Course Name:
Analysis and Design of Algorithms

Topics to be covered

Sorting Techniques and their analysis

Overview

Algorithmic Description and Analysis of

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort
- Quick Sort

Sorting - what for ?

Example:

Accessing (finding a specific value in) an **unsorted** and a **sorted** array:

Find the name of a person being 10 years old:

10	Bart
36	Homer
8	Lisa
35	Marge
1	Maggie

Sorting - what for ?

Unsorted:

Worst case: try n rows => order of magnitude: O(n)

Average case: try n/2 rows => O(n)

n

10	Bart
36	Homer
1	Maggie
35	Marge
8	Lisa

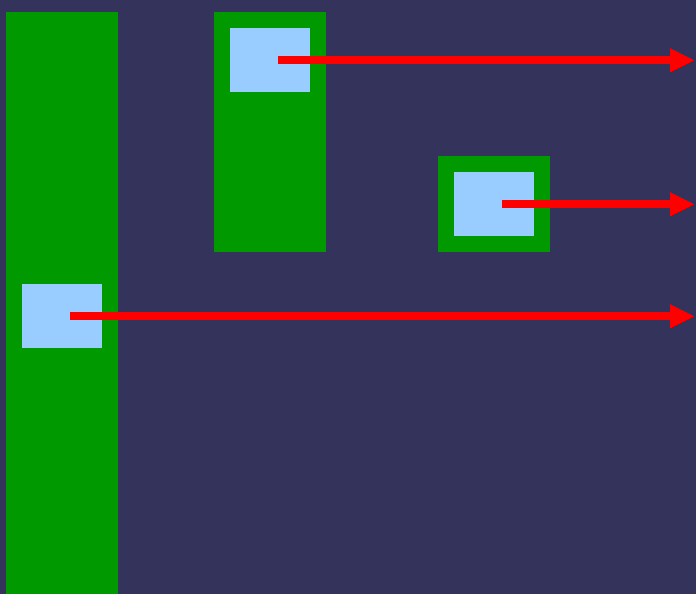
Sorting - what for ?

Sorted: Binary Search

Worst case: try log(n) <= k <= log(n)+1 rows => O(log n)

Average case: O(log n)

(for a proof see e.g. http://www.mcs.sdsmt.edu/~ecorwin/cs251/binavg/binavg.htm)



1	Maggie
8	Lisa
10	Bart
35	Marge
36	Homer

Sorting - what for ?

- Sorting and accessing is faster than accessing an unsorted dataset (if multiple (=k) queries occur):

$$n*\log(n) + k*\log(n) < k * n$$

(if k is big enough)

- Sorting is crucial to databases, databases are crucial to data-management, data-management is crucial to economy, economy is ... sorting seems to be pretty important !
- The question is WHAT (name or age ?) and HOW to sort.
- This lesson will answer the latter one.

Sorting

Quadratic Algorithms

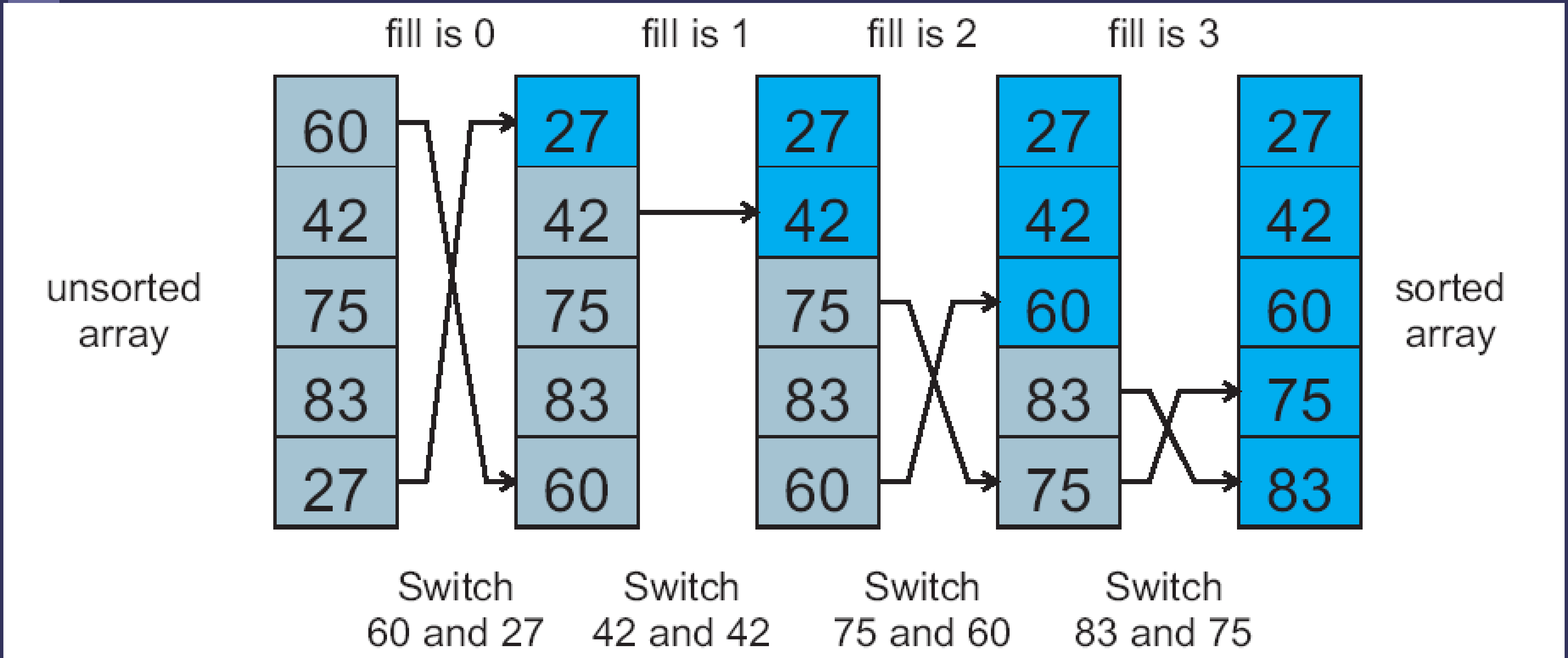
Quadratic Algorithms

Selection Sort

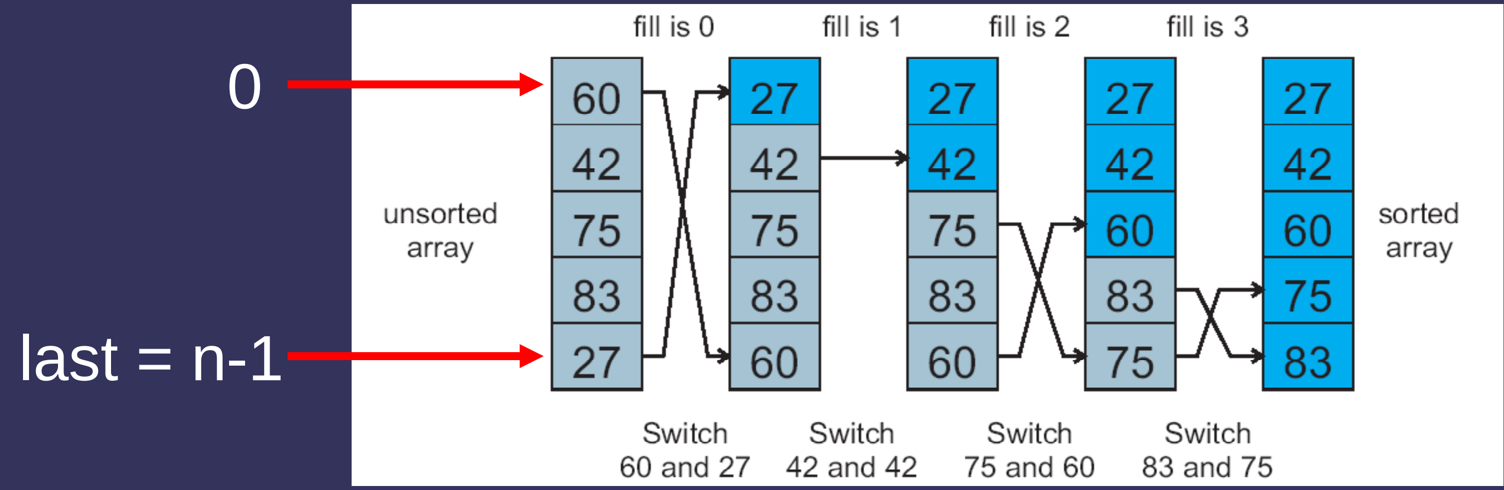
Selection Sort: Example

The Brute Force Method: Selection Sort

http://www.site.uottawa.ca/~stan/csi2514/applets/sort/sort.html



Selection Sort: Algorithm



Algorithm:

For i=0 .. last -1

find smallest element M in subarray i .. last

if M != element at i: swap elements

Next i (← this is for BASIC-freaks !)

Selection Sort: Analysis

Number of comparisons:

$(n-1) + (n-2) + ... + 3 + 2 + 1 =$

$n * (n-1)/2 =$

$(n^2 - n)/2$

→ $O(n^2)$

Number of exchanges (worst case):

$n – 1$

→ $O(n)$

Overall (worst case) $O(n) + O(n^2) = O(n^2)$ (‘quadratic sort‘)

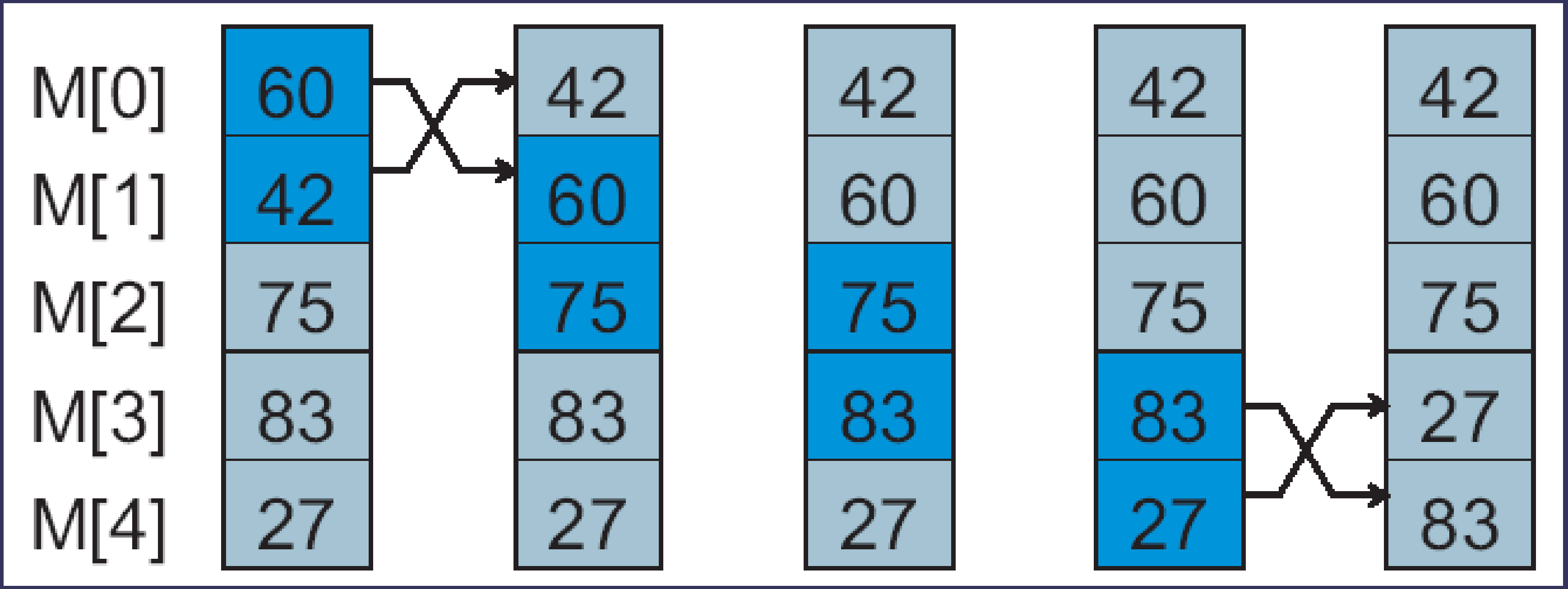
Quadratic Algorithms

Bubble Sort

Bubble Sort: Example

The Famous Method: Bubble Sort

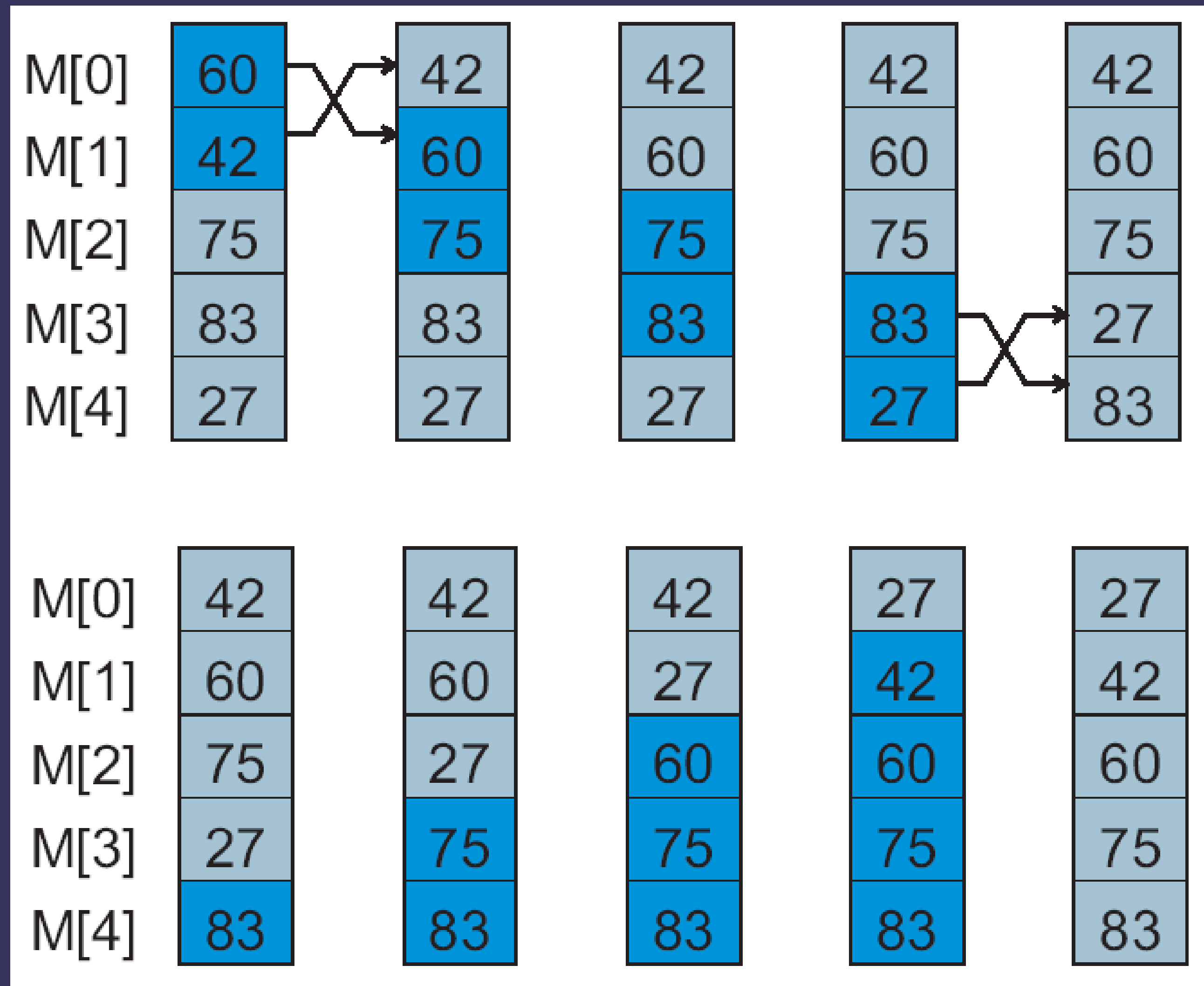
http://www.site.uottawa.ca/~stan/csi2514/applets/sort/sort.html



Bubble Sort: Example

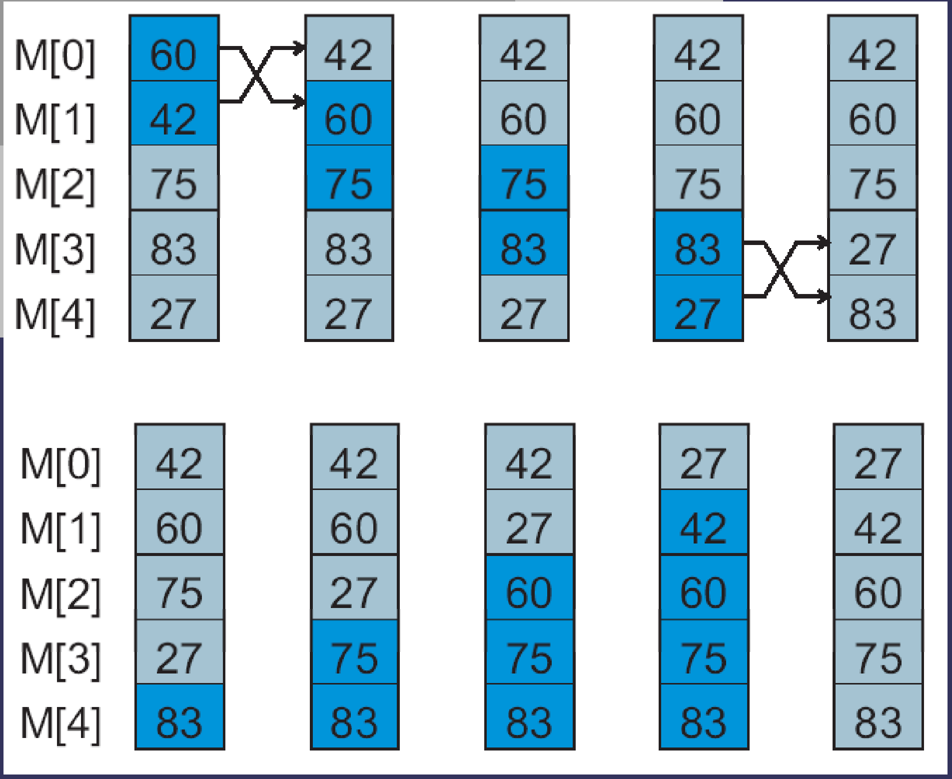
One Pass

Array after Completion of Each Pass



Bubble Sort: Algorithm

```
for pass = 1 .. n-1
    exchange = false
    for position = 1 .. n-pass
        if element at position < element at position +1
            exchange elements
            exchange = true
        end if
    next position
    if exchange = false BREAK
next pass
```



Bubble Sort: Analysis

Number of comparisons (worst case):

$(n-1) + (n-2) + ... + 3 + 2 + 1 \rightarrow O(n^2)$

Number of comparisons (best case):

$n - 1 \rightarrow O(n)$

Number of exchanges (worst case):

$(n-1) + (n-2) + ... + 3 + 2 + 1 \rightarrow O(n^2)$

Number of exchanges (best case):

$0 \rightarrow O(1)$

Overall worst case: $O(n^2) + O(n^2) = O(n^2)$

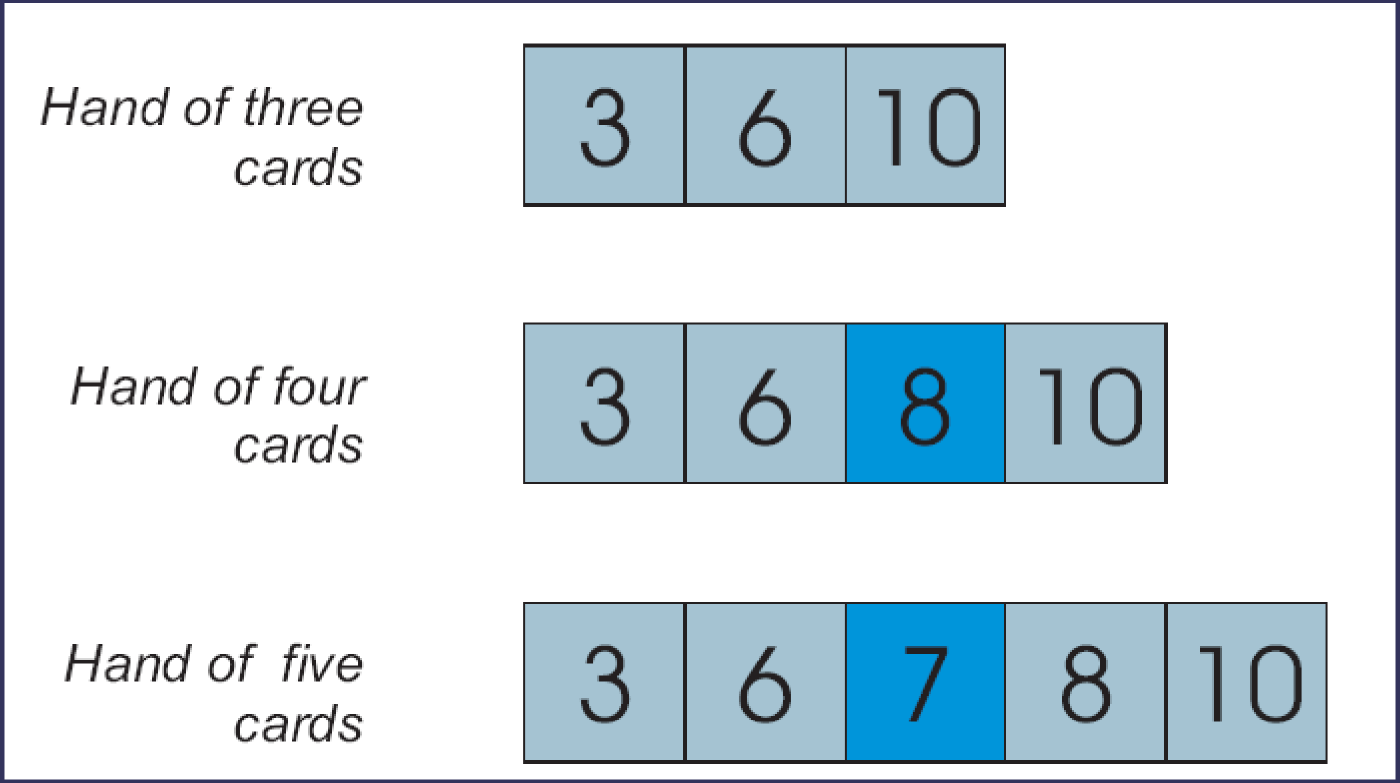
Quadratic Algorithms

Insertion Sort

Insertion Sort: Example

The Card Player's Method: Insertion Sort

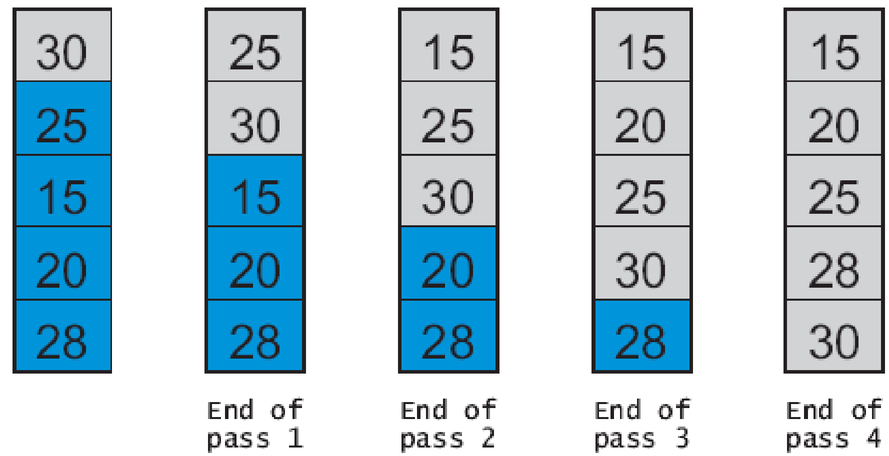
http://www.site.uottawa.ca/~stan/csi2514/applets/sort/sort.html



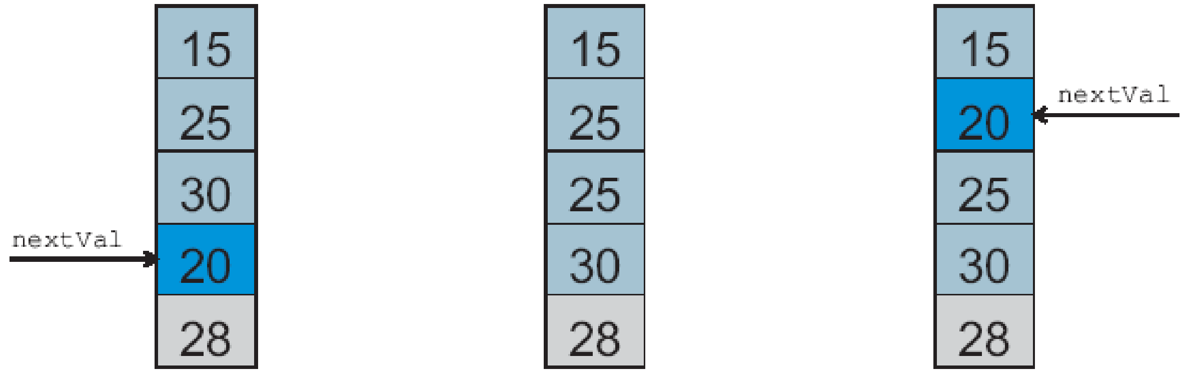
Insertion Sort: Example

Insertion Sort:

4 passes



Pass 3



Insertion Sort: Algorithm

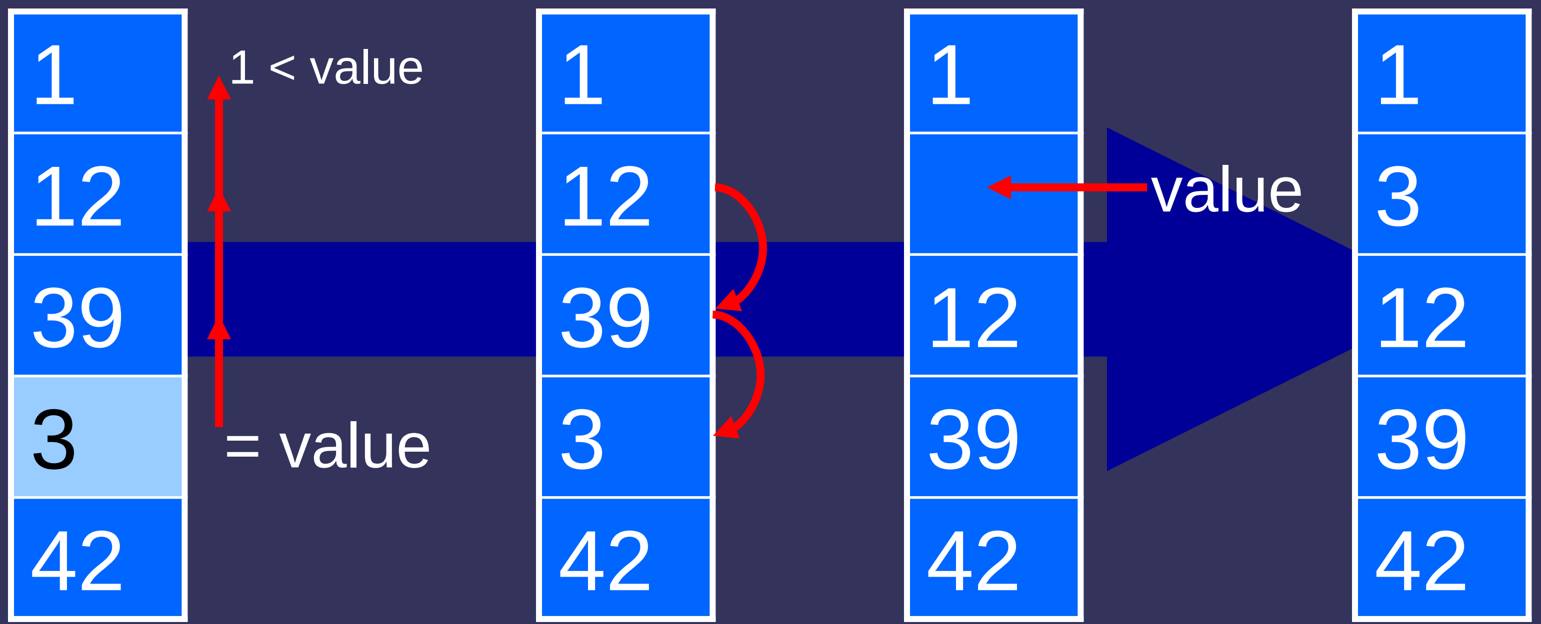
for pass = 2 .. n-1

value = element at pass

shift all elements > value in array 1..pass-1 one pos. right

place value in the array at the ‘vacant’ position

next pass



Insertion Sort: Analysis

Number of comparisons (worst case):

$(n-1) + (n-2) + ... + 3 + 2 + 1 \rightarrow O(n^2)$

Number of comparisons (best case):

$n - 1 \rightarrow O(n)$

Number of exchanges (worst case):

$(n-1) + (n-2) + ... + 3 + 2 + 1 \rightarrow O(n^2)$

Number of exchanges (best case):

$0 \rightarrow O(1)$

Overall worst case: $O(n^2) + O(n^2) = O(n^2)$

Comparison of Quadratic Sorts

	Comparisons		Exchanges	
	Best	Worst	Best	Worst
Selection Sort	$O(n^2)$	$O(n^2)$	$O(1)$	$O(n)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(1)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(1)$	$O(n^2)$

Result Quadratic Algorithms

	Pro	Contra
Selection Sort	If array is in 'total disorder'	If array is presorted
Bubble Sort	If array is presorted	If array is in 'total disorder'
Insertion Sort	If array is presorted	If array is in 'total disorder'

N	N^2	$N \times \log_2 N$
8	64	24
16	256	64
32	1,024	160
64	4,096	384
128	16,384	896
256	65,536	2,048
512	262,144	4,608

Overall: $O(n^2)$ is not acceptable since there are nLog(n) algorithms !

n*Log(n) Algorithms

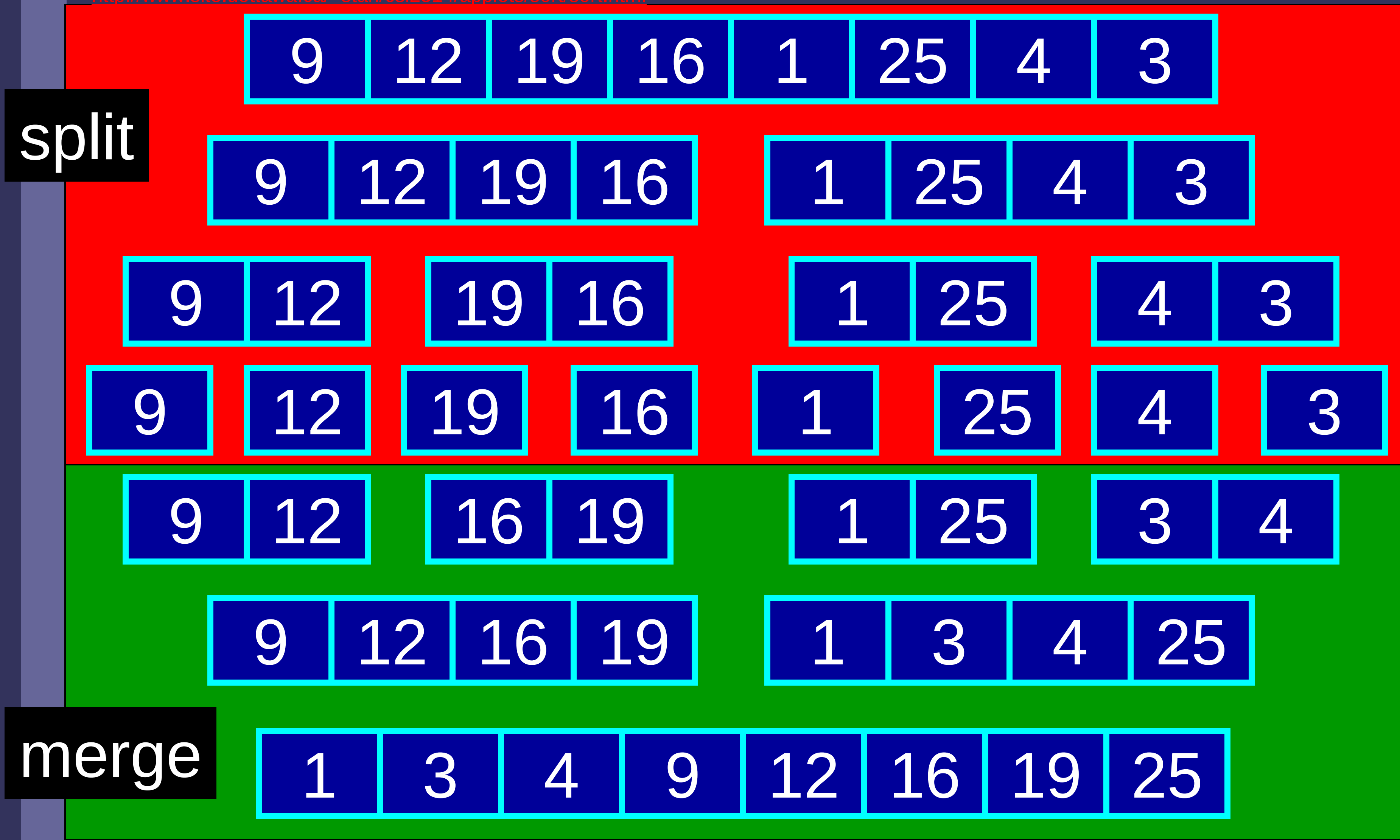
n Log(n) Algorithms

Merge Sort

Merge Sort: Example

Divide and Conquer: Merge Sort

http://www.site.uottawa.ca/~stan/csi2514/applets/sort/sort.html



Merge Sort: Algorithm

```
function  outArray = sort(array):
    n = array.length
    if n == 1
        return array
    else
        mid = n / 2
        leftarray = array 0..mid
        rightarray = array mid+1 .. n-1
        sort leftarray          <---- RECURSION !
        sort rightarray
        array = merge leftarray and rightarray
        return array
    end
```

Cont'd: …how to merge…

Merge Sort: Algorithm (merge)

The algorithm for merging the two sequences is as follows:

1. Extract the first item from both sequences.
2. **`while`** not at the end of either sequence
    3. Compare the current items from each sequence, append the smaller item to the output sequence, and extract the next item from the sequence whose item was just output.
4. **`while`** not at the end of the first sequence
    5. Copy any remaining items from the first sequence to the output.
6. **`while`** not at the end of the second sequence
    7. Copy any remaining items from the second sequence to the output.

Merge Sort: Analysis

- The complexity is O(n * Log(n))
- For details see textbook
- The idea is:
  - We need Log(n) merging steps
  - Each merging step has complexity O(n)

Problem: Merge Sort needs extra memory !

Merge Sort: Analysis

Memory used by recursive merge sort:

- N/2 for leftArray
- N/2 for rightArray

…on stack for each step !

Total for each subarray: N/2 + N/4 + … + 1 = N –1

- 2N bytes of memory needed if implemented the simple way !
- Solution: don't pass leftArray and rightArray, but only indices defining the bounds

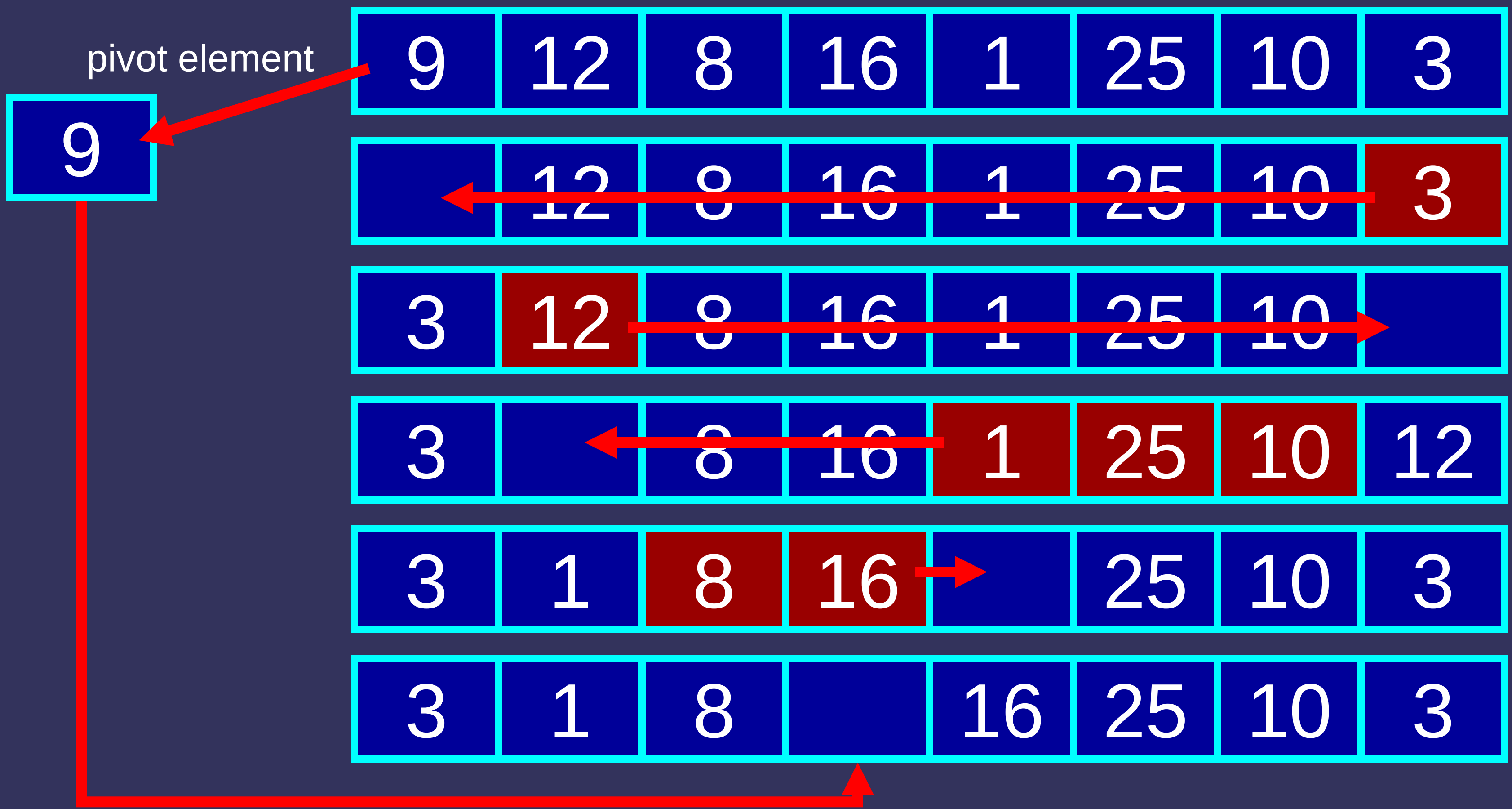
n Log(n) Algorithms

Quick Sort

Quick Sort: Example

Divide and Conquer II: Quick Sort

http://www.site.uottawa.ca/~stan/csi2514/applets/sort/sort.html



One step of Quick Sort (‘partitioning’)

Quick Sort: Algorithm

Inputs:

The array to be sorted

The first subscript (`first`)

The last subscript (`last`)

Outputs:

The sorted array

Steps:

1. **if** `first` < `last` then
2. Partition the elements in the subarray `first`…`last` so that the pivot value is in place (subscript `pivIndex`).
3. Recursively apply QuickSort to the subarray `first`…`pivIndex-1`.
4. Recursively apply QuickSort to the subarray `pivIndex+1`…`last`.

Quick Sort: Analysis

- Exact analysis is beyond scope of this course
- The complexity is O(n * Log(n))
  - Optimal case: pivot-index splits array into equal sizes
  - Worst Case: size left = 0, size right = n-1 (presorted list)
- Interesting case: presorted list:
  - Nothing is done, except (n+1) * n /2 comparisons
  - Complexity grows up to $O(n^2)$!
  - The better the list is presorted, the worse the algorithm performs !
- The pivot-selection is crucial. *In practical situations, a finely tuned implementation of quicksort beats most sort algorithms, including sort algorithms whose theoretical complexity is O(n log n) in the worst case.*
- Comparison to Merge Sort:
  - Comparable best case performance
  - No extra memory needed

Review of Algorithms

- Selection Sort
  - An algorithm which orders items by repeatedly looking through remaining items to find the least one and moving it to a final location
- Bubble Sort
  - Sort by comparing each adjacent pair of items in a list in turn, swapping the items if necessary, and repeating the pass through the list until no swaps are done
- Insertion Sort
  - Sort by repeatedly taking the next item and inserting it into the final data structure in its proper order with respect to items already inserted.
- Merge Sort
  - An algorithm which splits the items to be sorted into two groups, recursively sorts each group, and merges them into a final, sorted sequence
- Quick Sort
  - An in-place sort algorithm that uses the divide and conquer paradigm. It picks an element from the array (the pivot), partitions the remaining elements into those greater than and less than this pivot, and recursively sorts the partitions.

Definitions taken from www.nist.gov

Review

- There are thousands of different sorting algorithms out there
- Some of them (the most important ones) were presented
- Later we will meet another sorting algorithm using trees
- lots of images of these slides were taken from the textbook, for further details read there (Software Design & Data Structures in Java by Elliot B. Koffman + Paul A. T. Wolfgang) !