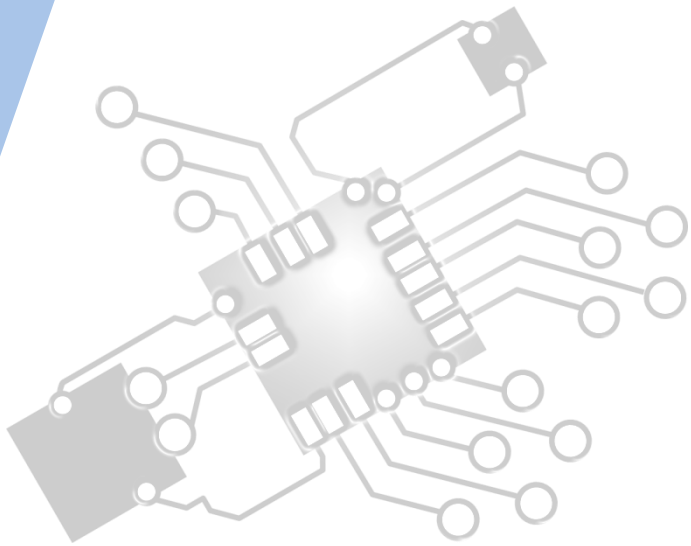
Computational thinking, problem-solving and programming:

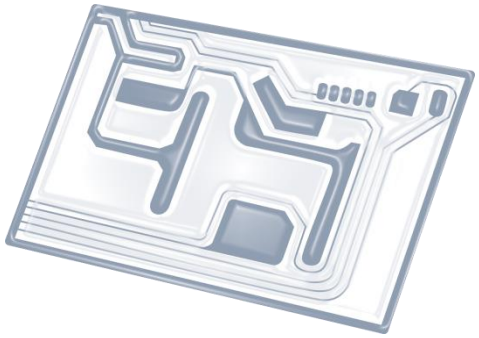
Connecting computational thinking and program design

IB Computer Science



Content developed by
Dartford Grammar School
Computer Science Department

HL Topics 1-7, D1-4



1: System design

2: Computer Organisation

3: Networks

4: Computational thinking

5: Abstract data structures

6: Resource management

7: Control

D: OOP

HL & SL 4.2 Overview

4.2.1 Describe the characteristics of standard algorithms on linear arrays

4.2.2 Outline the standard operations of collections

4.2.3 Discuss an algorithm to solve a specific problem

4.2.4 Analyse an algorithm presented as a flow chart

4.2.5 Analyse an algorithm presented as pseudocode

4.2.6 Construct pseudocode to represent an algorithm

4.2.7 Suggest suitable algorithms to solve a specific problem

4.2.8 Deduce the efficiency of an algorithm in the context of its use

4.2.9 Determine the number of times a step in an algorithm will be performed for given input data

1: System design

2: Computer Organisation

3: Networks

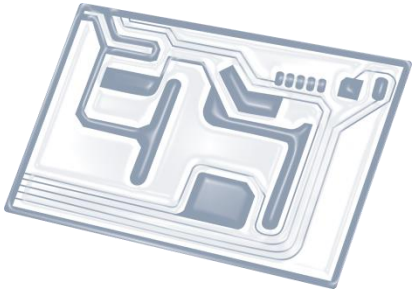
4: Computational thinking

5: Abstract data structures

6: Resource management

7: Control

D: OOP

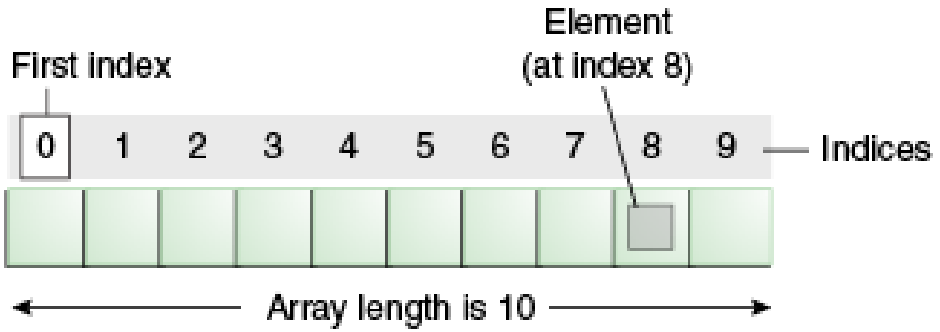


Topic 4.2.1

Describe the **characteristics** of **standard algorithms** on **linear arrays**

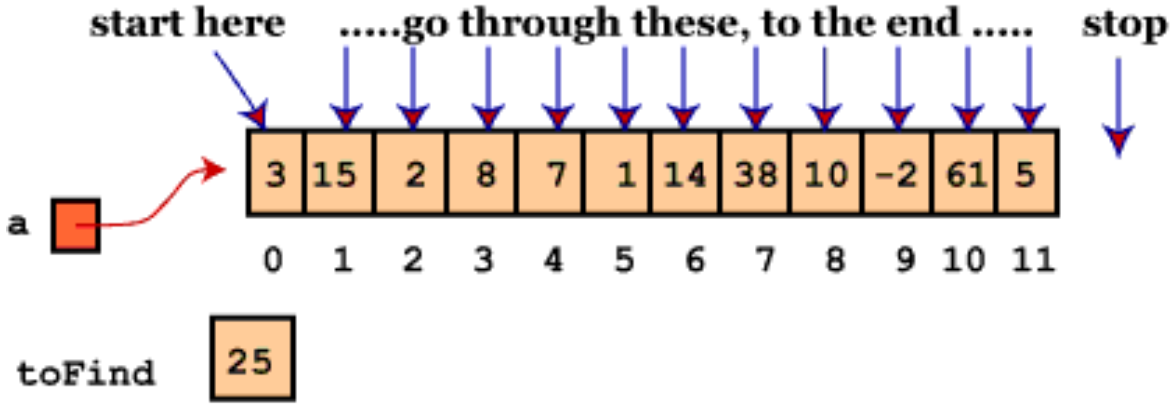
The four key standard algorithms:

- Sequential **search**
- Binary **search**
- Bubble **sort**
- Selection **sort**

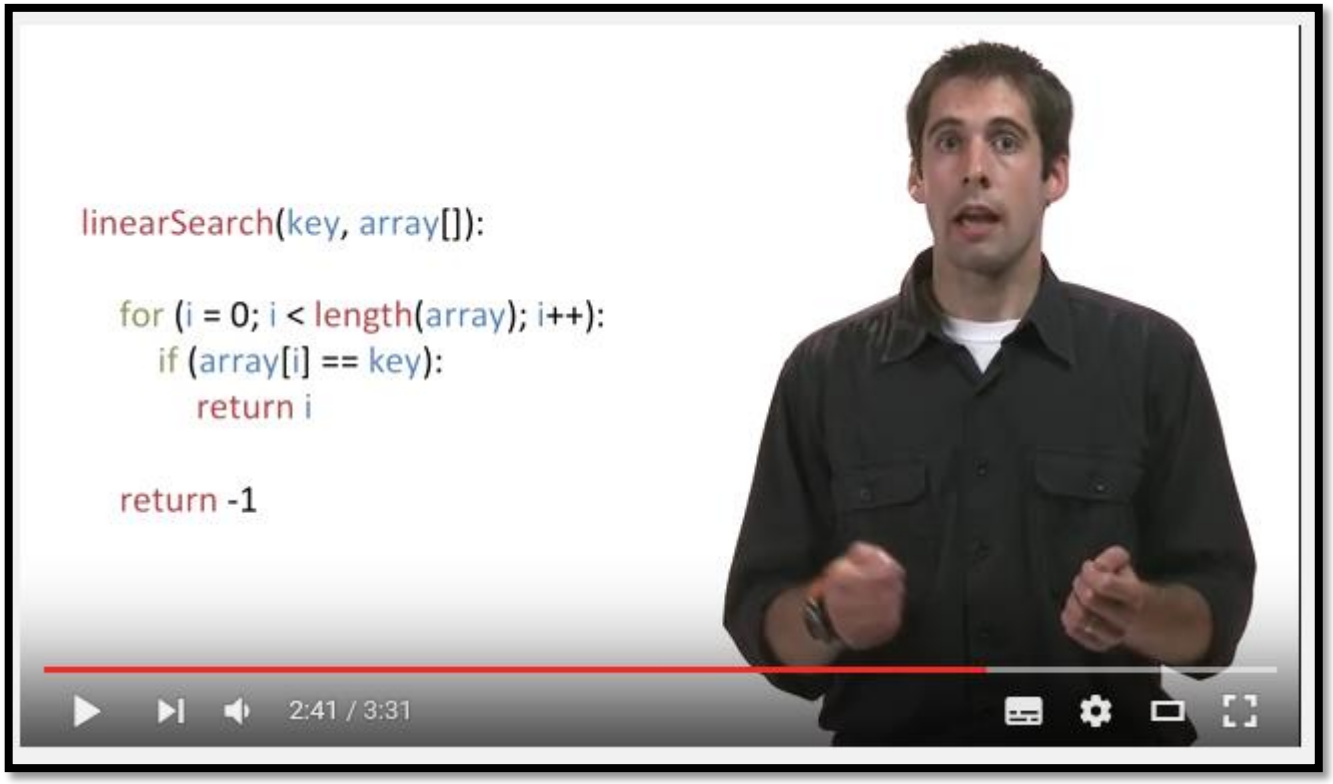


Sequential search

- **Linear search** or **sequential search** is an algorithm to find an item in a list.
- It starts at the first element and compares each element to the one it's looking for until it finds it.
- Commonly used with **collections** (which are unsorted lists of items) and text/csv **file reading.**



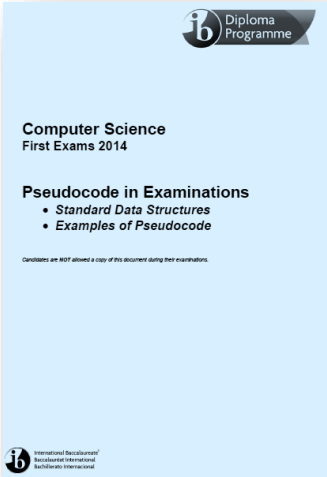
Sequential search (video)



https://www.youtube.com/watch?v=CX2CYIJLwfg

Sequential search (Pseudocode)

```
NAMES = "Bob","Betty","Kim","Lucy","Dave"

output "These names start with D"

loop while NAMES.hasNext()
    NAME = NAMES.getNext()
    if firstLetter(NAME) = "D" then
      output NAME
    end if
end loop
```



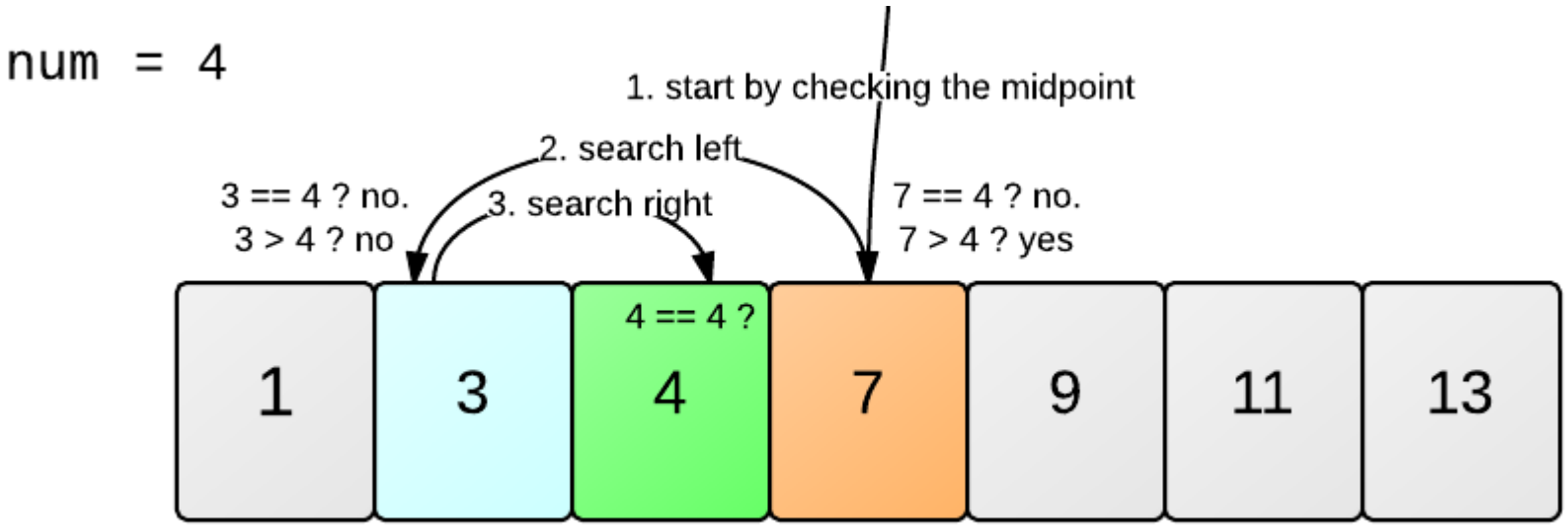
Binary search

- **Binary search**, also known as **half-interval search**, is a search algorithm that finds the position of a target value within a sorted array.
- It works by comparing the target value to the **middle element** of the array;
- If they are unequal, the lower or upper half of the array is eliminated depending on the result and the search is repeated in the remaining sub-array until it is successful.
- It only applies to **SORTED arrays** (where there are usually no duplicate values, or duplicates do not matter)

Binary search (video)

https://www.youtube.com/watch?v=D5SrAga1pno

Binary search



Binary search

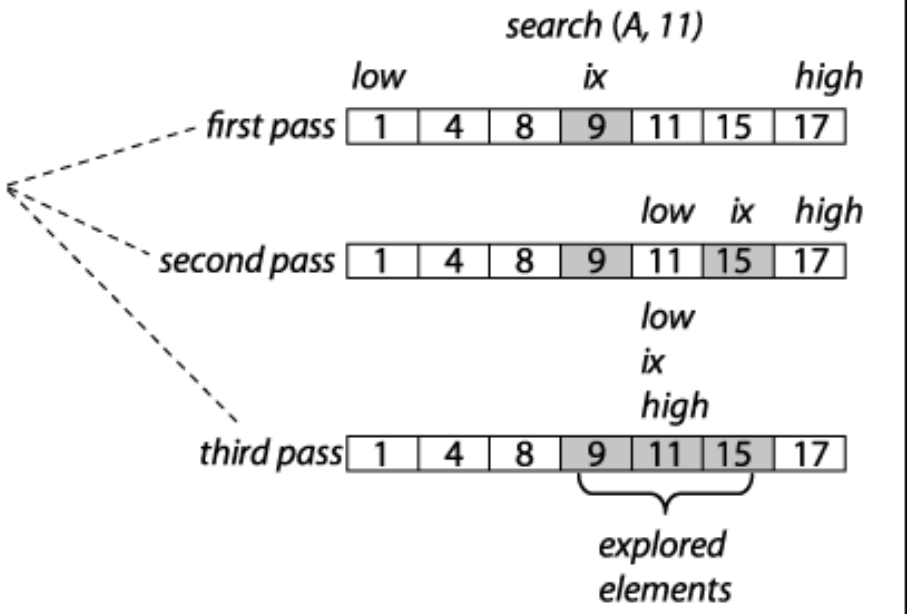
BINARY SEARCH

Best	Average	Worst
O (1)	O (log n)	O (log n)


Array

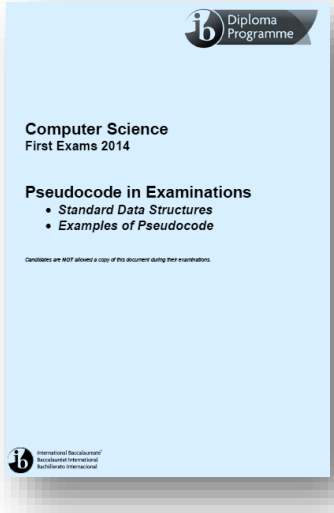

Divide and Conquer

```
search (A, t)
1.    low = 0
2.    high = n –1
3.    while (low ≤ high) do
4.       ix = (low + high)/2
5.       if (t = A[ix]) then
6.          return true
7.       else if (t < A[ix]) then
8.          high = ix – 1
9.       else low = ix + 1
10.   return false
end
```



Binary search (Pseudocode)

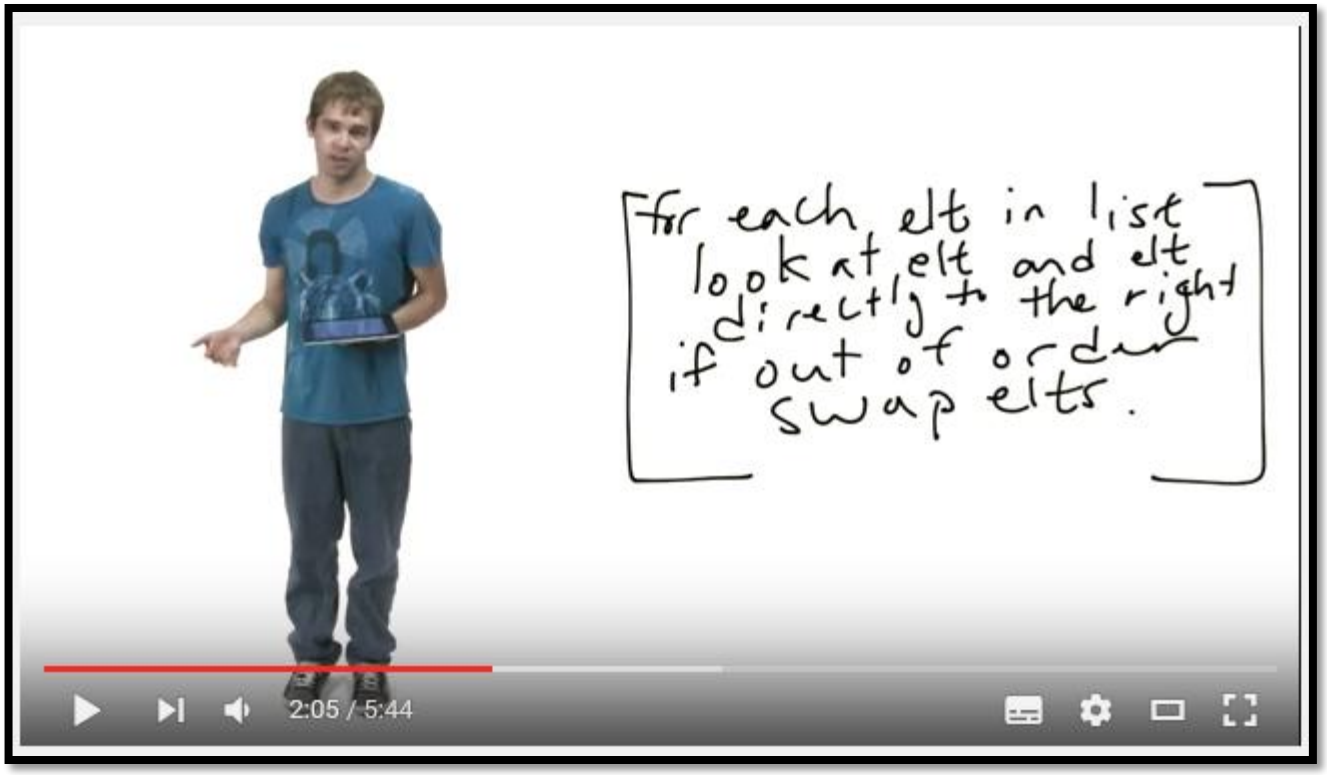
```
ID = [1001,1002,1050,1100,1120,1180,1200,1400]
NAME = ["Apple","Cherry","Peach","Banana","Fig","Grape","Olive","Mango"]

output "Type the ID number that you wish to find"
input TARGET

LOW = 0
HIGH = 7
FOUND = -1

loop while FOUND = -1 AND LOW <= HIGH
  MID = LOW + HIGH  div 2
  if ID[MID] = TARGET then
      FOUND = MID
  else if TARGET < ID[MID] then
      HIGH = MID - 1
  else
      LOW = MID + 1
  end if
end while

if FOUND >= 0 then
    output TARGET , ":" , NAME[FOUND]
else
    output TARGET , " was not found"
end if
```



Bubble sort

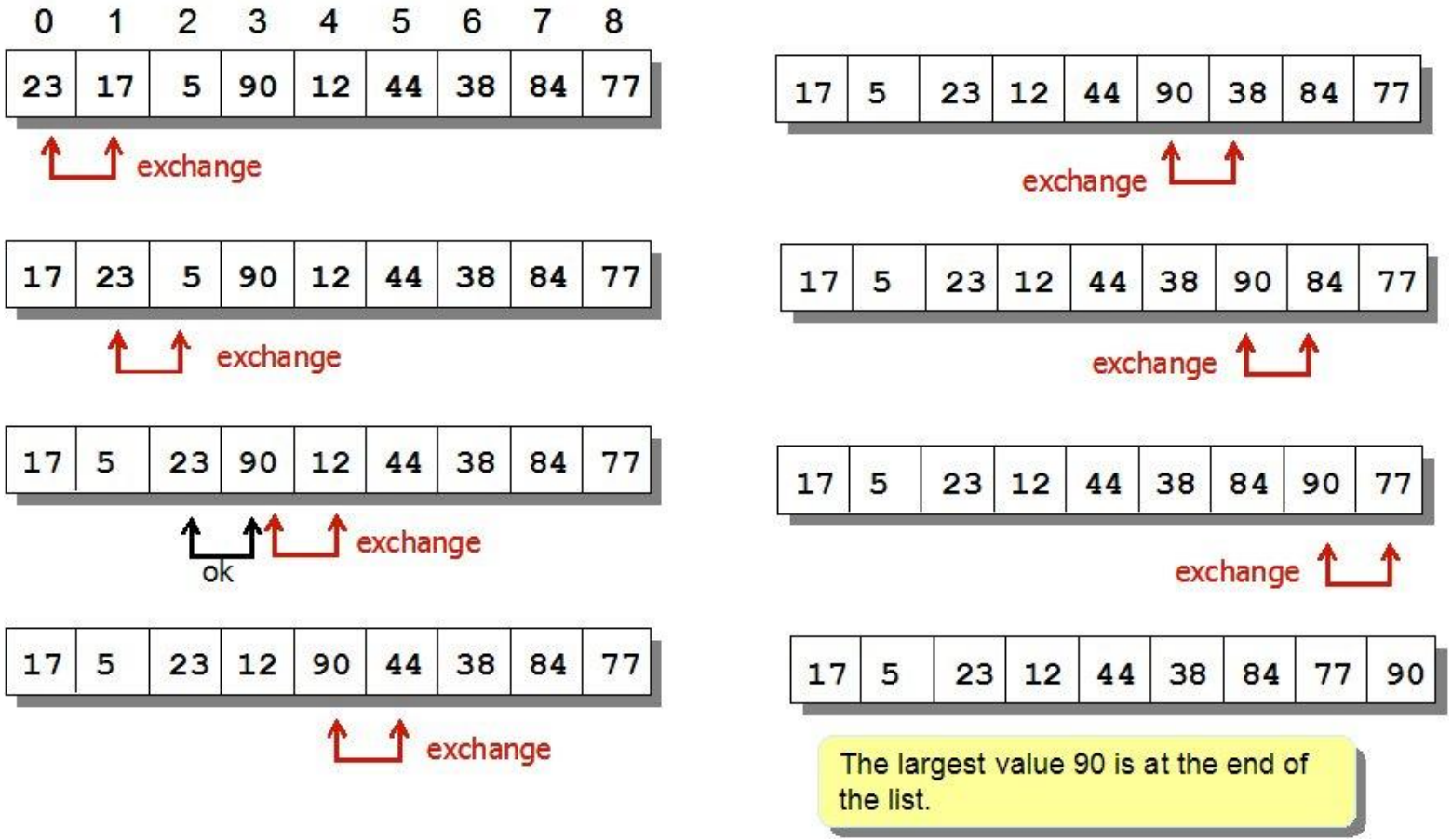
- Bubble sort is a simple sorting algorithm that repeatedly steps through the list to be sorted, **compares each pair** of adjacent items and **swaps them if they are in the wrong** order.
- The pass through the list is **repeated until no swaps are needed**, which indicates that the list is sorted.
- The algorithm, which is a comparison sort, is named for the way smaller elements "bubble" to the top of the list.
- Although the algorithm is simple, it is **too slow and impractical for most problems**

Bubble sort (video)



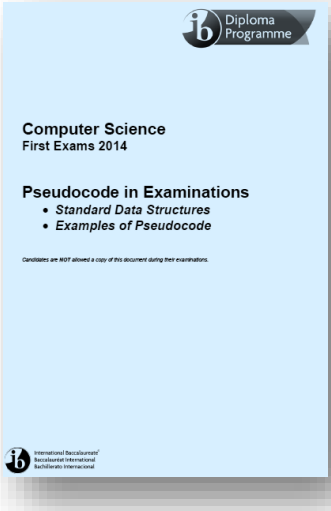
https://www.youtube.com/watch?v=8Kp-8OGwphY

Bubble sort


0 1 2 3 4 5 6 7 8
23 17 5 90 12 44 38 84 77
exchange
17 23 5 90 12 44 38 84 77
exchange
17 5 23 90 12 44 38 84 77
ok
exchange
17 5 23 12 90 44 38 84 77
exchange
17 5 23 12 44 90 38 84 77
exchange
17 5 23 12 44 38 90 84 77
exchange
17 5 23 12 44 38 84 90 77
exchange
17 5 23 12 44 38 84 77 90
The largest value 90 is at the end of the list.

Bubble sort (Pseudocode)

```
NUMS = [15,30,85,25,40,90,50,65,20,60]

output "Before sorting"
loop C from 0 to 9
   output NUMS[C]
end loop

loop PASS from 0 to 8
    loop CURRENT from 0 to 8
        if NUMS[CURRENT] < NUMS[CURRENT + 1] then
          TEMP = NUMS[CURRENT]
          NUMS[CURRENT] = NUMS[CURRENT+1]
          NUMS[CURRENT+1] = TEMP
        end if
    end loop
end loop
```

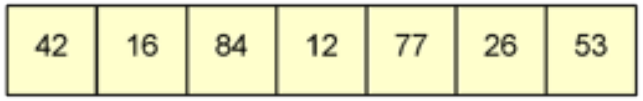


Selection sort

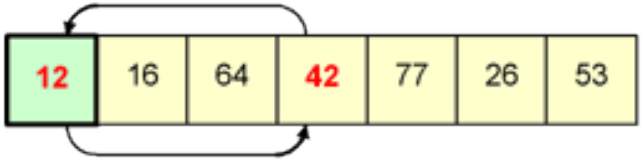
- Selection sort is a sorting algorithm and it is **inefficient** on **large lists**
- Selection sort is noted for its **simplicity**, and it has performance advantages over more complicated algorithms in certain situations, **particularly where memory is limited**.
- The algorithm **divides** the input list into two parts: the sublist of items **already sorted**, which is built up from left to right at the front (left) of the list, and the sublist of **items remaining to be sorted** that occupy the rest of the list.
- Initially, the sorted sublist is **empty** and the unsorted sublist is **the entire input list**.
- The algorithm proceeds by finding the **smallest** (or **largest**, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

Selection sort (video)

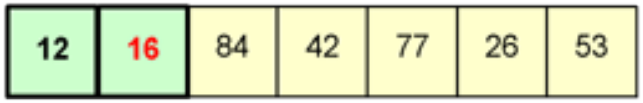
https://www.youtube.com/watch?v=f8hXR_Hvybo



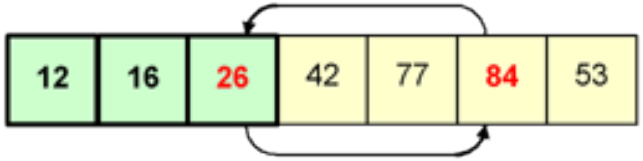
The array, before the selection sort operation begins.



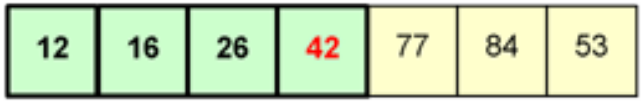
The smallest number (**12**) is swapped into the first element in the structure.



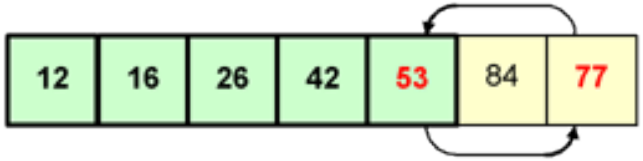
In the data that remains, **16** is the smallest; and it does not need to be moved.



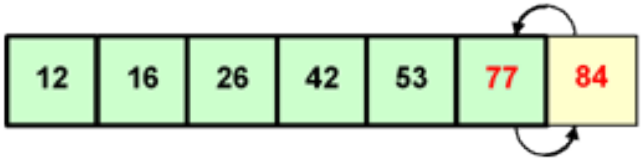
26 is the next smallest number, and it is swapped into the third position.



42 is the next smallest number; it is already in the correct position.



53 is the smallest number in the data that remains; and it is swapped to the appropriate position.



Of the two remaining data items, **77** is the smaller; the items are swapped. *The selection sort is now complete.*

Selection sort (Pseudocode)

```
A - an array containing the list of numbers
numItems - the number of numbers in the list

for i = 0 to numItems - 1
    for  j = i+1 to numItems
        if A[i] > A[j]
            // Swap the entries
            Temp = A[i]
            A[i] = A[j]
            A[j] = Temp
        end if
    end loop
end loop
```