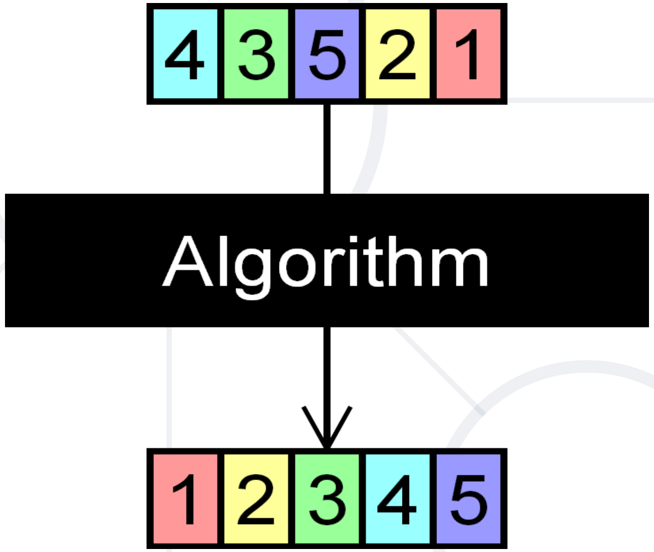
Sorting and Searching Algorithms

Simple and Advanced Sorting and Searching Algorithms



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

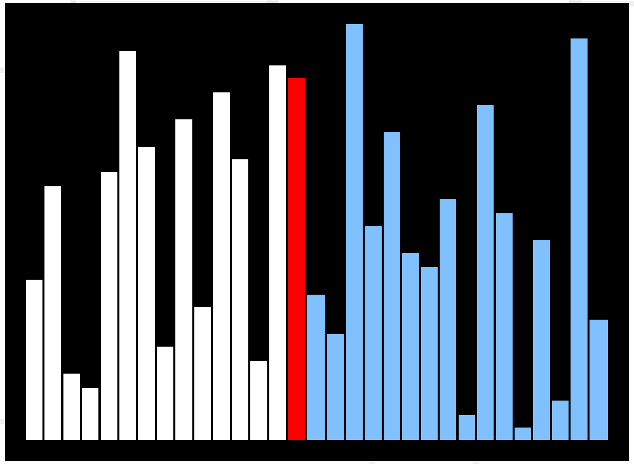
https://about.softuni.bg

Table of Contents

Selection Sort and Bubble Sort

What is a Sorting Algorithm?

- **Sorting algorithm**
  - An algorithm that rearranges elements in a list
    - In non-decreasing order
  - Elements must be **comparable**
- More formally
  - The **input** is a sequence / list of elements
  - The **output** is a rearrangement / **permutation** of elements
    - In non-decreasing order



Sorting – Example

- Efficient **sorting algorithms** are important for:
  - Producing human-readable output
  - Canonicalizing data – making data uniquely **arranged**
  - In conjunction with other algorithms, like **binary searching**
- Example of sorting:

Unsorted list

10	3	7	3	4

→ sorting →

Sorted list

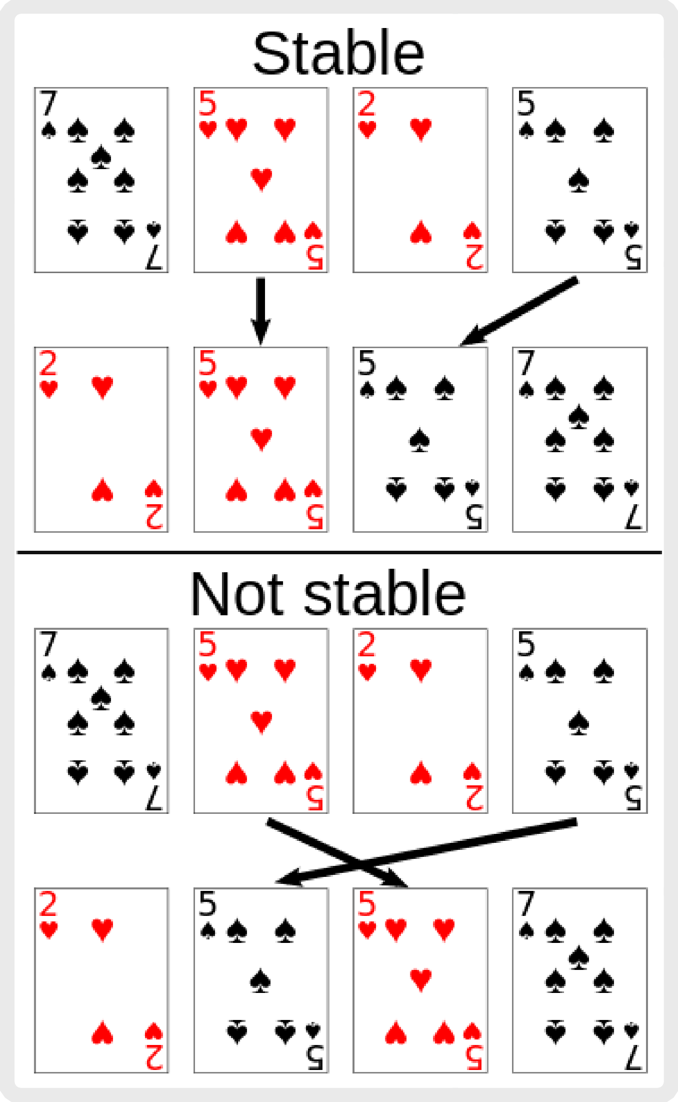
3	3	4	7	10

Sorting Algorithms: Classification

- **Sorting algorithms** are often classified by:
  - Computational **complexity** and memory usage
    - Worst, average and best-case behavior
  - **Recursive** / non-recursive
  - **Stability** – stable / unstable
  - **Comparison-based** sort / non-comparison based

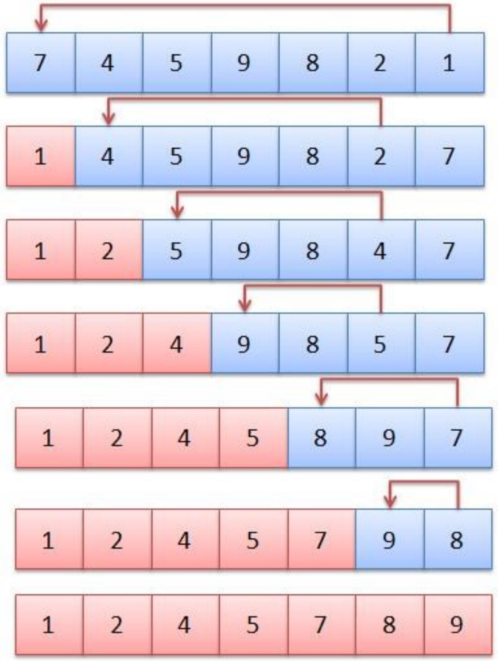
Stability of Sorting

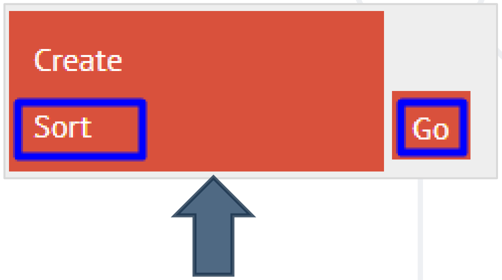
- **Stable** sorting algorithms
  - Maintain the order of equal elements
  - If two items compare as equal, their relative order is preserved
- **Unstable** sorting algorithms
  - Rearrange the equal elements in unpredictable order
- Often **different elements** have **same key** used for equality comparing

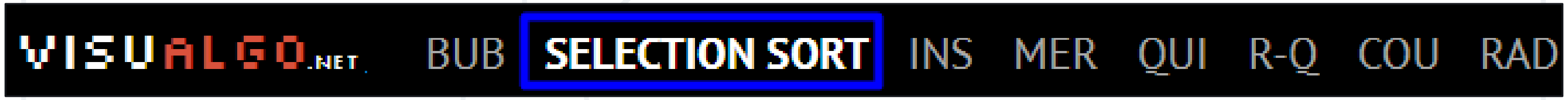


Selection Sort

- **Selection sort** – simple, but inefficient algorithm
  - Swap the first with the min element on the right, then the second, etc.
  - Memory: **O(1)**, Time: **O(n²)**, Stable: **No**
  - Method: Selection
  - See the visualization: https://visualgo.net/en/sorting → choose **Selection**:







Selection Sort: Code

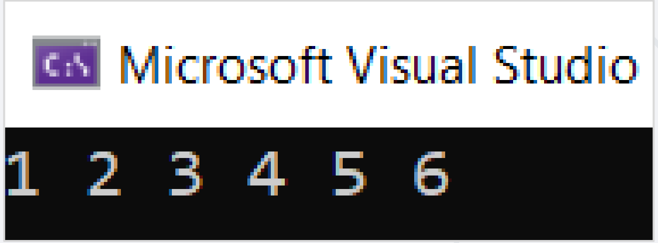
```
var nums = new[] { 1, 3, 4, 2, 5, 6 };

for (int start = 0; start < nums.Length - 1; start++)
{
    // posMin is position of min, set to current array index
    int posMin = start;

    for (int next = start + 1; next < nums.Length; next++)
        if (nums[next] < nums[posMin])
            posMin = next;

    // if posMin no longer equals i → a smaller value was found →
    // a swap must occur
    if (posMin != start)
        Swap(nums, posMin, start);
}

Console.WriteLine(string.Join(" ", nums));
```



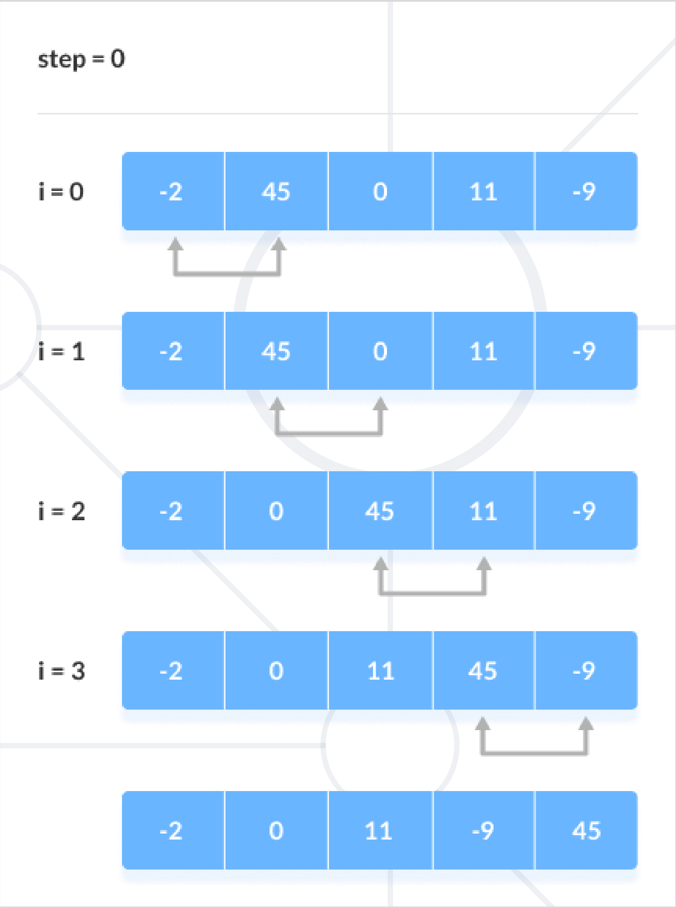
Swap Method: Code

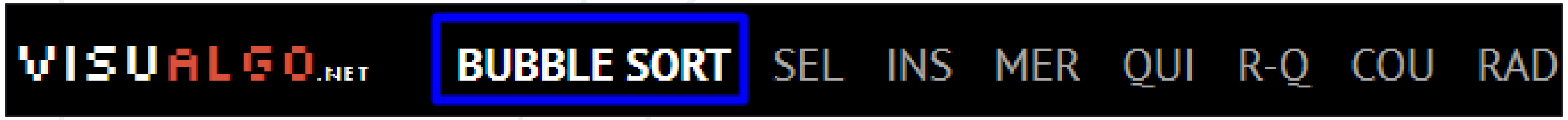
- Interchanging two elements (swap):

```
static void Swap(int[] nums, int index1, int index2)
{
    int oldNum = nums[index1];
    nums[index1] = nums[index2];
    nums[index2] = oldNum;
}
```

Bubble Sort

- **Bubble sort** – simple, but inefficient algorithm
- Swaps to neighbor elements when not in order until sorted
  - Memory: **O(1)**, Time: **O(n^2)**, Stable: **Yes**
  - Method: Exchanging
  - See the visualization: https://visualgo.net/en/sorting → choose **Bubble sort**:





Bubble Sort: Code

```
var nums = new[] { 1, 3, 4, 2, 5, 6 };
for (int i = 0; i < nums.Length; i++)
{
  for (int j = 1; j < nums.Length - i; j++)
  {
    if (nums[j - 1] > nums[j])
      Swap(nums, j - 1, j);
  }
}
Console.WriteLine(string.Join(" ", nums));
```

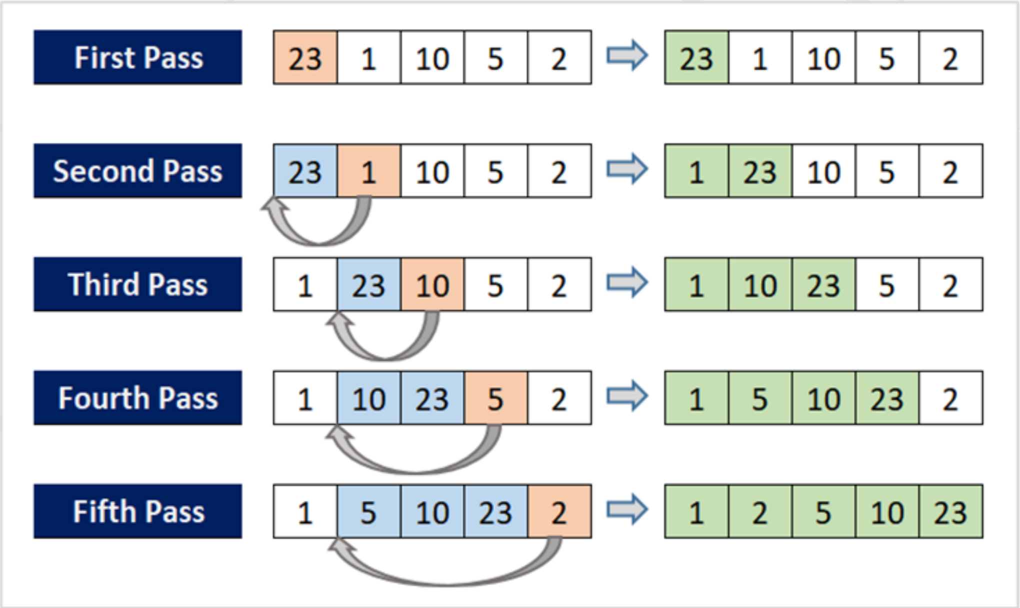
Comparison of Sorting Algorithms

- **Selection** sort vs. **Bubble** sort:

Name	Best	Average	Worst	Memory	Stable	Method
Selection	n^2	n^2	n^2	1	No	Selection
Bubble	n	n^2	n^2	1	Yes	Exchanging

Insertion Sort

- **Insertion Sort** – simple, but inefficient algorithm
  - Move the first unsorted element left to its place
  - Memory: **O(1)**, Time: **O(n²)**
  - Stable: **Yes**
  - Method: **Insertion**
  - See the visualization: https://visualgo.net/en/sorting → choose **Insertion sort**:



Insertion Sort

Software University

```
var nums = new[] { 1, 3, 4, 2, 5, 6 };
for (int startIndex = 1; startIndex < nums.Length; startIndex++)
{
  var currIndex = startIndex;
  while (currIndex > 0 && nums[currIndex] < nums[currIndex - 1])
  {
    Swap(nums, currIndex, currIndex - 1);
    currIndex--;
  }
}
Console.WriteLine(string.Join(" ", nums));
```

Comparison of Sorting Algorithms

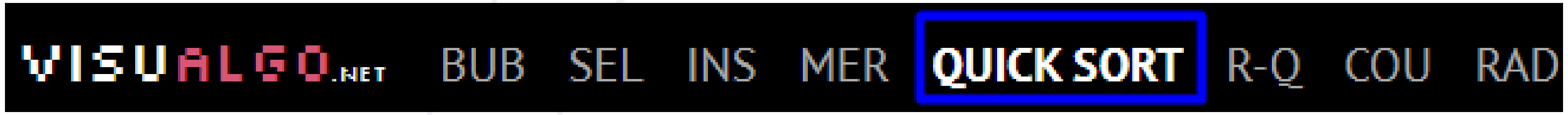
- **Selection** sort vs. **Bubble** sort vs. **Insertion** sort:

Name	Best	Average	Worst	Memory	Stable	Method
Selection	n^2	n^2	n^2	1	No	Selection
Bubble	n	n^2	n^2	1	Yes	Exchanging
Insertion	n	n^2	n^2	1	Yes	Insertion

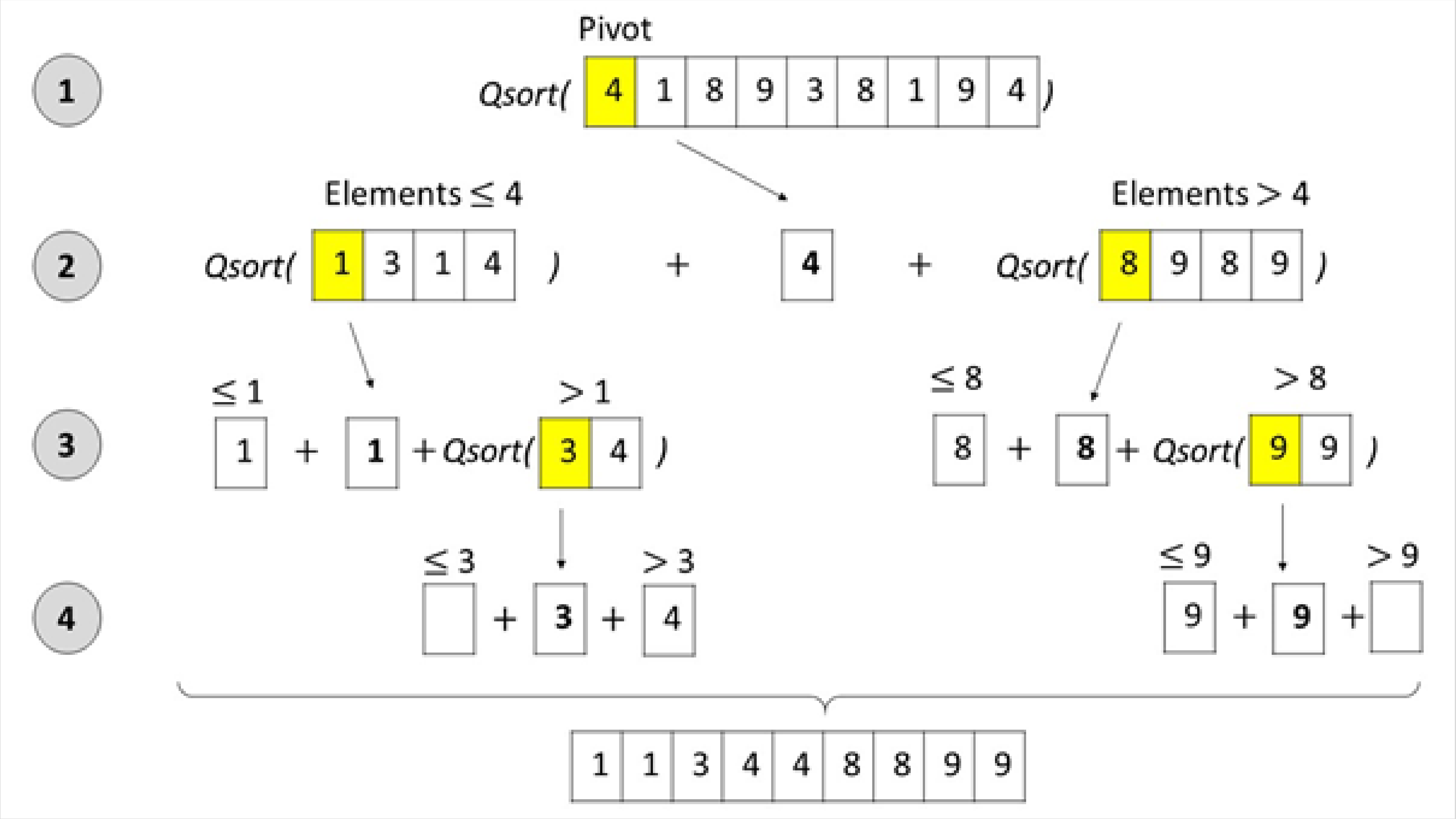
QuickSort, MergeSort

Quick Sort

- **QuickSort** – efficient sorting algorithm
  - Choose a **pivot**; move smaller elements left & larger right; sort left & right
  - Memory: **O(log(n))** stack space (recursion), Time: **$O(n^2)$**, Stable: **Depends**
  - Method: **Partitioning**
  - See the visualization: https://visualgo.net/en/sorting → choose **Quick sort**:



Quick Sort: Conceptual Overview



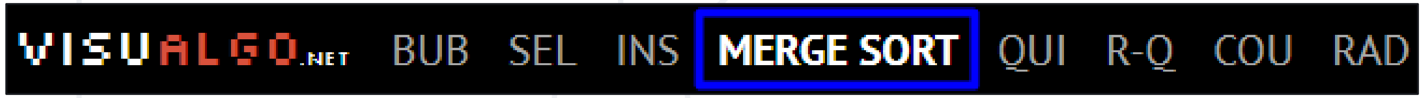
Comparison of Sorting Algorithms

- **Selection** sort vs. **Bubble** sort vs. **Insertion** sort vs. **Quick** sort:

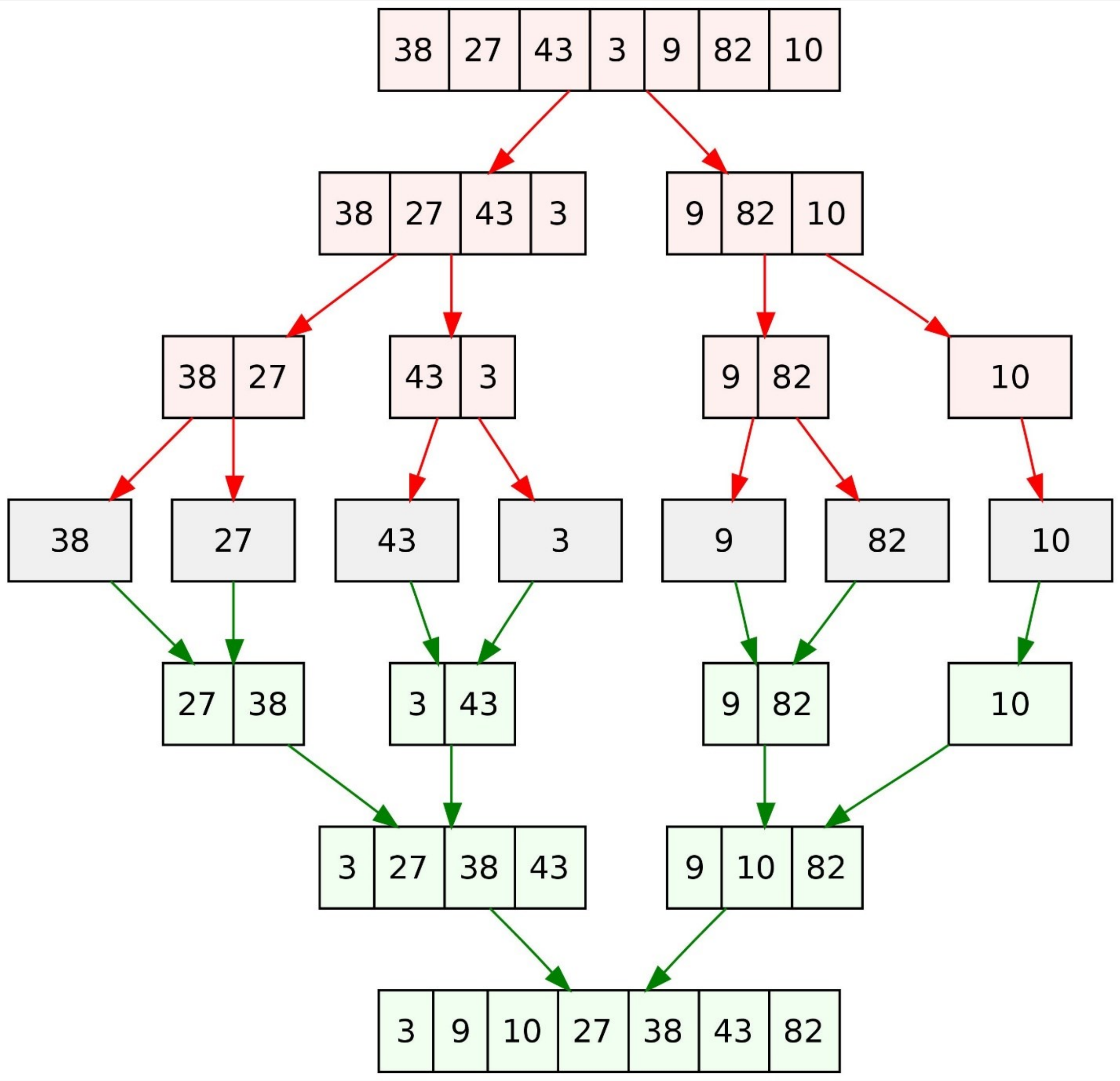
Name	Best	Average	Worst	Memory	Stable	Method
Selection	n^2	n^2	n^2	1	No	Selection
Bubble	n	n^2	n^2	1	Yes	Exchanging
Insertion	n	n^2	n^2	1	Yes	Insertion
Quick	n * log(n)	n * log(n)	n^2	1	Depends	Partitioning

Merge Sort

- **Merge sort** is efficient sorting algorithm
- Divide the list into **sub-lists** (typically 2 sub-lists)
  1. Sort each sub-list (recursively call merge-sort)
  2. Merge the sorted sub-lists into a single list
- Memory: **O(n)** / **O(n*log(n))**, Time: **O(n*log(n))**, Stable: **Yes**
- Highly parallelizable on multiple cores → up to **O(log(n))**
- See the visualization: https://visualgo.net/en/sorting → choose **Merge sort**:



Merge Sort: Conceptual Overview

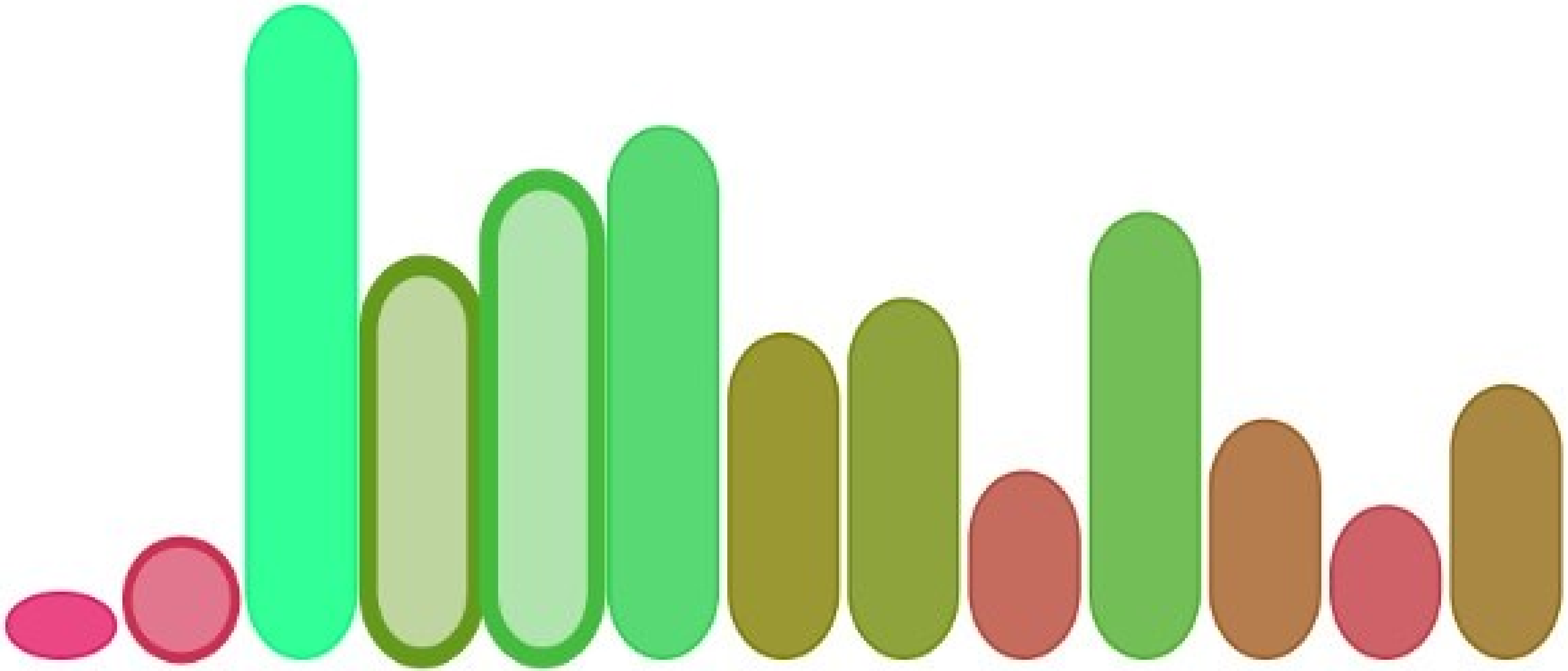


Comparison of Sorting Algorithms

- **Selection** sort vs. **Bubble** sort vs. **Insertion** sort vs. **Quick** sort vs. **Merge** sort:

Name	Best	Average	Worst	Memory	Stable	Method
Selection	n^2	n^2	n^2	1	No	Selection
Bubble	n	n^2	n^2	1	Yes	Exchanging
Insertion	n	n^2	n^2	1	Yes	Insertion
Quick	n * log(n)	n * log(n)	n^2	1	Depends	Partitioning
Merge	n * log(n)	n * log(n)	n * log(n)	1	Yes	Merging

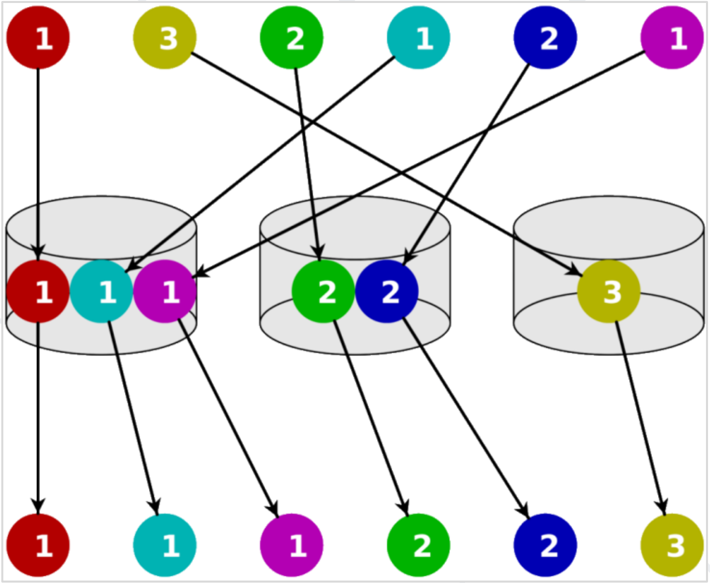
- Additional algorithms comparison: https://www.toptal.com/developers/sorting-algorithms



Built-in and Special Algorithms

How to Choose a Sorting Algorithm

- Use **Insertion sort** for small and **QuickSort** for larger arrays
- Use the **built-in sorting** from C# / .NET
  - `Array.Sort()`
  - `List<T>.Sort()`
  - Other built-in sorting
- Use special sorting in special cases:
  - Example: **Bucket sort** (runs in linear time)
  - See the visualization: http://www.algostructure.com/sorting/bucketsort.php



Linear, Binary and Interpolation Search

Search Algorithm

- **Search algorithm** == an algorithm for finding an item with specified properties among a collection of items
- Different types of **searching algorithms**:
  - For **virtual** search spaces
    - Satisfy specific mathematical equations
    - Try to exploit partial knowledge about a structure
  - For **sub-structures** of a given structure
    - A graph, a string, a finite group
  - Search for the **min / max** of a **function**, etc.

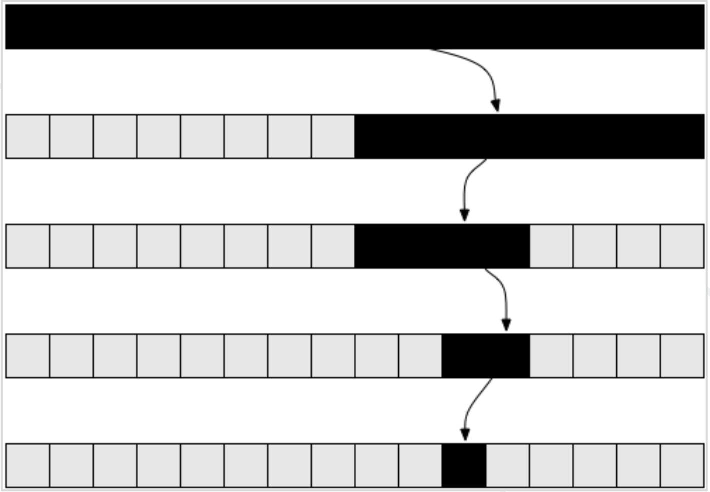
Linear Search

- **Linear search** finds a particular **value** in a list
  - Checking every one of the elements
  - One at a time, in sequence
  - Until the desired one is found
- Worst & average performance: **`O(n)`**
- See the [visualization]

```
for each item in the list:
  if that item has the desired value,
    return the item's location
return nothing
```

Binary Search

- **Binary search** finds an item within an **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
- Average performance: **O(log(n))**
- See the visualization



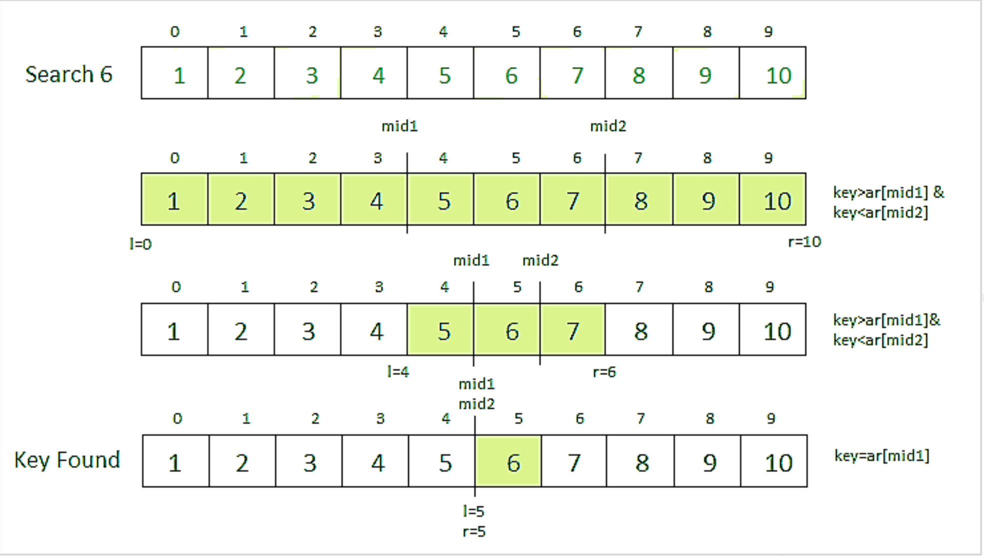
Binary Search (Iterative): Code

```
static int BinarySearch(int[] numbers, int searchNumber) {
  var left = 0;
  var right = numbers.Length - 1;
  while (left <= right) {
    var mid = (left + right) / 2;
    if (numbers[mid] == searchNumber)
      return mid;
    if (searchNumber > numbers[mid])
      left = mid + 1;
    else
      right = mid - 1;
  }
  return KEY_NOT_FOUND;  // const KEY_NOT_FOUND = -1;
}
```

Interpolation Search

Software University

- Interpolation search == an algorithm for searching for a given key in an ordered indexed array
  - Similar to how humans search through a telephone book
  - Calculates where in the remaining search space the item may be
    - Binary search always chooses the middle element
- Average case: log(log(n))
- Worst case: O(n)
- See the visualization



Interpolation Search: Code

```
int InterpolationSearch(int[] sortedArray, int key) {
  int low = 0;
  int high = sortedArray.Length - 1;
  while (sortedArray[low] <= key && sortedArray[high] >= key) {
    int mid = low + ((key - sortedArray[low]) * (high - low))
      / (sortedArray[high] - sortedArray[low]);
    if (sortedArray[mid] < key)
      low = mid + 1;
    else if (sortedArray[mid] > key)
      high = mid - 1;
    else
      return mid;
  }
  if (sortedArray[low] == key) return low;
  else return KEY_NOT_FOUND; // const KEY_NOT_FOUND = -1;
}
```

Fisher-Yates Shuffle

Shuffling

- **Shuffling** == randomizing the order of items in a collection
  - Generate a **random** permutation

A	B	C	D

→

D	B	A	C

> *"The generation of random numbers is too important to be left to chance."*
>
> —Robert R. Coveyou

```
Input: arr[], holding n elements

Shuffle:
  for i = 0 … n:
    next = random in the range [i … n-1]
    Exchange(arr[i], arr[next])
```

Fisher–Yates Shuffle Algorithm: Code

```
public static void Shuffle(T[] elements)
{
  Random rnd = new Random();

  for (int i = 0; i < elements.Length; i++)
  {
    // Exchange array[i] with random element in array[i … n-1]
    int next = rnd.Next(i, elements.Length);
    T oldElement = elements[i];
    elements[i] = elements[next];
    elements[next] = oldElement;
  }
}
```

Shuffle algorithms: visualization

Summary

- **Slow** sorting algorithms:
  - Selection sort, Bubble sort, Insertion sort
- **Fast** sorting algorithms:
  - Quick sort, Merge sort, Bucket sort, etc.
- **Searching** algorithms
  - Binary Search, Linear Search, Interpolation Search
- **Shuffling**
  - Randomizing the order of items in a collection
  - Fisher-Yates Shuffle

Questions?

SoftUni

- Software University
- SoftUni Svetlina
- SoftUni Creative
- SoftUni Digital
- SoftUni Foundation
- SoftUni Kids

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

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