BTH001
Object Oriented Programming
Lesson 09
Sorting and Searching
Templates

Sorting

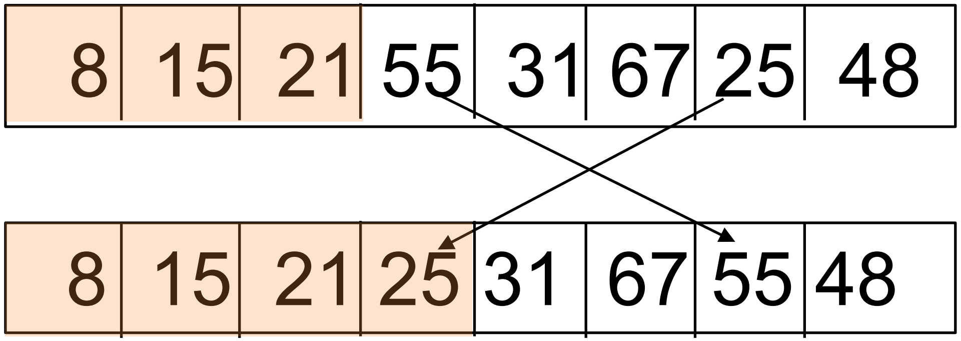
- Arrange the element in a sequence in a specific order (increasingly or decreasingly)
- Example of ”simple” sorting algorithms
  - Selectionsort
  - Insertionsort

Selectionsort

- Start at subscript 0 and find among the rest, the subscript of the smallest element
- Swap the elements on these two subscripts (now the smallest element is on subscript 0)
- Continue in the same way but start from subscript 1 (now the second smallest element is on subscript 1)
- Do the same for the rest of the sequence, from subscript 2 and forward.

Selectionsort (cont.)

”Principle”: locate the smallest element among the unsorted and place it after all already sorted

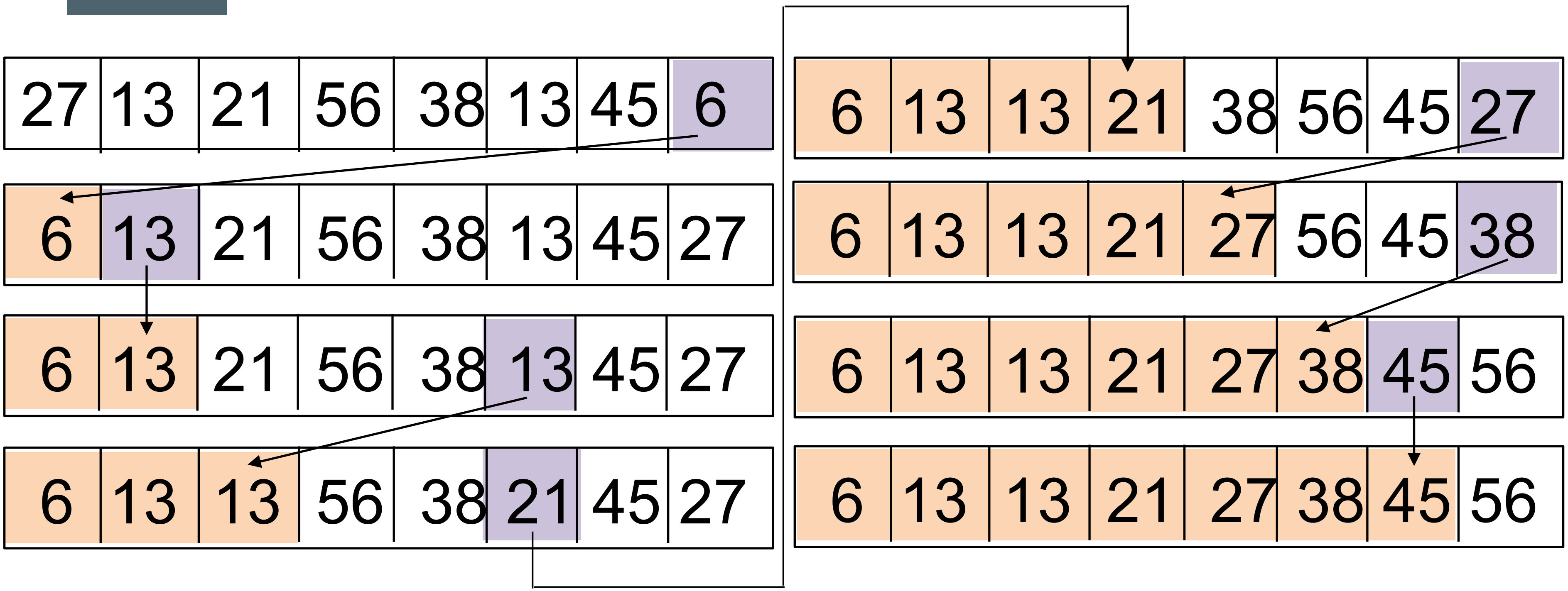


8 15 21 55 31 67 25 48

8 15 21 25 31 67 55 48

Continue until all elements are sorted

Example: Selectionsort



27	13	21	56	38	13	45	6
6	13	21	56	38	13	45	27
6	13	21	56	38	13	45	27
6	13	13	56	38	21	45	27
6	13	13	21	38	56	45	27
6	13	13	21	27	56	45	38
6	13	13	21	27	38	45	56
6	13	13	21	27	38	45	56

Algorithm of Selectionsort

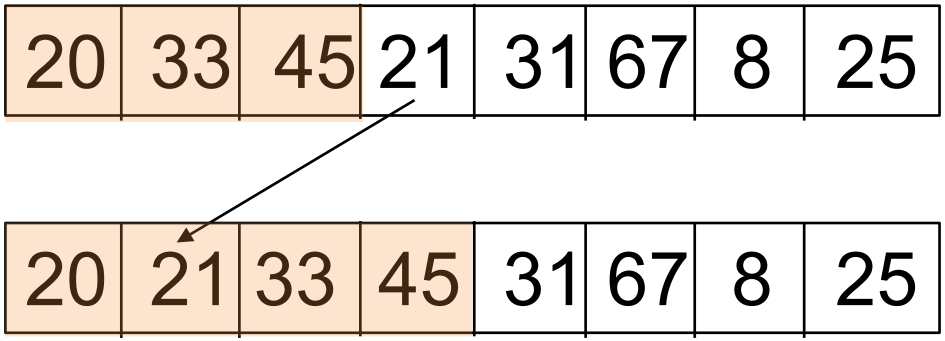
```
Selectionsort(arr, n)

for i=0 to n-1
    indexOfSmallest = i;
    for k=i+1 to n
        if arr[k]<arr[indexOfSmallest]
            indexOfSmallest = k
    swap arr[i] and arr[indexOfSmallest]
```

Insertionsort

- Consider the first element (in subscript 0) as being ordered
- Take the next element (in subscript 1) and put it in the correct subscript in the ordered part of the sequence (now the first two elements are ordered)
- Continue in the same way but start from subscript 2 (now the first three elements are ordered)
- Do the same for the rest of the sequence, from subscript 3 and forward

Insertionsort (cont.)

”Principle”: take one element at a time and put it at the correct place among all already sorted elements

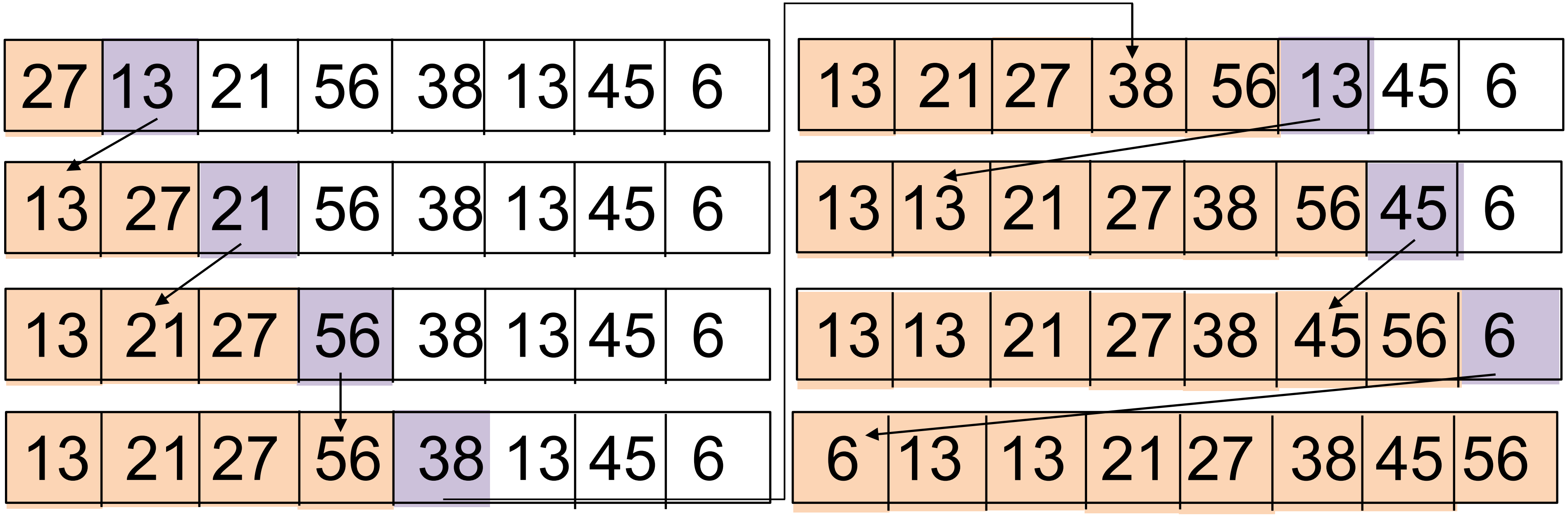


20	33	45	21	31	67	8	25

20	21	33	45	31	67	8	25

Continue until all elements are sorted

Example: Insertionsort



27 13 21 56 38 13 45 6

13 27 21 56 38 13 45 6

13 21 27 56 38 13 45 6

13 21 27 56 38 13 45 6

13 21 27 38 56 13 45 6

13 13 21 27 38 56 45 6

13 13 21 27 38 45 56 6

6 13 13 21 27 38 45 56

Algorithm of Insertionsort

```
Insertionsort(arr, n)

for i=1 to n
    elemToInsert = arr[i];
    k = i-1
    while k>= 0 && elemToInsert<arr[k]
        arr[k+1] = arr[k]
        k--
    arr[k+1] = elemToInsert
```

More sorting algorithms

There are more sofisticated and time efficient sorting algoritms. For example

- Quicksort
- Mergesort
- Heapsort
- …

Searching

- Finding and returning the subscript of a specific element in a sequence (array), -1 if not found
- Lineary search
  - The sequence needs NOT to be sorted
  - Starts from the beginning of the sequence
- Binary search
  - The sequence has to be sorted
  - Starts from the middle of the sequence
  - Excludes half of the remaining sequence in each iteration step

Algorithm Linear search

```
LinearSearch(arr, elemToFind, n)

  i = 0
  while i < n AND arr[i] != elemToFind
      increase i by 1
  if i == n
      i = -1
  return i
```

Lineary search

- Suitable if it is small number of elements
- A good implementation stops when the element is found
- Example:
    - An array contains 1000 elements
    - Best case: number of comparisons?
    - Worst case: number of comparisons?

Binary search

- Calculate the mid subscript based on start subscript and end subscript
- If searched element is at the mid subscript then the searching is done
- Otherwise if the searched element is smaller than the element at the mid subscript the searched element can only be found on the left hand side (the sequence is sorted)
- Otherwise if the searched element is bigger than the element at the mid subscript the searched element can only be found on the right hand side
- Calculate a new mid subscript and continue until the searched element is found or the sub sequence is empty (not found)

Example of Binary search

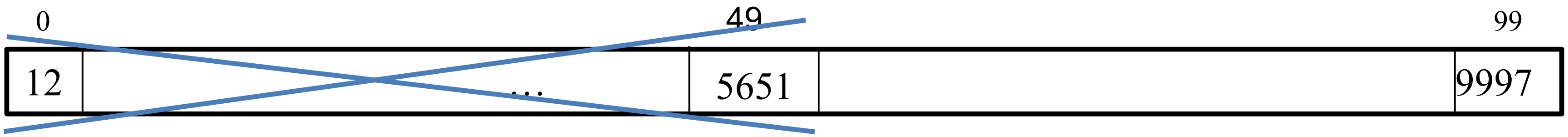
A sorted array containing 100 integers.

The smallest value is 12 and the biggest value is 9997.

The value to find is 7526.

Calculate mid (0+99)/2 = 49

5651 < 7526 => 7526 can only be found on the right side of 5651 (from subscript 50 until subscript 99)

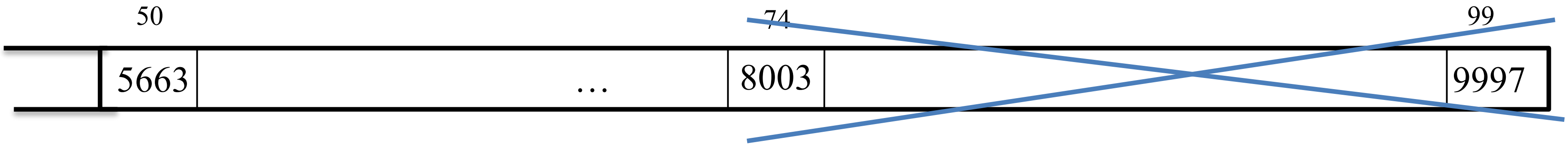


Example Binary search (cont.)

Actual part of the array is the part from subscript 50 until subscript 99

Calculate mid (50+99)/2 = 74

7526 < 8003 => 7526 can only be found on the left side of 8003 (from subscript 50 until subscript 73)

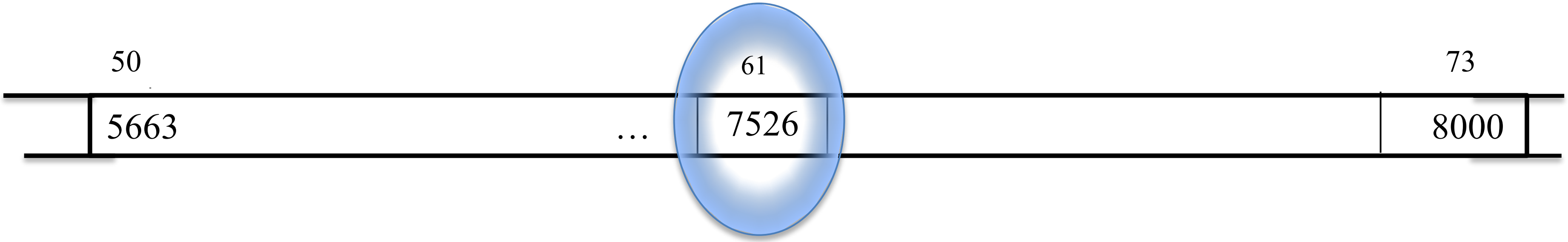


Example Binary search (cont.)

Actual part of the array is the part from subscript 50 until subscript 73

Calculate mid (50+73)/2 = 61

7526 is found at subscript 61!



Algorithm of Binary search

```
BinarySearch(arr, elemToFind, n)

    start = 0, end = n-1, mid = (start + end)/2
    while start <= end AND arr[mid]  != elemToFind
        if elemToFind < arr[mid]
            end = mid – 1
        else
            start = mid + 1
        mid = (start + end)/2
    if start > end
        mid = -1
    return mid
```

Binary search

- Suitable if bigger amount of elements
- Requires that the array is sorted

- Example:
  - An array contains 1000 elements
  - Best case: number of comparisons?
  - Worst case: number of comparisons?

Template function

- Searching and sorting is relevant in many situations and for many datatypes
- It is possible to implement algoritms as functions independent of the datatype
- The concept is known as function templates

Syntax:
template <typename T>
returntype functionName(parameterlist)

- At least one of the parameters must be of the general type T
- The datatype T is decided as the function is called

Class templates

It is also possible to implement classes that are type independent (Generic classes)

Syntax:

```
template<typename T>
class Name
{
   // use the type T
};
```