Data structure and Programming II

Sorting Algorithms



Original array:

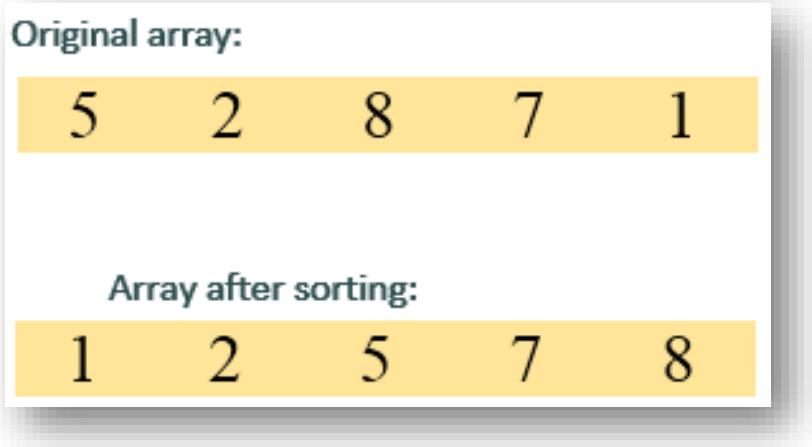
5	2	8	7	1

Array after sorting:

1	2	5	7	8

Outline

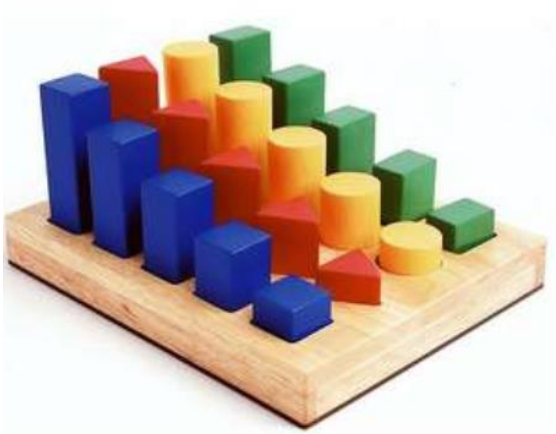
- Introduction
- Types of sorting
- Complexity of algorithm
- Examples



What is sorting?

❑ Definition

- **Sorting** is any process of arranging items systematically, with two distinct meanings:
  - Ordering : arranging items in a sequence ordered by some criterion
  - Categorizing : grouping items with similar properties





What is sorting algorithm?

❑ Definition

- **A sorting algorithm** is an algorithm that puts items of a list in a certain order



Sort data in an ascending order

How to compare elements (Examples)

- Some examples
  1. **Order numbers from the smallest to the largest or vice versa**
  2. **Order words alphabetically**
  3. **Order people alphabetically by name**
     - People having the same **firstname** are then ordered according to their **lastname**
     - People have the same **firstname** and **lastname** could be then ordered according to their age
- Hence, comparisons could be based on rules that are used to sort things

What do we sort?

❑ Definition

- We order collections of data
- Example
  1. **Array**
  2. **Linked list**

- ❑ Insertion sort
- ❑ **Selection sor**t
- ❑ Merge sort
- ❑ Heap sort
- ❑ Quick sort
- ❑ **Bubble sort**
- ❑ Shell sort
- ❑ Comb sort
- ❑ Counting sort
- ❑ Bucket sort
- ❑ Radix sort

Simple sorting algorithms

Efficiency sorting algorithms

Bubble sort and variant algorithms

Distribution sort algorithms

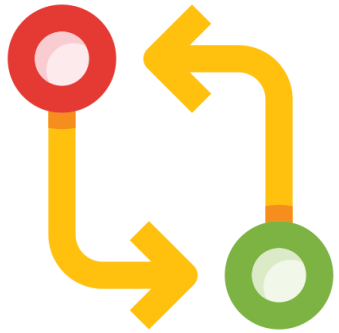
Etc….

Common sorting algorithms

❑ Remark

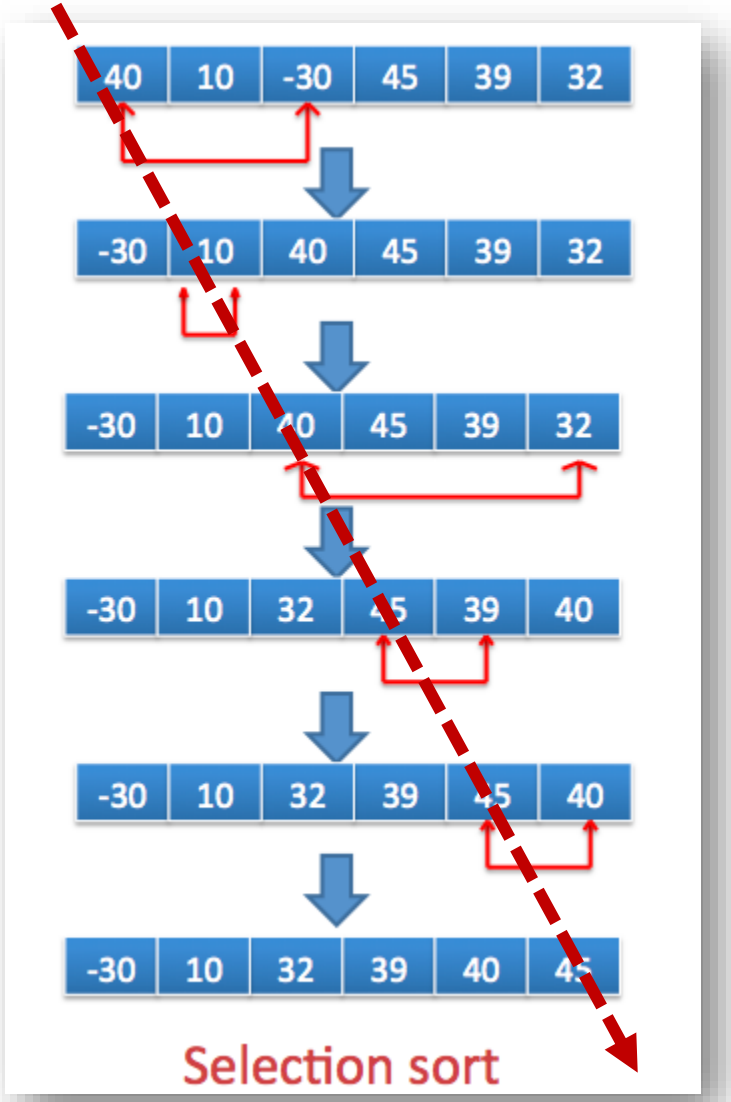
- There are many algorithms that we can use to order a list of elements

- Although they are all different, they share a common building block
  - ✓ Compare pairs of elements to decide which is their relative order
  - ✓ When needed, they swap the elements to restore their order



Selection sort

- **Loop from index i=0 to last element index**
  - Find index j of min element
  - Swap min element with element at position i
    - Use swap function
  - i = i+1
  - Repeat



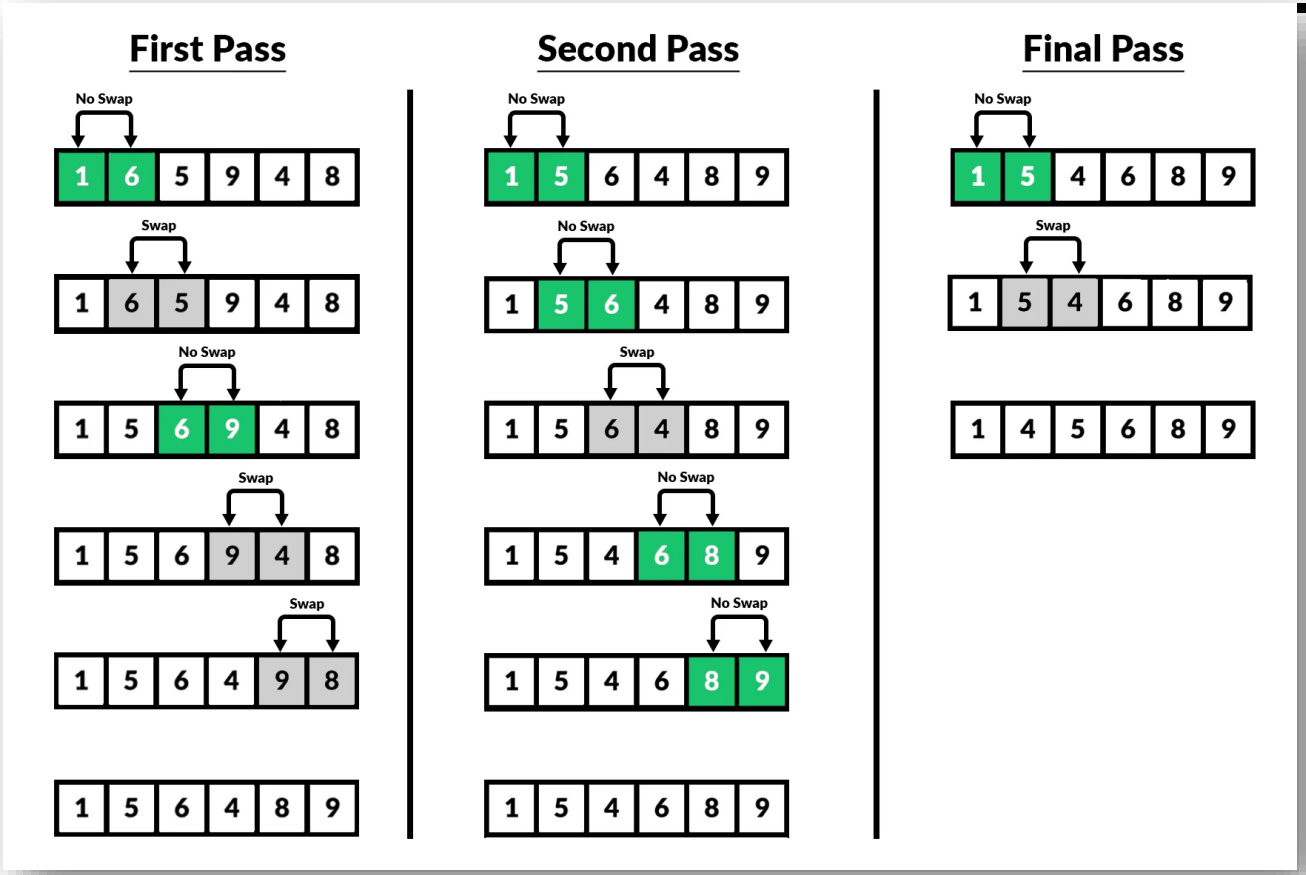

```
4   void swap(int *x, int *y){
5       int temp;
6       temp = *x;
7       *x = *y;
8       *y = temp;
9   }
10
11  void selectionSort(int arr[], int n){
12      int i, j, minIndex;
13
14      for (i = 0; i < n-1; i++){
15          // Find index of min element
16          minIndex = i;
17          for (j = i+1; j < n; j++){
18              if (arr[j] < arr[minIndex]){
19                  minIndex = j;
20              }
21          }
22
23          // Swap the min element with the element at index i
24          swap(&arr[minIndex], &arr[i]);
25      }
26  }
```

A function to exchange data via pointer operation

Find index of min element

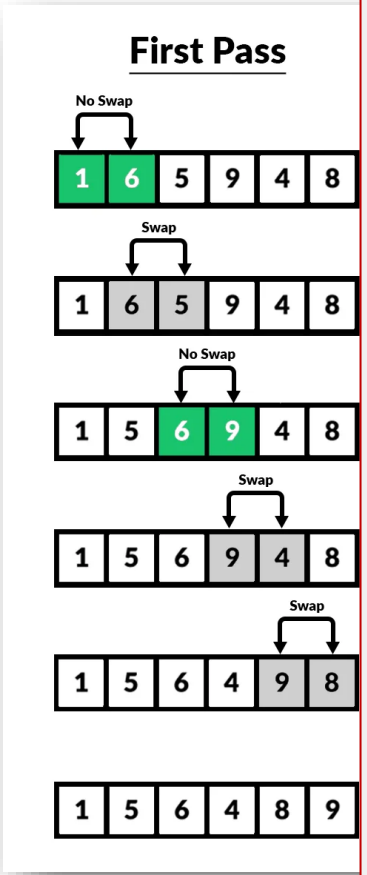
Putting the min element to position i

Bubble sort



- Step 1
  - Compare two first elements and swap if not yet in order we want
    - Element at i and i+1
- Step 2
  - Repeat step 1
    - Element at i+1 and i+2
- If no swap, sorted already.

Bubble sort



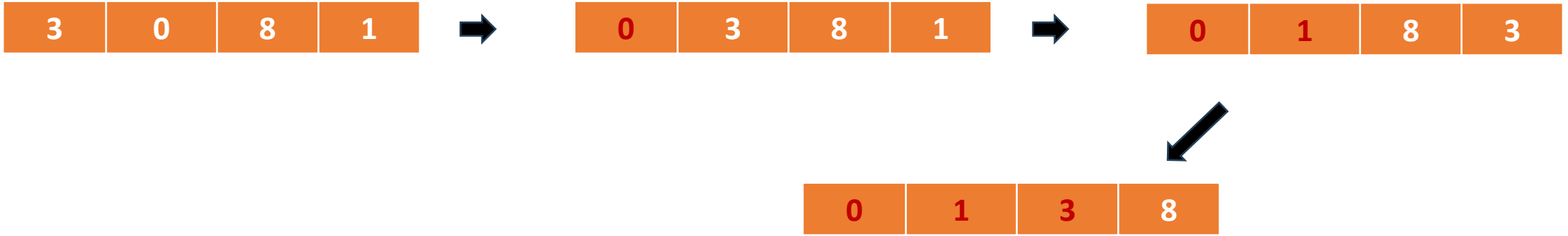

```
4   void exchange(int *a, int *b){
5       int tmp;
6
7       tmp = *a;
8       *a = *b;
9       *b = tmp;
10  }
11
12  void sortBubble(int a[], int n){
13      bool state;
14
15      for(int k=0; k<n-1; k=k+1){
16          state = false;
17
18          for(int p=0; p<n-1; p=p+1){
19              if(a[p] > a[p+1]){ //compare if it is not in the correct order
20                  exchange(&a[p], &a[p+1]); //swap to get the correct order
21                  state = true;
22              }
23          }
24          //If no swap at all, array already sorted
25          if(state == false){
26              break;
27          }
28      }
29  }
```

Note for Selection sort and Bubble sort

- With **bubble sort**, the correct position data are filled backward from the position n to 0. Correct data is push to the end first and so on.



- With selection sort, the correct position data are filled forward from position to 0 to position n at the end.



Classification criteria for sorting algorithms

❑ Criteria

1. Total number of steps (complexity)
2. Number of comparisons performed
3. Number of interchanges performed
4. Stability
5. Type of memory used
   - Internal algorithms, External algorithms

Complexity of Algorithm (time consuming)

❑ Definition

- **Time complexity of an algorithm** is the amount of time taken by the algorithm to run
- Time complexity
  - Is expressed as a function of the length of the input
  - Is commonly expressed using **Big-O Notation**

Complexity of Algorithm

❑ Big-O Notation

- Big-O Notation describes the limiting behavior of a function when the number of elements it has to process tends to infinity
- The concept is simplified in this way:



Complexity of Algorithm

❑ Big-O Notation

- Which algorithm is the best?
  - *To us, algorithm is good if it consumes less time to run*
- Let N be the number of elements we want to process
  - E.g: number of elements in an array
- What if we have 4 algorithms whose time complexities are as follows?

Algorithm	Time Complexity
Algorithm 1	O(N)
Algorithm 2	$O(N^2)$
Algorithm 3	O(log(N))
Algorithm 4	O(N*log(N))

Complexity of Algorithm

❑ Big-O Notation

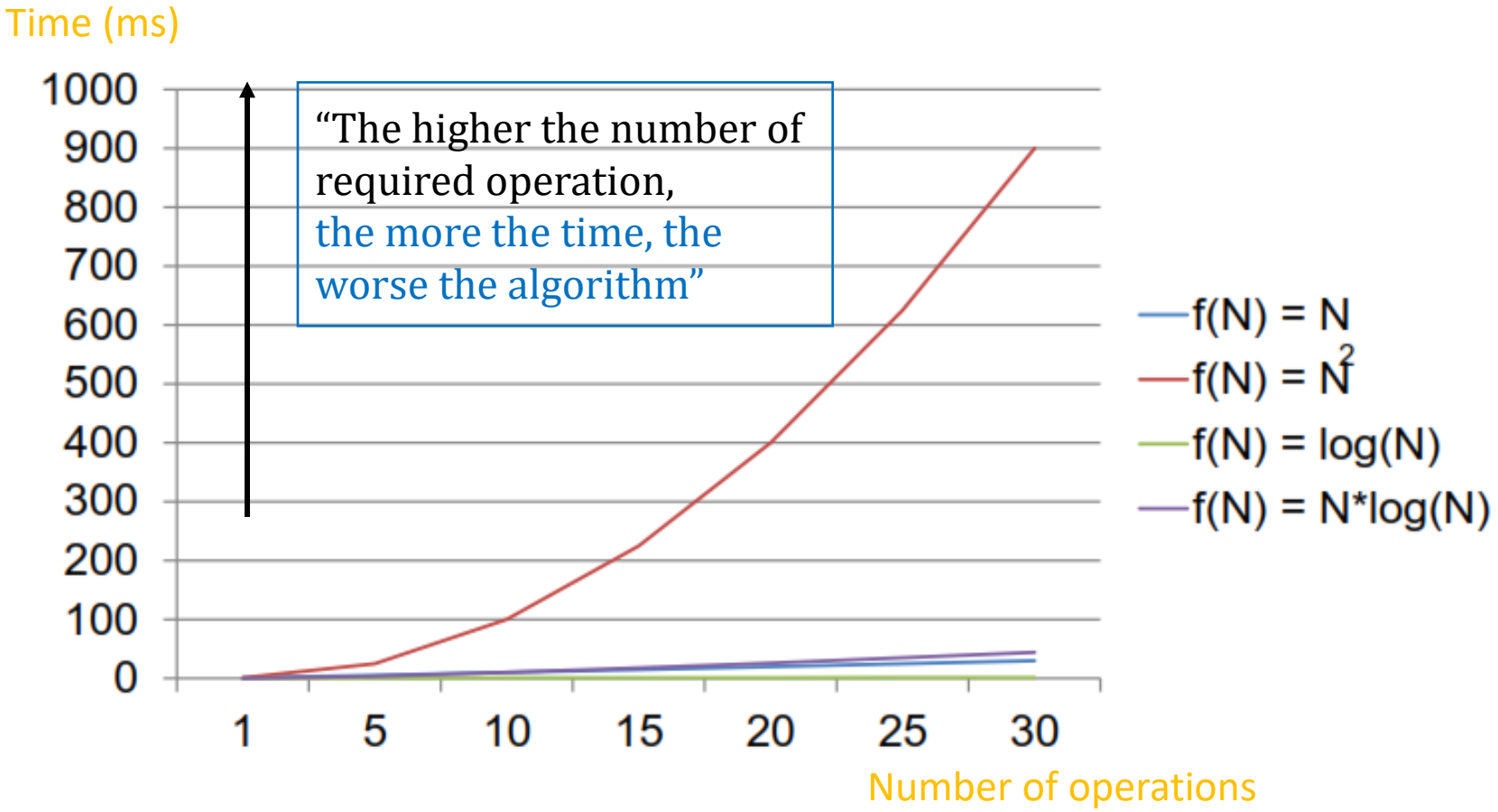
- These time complexity can be treated as if it were a mathematical function

Algorithm	Time Complexity	Math function
Algorithm 1	O(N)	$f(N) = N$
Algorithm 2	$O(N^2)$	$f(N) = N^2$
Algorithm 3	O(log(N))	f(N) = log(N)
Algorithm 4	O(N*log(N))	f(N) = N*log(N)

Complexity of Algorithm

❑ Big-O Notation

- These time complexity can be treated as if it were a mathematical function



Algorithm	Time Complexity	Math function
Algorithm 1	O(N)	f(N) = N
Algorithm 2	$O(N^2)$	$f(N) = N^2$
Algorithm 3	O(log(N))	f(N) = log(N)
Algorithm 4	O(N*log(N))	f(N) = N*log(N)

Sorting Algorithm Techniques

❑ Following are some families of sorting algorithms

- ***Simple sorts***
- Efficient sorts
- Bubble sort and variants
- Distribution sorts

Simple sorting algorithms are efficient on small data amounts but generally do not perform well on large lists

- ❑ Insertion sort
- ❑ Selection sort

Sorting Algorithm Techniques

❑ Following are families of sorting algorithms

- Simple sorts
- ***Efficient sorts***
- Bubble sort and variants
- Distribution sorts

Efficient sorting algorithms are those algorithms whose average complexity is the best you can find O(N*log(N))

- ❑ Merge sort
- ❑ Heap sort
- ❑ Quick sort

Sorting Algorithm Techniques

❑ Following are families of sorting algorithms

- Simple sorts
- Efficient sorts
- ***Bubble sort and variants***
- Distribution sorts

Bubble sorting algorithms is very simple, and the same characteristics is inherited by all its variants. However, it is highly inefficient (high time complexity $O(N^2)$)

- ❑ Bubble sort
- ❑ Shell sort
- ❑ Comb sort

Sorting Algorithm Techniques

❑ Following are families of sorting algorithms

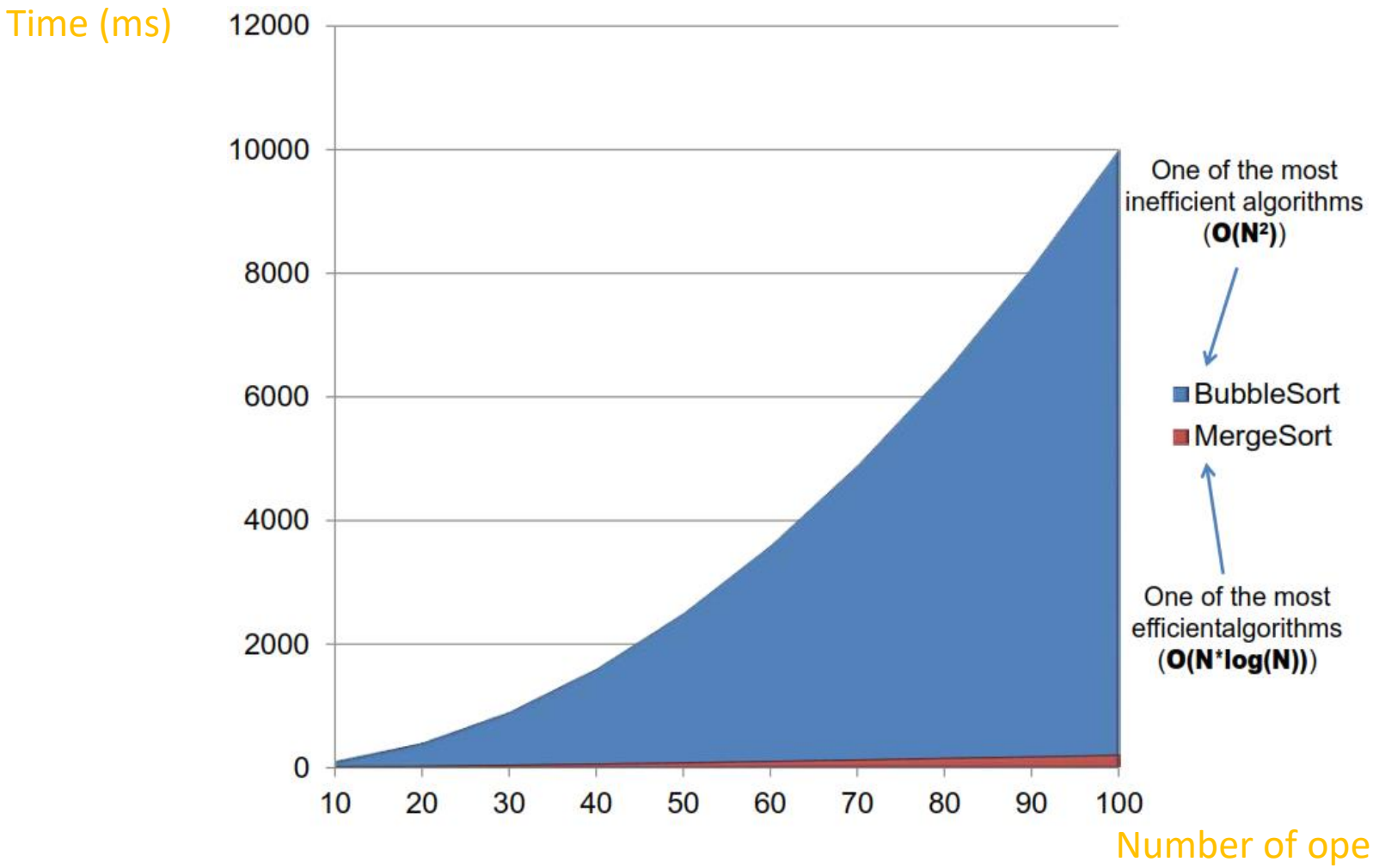
- Simple sorts
- Efficient sorts
- Bubble sort and variants
- ***Distribution sorts***

Distribution sorting algorithms distribute the input to intermediate structures, which are gathered and placed on the output. They are useful in case of very large data sets that do not fit in memory. Intermediate structures can be deployed on different machine

- ❑ Counting sort
- ❑ Bucket sort
- ❑ Radix sort

Sorting Algorithm Techniques

❑ Performance: Bubble Vs. Merge sort



Sorting Algorithm Techniques

❑ In summary

- The two best families of sorting algorithms are
  - Efficient sorts : have best average time complexity O(N*log(N))
    - ❑ Merge sort
    - ❑ Heap sort
    - ❑ Quick sort
  - Distribution sorts : perform wells on large data
    - ❑ Counting sort
    - ❑ Bucket sort
    - ❑ Radix sort

Sorting algorithms

Sorting Algorithm	Definition	Complexity	Stability?
Insertion sort	is a simple sorting algorithm that **builds the final sorted array one item at a time**. Method: Insertion	Worst case: n^2 Average case: n^2 Best case: n	Yes
Selection sort	is a sorting algorithm, specifically **an in-place comparison sort**. inefficient on large lists, and performs worse than the similar insertion sort. Method: Selection	Worst case: n^2 Average case: n^2 Best case: n^2	No
Merge sort	is an efficient, general-purpose, comparison-based sorting algorithm. It is a **divide and conquer algorithm** Method: Merging	Worst case: n*log(n) Average case: n*log(n) Best case: n*log(n)	Yes
Quick sort	is an efficient sorting algorithm, serving as a systematic method for **placing the elements of an array in order**. Method: Partitioning	Worst case: n^2 Average case: n*log(n) Best case: n*log(n)	No
Bubble sort (or called Sinking sort)	is a simple sorting algorithm that repeatedly steps through the list, compares adjacent pairs and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. Method: Exchanging	Worst case: n^2 Average case: n^2 Best case: n	Yes

Q&A