Sorting Algorithms (Tutorial 2)

Name:

Date:

Contents

Concepts

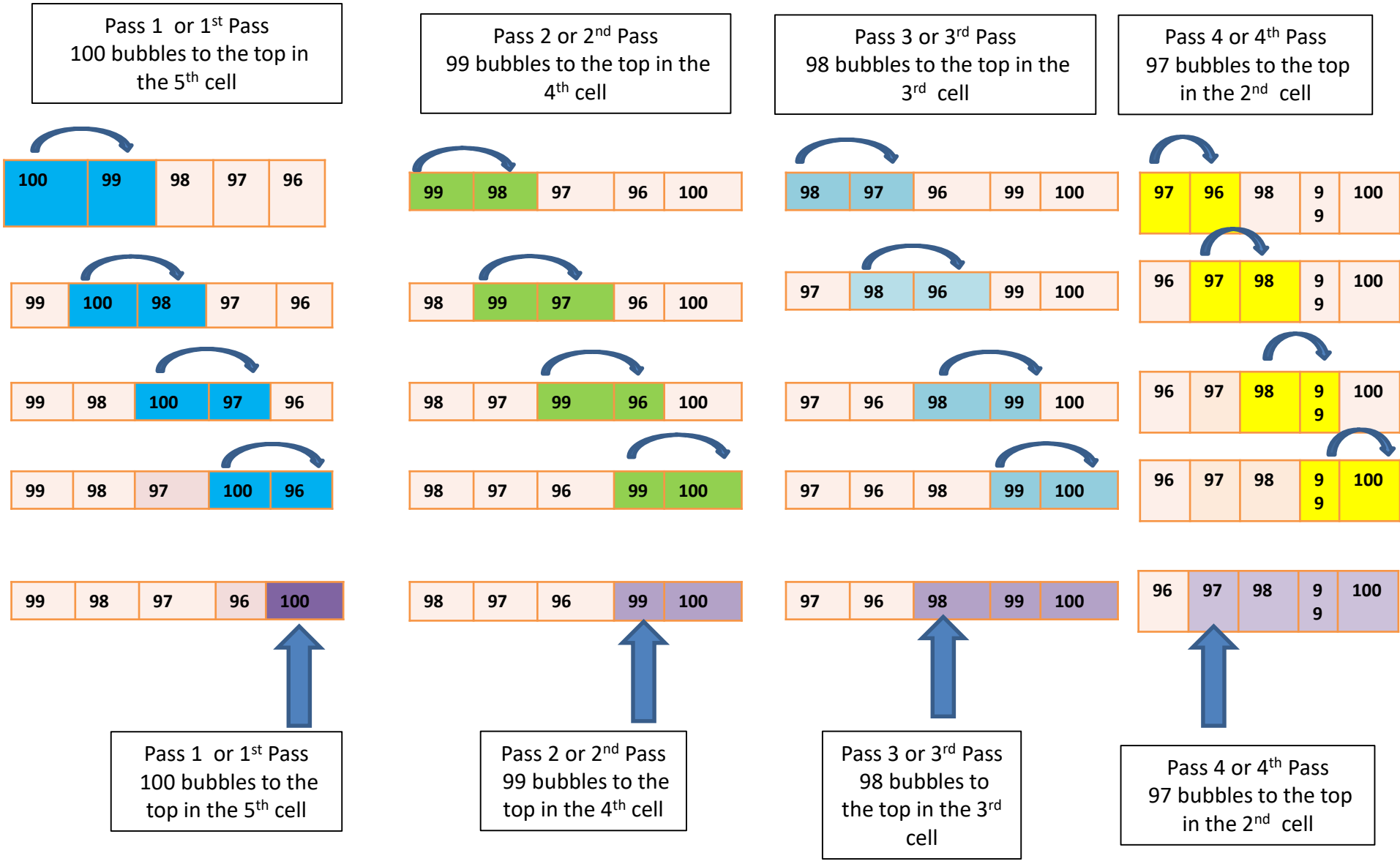
1) Algorithm

2) Decomposition

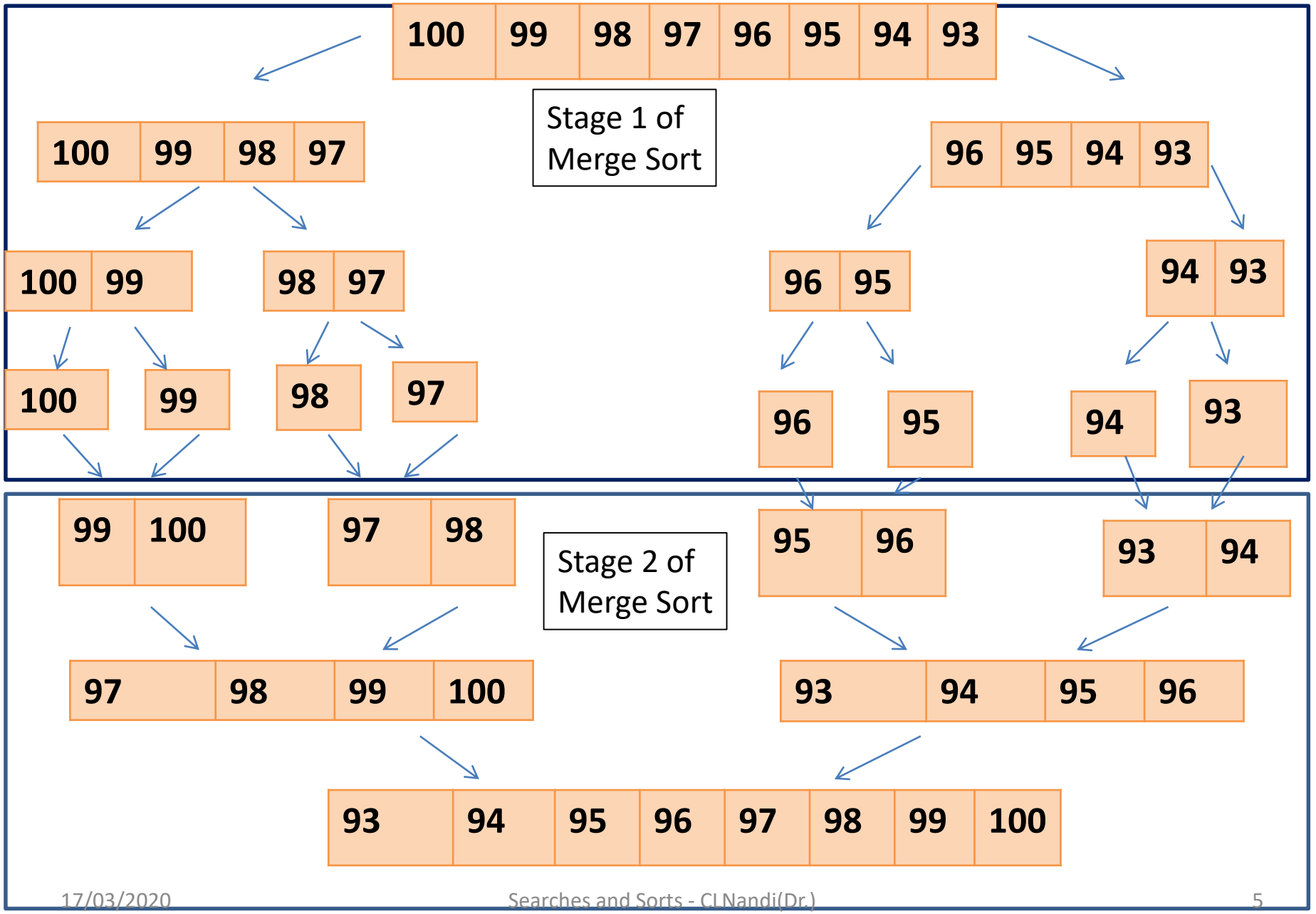
3) Sorting Algorithms
   a) Bubble Sort
      I. Mechanics of Bubble Sort
      II. (Time) Efficiency of Bubble Sort
      III. Advantages and Disadvantages compared to Merge Sort
      IV. Simple Code not using a flag for Bubble Sort (makes use of a swap routine)
      V. Description of Bubble Sort Code using flag

   b) Merge Sort (a “Divide and Conquer” routine)
      I. Mechanics of Merge Sort
      II. (Time) Efficiency of Merge Sort
      III. Advantages and Disadvantages compared to Bubble Sort
4) Efficiency
5) The Swap Routine

Bubble Sort Illustration



Merge Sort – a "divide and conquer routine" - Illustration



Bubble and Merge Sorts Description

- A **bubble sort** works by repeatedly going through the list to be sorted comparing each pair of adjacent elements.
- If the elements are in the wrong order they are swapped.

- **Merge Sort**
- This is a two stage sort.
- In the first stage, the list is successively divided in half, forming two sublists, until each sublist is of length one.
- In the 2nd stage, each pair of sublists is repeatedly merged to produce new sorted sublists until there is only one sublist remaining.
- As each sublist is merged, they are merged in order. Merging the final 2 sublists results in the sorted list.

Efficiencies (1) - Comparing Bubble and Merge Sorts

- The way to analyse competing algorithms is to count the number of steps it takes.

	10 Items	**500**
Bubble Sort	~100 steps	~250000 steps
Merge Sort	~33 Steps	~4482 steps

Efficiencies(2) of Bubble and Merge Sort

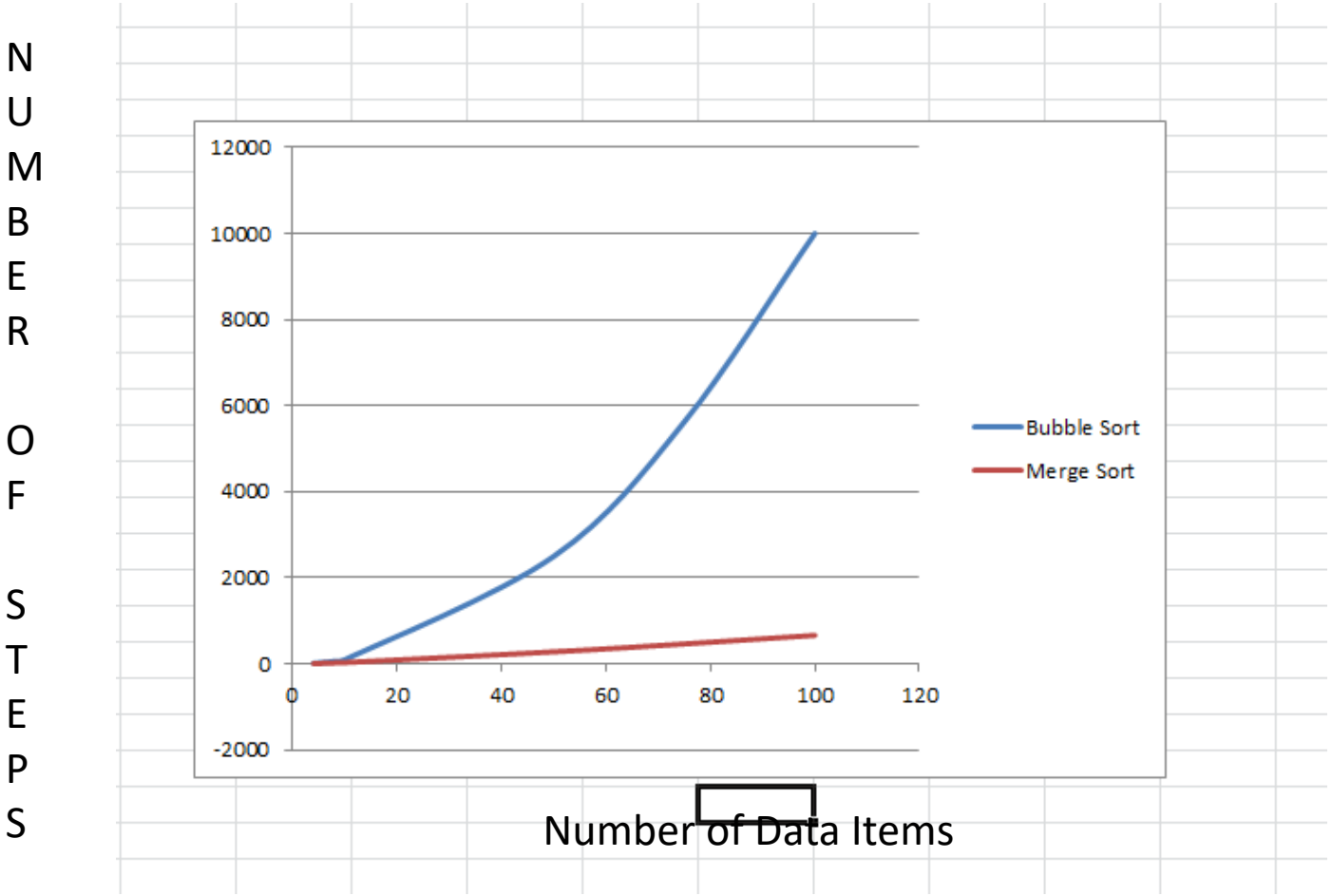
- **Bubble Sort**
- With bubble sort the
- (time) efficiency is:

- $O(n^2)$

- **Merge Sort**
- With merge sort we say the (time) efficiency is:

- $O(n\log_2 n)$

Efficiencies (3) - Graph Comparing Bubble and Merge Sort Efficiencies



Differences Between Bubble and Merge Sort

	Bubble Sort	Merge Sort
1	Less efficient than merge sort as it takes more time to execute (i.e more number of steps/instructions executed)	More efficient than bubble sort as it takes less time to execute (fewer number of steps)
2	Easier to program/code/implement than merge sort.	Disadvantage - Harder to program/code/implement than bubble sort.
3	Bubble sort requires less memory than a merge sort./	Disadvantage – merge sort requires more memory/space than bubble sort. Memory is required to store the sublists. This can be a problem for a large list.

Simple Bubble Sort Code without Flag

```
# Array to be sorted
numbers = [45,62,13,98,9,50]

# Finding length of array
numItems = len(numbers)

# Doing the comparisons between adjacent elements
# and swapping values if necessary

for i in range(numItems-1):
    print ("i =",i)
    for j in range(numItems - i - 1):
        print("j =",j)

        # Swapping routine
        if numbers[j] > numbers[j + 1]:
            temp = numbers[j]
            numbers[j] = numbers[j + 1]
            numbers[j + 1] = temp

# Printing the Output
 print (numbers)
```

Swapping Code needed in Bubble Sort

- Bubble Sort works by Swapping Variables

Temp = a

a = b

b = temp

- If a = 9 and b = 6 the
- trace table is:-

Temp	a	b
	9	6
9	6	9

Pseudo-code for Simple Bubble Sort

Here element of the array is referred to using an index in square brackets. Each element of the array is referred to using an index in square brackets. In the array below, the 1st element of the array, 9, is held in numbers[0[, and the last element 2, is held in numbers[7].

```
numbers ← [9,5,4,15,3,8,11,2]
numItems ← len(numbers)   #get number of items in the array
FOR i ←  TO numItems  -  2
    FOR j ← 0 to numItems –  i – 2
       IF numbers[j] > numbers[j+1] THEN
            temp ← numbers[ j ]
            numbers[ j ] ← numbers[ j + 1 ]
            numbers [j + 1] ← temp
      ENDIF
   ENDFOR
     OUTPUT numbers
 ENDFOR
```

If you run the program the **output** is:

[5, 4, 9, 3, 8, 11, 2, 15]
[4, 5, 3, 8, 9, 2, 11, 15]
[4, 3, 5, 8, 2, 9, 11, 15]
[3, 4, 5, 2, 8, 9, 11, 15]
[3, 4, 2, 5, 8, 9, 11, 15]
[3, 2, 4, 5, 8, 9, 11, 15]
[2, 3, 4, 5, 8, 9, 11, 15]

Description of Code Using a flag for bubble sort

- Suppose you have a list of numbers
- [7,2,15,24]
- After the 1st pass the list will be sorted [2,7,15,24]
- We can use a variable called Flag and at the beginning of a pass set it to False.
- If a swap is made during a pass the Flag variable is changed to True.

- Then the variable is reset to False at the beginning of the 2nd pass and if it remains False after this pass – which means no swaps have taken place, then the program stops running.

Amended Algorithm for Bubble soet using a Flag

```
numbers ← [9,5,4,15,3,8,11,2]
numItems ← len(numbers)     #get number of items in the array
Flag ← True           #indicates when a swap is made
 i = 0
While i < (numItems - 1 ) AND  (flag = TRUE)
    Flag ← False
    FOR j ← 0 to numItems –  i – 2
       IF numbers[j] > numbers[j+1] THEN
            temp ← numbers[ j ]
            numbers[ j ] ← numbers[ j + 1 ]
            numbers [j + 1] ← temp
            Flag ← True
      ENDIF
    ENDFOR
     i = i + 1

 ENDWHILE
 OUTPUT numbers
```

Questions on Bubble and Merge Sort(1)

(1) Carry out a bubble sort on the following set of numbers. The numbers are to be in descending order (highest to lowest)

6 8 1 17 27 11 15 3 14 42 5

(a) What is the order of the items after the first pass?

(b) (i) Using the simple bubble sort method which does not use a flag, how many passes through the data will be made?

(ii) What is the maximum number of passes on a list of 2 items?

(iii) What is the maximum number of passes on a list of 3 items?

(iv) What is the maximum number of passes on a list of 10,000 items?

Working Out

Questions (2)

- (2) Carry out a merge sort on the following set of numbers. The numbers are to be sorted in ascending order.

  - 6 8 1 17 27 11 15 3

- (a) Write out the four sorted sublists after the first phase of Stage 2(the merge process).

(b) Write out the two sorted sublists after the second phase of the merge process.

© Write out the complete list after the 3rd phase of the merge process.

Working Out

Questions (3)

(3) Which algorithm , the bubble sort or the merge sort, do you think is more efficient?

(4) Is a binary search always faster than a linear search? Explain your answer.

Working Out

Questions (4)

- (5) A bubble sort is used to sort the following numbers in ascending order:
  - 34 56 89 23 12 77 49 44
  - (i) What order will the numbers be in after the 1st pass?

    (ii) How many passes will be required to sort the items? (No flag is used to indicate a sorted list).

Working Out

Questions(5)

- (6) A bubble sort is used to sort the same list – see below.
  - 34 56 89 23 12 77 49 44
  - During the merge phase, the following 4 pairs of numbers need to be merged into two groups of four:
  - (34,56), (23,89), (12,77),(44,49)
  - (i) What will be the contents of each group of four numbers after the next phase of the merge?

Working Out

- The End

Questions(6)

(7) A list of surnames is held in sorted order. The names are:

Beck, Coe, Ford, Grey, Hill, Lunn, Pugh, Roiss, Shaw, Taft, Ward

(a) State which names would be examined when searching for the name 'Grey' using :-

(i) a linear search
(ii) a binary search

(b) State which names would be examined when searching for the name 'James' using:-

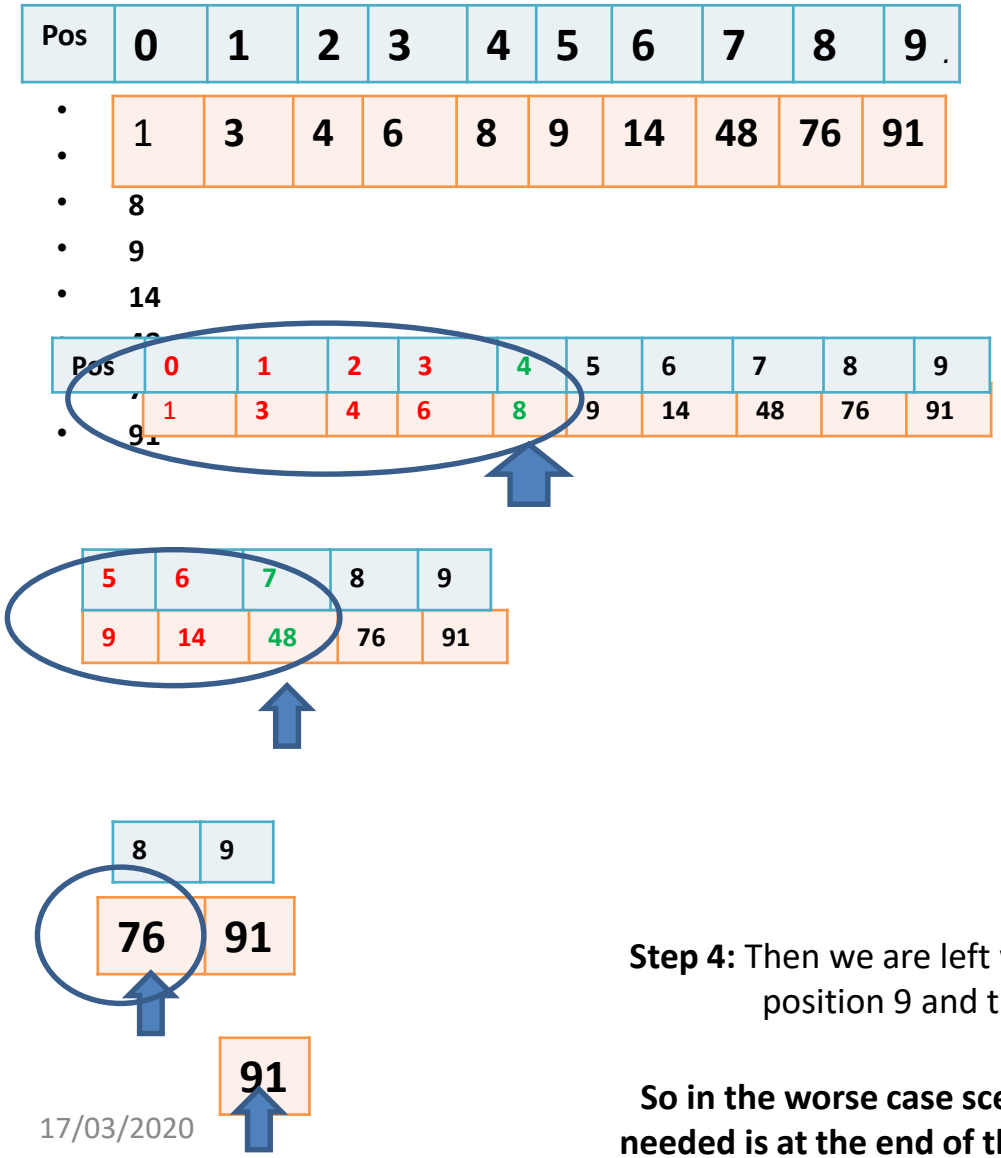
(i) a linear search
(ii) a binary search

© In a list of 1000 items, state the maximum number of names that would have to be searched to find a particular name using:-

(i) a linear search
(ii) a binary search

Working Out

Binary Search – Worse Case Scenario



To find any number, such as 91, we take the middle element of the array.

- **Step 1:** But in this case strictly speaking there is no middle array, so we look at the middle & arbitrarily take the left element of the middle, which is 8 in this case.

- Then we ask if 76 is less than or more than 8. As 76 is more than 8, we discard the LHS of the array,

Step 2: Then we are left with values from positions 5 to 6. And look at the middle value which is 48. As 91 is greater than 48 we discard 48 and to its left.

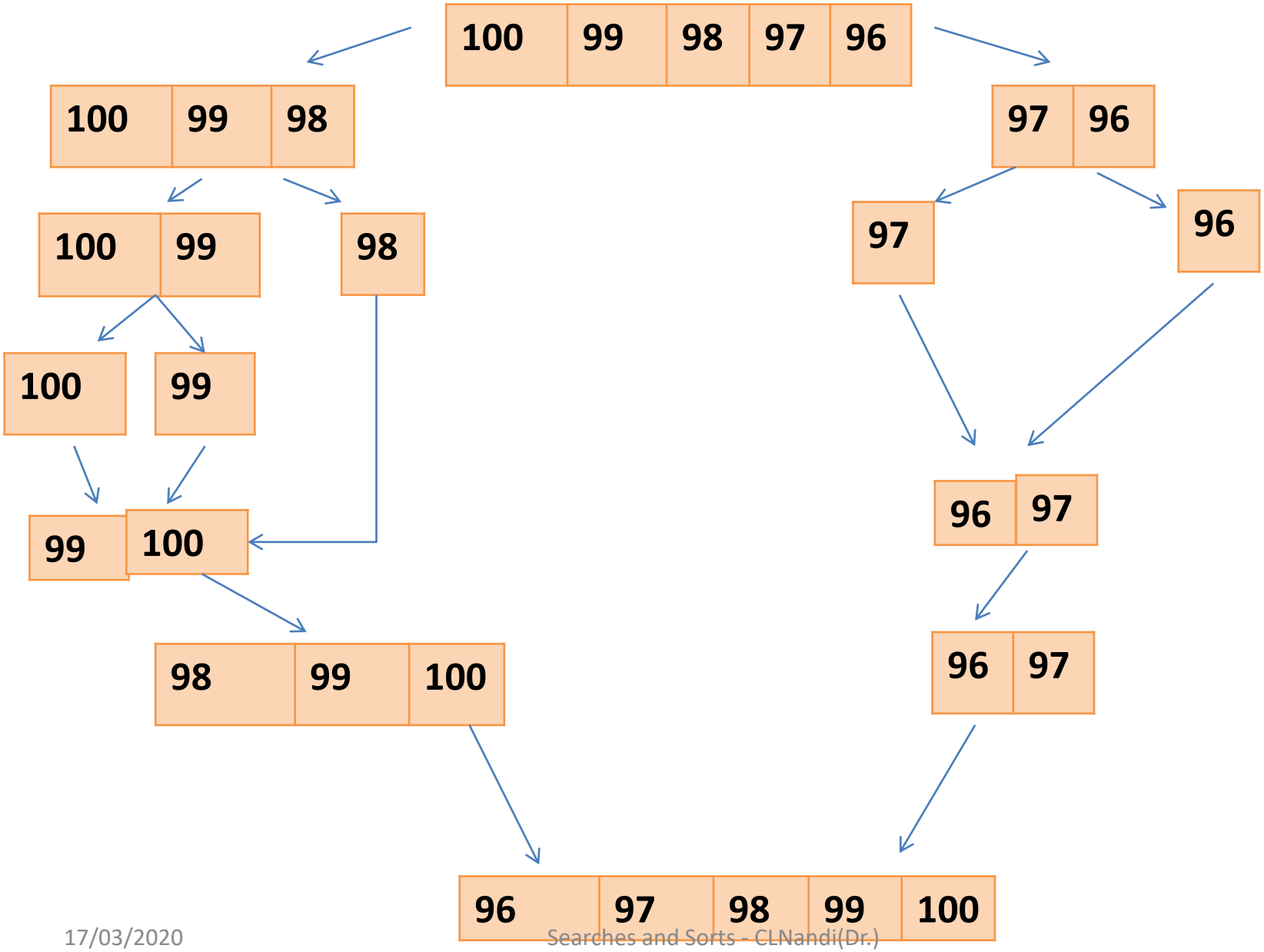
Step 3: Then we are left with values in positions 8 and 9. We look for the middle value and arbitrarily take the value to the left of it, which is 76 and we discard that.

Step 4: Then we are left with one value in position 9 and that is 91

So in the worse case scenario of binary search where the value needed is at the end of the array, it took 4 steps to find the value.

- End -

Merge Sort


100 99 98 97 96
100 99 98
97 96
100 99
98
97
96
100
99
96 97
99 100
98 99 100
96 97
96 97 98 99 100