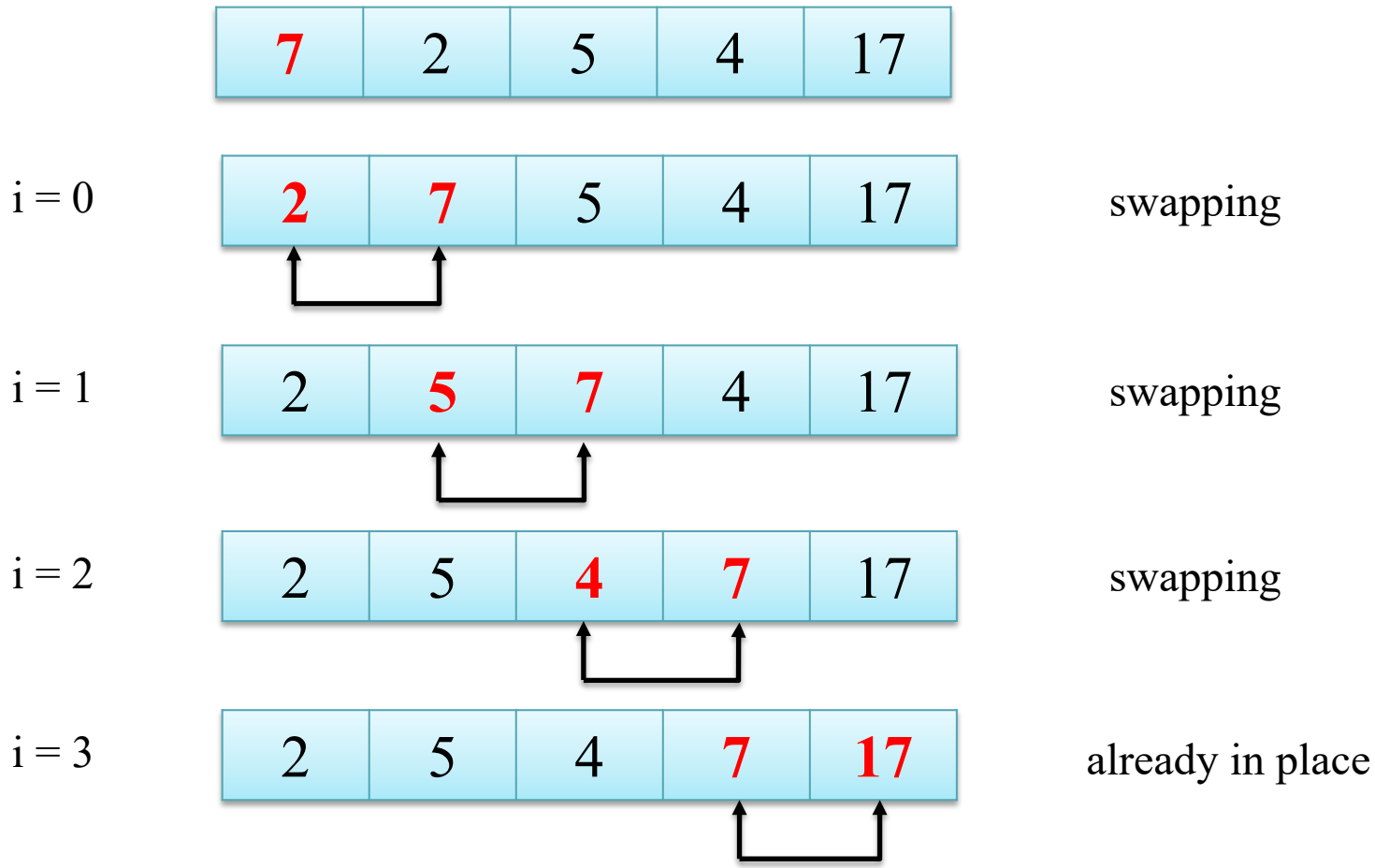
- After each iteration, maximum is placed in the end of the unsorted list

2	5	4	7	17

1 – Brute Force

1.2. Sorting problem: BUBBLE SORT

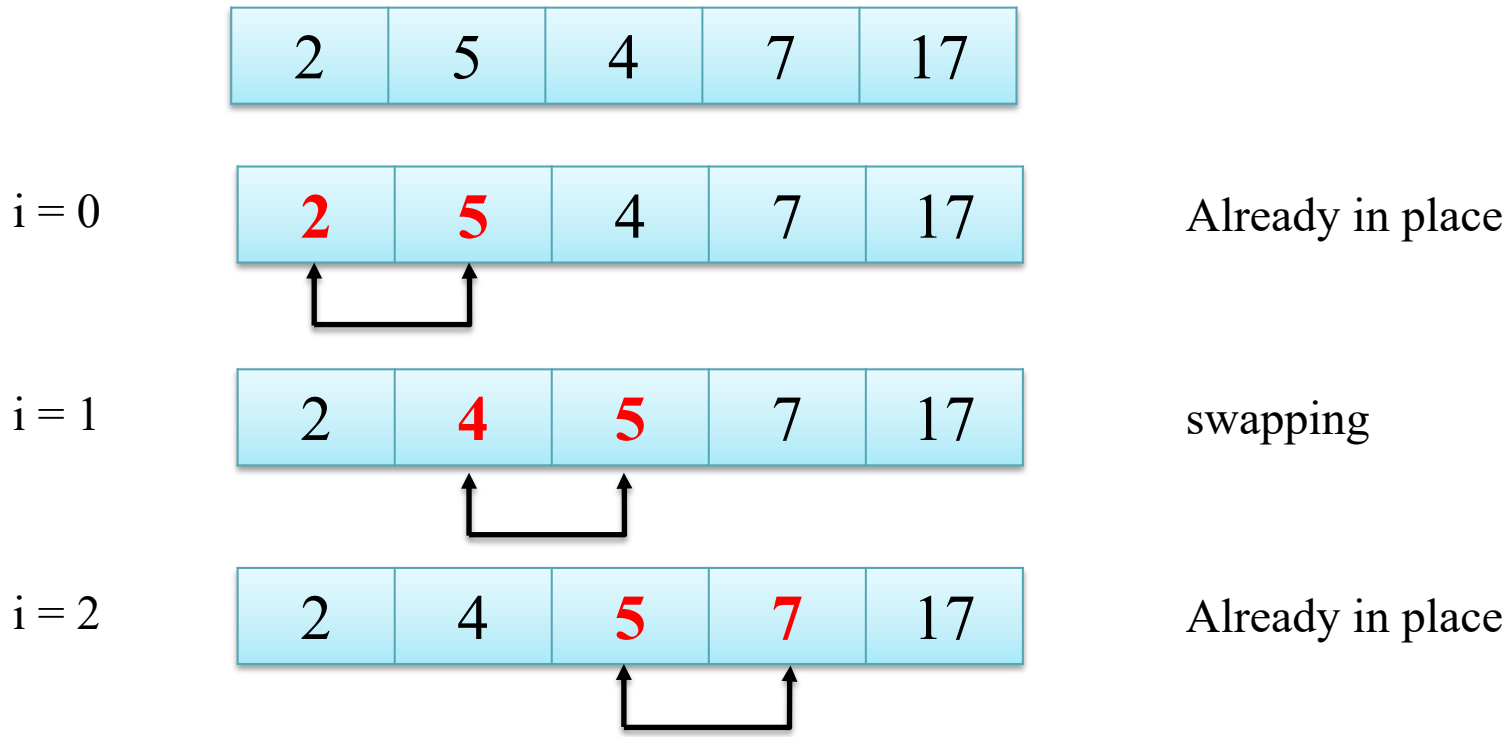
Step 0



1 – Brute Force

1.2. Sorting problem: BUBBLE SORT

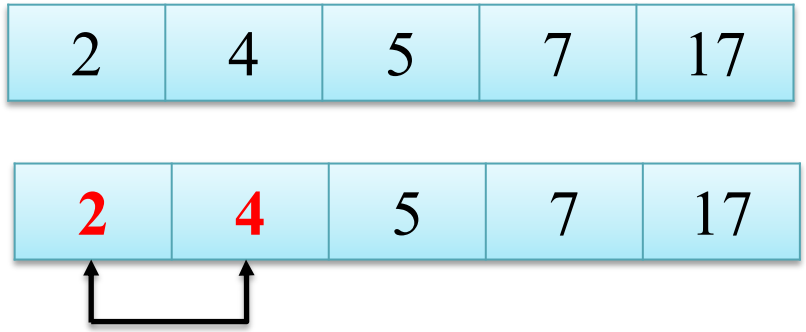
Step 1



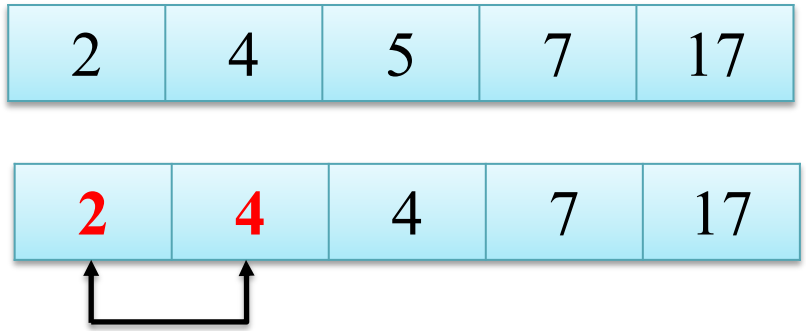
1 – Brute Force

1.2. Sorting problem: BUBBLE SORT

Step 2



Step 3



1 – Brute Force

1.2. Sorting problem: BUBBLE SORT

- Number of comparisons:

Cycle	Number of comparisions
1st	(n-1)
2nd	(n-2)
3rd	(n-3)
…	…
last	1

$\Rightarrow (n-1) + (n-2) + (n-3) + \ldots + 1$
$= n(n-2)/2$

```python
# aivietnam

def bubble_sort(array):
    n = len(array)

    for step in range(n):

        for i in range(0, n-step-1):

            if array[i] > array[i+1]:
                temp = array[i]
                array[i] = array[i+1]
                array[i+1] = temp

    return array

arr = [7, 2, 5, 4, 17]
bubble_sort(arr)
```

```
[2, 4, 5, 7, 17]
```

1 – Brute Force

1.2. Sorting problem: BUBBLE SORT

Time Complexities:

- Best case: $O(n^2)$

The array is already sorted

- Worst case: $O(n^2)$

The array is sorted

- Average case: $O(n^2)$

```python
# aivietnam

def bubble_sort(array):
    n = len(array)

    for step in range(n):

        for i in range(0, n-step-1):

            if array[i] > array[i+1]:
                temp = array[i]
                array[i] = array[i+1]
                array[i+1] = temp

    return array

arr = [7, 2, 5, 4, 17]
bubble_sort(arr)
```

```
[2, 4, 5, 7, 17]
```

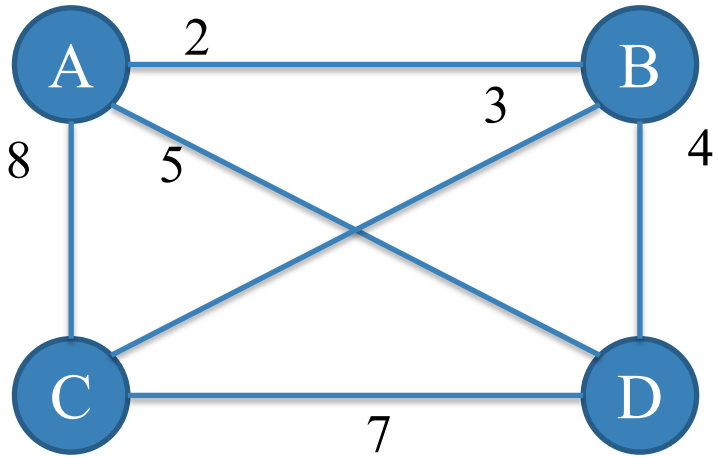
2 – Exhaustive Search

- A brute force solution to a problem involving search for an element with a special property or constraint by exploring every possible combination from a set of choices or values
- Method:
  - 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 search ends, output the solution found

2 – Exhaustive Search

Traveling Salesman Problem

- Given *n* cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city
- Example:

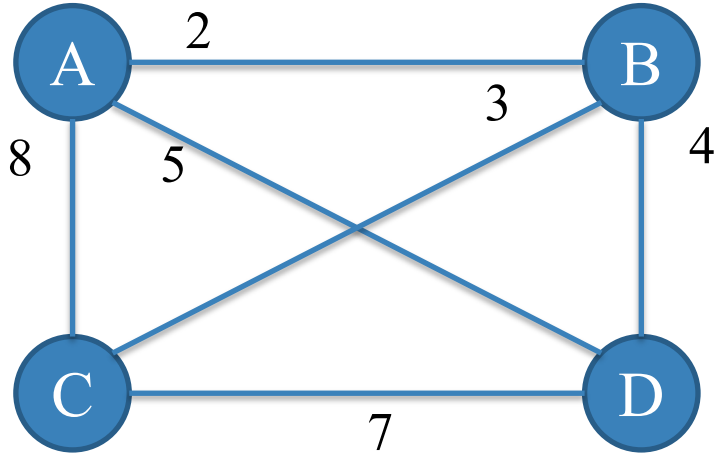


2 – Exhaustive Search

Traveling Salesman Problem

- Given n cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city
- **Step 1: Create all possible solutions (start: A)**

Tour
A => B => C => D => A
A => B => D => C => A
A => C => B => D => A
A => C => D => B => A
A => D => B => C => A
A => D => C => B => A
…

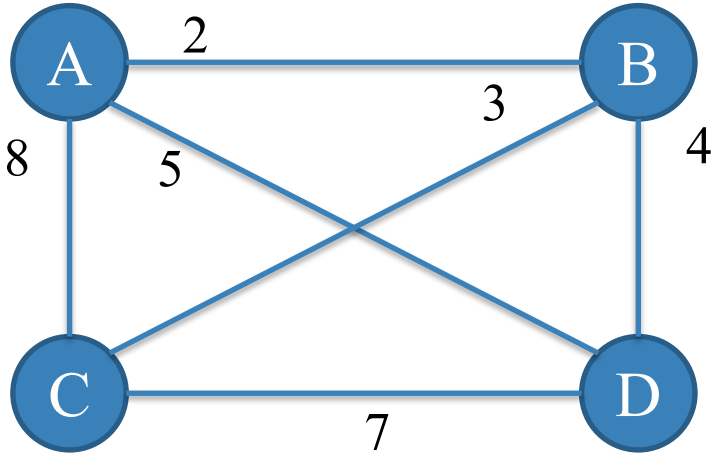


2 – Exhaustive Search

Traveling Salesman Problem

- Given n cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city
- **Step 2: Evaluate potential solutions**

Tour	Cost
A => B => C => D => A	2 + 3 + 7 + 5 = 17
A => B => D => C => A	2 + 4 + 7 + 8 = 21
A => C => B => D => A	8 + 3 + 4 + 5 = 20
A => C => D => B => A	8 + 7 + 4 + 2 = 21
A => D => B => C => A	5 + 4 + 3 + 8 = 20
A => D => C => B => A	5 + 7 + 3 + 2 = 17



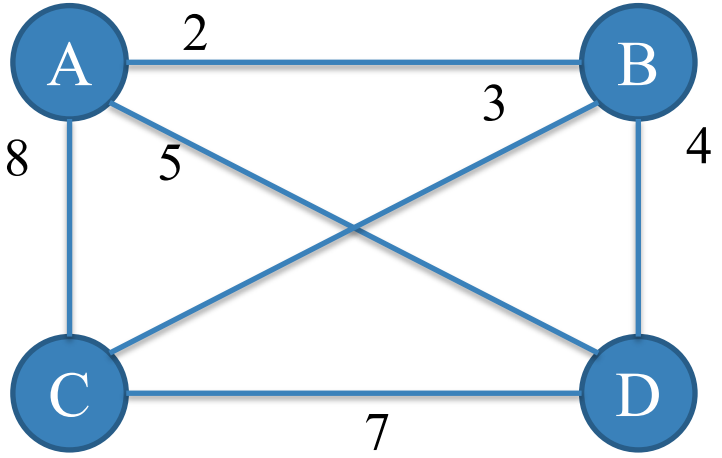
2 – Exhaustive Search

Traveling Salesman Problem

- Given n cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city
- **Step 2: Evaluate potential solutions**

Tour	Cost
A => B => C => D => A	2 + 3 + 7 + 5 = 17
A => B => D => C => A	2 + 4 + 7 + 8 = 21
A => C => B => D => A	8 + 3 + 4 + 5 = 20
A => C => D => B => A	8 + 7 + 4 + 2 = 21
A => D => B => C => A	5 + 4 + 3 + 8 = 20
A => D => C => B => A	5 + 7 + 3 + 2 = 17

Shortest Tour: A => D => C => B => A



AI VIET NAM
@aivietnam.edu.vn

2 – Exhaustive Search

Traveling Salesman Problem

```python
# aivietnam

def permute(nums):
    L = []
    n = len(nums)
    i = 0
    while i < n:
        if not L:
            L.append([nums[i]])
        else:
            newl = []
            for elem in L:
                for j in range(0, i+1):
                    e = elem[:]
                    e.insert(j, nums[i])
                    newl.append(e)
            L = newl
        i += 1

    return L
S = [1, 2, 3]
permute(S)
```

```
[[3, 2, 1], [2, 3, 1], [2, 1, 3], [3, 1, 2], [1, 3, 2], [1, 2, 3]]
```

```python
# aivietnam

from sys import maxsize
from itertools import permutations

def travelling_salesman_problem(distances, s, num_cities):
    cities = []
    for i in range(num_cities):
        if i != s:
            cities.append(i)
    min_path = maxsize
    next_permutation = permutations(cities)
    for i in next_permutation:
        current_pathweight = 0
        k = s
        for j in i:
            current_pathweight += distances[k][j]
            k = j
        current_pathweight += distances[k][s]
        min_path = min(min_path, current_pathweight)

    return min_path

distances = [
    [0, 2, 8, 5], [2, 0, 3, 4],
    [8, 3, 0, 7], [5, 4, 7, 0]
]
s = 0
num_cities = 4

travelling_salesman_problem(distances, s, num_cities)
```

Summary

1 Brute Force

Searching

```
LINEAR-SEARCH(arr, key)
1     for idx = 0 to (arr.length-1)
2         element = arr[id]
3         // Compare element with key
4         if element = key
5             return idx
6     return
7         -1
```

Sorting

- Selection Sort

Moving the smaller elements to the first positional in the sequence

- Bubble Sort

Moving the larger elements to the last positional in the sequence

2 Exhaustive Search

Travelling Salesman Problem

- Search with a special property or constraint

(1) Create all possible solutions

(2) Evaluate potential solutions

(3) Return the solution found

Reference

(1) Introduction to Algorithms, 3rd Edition; Thomas H.Cormen et al; 2009

(2) Data Structures & Algorithms; Michael T.Goodrich et al; 2013

(3) Algorithms, 4th; Robert Sedgewick et al; 2011

AI VIET NAM

@aivietnam.edu.vn

Thanks!

Any questions?