Data Representation and Manipulation

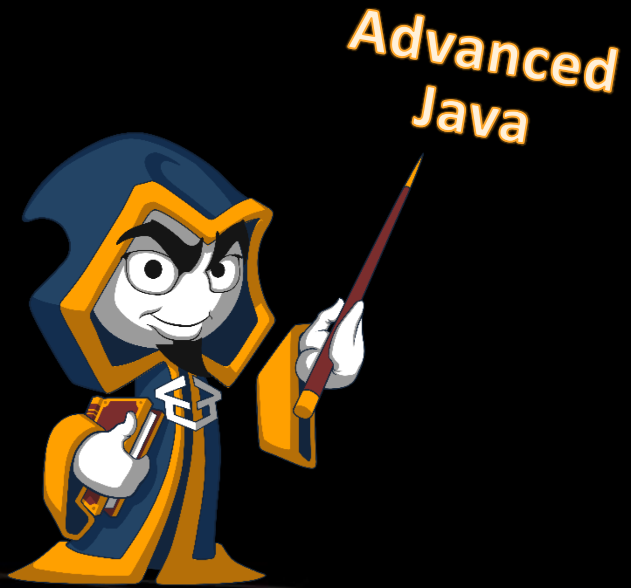
Recursion, Sorting and Searching Algorithms

SoftUni Team

Technical Trainers

Software University

http://softuni.bg



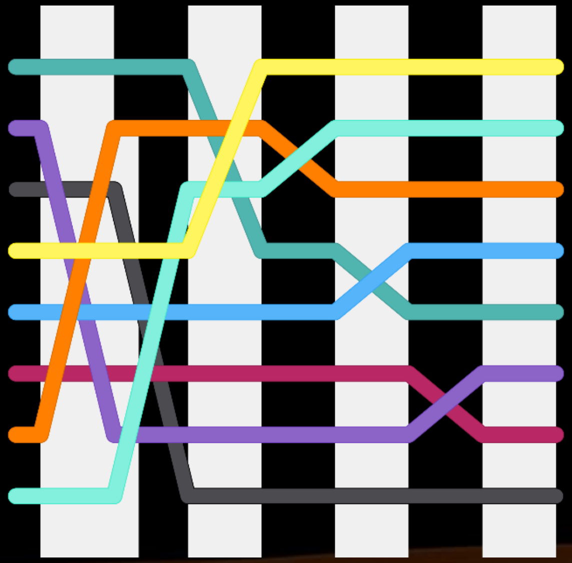


Table of Contents

1. Simple Sorting Algorithms
   - Selection Sort
   - Bubble Sort
   - Comparing Sorting Algorithms
2. Basic Search Algorithms
   - Linear Search
   - Binary Search
3. Recursion

Have a Question?

SoftUni Foundation

sli.do

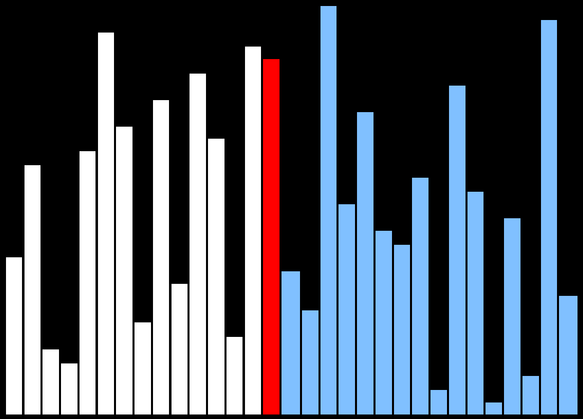
#JavaAdvanced

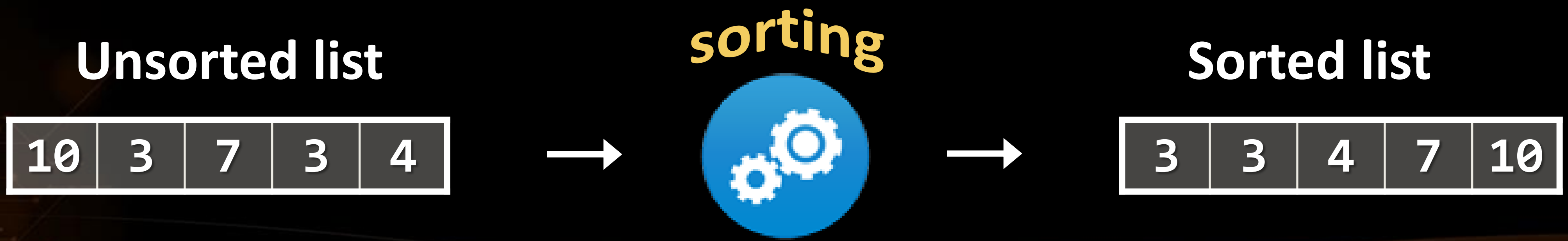
Simple Sorting Algorithms

Selection Sort and Bubble Sort

What is a Sorting Algorithm?

- **Sorting algorithm**
  - An algorithm that rearranges elements in a collection
    - In non-decreasing order
  - Elements must be **comparable**





Unsorted list

10	3	7	3	4

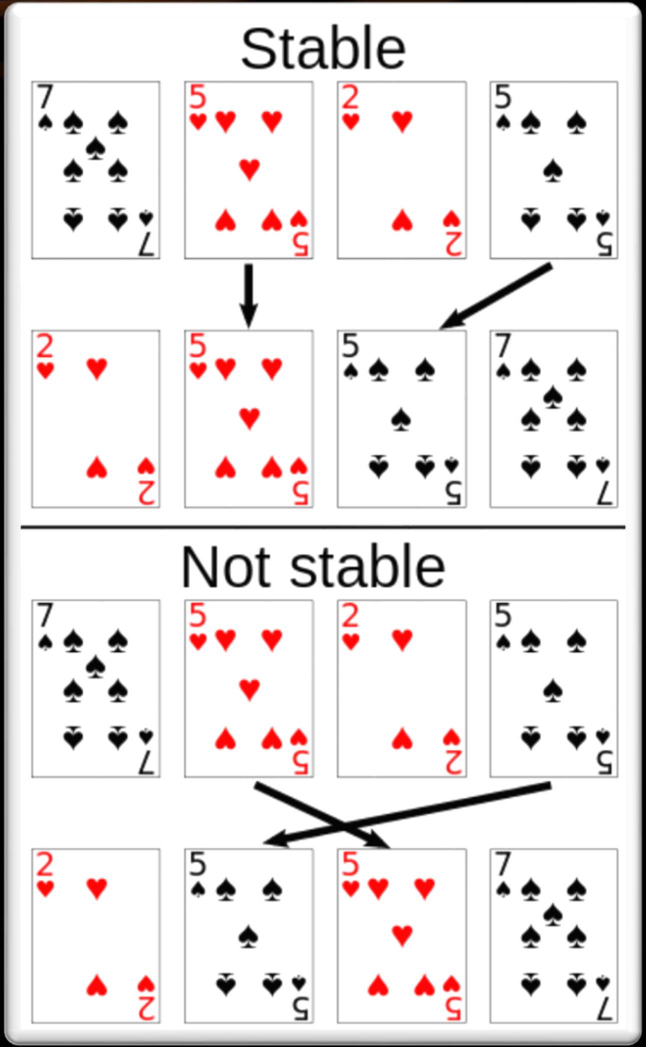
→ sorting →

Sorted list

3	3	4	7	10

Stability of Sorting

- **Stable** sorting algorithms
  - Maintain the order of equal elements
  - If two items compare as equal, their relative order is preserved
- **Unstable** sorting algorithms
  - Rearrange the equal elements in unpredictable order



Selection Sort

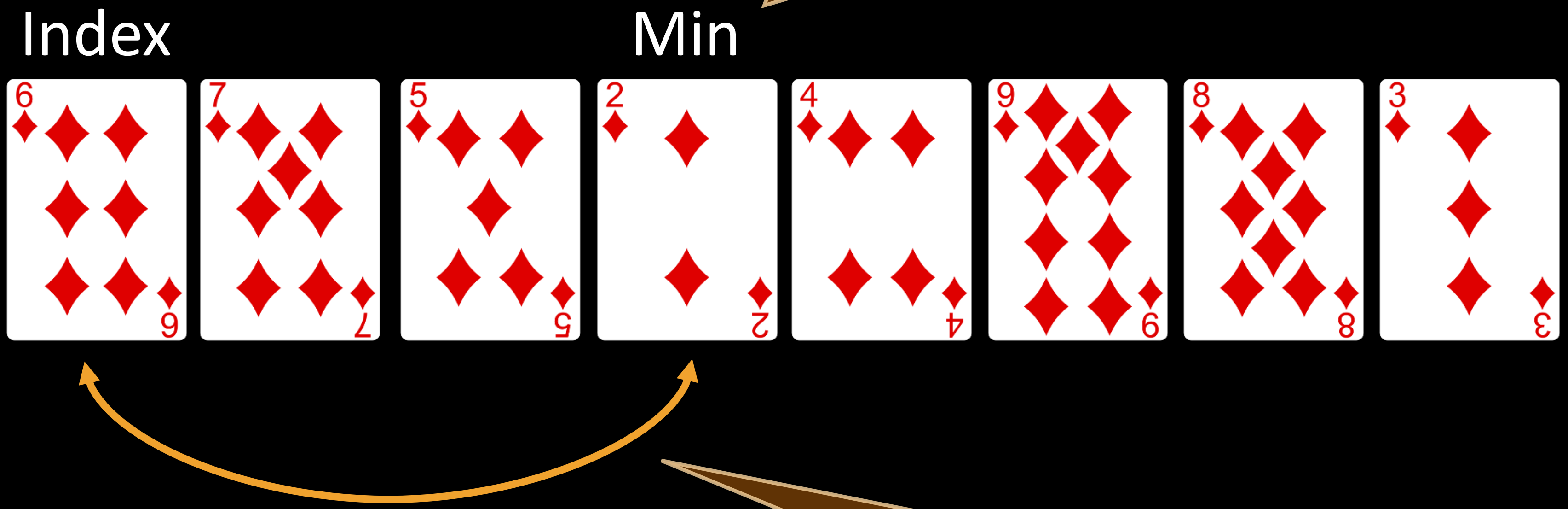
- Swap each element with the min element on its right
- Visualize

```
repeat (numOfElements - 1) times
  set the first element as min
  for each of the next elements
    if element < currentMinimum
      set element as new minimum
  swap minimum with first element
```

Selection Sort Visualization

Steps count: 8 + 1 ⇨ 9

Finding the **smallest** element takes **8** steps

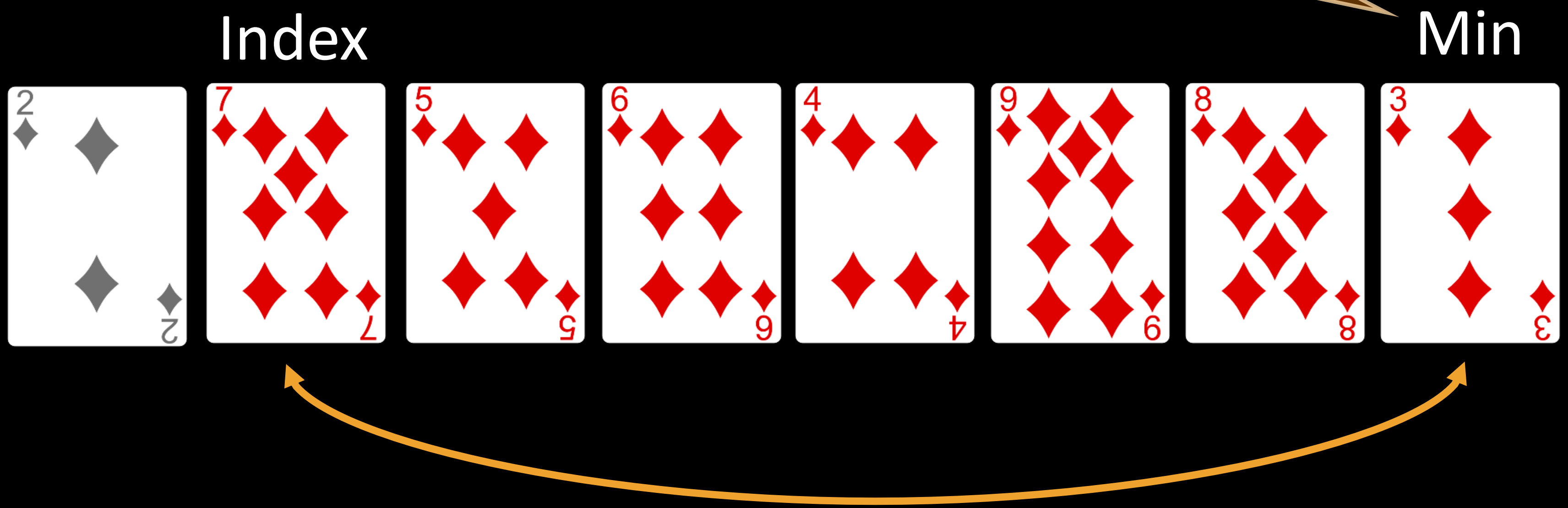


Swapping elements counts as an extra step

Selection Sort Visualization

Steps count: 9 + 7 + 1 ⇨ 17

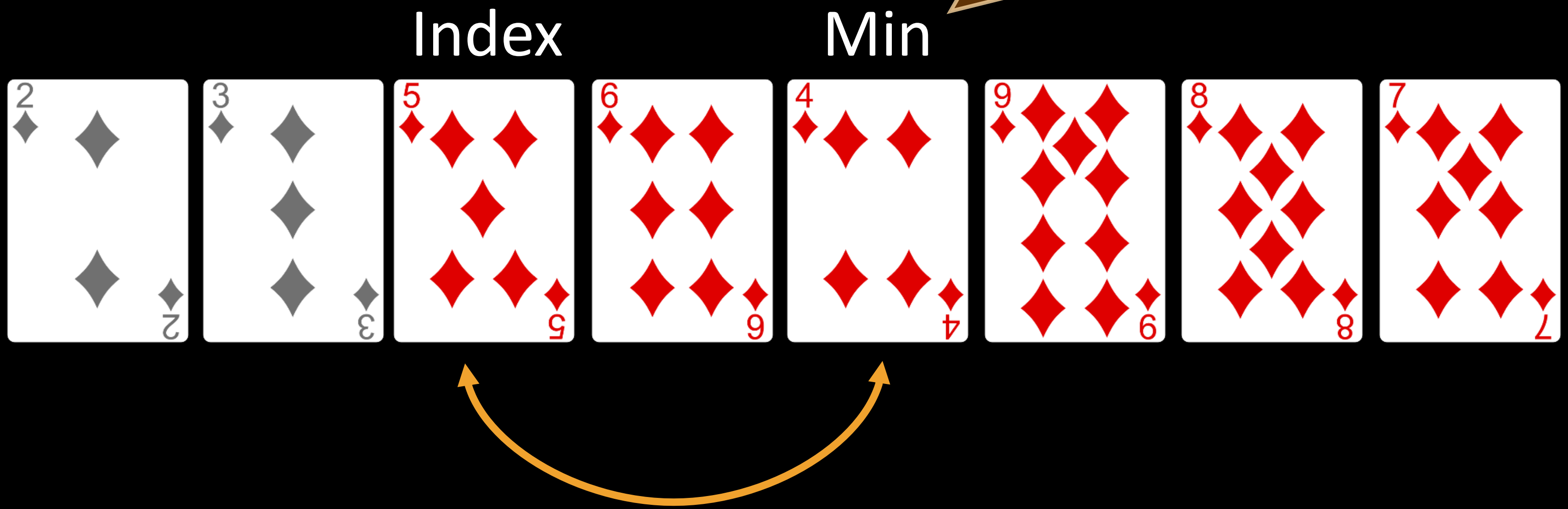
Finding the **smallest** element takes **7** steps



Selection Sort Visualization

Steps count: 17 + 6 + 1 ⇨ 24

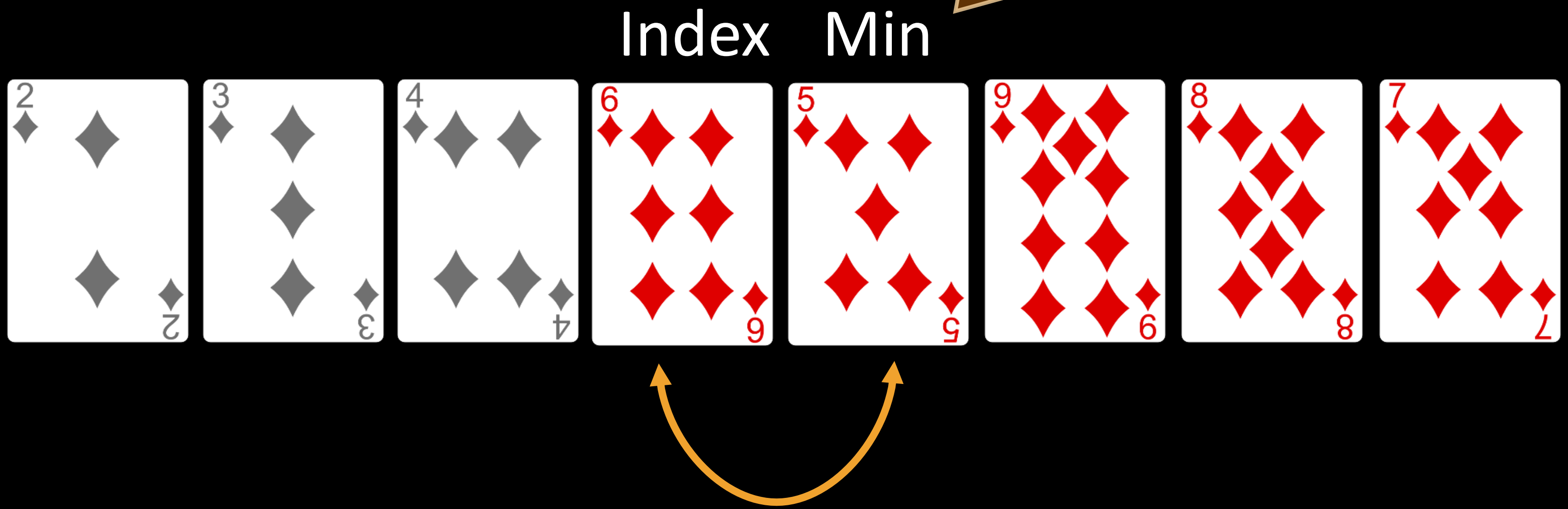
Finding the **smallest** element takes **6** steps



Selection Sort Visualization

Steps count: 24 + 5 + 1 ⇨ 30

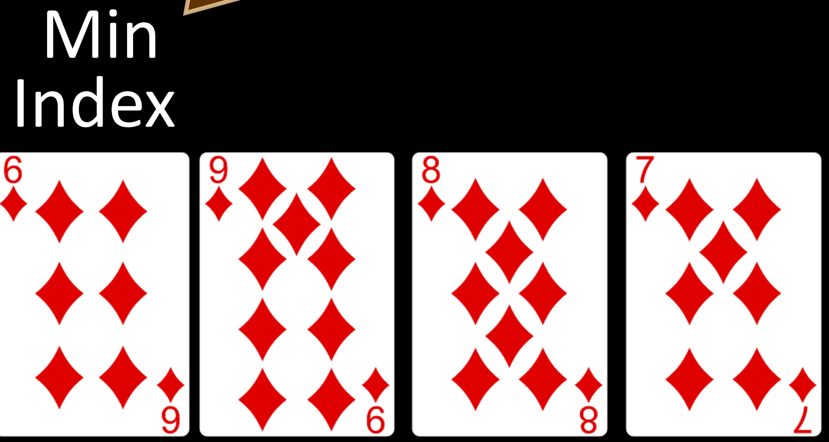
Finding the **smallest** element takes **5** steps



Selection Sort Visualization

Steps count: 30 + 4 ⇨ 34

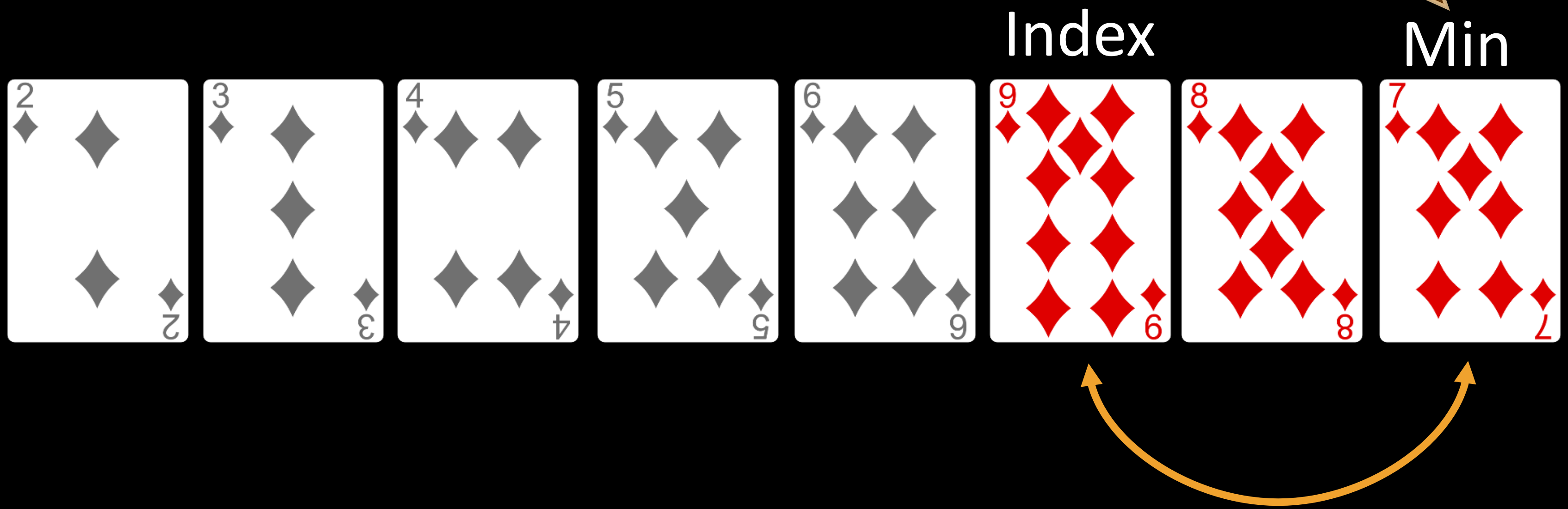
Finding the **smallest** element takes **4** steps



Selection Sort Visualization

Steps count: 34 + 3 + 1 ⇨ 38

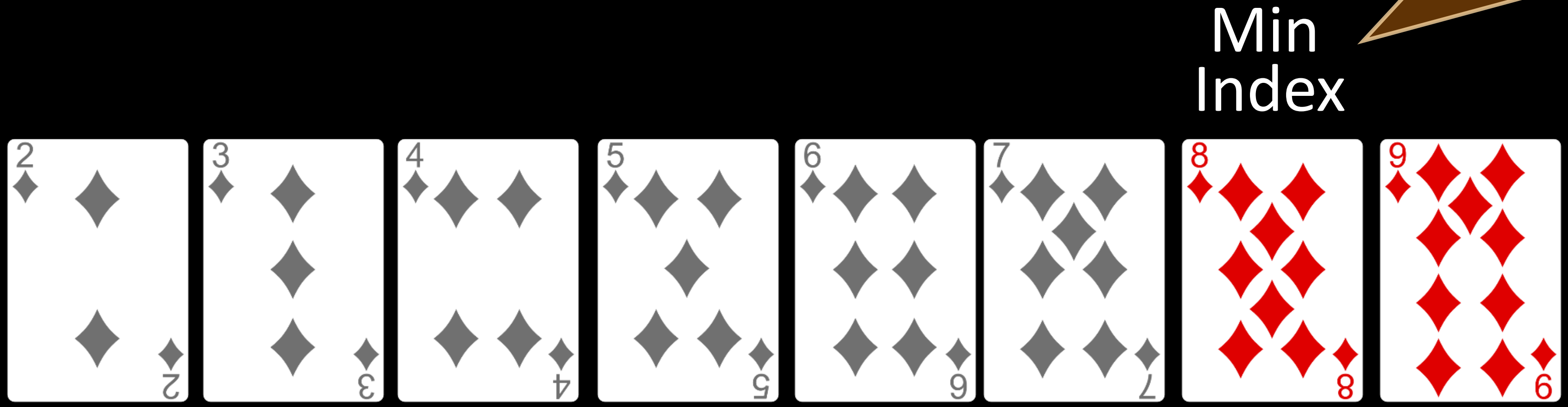
Finding the **smallest** element takes **3** steps



Selection Sort Visualization

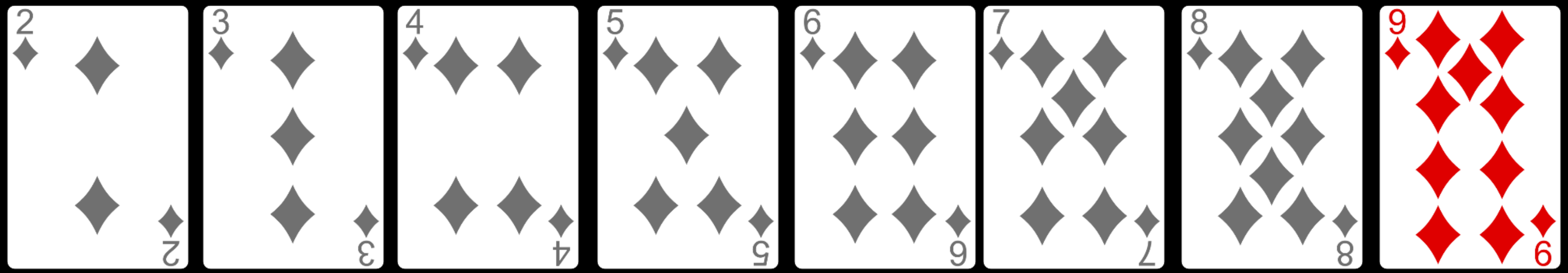
Steps count: 38 + 2 ⇨ 40

Finding the **smallest** element takes **2** steps



Selection Sort Visualization

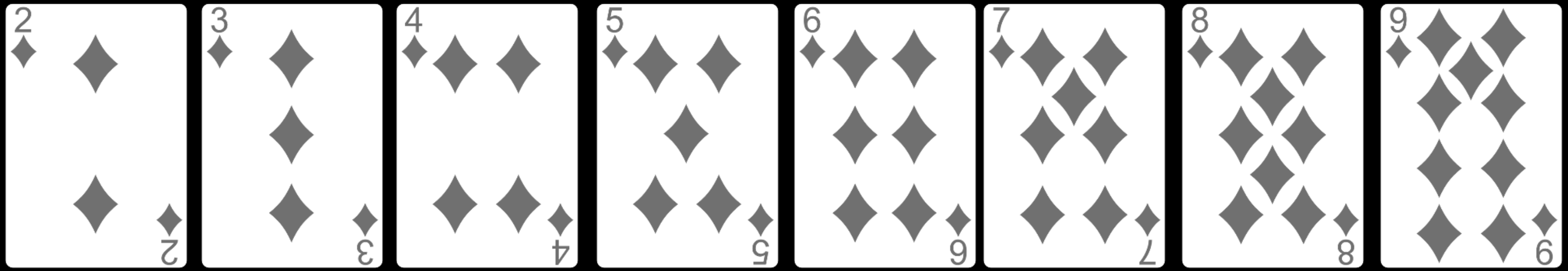
Steps count: 40 + 1 ⇨ 41



Finding the **smallest** element takes **1** step

Selection Sort Visualization

Total count of steps : 41



Selection Sort Code

```
for (int index = 0; index < collection.length; index++){
  int min = index;
  for (int curr = index + 1; curr < collection.length; curr++){
    if (collection[curr] < collection[min]){
       min = curr;
    }
  }
  swap(collection, index, min);
}
```

Find the **smallest** element

Swap current with it

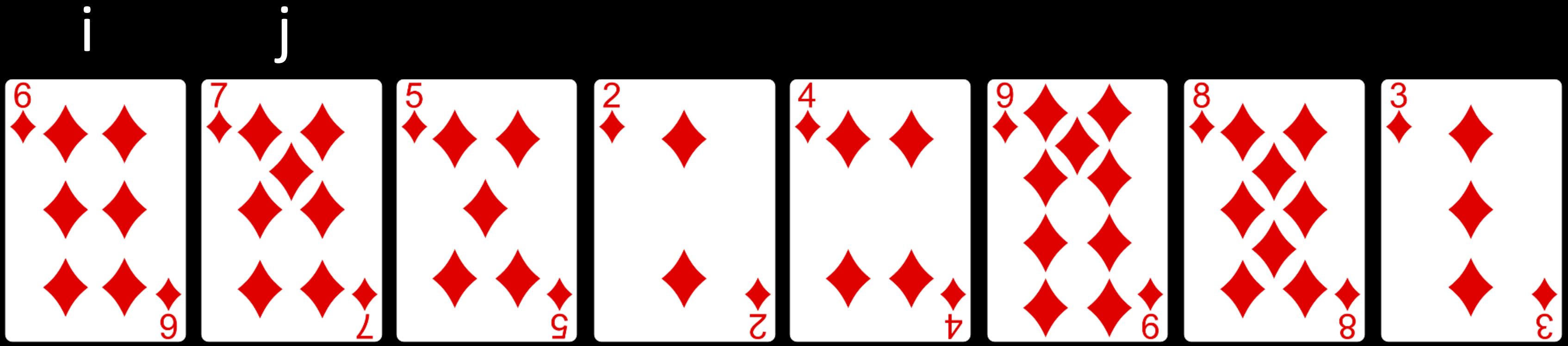
Bubble Sort

- Swaps neighbor elements when not in order until sorted
- Visualize

```
do
 swapped = false
 for i = 1 to collection length
  if leftElement > rightElement
   swap(leftElement, rightElement)
   swapped = true
while swapped
```

Bubble Sort Visualization

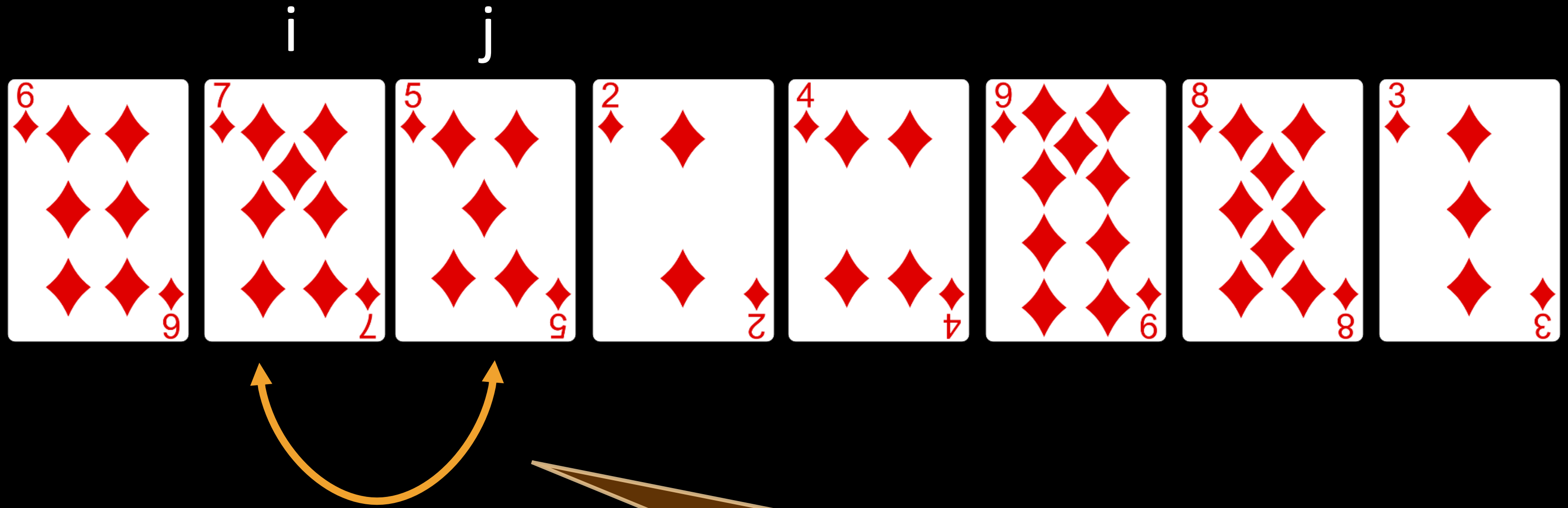
Steps count: 1



Bubble Sort Visualization

SoftUni Foundation

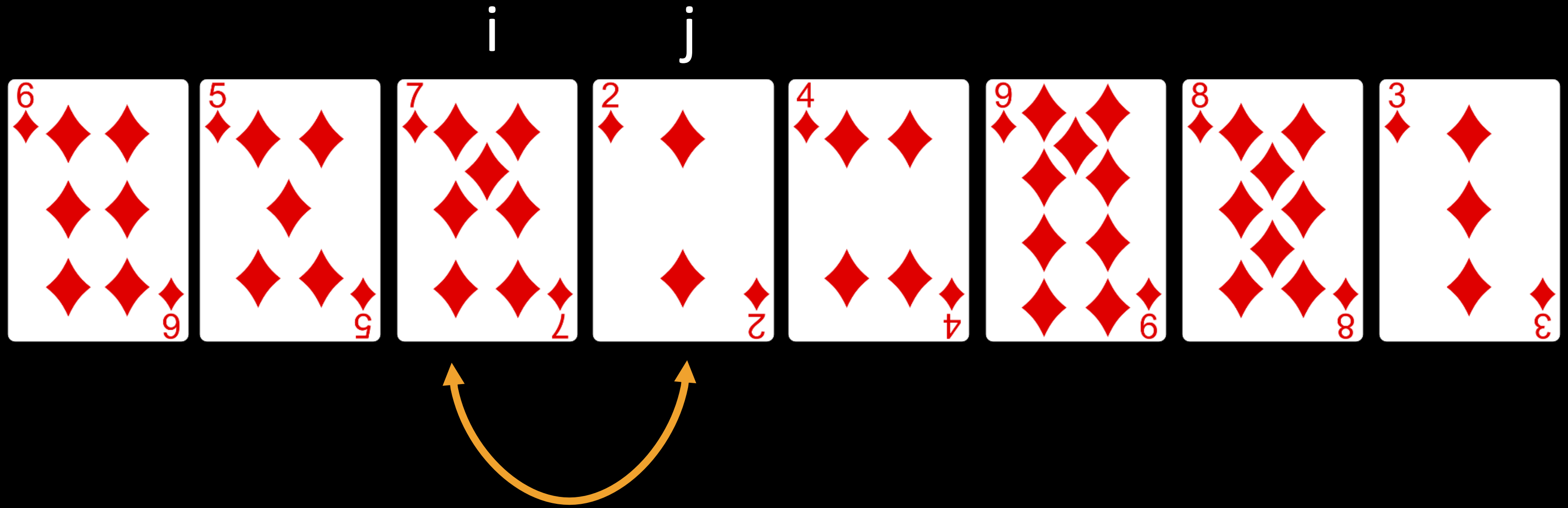
Steps count: 1 + 1 + 1 ⇨ 3



Swapping elements counts as an extra step

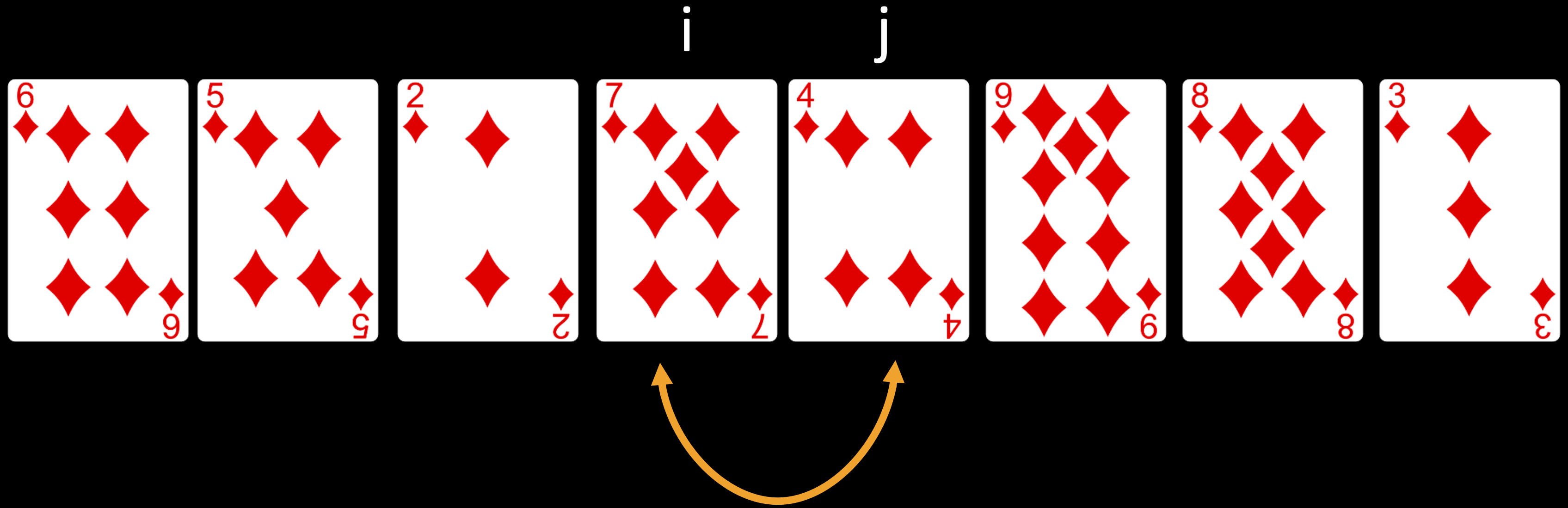
Bubble Sort Visualization

Steps count: 3 + 1 + 1 ⇨ 5



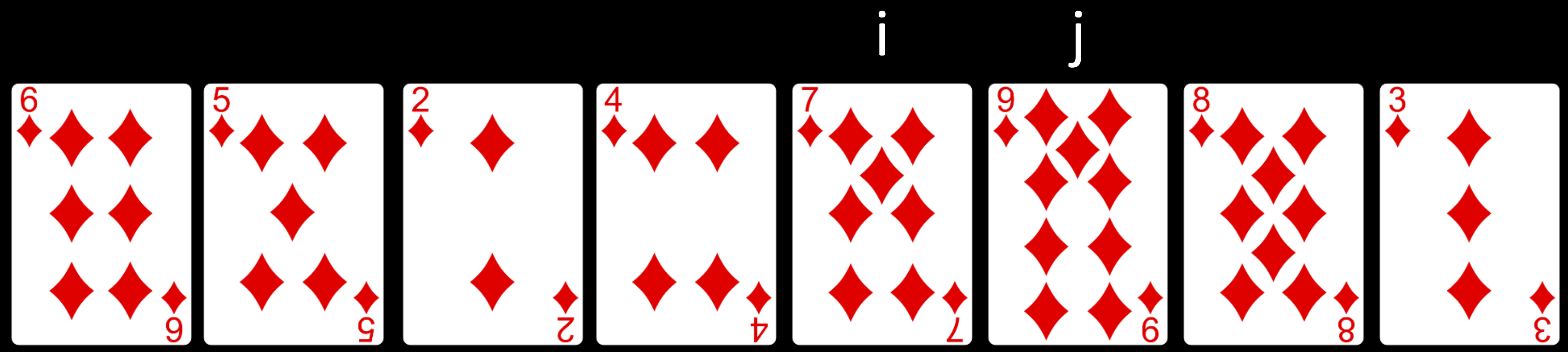
Bubble Sort Visualization

Steps count: 5 + 1 + 1 ⇨ 7



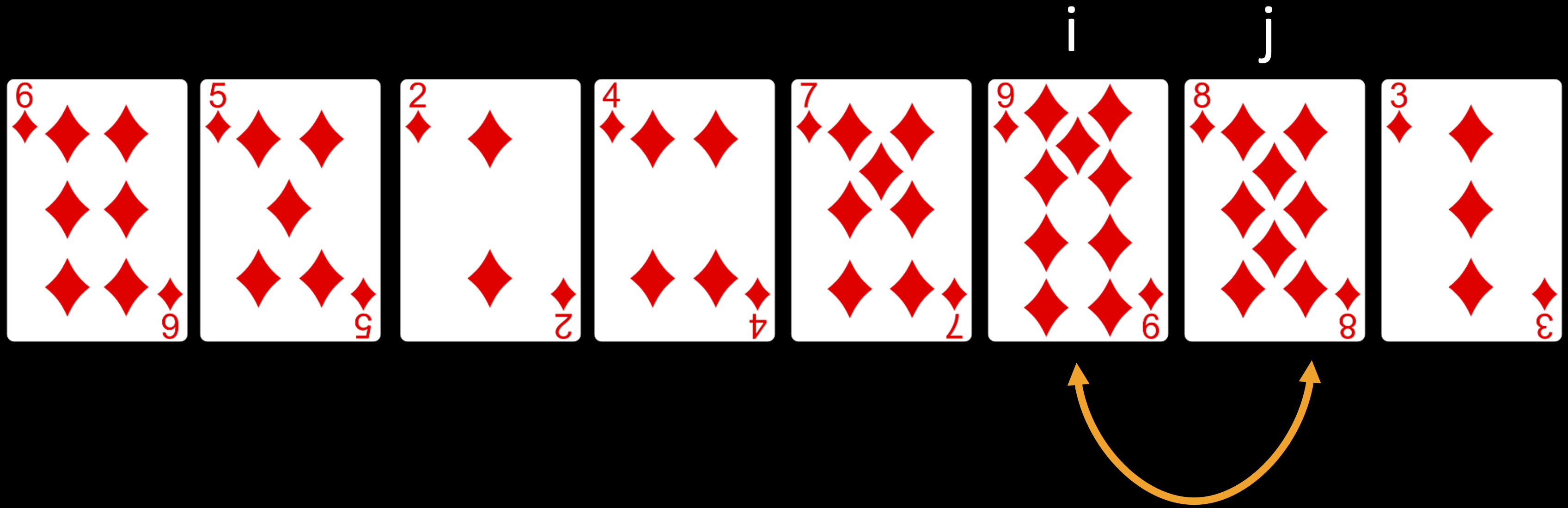
Bubble Sort Visualization

Steps count: 7 + 1 ⇨ 8



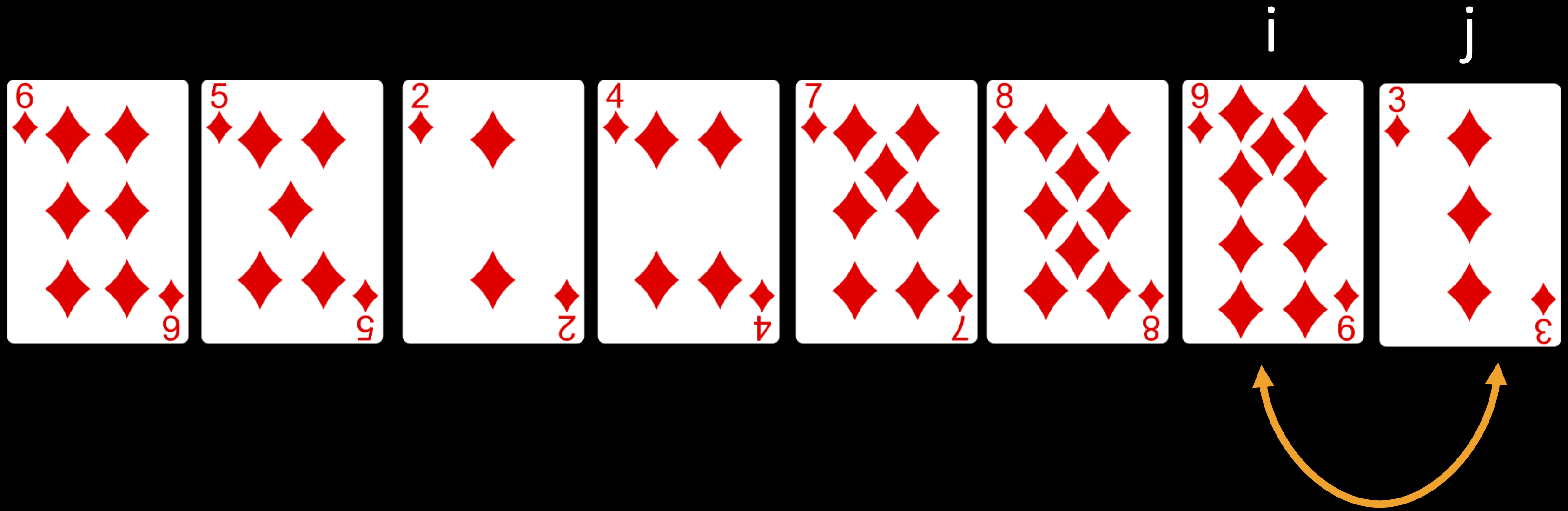
Bubble Sort Visualization

Steps count: 8 + 1 + 1 ⇨ 10



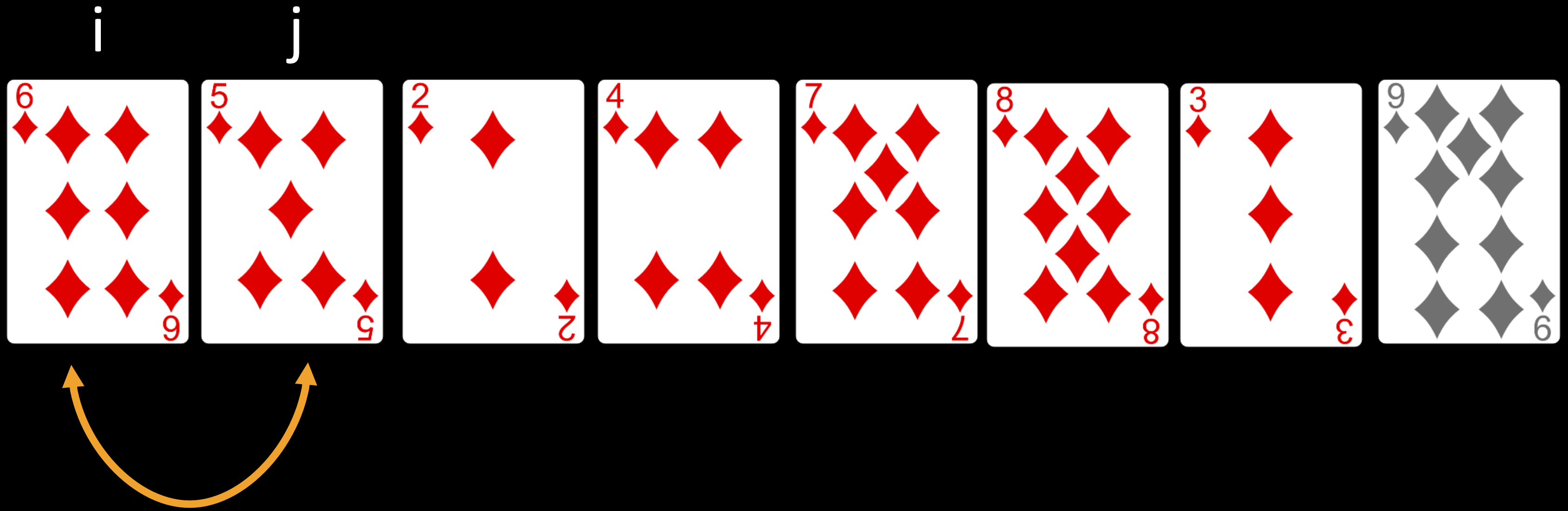
Bubble Sort Visualization

Steps count: 10 + 1 + 1 ⇨ 12



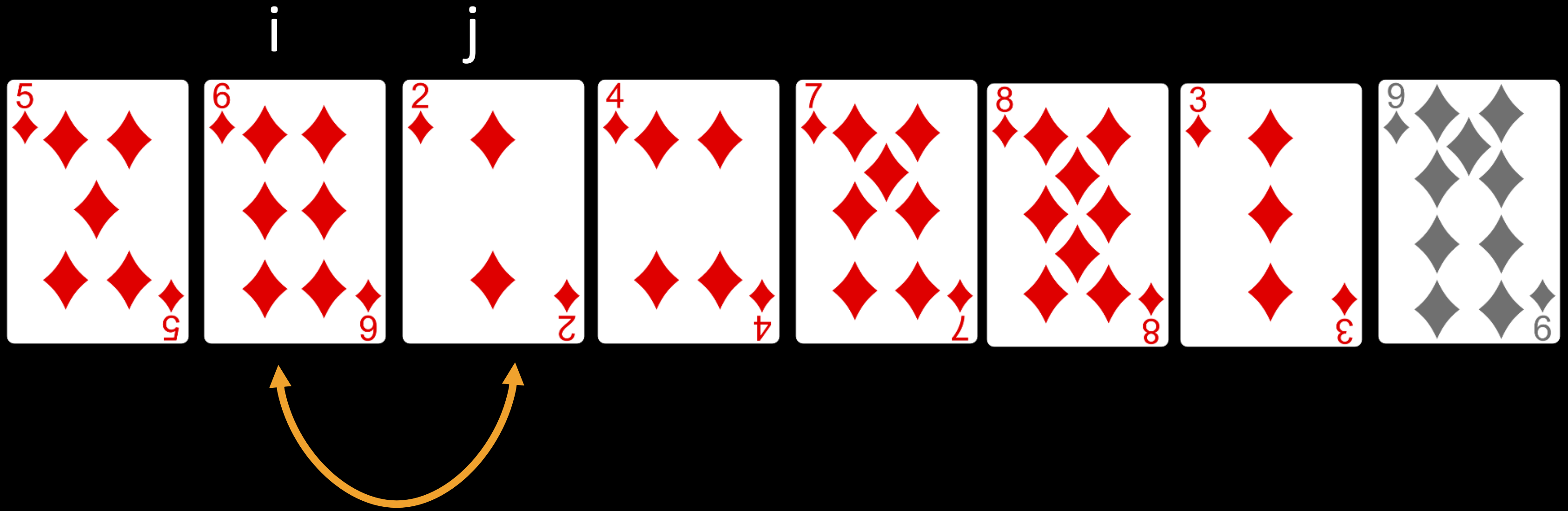
Bubble Sort Visualization

Steps count: 12 + 1 + 1 ⇨ 14



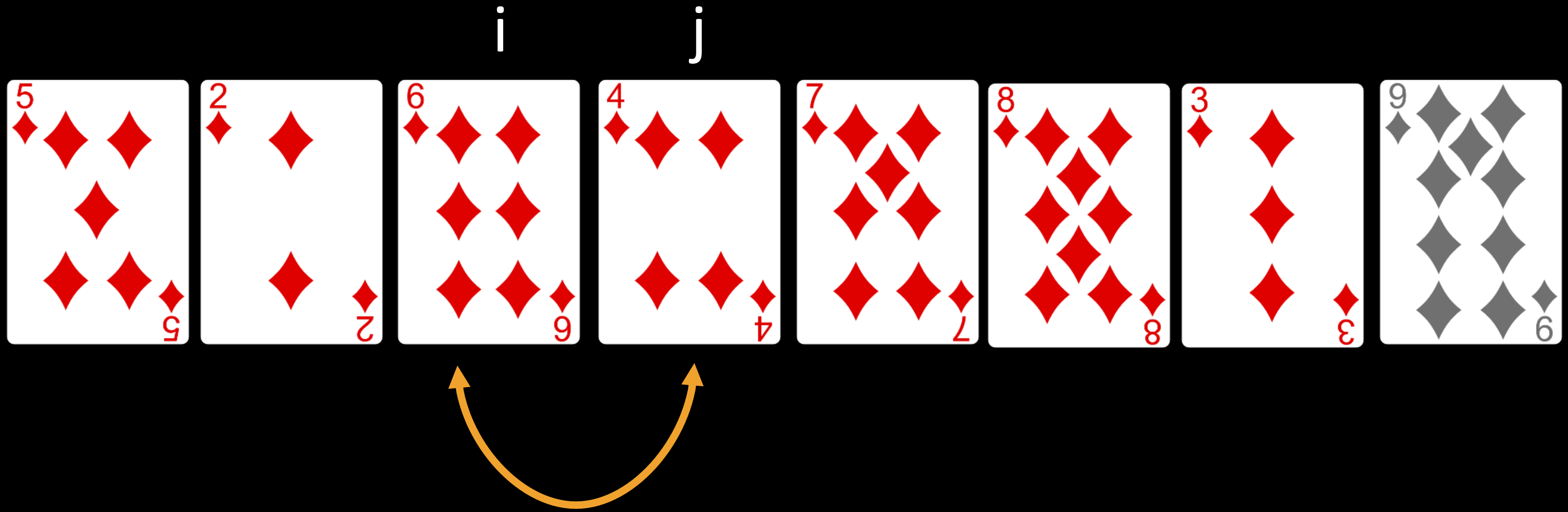
Bubble Sort Visualization

Steps count: 14 + 1 + 1 ⇨ 16



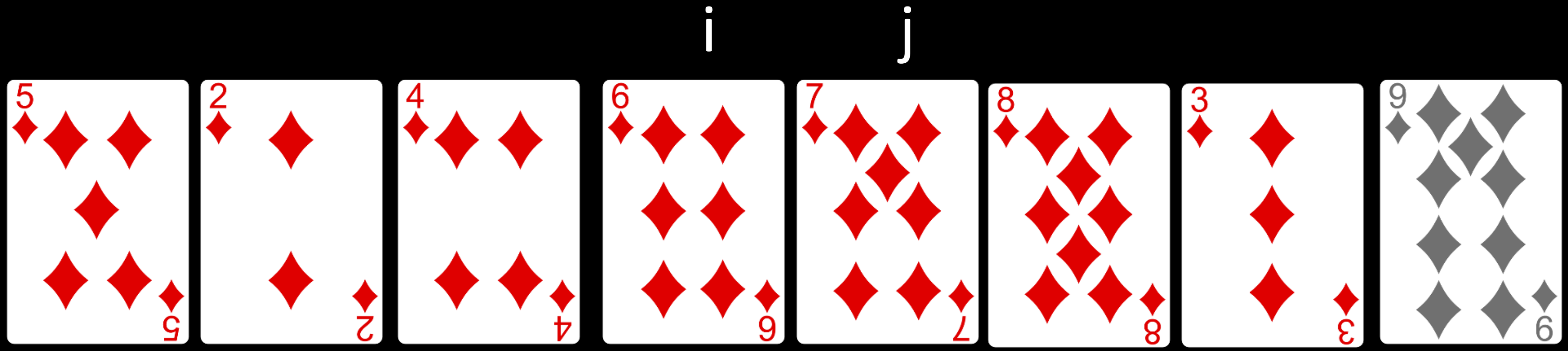
Bubble Sort Visualization

Steps count: 16 + 1 + 1 ⇨ 18



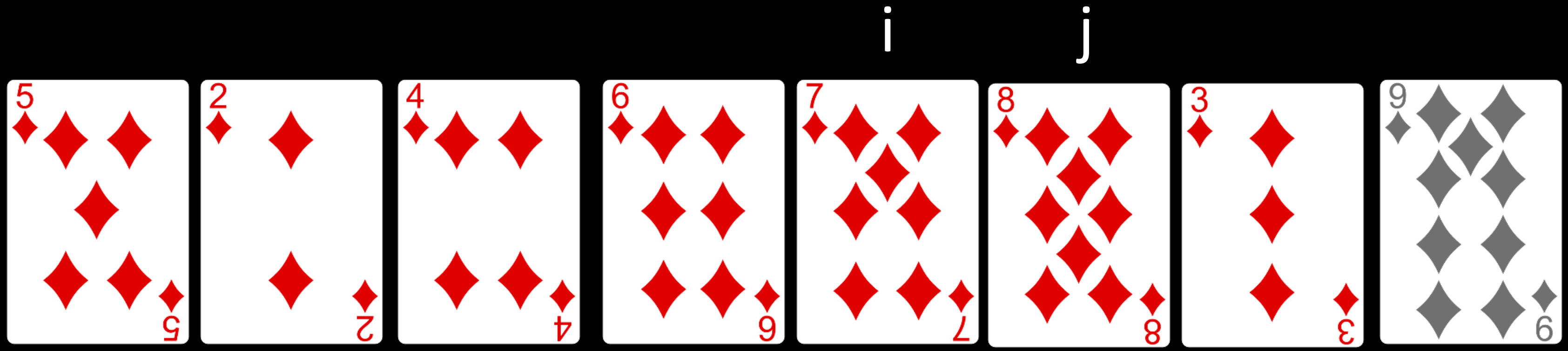
Bubble Sort Visualization

Steps count: 18 + 1 ⇨ 19



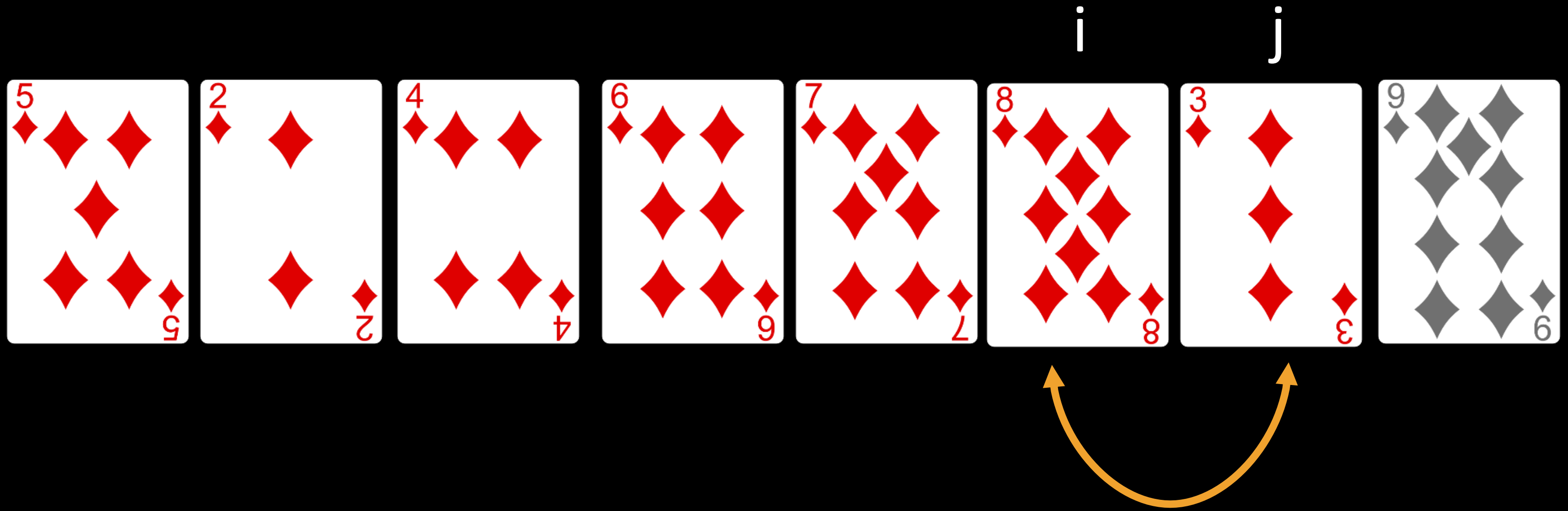
Bubble Sort Visualization

Steps count: 19 + 1 ⇨ 20



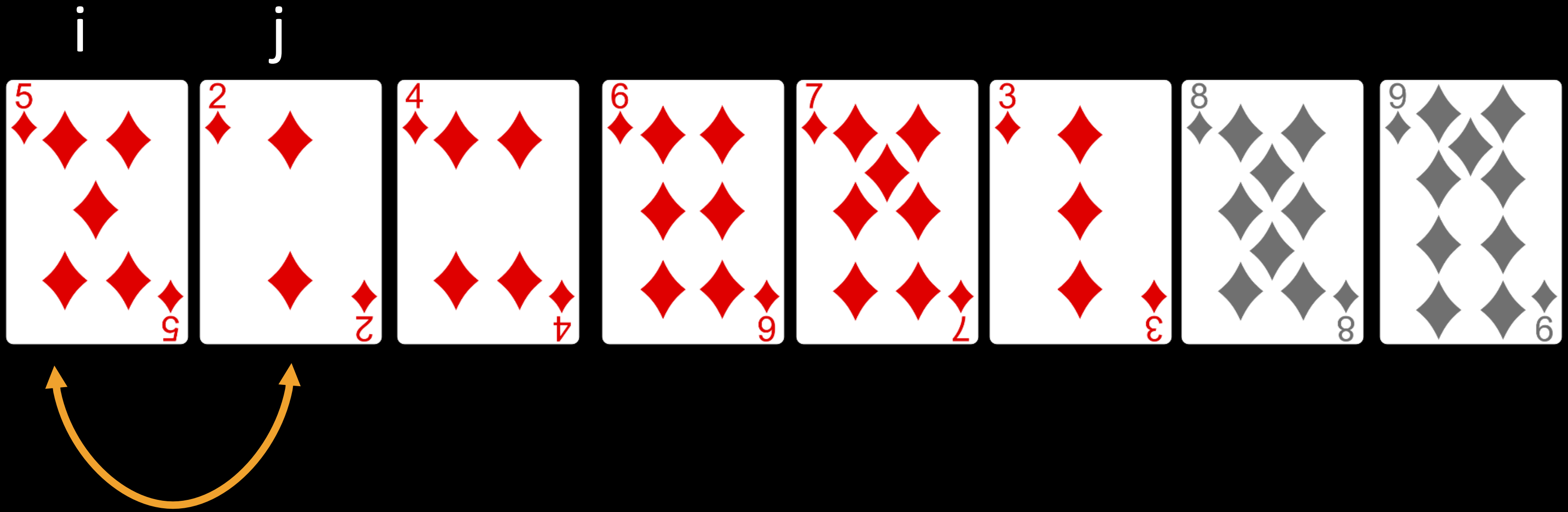
Bubble Sort Visualization

Steps count: 20 + 1 + 1 ⇨ 22



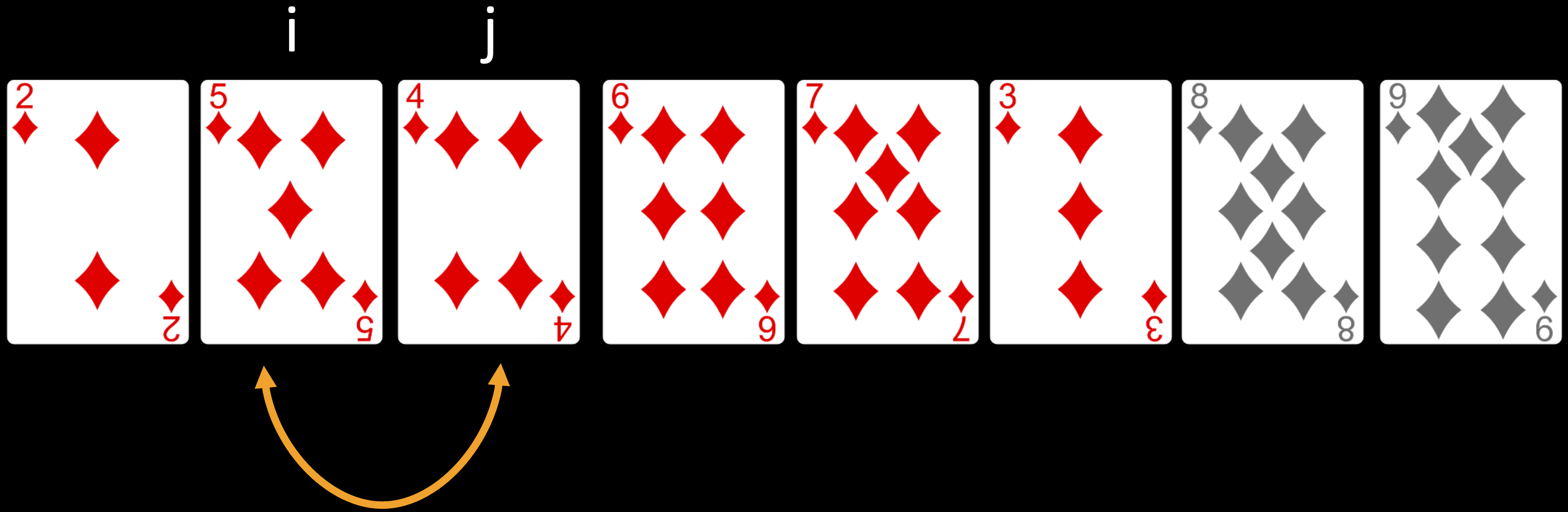
Bubble Sort Visualization

Steps count: 22 + 1 + 1 ⇨ 24



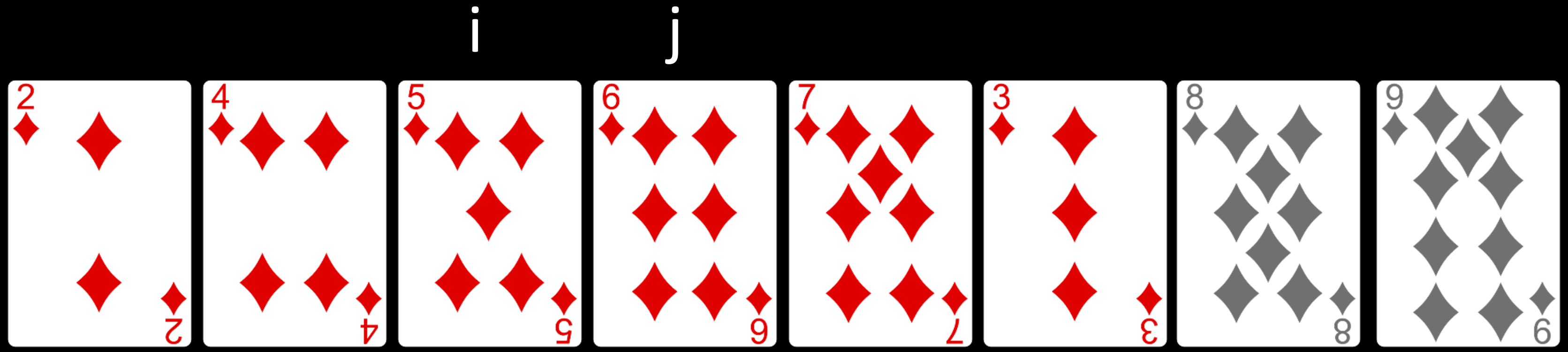
Bubble Sort Visualization

Steps count: 24 + 1 + 1 ⇨ 26



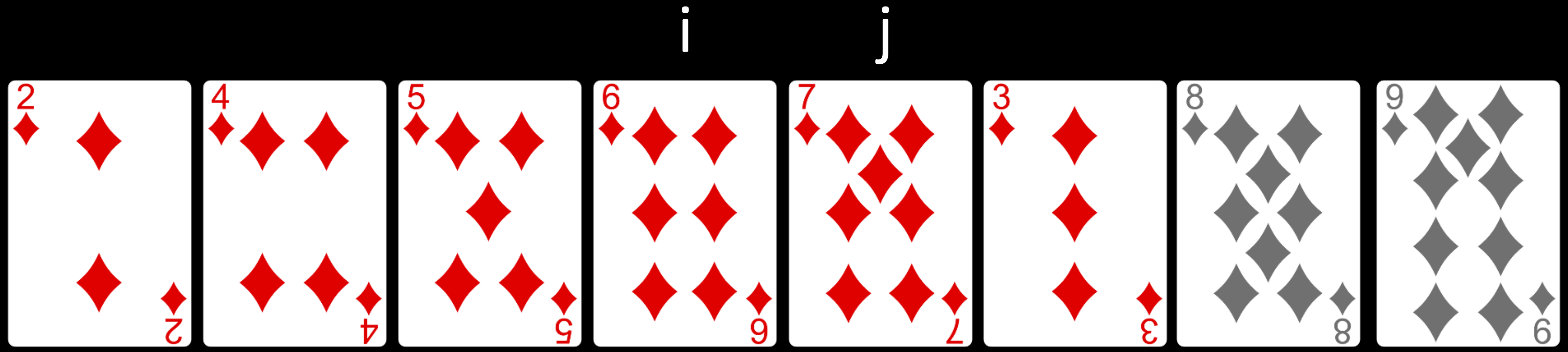
Bubble Sort Visualization

Steps count: 26 + 1 ⇨ 27



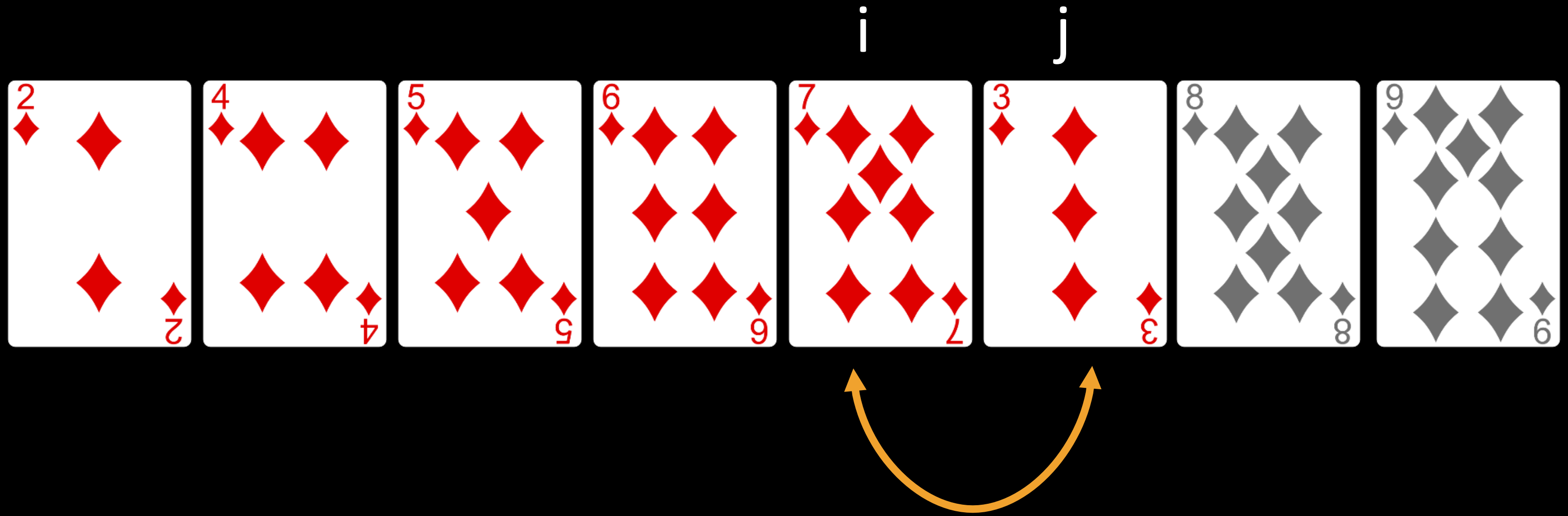
Bubble Sort Visualization

Steps count: 27 + 1 ⇨ 28



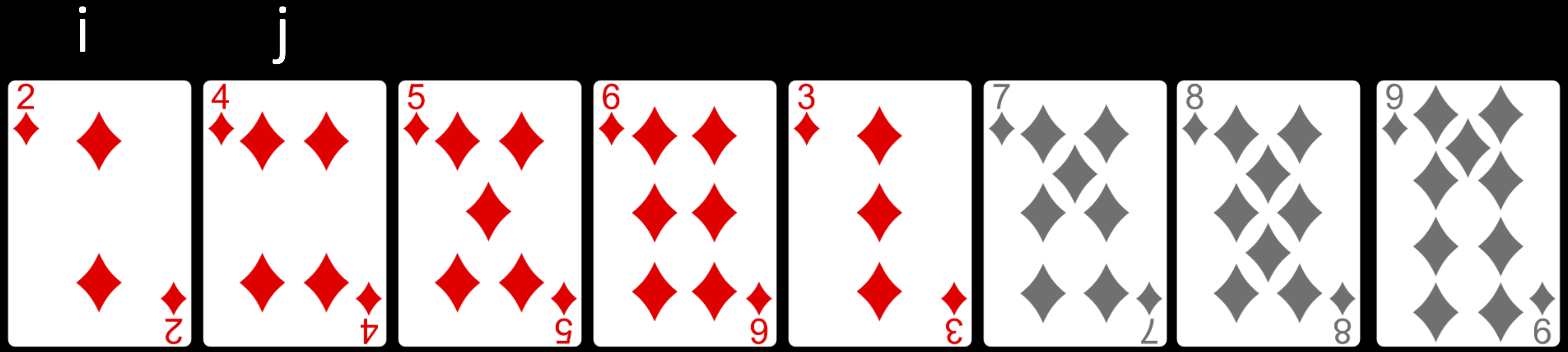
Bubble Sort Visualization

Steps count: 28 + 1 + 1 ⇨ 30



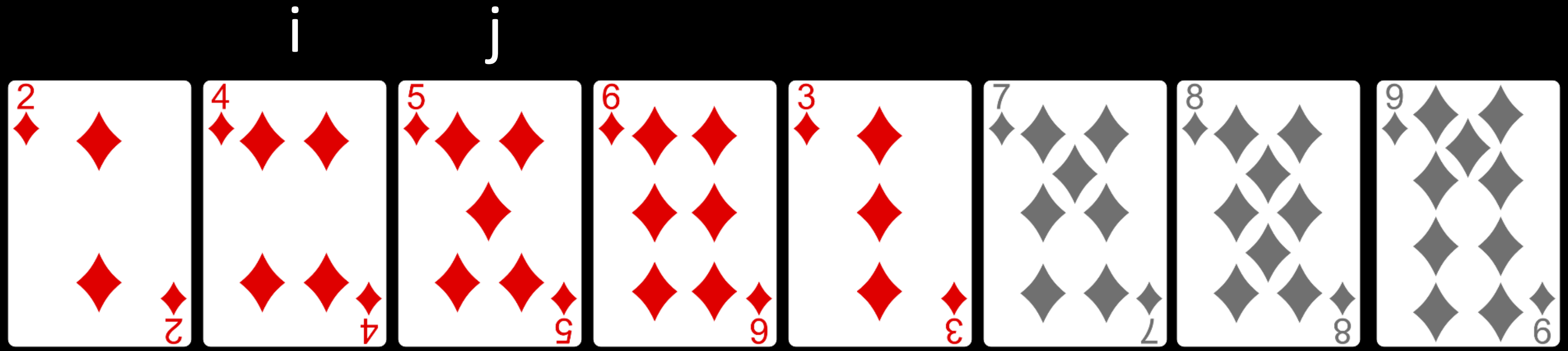
Bubble Sort Visualization

Steps count: 30 + 1 ⇨ 31



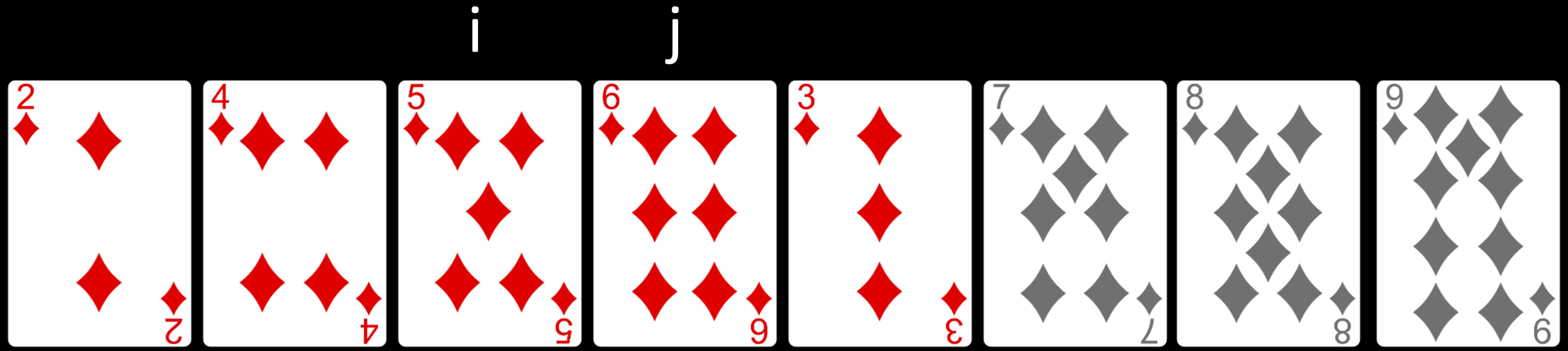
Bubble Sort Visualization

Steps count: 31 + 1 ⇨ 32



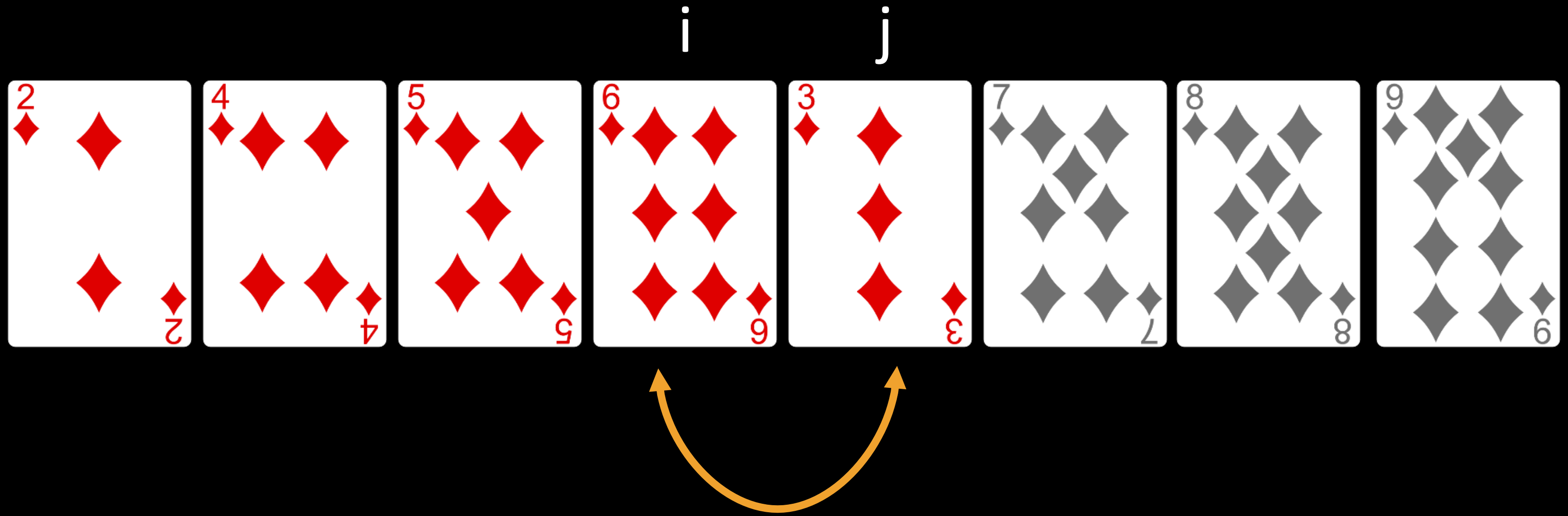
Bubble Sort Visualization

Steps count: 32 + 1 ⇨ 33



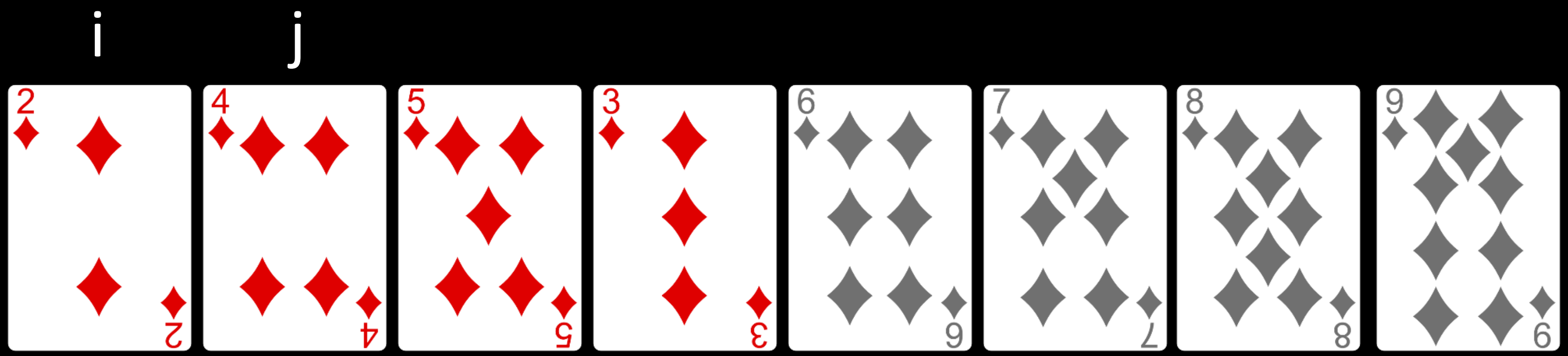
Bubble Sort Visualization

Steps count: 33 + 1 + 1 ⇨ 35



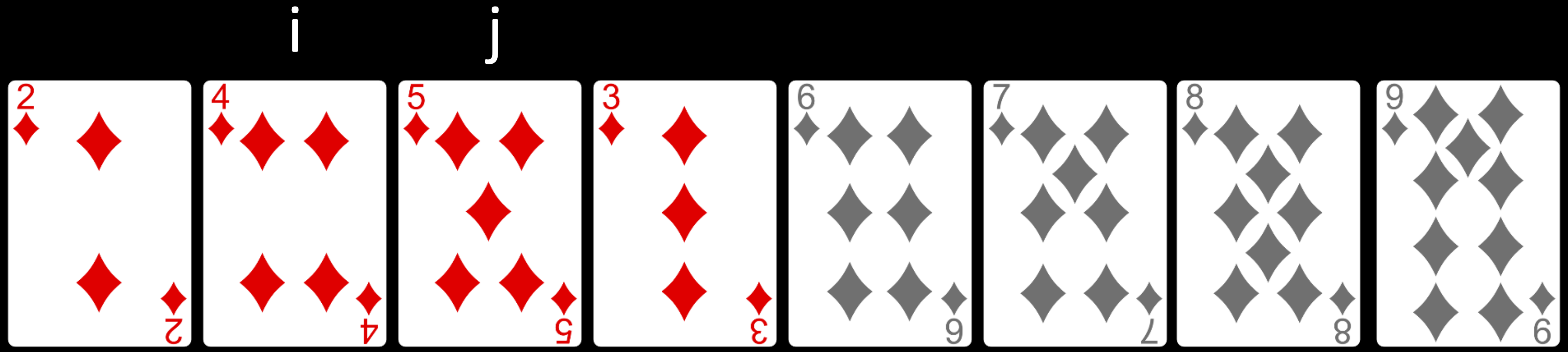
Bubble Sort Visualization

Steps count: 35 + 1 ⇨ 36



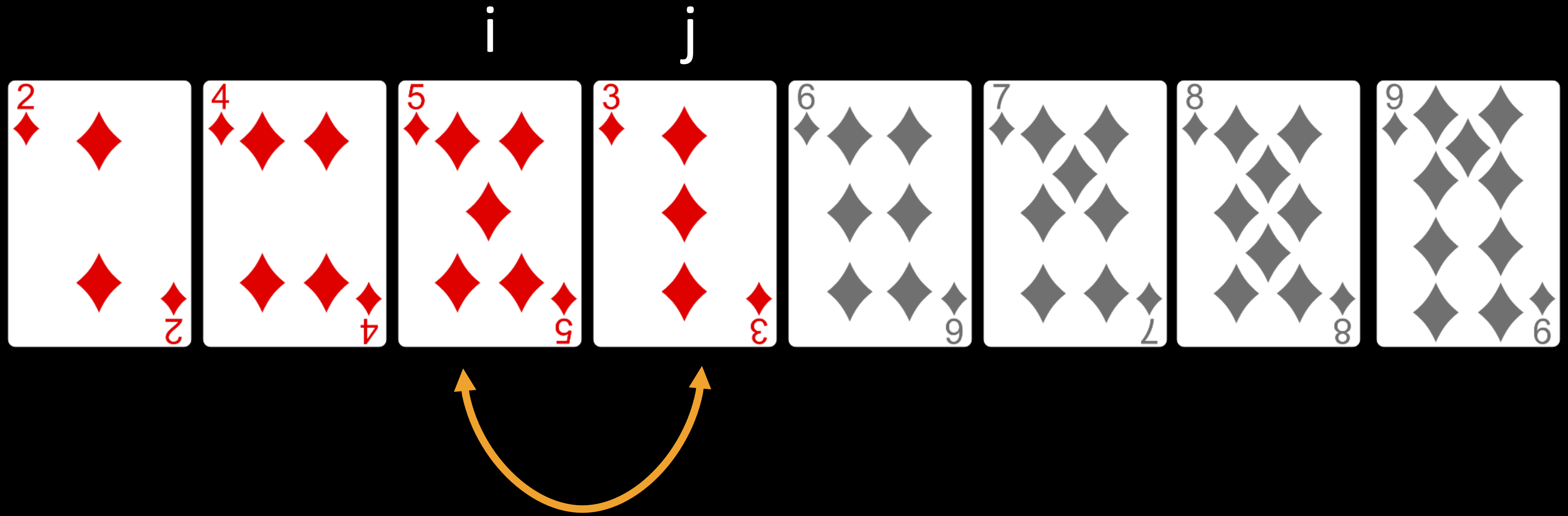
Bubble Sort Visualization

Steps count: 36 + 1 ⇨ 37



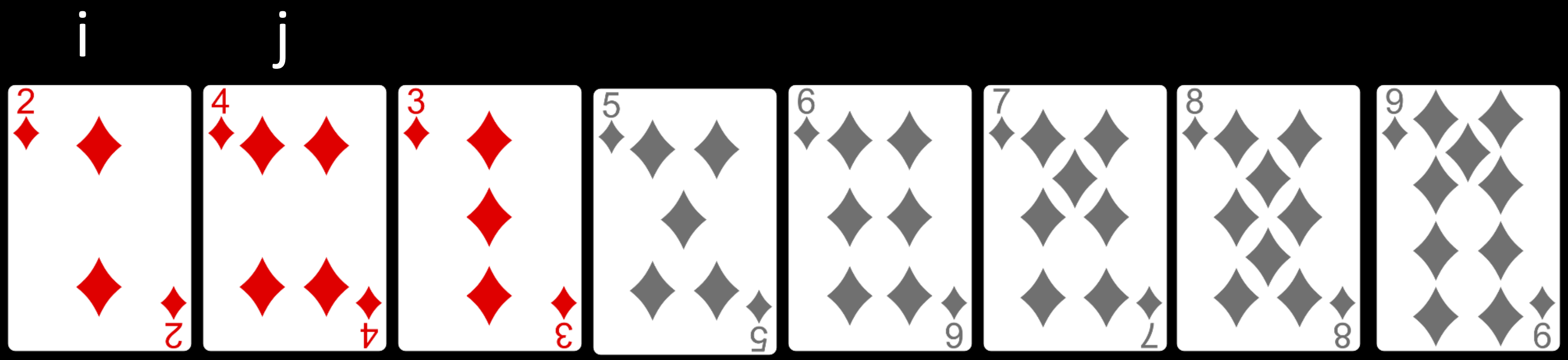
Bubble Sort Visualization

Steps count: 37 + 1 + 1 ⇨ 39



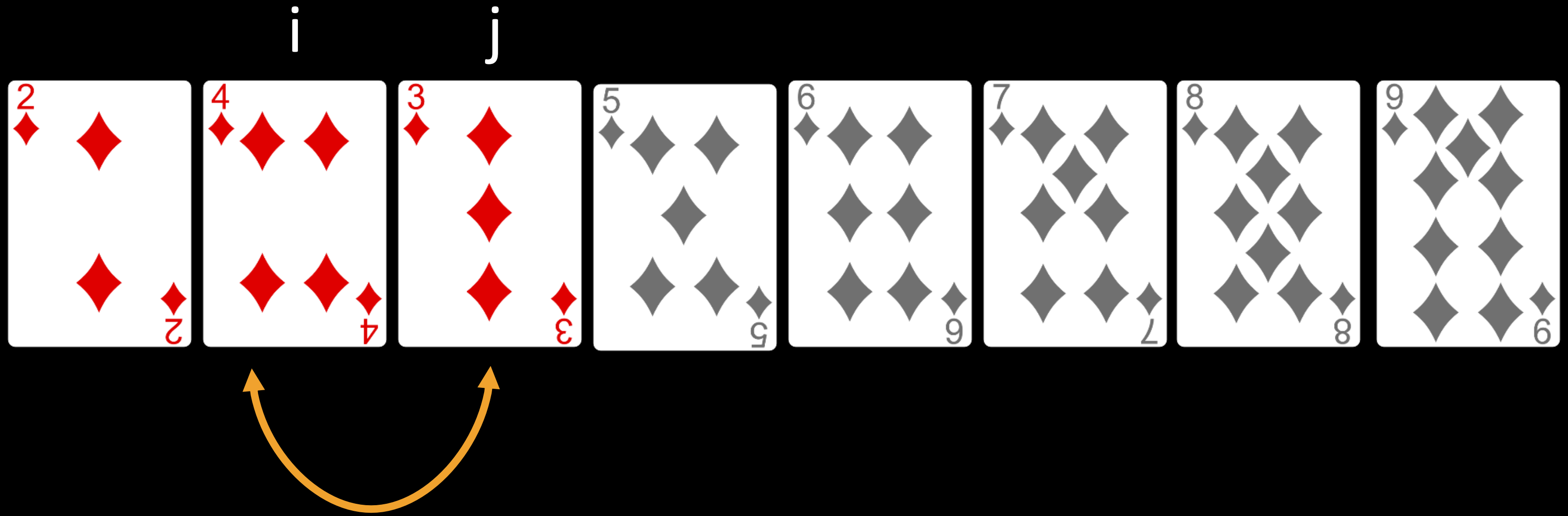
Bubble Sort Visualization

Steps count: 39 + 1 ⇨ 40



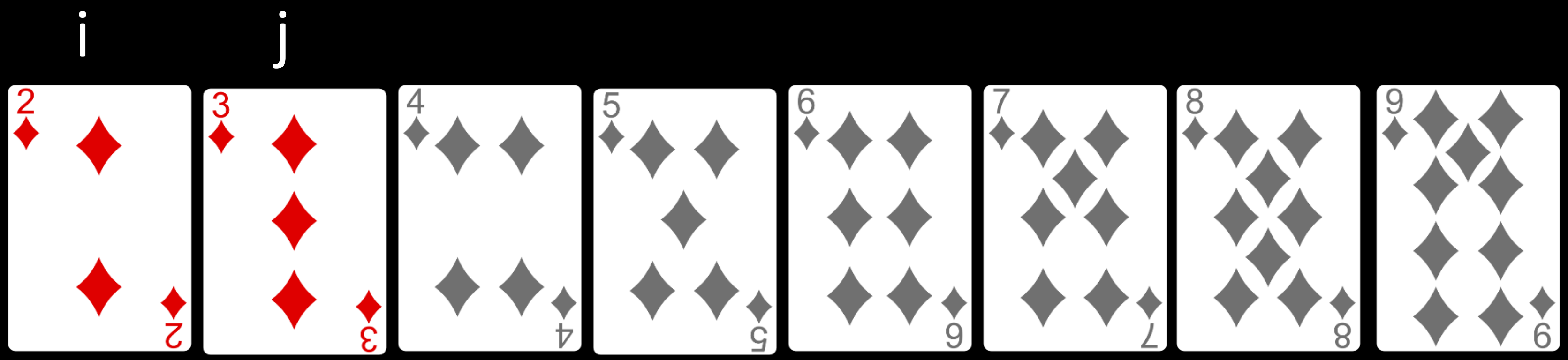
Bubble Sort Visualization

Steps count: 40 + 1 + 1 ⇨ 42



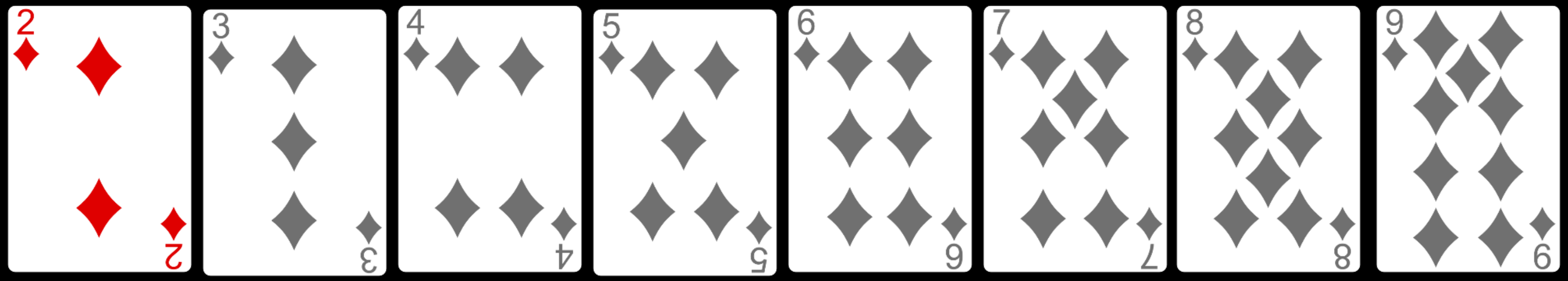
Bubble Sort Visualization

Steps count: 42 + 1 ⇨ 43



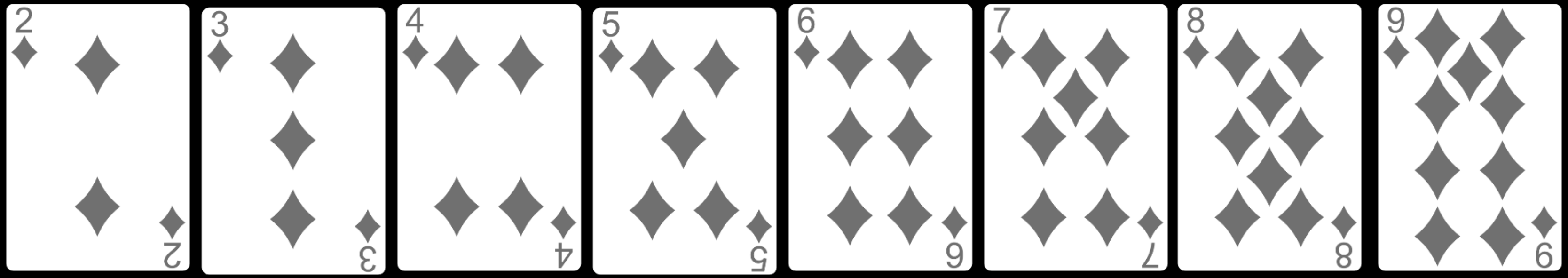
Bubble Sort Visualization

Steps count: 43 + 1 ⇨ 44



Bubble Sort Visualization

Total count of steps : 44



Bubble Sort Code

```
boolean swapped = true;
  do {
    swapped = false;
    for (int ind = 0; ind < collection.length - 1; ind++){
      if (collection[ind] > collection[ind + 1]){
        swap(collection, ind, ind + 1);
        swapped = true;
      }
    }
  } while (swapped)
}
```

Swap with next element, if its **smaller**

Stop if the collection is already sorted

Comparing Sorting Algorithms

SoftUni Foundation

Counting steps helps defininig the algoirthm‘s **efficiency**

Name	Steps Count
Selection Sort	41
Bubble Sort	44
Merge Sort	24
Quick Sort	35

The number of steps is **always similar**

There are **sorting algorithms** that can sort the **same deck** of cards with much less steps

What is algorithm complexity?

- A rough estimation of the **number of steps**
- Steps count depends on the **quantity of data** being processed
  - The **bigger** the collection, the **slower** the algorithm
  - Numbers can’t **accurately** describe it
- Instead we use **functions** to notate complexity:

n is the problem size

$$f(n) = 2n$$

Number of instructions needed in the **worst-case**, given a **n**

Comparing Sorting Algorithms (2)

Name	Complexity	n	f(n)
Selection Sort	n^2	100	≈ 100 00
Bubble Sort	n^2	100	≈ 100 00
Merge Sort	n * log(n)	100	≈ 200
Quick Sort	n * log(n)	100	≈ 200

Merge Sort and **Quick Sort** have much better **performance** when processing big amounts of data

Why should we analyze algorithms?

- The expected **running time** of an algorithm is:
  - The total number of **primitive operations** executed
  - The algorithm **efficiency**

Less steps == higher **efficiency**

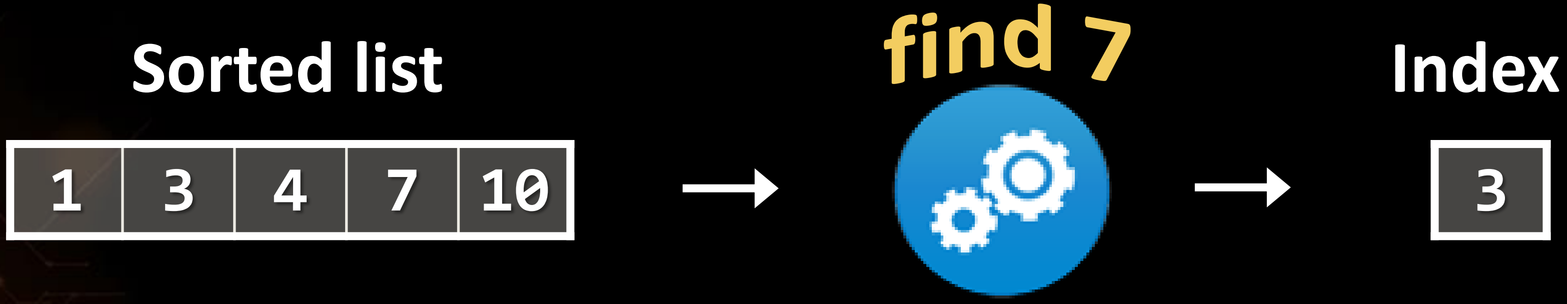
- Predict the **resources** the algorithm will need
  - Computational time (**CPU** consumption)
  - **Hard disk** operations

Searching Algorithms

Linear, Binary and Interpolation

Searching Algorithm

- **Searching algorithm** == an algorithm for finding an item with specified properties among a collection of items
  - Returns the **index** of the item
  - Returns **-1** if the element is not present



Sorted list

1	3	4	7	10

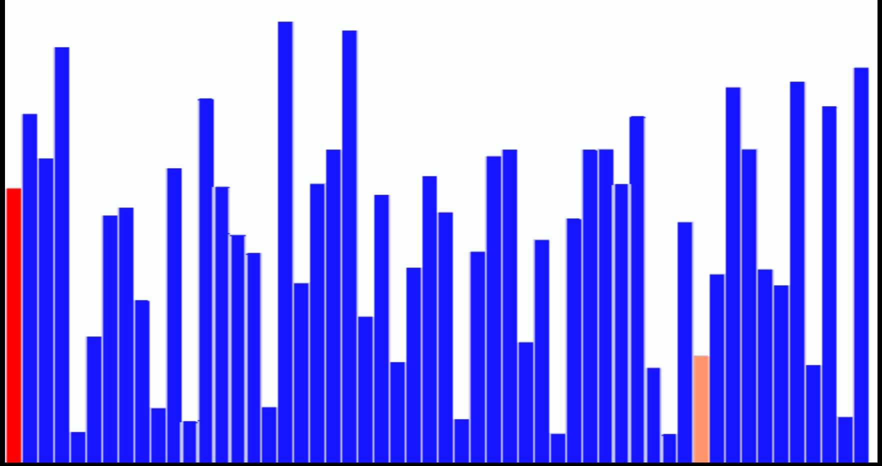
find 7

Index

3

Linear Search

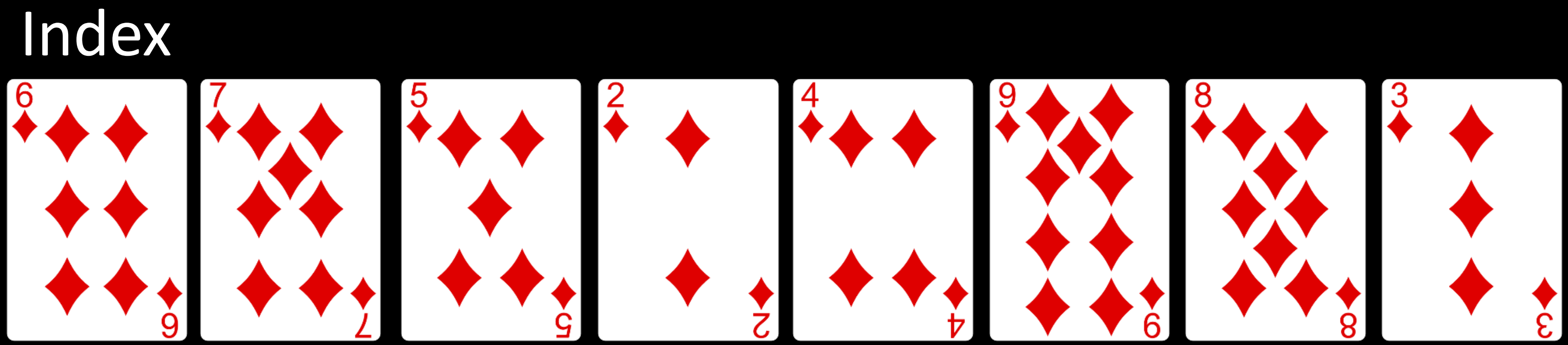
- Linear search finds an item within a **unordered data structure**
- Check every element
  - One at a time, in sequence
- Stop if the desired one is found
- Visualize



```
for each item in the list:
  if that item has the desired value
    return the item's index
return -1
```

Linear Search Visualization

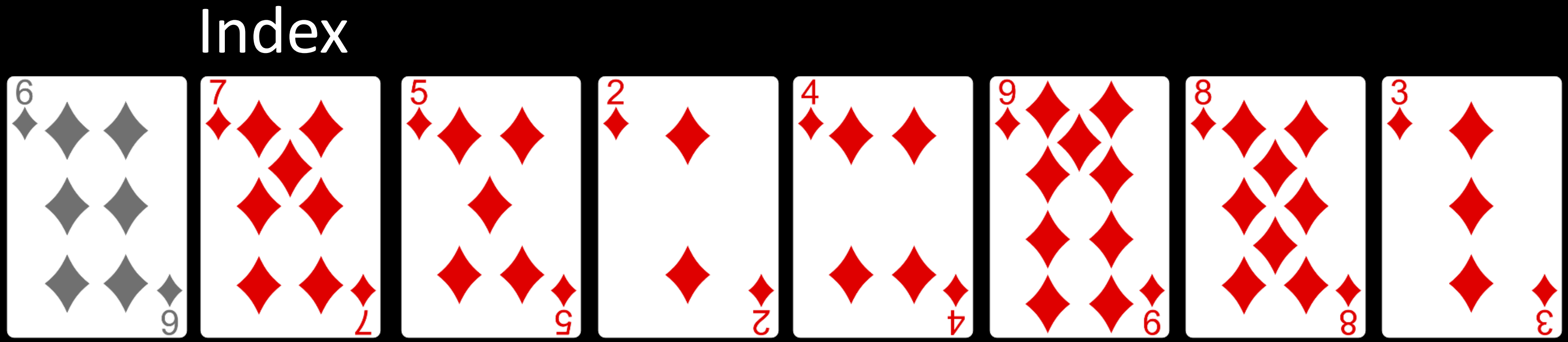
Steps count: 1



Look for **9**

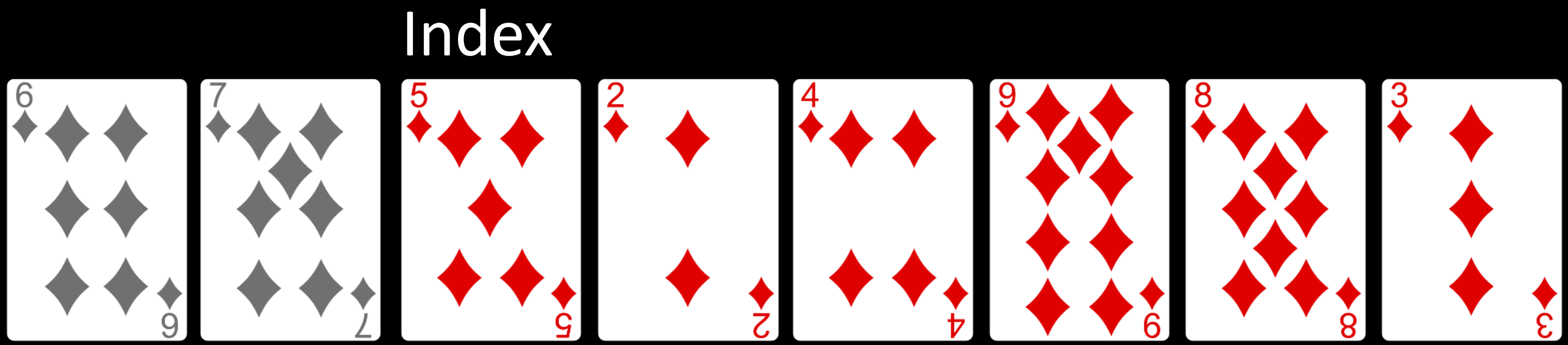
Linear Search Visualization

Steps count: 2



Linear Search Visualization

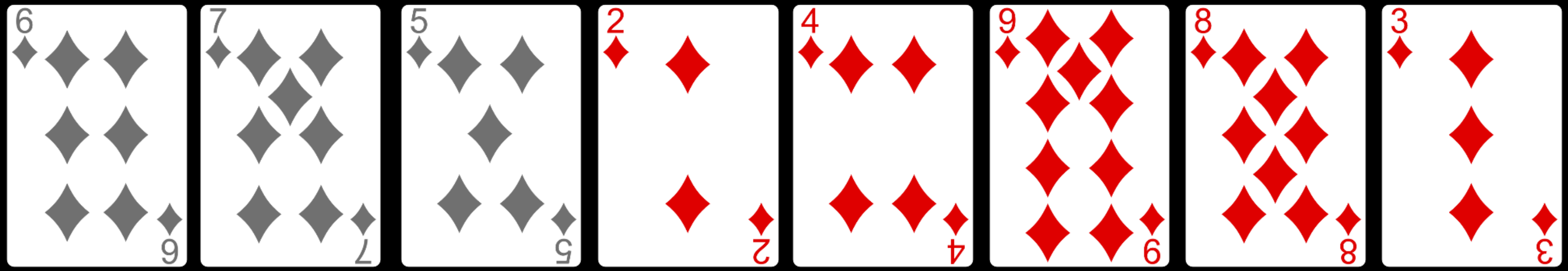
Steps count: 3



Linear Search Visualization

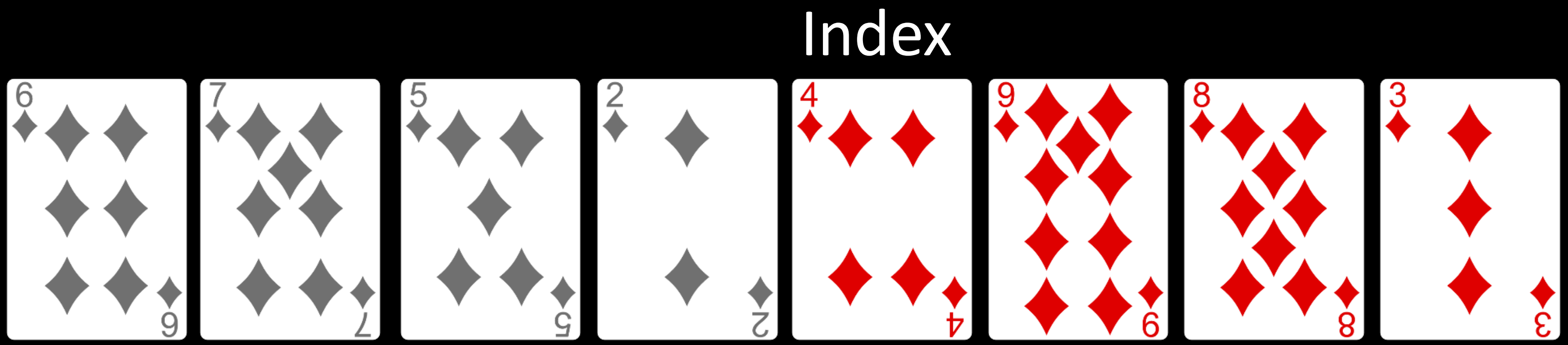
Steps count: 4

Index



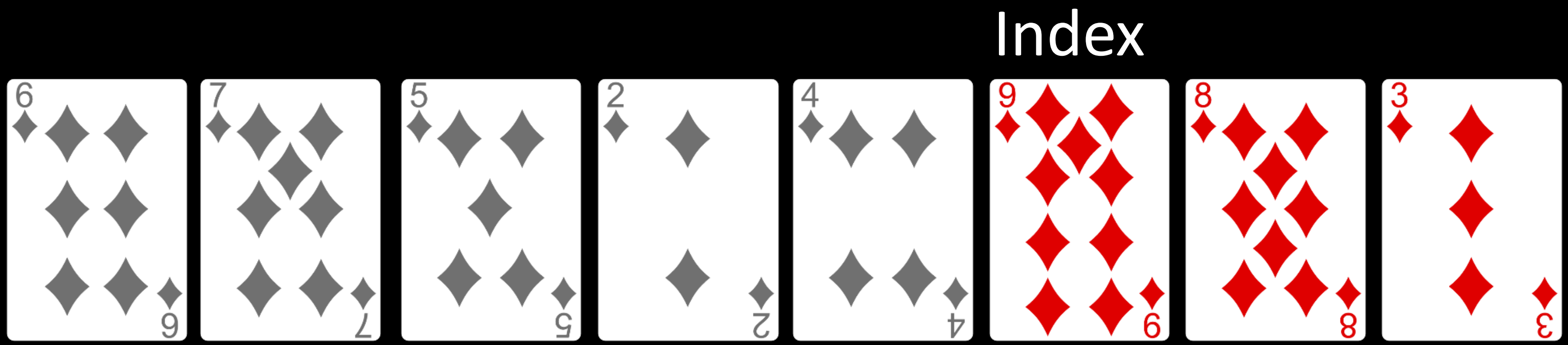
Linear Search Visualization

Steps count: 5



Linear Search Visualization

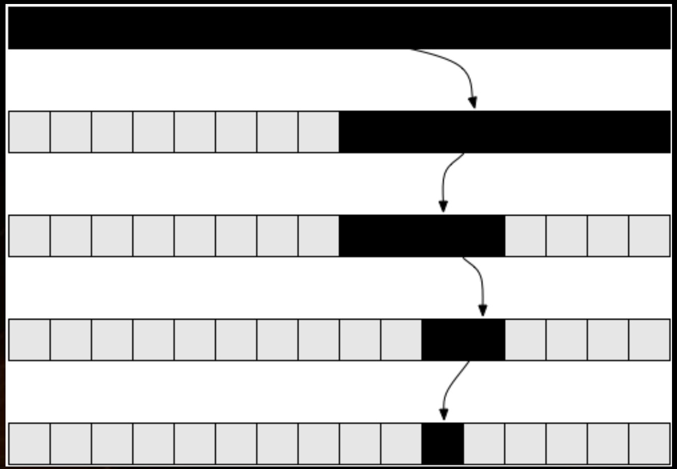
Steps count: 6 ➡ Total count of steps : 6

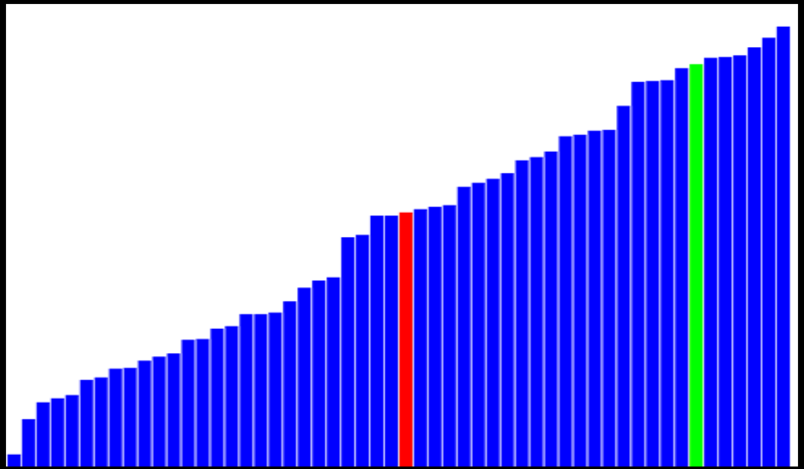


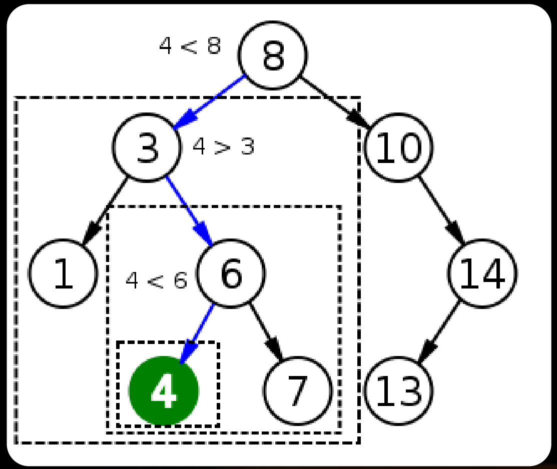
Found 9

Binary Search

- Binary search finds an item within a **ordered data structure**
- At each step, compare the input with the middle element
    - The algorithm repeats its action to the left or right sub-structure
- Visualization

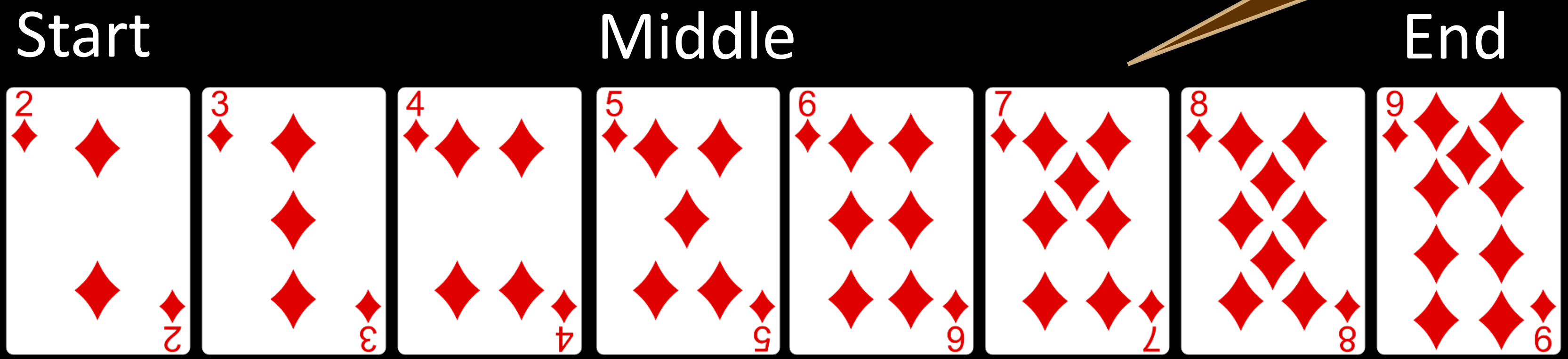






Binary Search Visualization

Steps count: 1



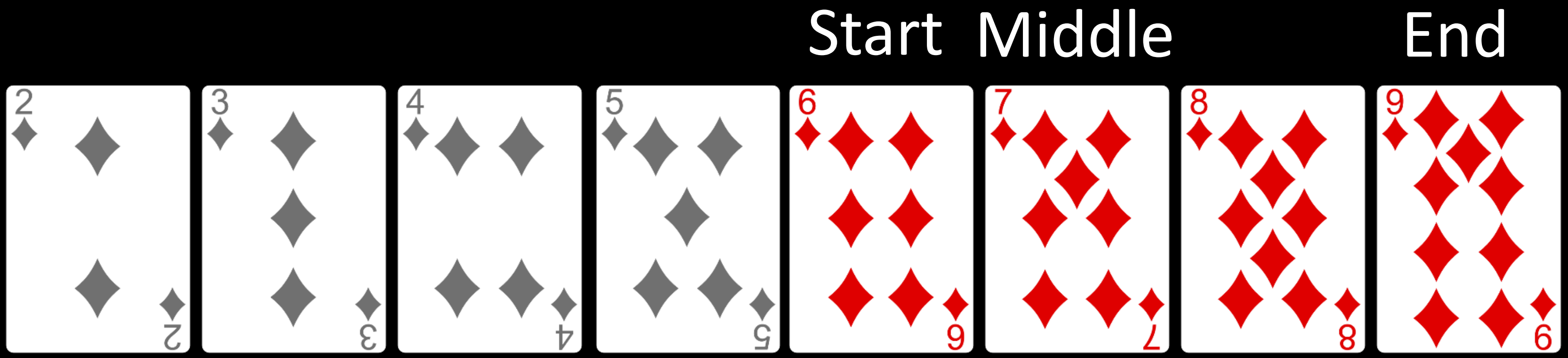
The deck is **sorted**

9 > 5 ⇨ search in the **right half**

Look for **9**

Binary Search Visualization

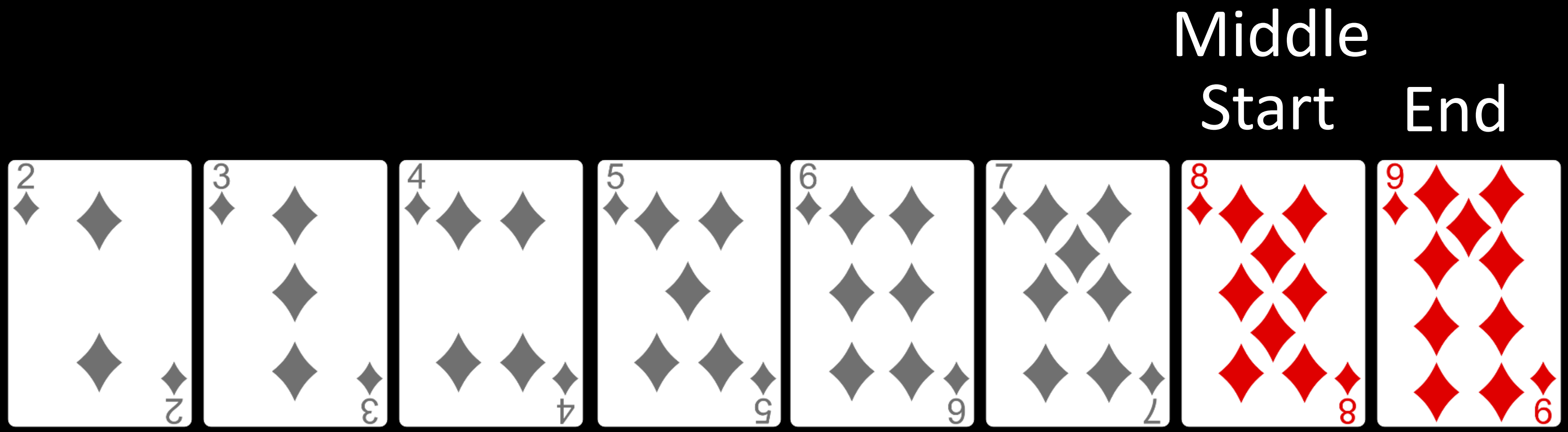
Steps count: 2



9 > 7 ⇨ search in the **right half**

Binary Search Visualization

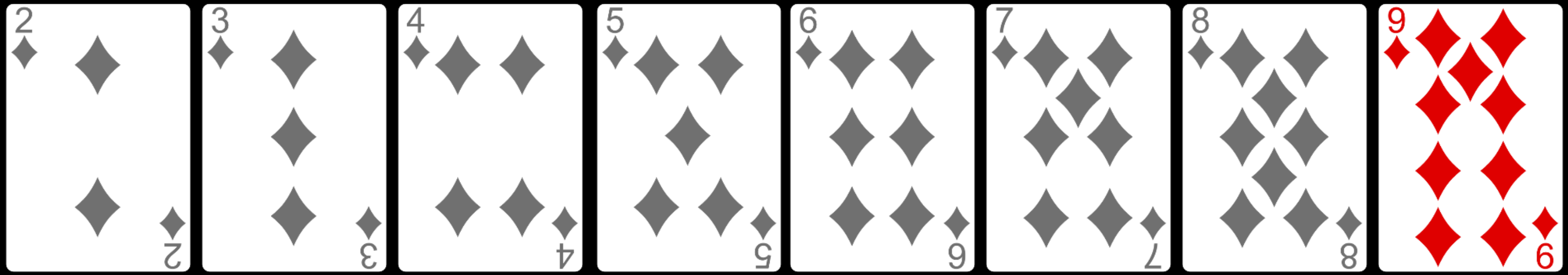
Steps count: 3



9 > 8 ⇨ search in the **right half**

Binary Search Visualization

Steps count: 4



9 == 9 ⇨ return the index 7

Binary Search

SoftUni Foundation

```
int binarySearch(int arr[], int key, int start, int end) {
  while (end >= start) {
    int mid = (start + end) / 2;
    if (arr[mid] > key)
      end = mid - 1;
    else if (arr[mid] < key)
      start = mid + 1;
    else
      return mid;
  }
  return KEY_NOT_FOUND;
}
```

Search in the left half of the collection

Search in the right half of the collection

Comparing Searching Algorithms

We need go trough **every** element

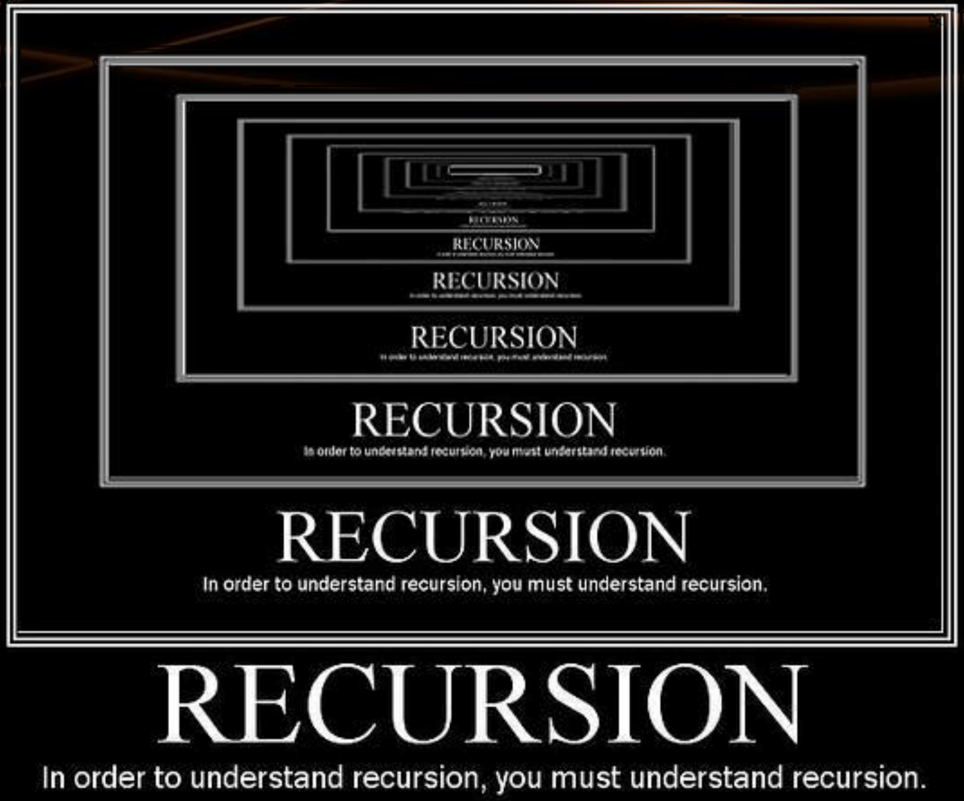
Name	Complexity	n	f(n)
Linear Search	n^2	100	≈ 100
Binary Search	n * log(n)	100	≈ 6,64

Binary search can also be implemented **iteratively** and **recursively**

On each step we **halve** the collection

Practice: Sorting and Searching Algorithms

Exercises in class (Lab)



RECURSION
In order to understand recursion, you must understand recursion.
RECURSION
In order to understand recursion, you must understand recursion.
RECURSION
In order to understand recursion, you must understand recursion.
RECURSION
RECURSION
RECURSION

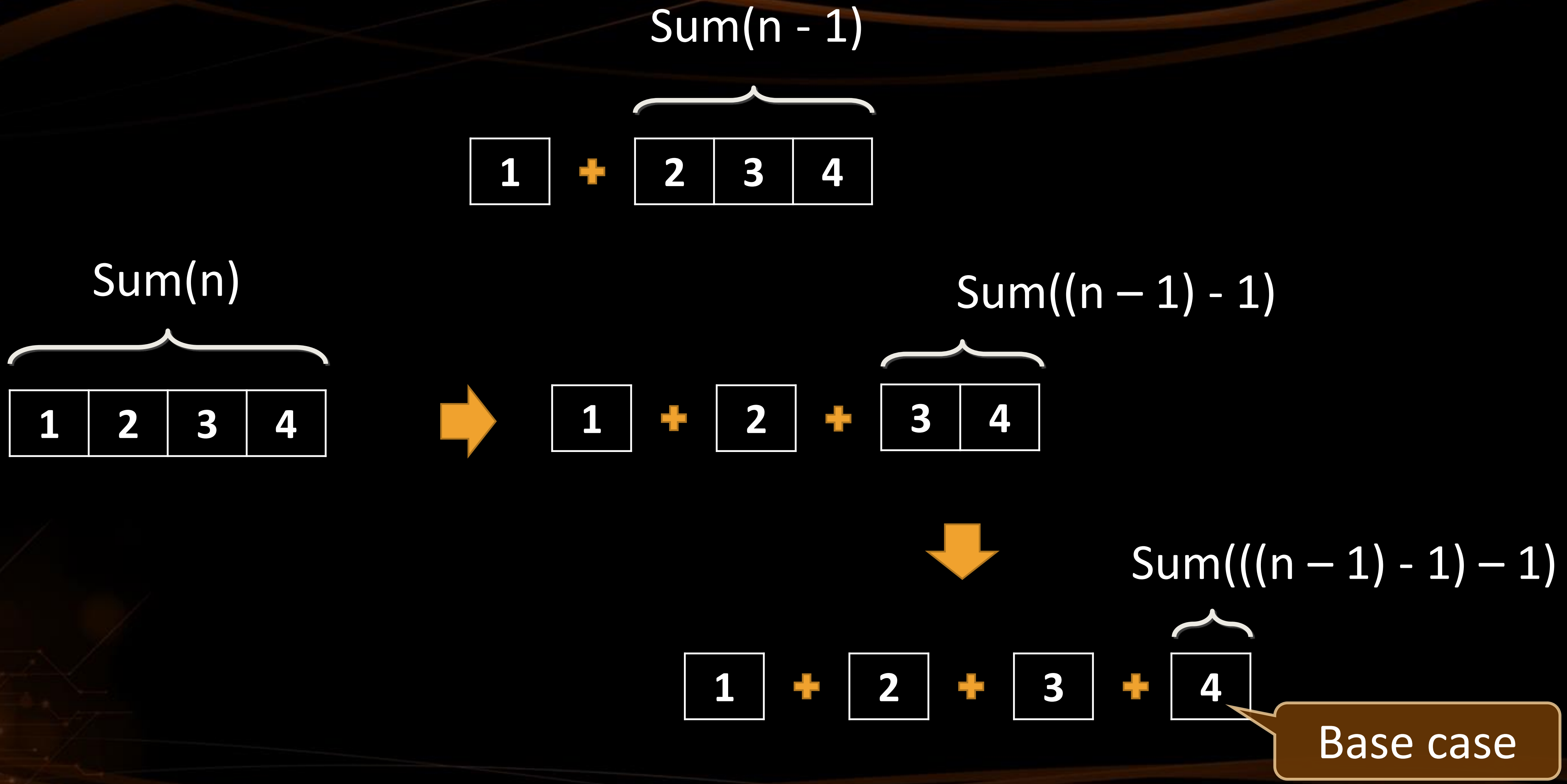
RECURSION

In order to understand recursion, you must understand recursion.

Recursion

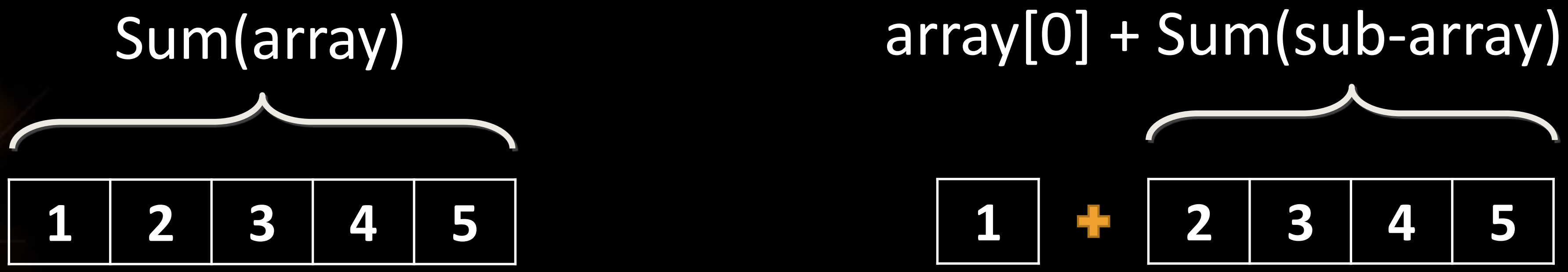
Recursive Algorithms

Array Sum – Example



What is Recursion?

- Problem solving technique
- Divides a problem into **subproblems of the same type**
  - Involves a **function calling itself**
  - The function should have a **base case**
  - **Each step** of the recursion should **move towards** the **base case**



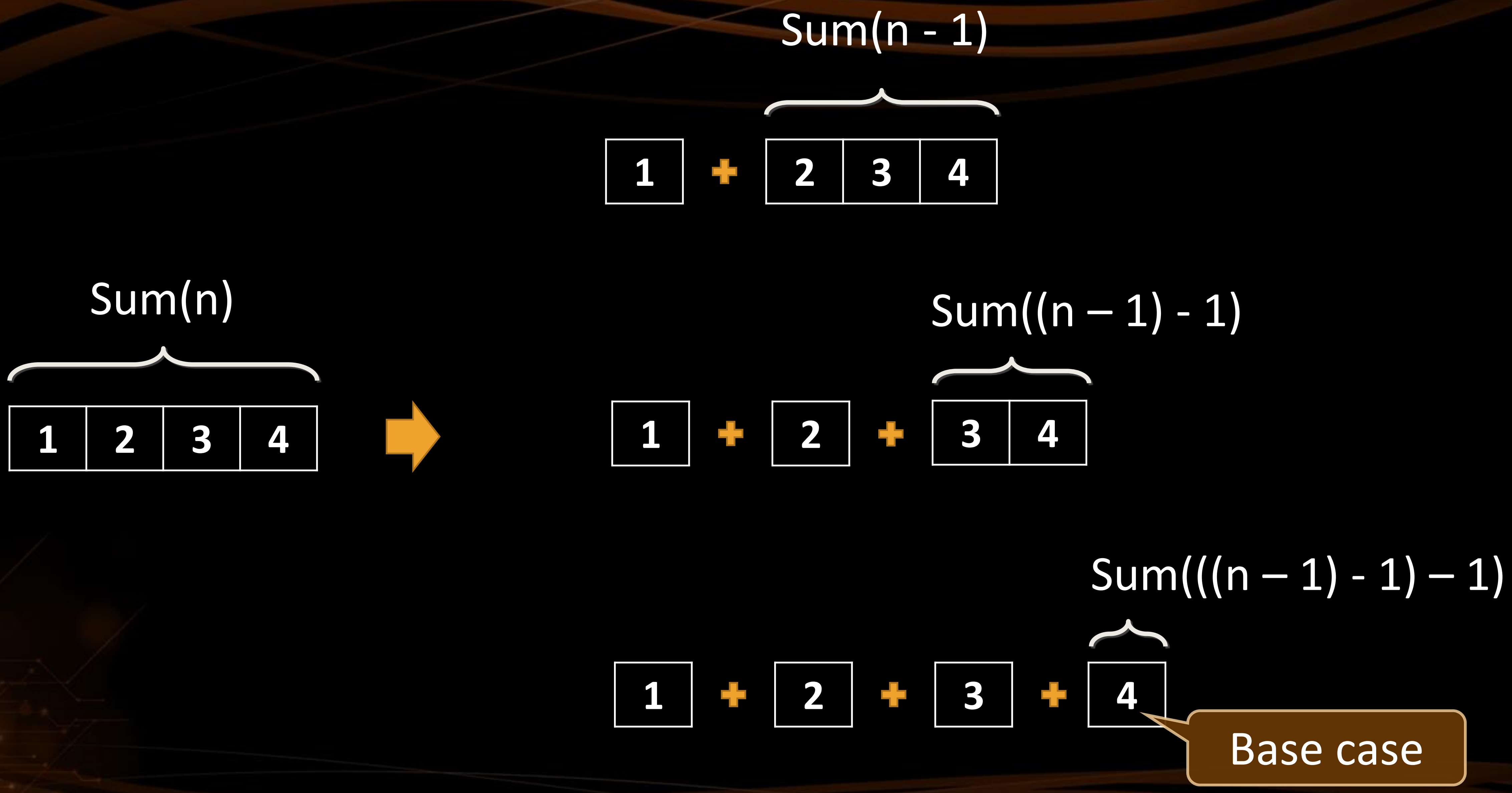
Sum(array)

1 2 3 4 5

array[0] + Sum(sub-array)

1 + 2 3 4 5

Array Sum – Example



Sum(n - 1)

1 + 2 3 4

Sum(n)

1 2 3 4

Sum((n – 1) - 1)

1 + 2 + 3 4

Sum(((n – 1) - 1) – 1)

1 + 2 + 3 + 4

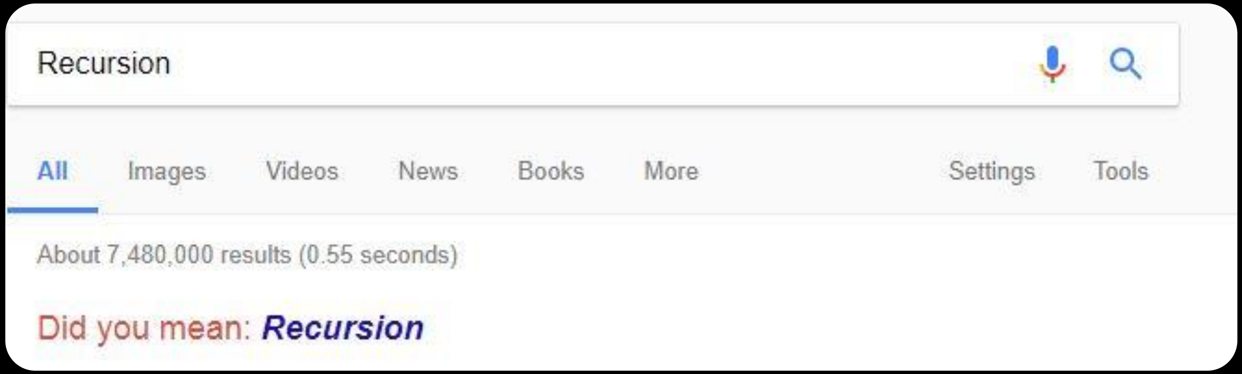
Base case

Problem: Array Sum

- Create a **recursive method** that
  - Reads numbers from the console and stores them in an **`int[] array`**
  - Finds the **sum** of all numbers

`1 2 3 4` → `10`

`-1 0 1` → `0`


Recursion

About 7,480,000 results (0.55 seconds)

Did you mean: ***Recursion***

Check your solution here: https://judge.softuni.bg/Contests/779

Solution: Array Sum

```
static int sum(int[] array, int index){
  if (index == array.length - 1)
  {
    return array[index];
  }

  return array[index] + sum(array, index + 1);
}
```

Base case

Check your solution here: https://judge.softuni.bg/Contests/779

Problem: Recursive Factorial

- Create a **recursive method** that calculates **n!**
- Recursive definition of **n!**:

```
n! = n * (n-1)! for n > 0
```

  - 5! = 5 * 4!
    - 4! = 4 * 3!
      - 3! = 3 * 2!
        - 2! = 2 * 1!
          - 1! = 1 * 0!

0! = 1

```
5 → 120
```

```
10 → 3628800
```

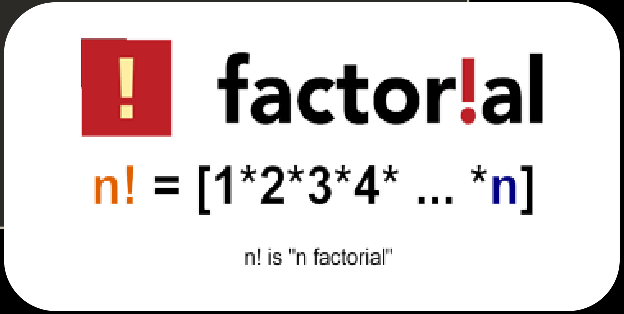
Check your solution here: https://judge.softuni.bg/Contests/779

Solution: Recursive Factorial

```
static long factorial(int num){
  if (num == 0)
  {
    return 1;
  }

  return num * factorial(num - 1);
}
```

Base case



Check your solution here: https://judge.softuni.bg/Contests/779

Recursion Pre-Actions and Post-Actions

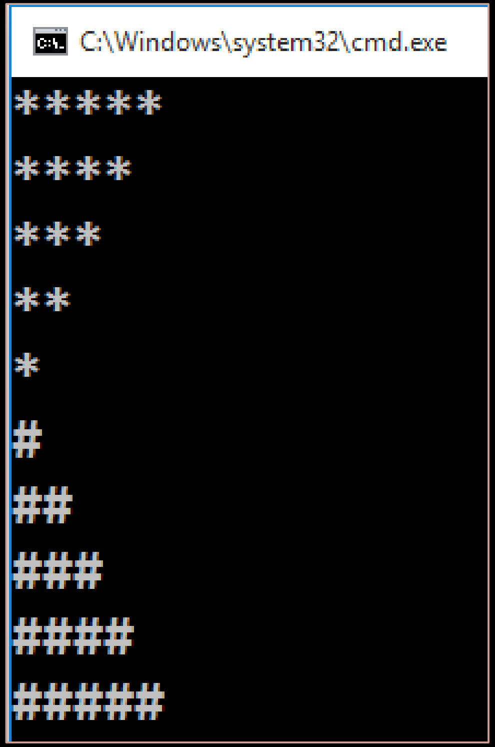
- Recursive methods have 3 parts:
  - **Pre-actions** (before calling the recursion)
  - **Recursive calls** (step-in)
  - **Post-actions** (after returning from recursion)

```
static void Recursion(){
  // Pre-actions
  Recursion();
  // Post-actions
}
```

Problem: Recursive Drawing

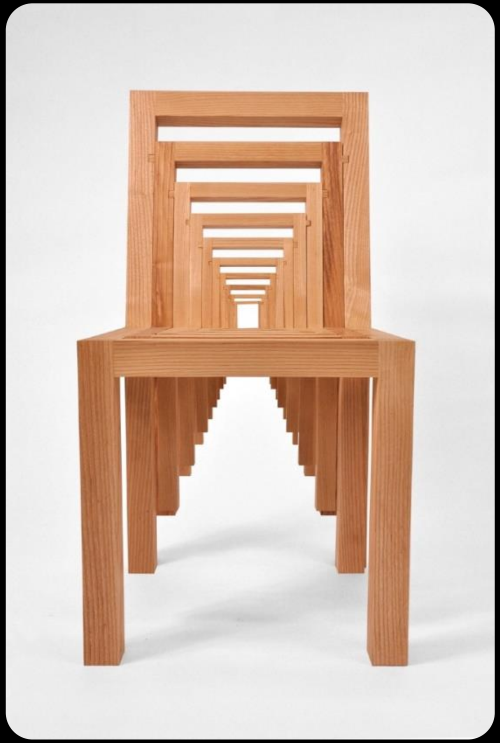
- Create a **recursive method** that draws the following figure

```
5
```



C:\Windows\system32\cmd.exe

```
*****
****
***
**
*
#
##
###
####
#####
```



Check your solution here: https://judge.softuni.bg/Contests/779

Solution: Recursive Drawing

```
static void printFigure(int n)
    if (n == 0) // Bottom of the recursion
      return;
    // Pre-action: print n asterisks
    System.out.println(
       String.join("", Collections.nCopies(n, "*")));

    // Recursive call: print figure of size n-1
    printFigure(n - 1);

    // Post-action: print n hashtags
    System.out.println(
         String.join("", Collections.nCopies(n, "#")));
```

Returns a **String** consisting of n copies of '*'.

Check your solution here: https://judge.softuni.bg/Contests/779

Performance: Recursion vs. Iteration

- Recursive calls are **slightly slower** than iteration
  - Parameters and return values travel through the stack at each step
  - Prefer iteration for linear calculations (**without branched calls**)

Recursive factorial:

```
static long fact(int n){
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
```

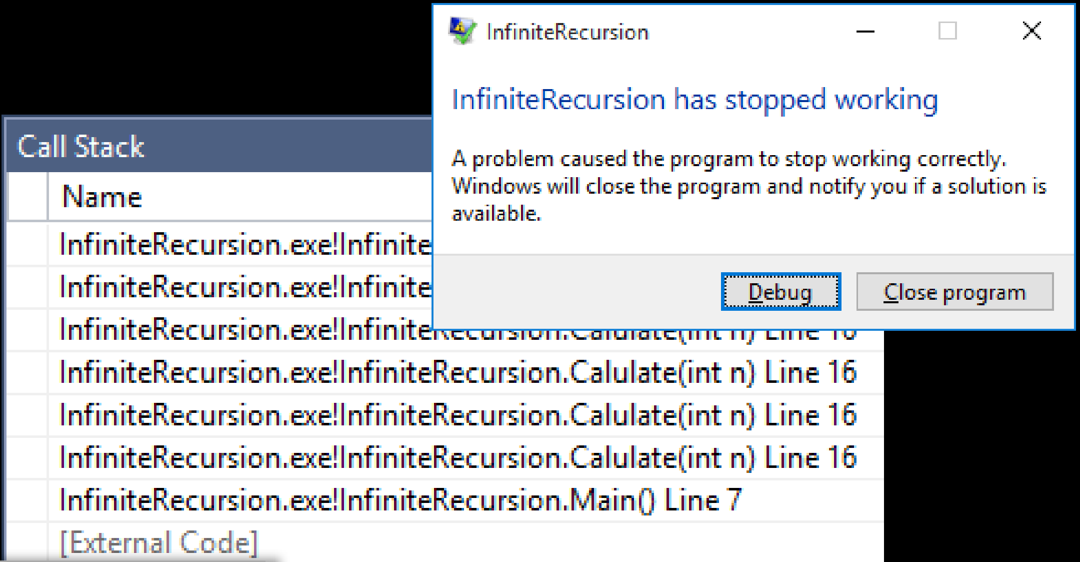
Iterative factorial:

```
static long iterFact(int num){
  long result = 1;
  for (int i = 1; i <= n; i++)
    result *= i;
  return result;
}
```

Infinite Recursion

- **Infinite recursion** == a method calls itself **infinitely**
  - Typically, infinite recursion is a bug in the program
  - The bottom of the recursion is missing or wrong
  - Causes "**stack overflow**" exception

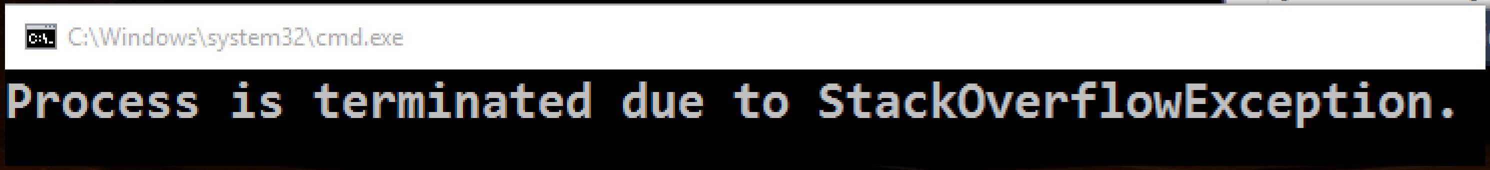
```
static long Calulate(int n){
    return Calulate(n + 1);
}
```



InfiniteRecursion has stopped working

A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available.

```
InfiniteRecursion.exe!Infinite
InfiniteRecursion.exe!Infinite
InfiniteRecursion.exe!Infinite
InfiniteRecursion.exe!InfiniteRecursion.Calulate(int n) Line 16
InfiniteRecursion.exe!InfiniteRecursion.Calulate(int n) Line 16
InfiniteRecursion.exe!InfiniteRecursion.Calulate(int n) Line 16
InfiniteRecursion.exe!InfiniteRecursion.Main() Line 7
[External Code]
```



```
Process is terminated due to StackOverflowException.
```

Practice: Recursion

Exercises in class (Lab)

Summary

- **Sorting** == an algorithm that rearranges elements in a list
  - In non-decreasing order
- **Searching** == an algorithm for finding an item among a collection of items
- **Recursion** means to **call a method from itself**
  - It should always have a **bottom**
  - **Each step** should **move towards** the **bottom**

Data Representation and Manipulation

SoftUni Foundation

xssoftware

SmartIT

NETPEAK
SEO and PPC for Business

Questions?

SUPERHOSTING®.BG

INDEAVR
Serving the high achievers

telenor

SOFTWARE GROUP

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

https://softuni.bg/opencourses/algorithms

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

SoftUni Foundation

FORUM