Problem Solving Techniques and Data Structures

Course Objectives

- To introduce problem solving and develop the ability to analyze a givenproblem, understand and apply various techniques to solve problem.
- To introduce the concepts of Data Structures

Session Plan

- To understand problem solving, problem classification and problem solvingtechniques
- To understand Data Structures, Why Data Structures, Types of Data Structures andoperations permitted on different Data Structures

Introduction to Introduction to Problem Solving Techniques

Skills of Software Developer

- The following are the ten skills to be possessed by a Software Developer
  - Analytical ability
  - Analysis
  - Design
  - Technical knowledge
  - Programming ability
  - Testing
  - Quality planning and Practice
  - Innovation
  - Team working
  - Communication

Performance measures

- The following are the five points deciding the performance of a softwaredeveloper
  - Timeliness
  - Quality of work
  - Customer Orientation
  - Optimal solution
  - Team satisfaction

Problem-Definition

- **Definition:** A *problem* is a puzzle that requires logical thought or mathematics tosolve
- What is **Problem solving** ?

  The act of defining a problem; determining its cause; identifying, prioritizing andselecting alternatives for a solution; and implementing that solution.

Problem Solving-Steps

Analyze and understand the Problem

↓

Select a method to solve the problem

↓

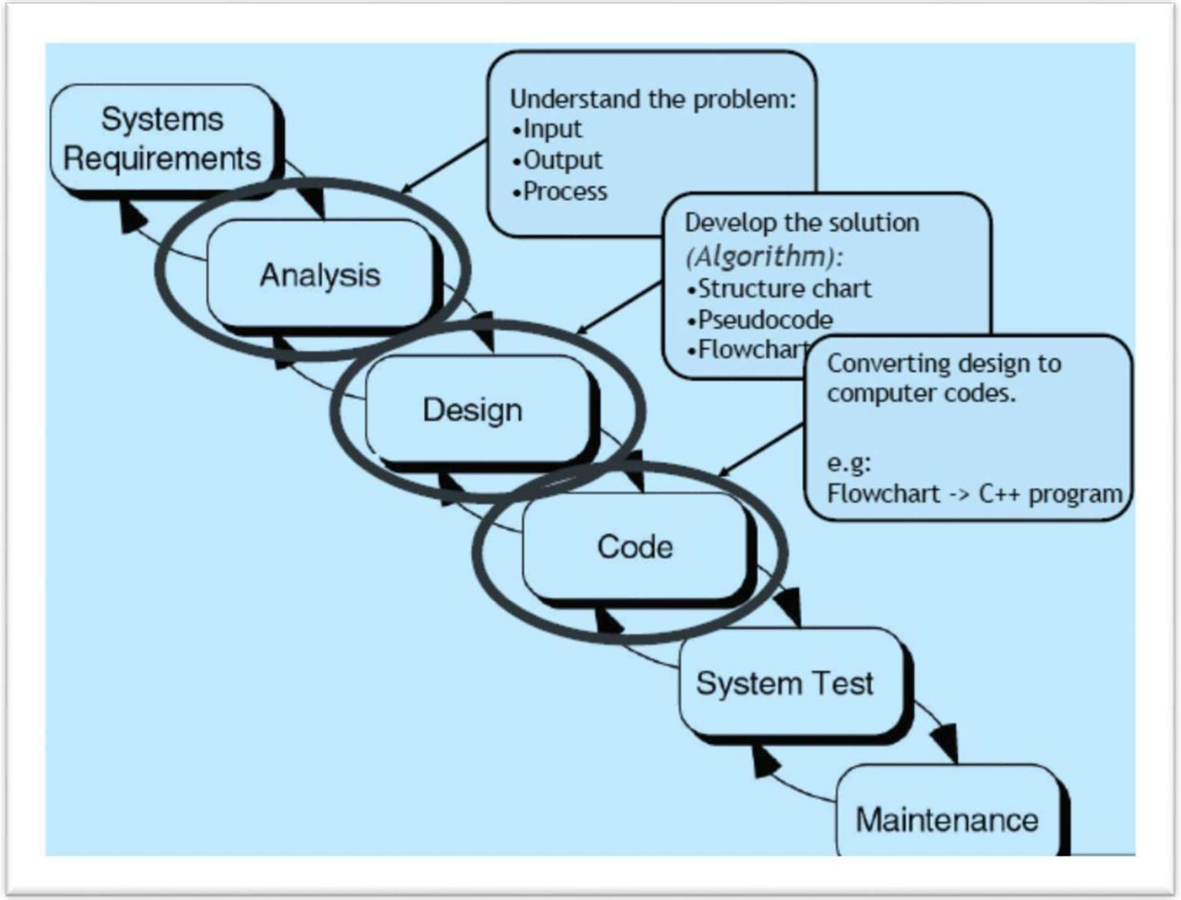
Design a solution

↓

Develop the solution

↓

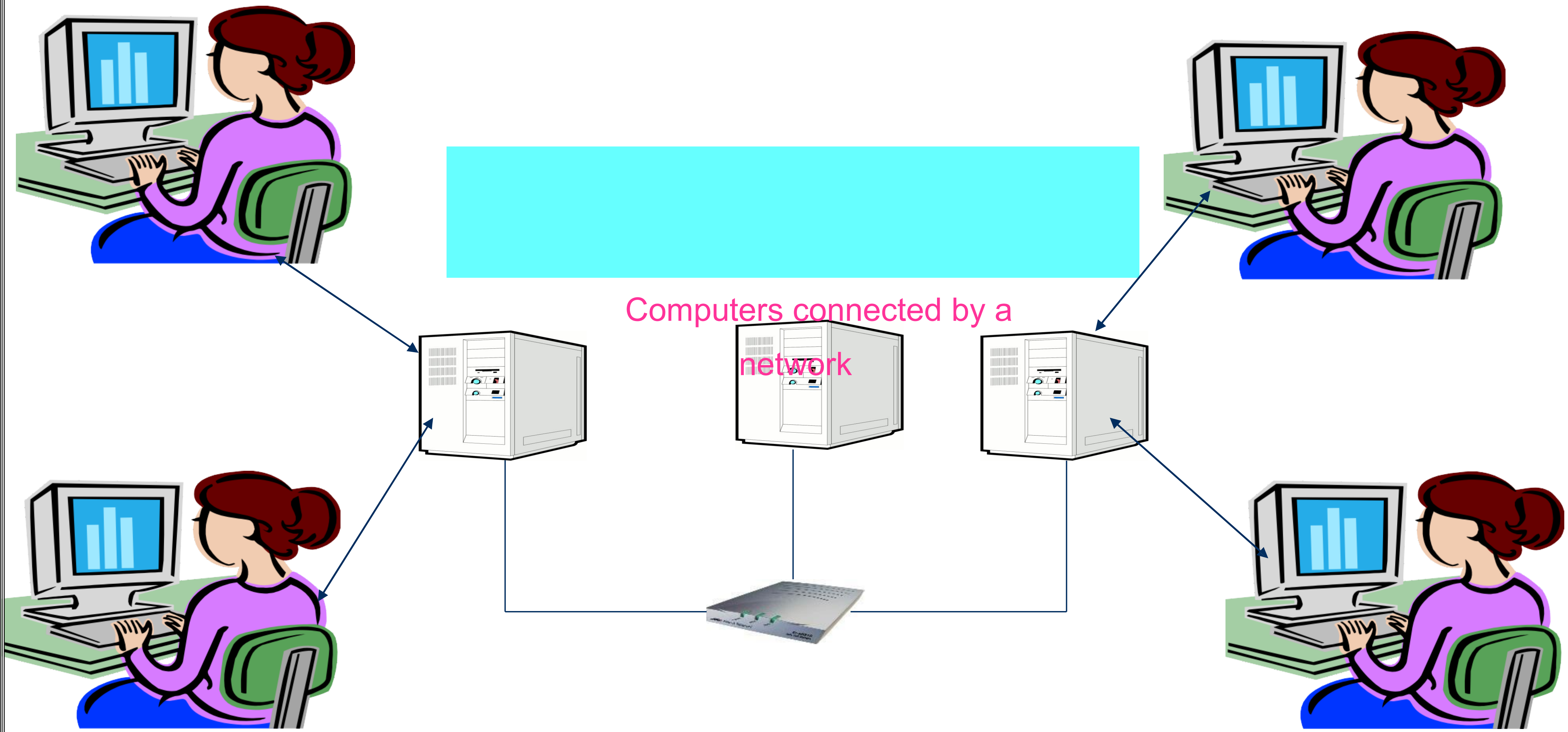
Test the solution



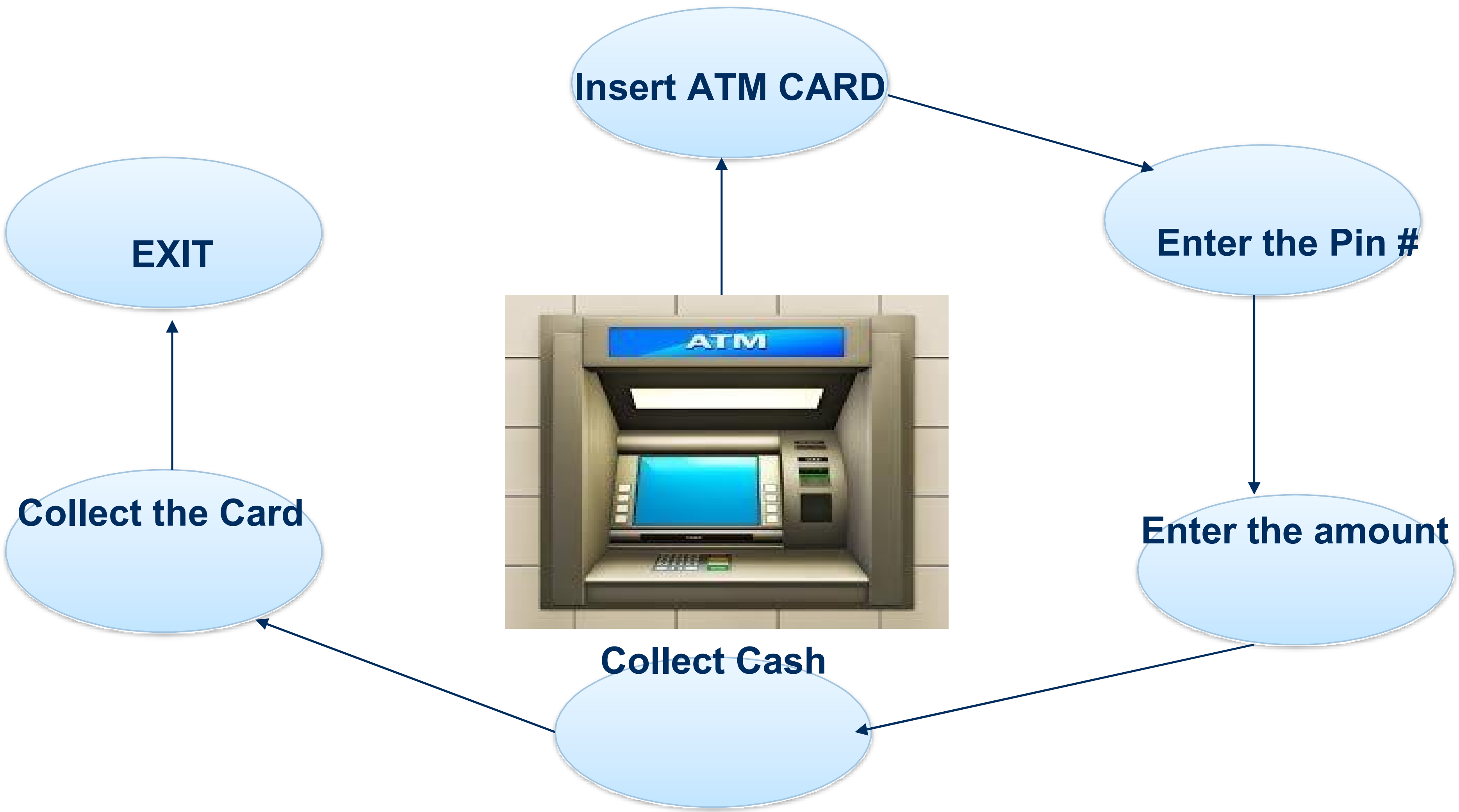
Problem Classification

- Concurrent: Operations overlap in time
- Sequential: Operations are performed in a step-by-step manner
- Distributed: Operations are performed at different locations
- Event-Based: Operations are performed based on the input

Distributed/Concurrent Problems



Sequential/Event based-Example



Problem solving methods

- Heuristic approach/ Brute Force technique
- Greedy approach
- Divide and Conquer technique
- Dynamic Programming technique

Heuristic/ Brute Force approach

- Brute force approach is a straight forwardapproach to solve the problem. It is directly based on the problem statement and the concepts
- Brute force is a simple but a very costlytechnique
- Example: Breaking Password

Watch the video to get more clarity on Heuristic approach

https://www.youtube.com/wa

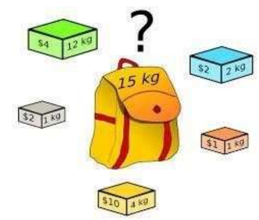
Greedy Approach

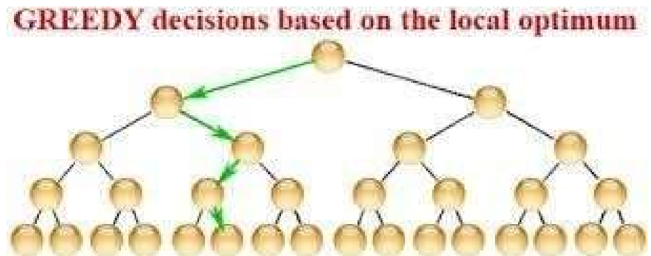
- Greedy design technique is primarily used in Optimization problems

- The Greedy approach helps in constructing a solution for a problem through a sequence of steps where each step is considered to be a partial solution. This partialsolution is extended progressively to get the complete solution

- The choice of each step in a greedy approach is done based on the following
  - It must be feasible
  - It must be locally optimal
  - It must be irrevocable
- Example: TSP- Traveling Salesman Problem

- https://www.youtube.com/watch?v=SC5CX8drAtU





Divide-and-Conquer

The most-well known algorithm design strategy:

1. Divide instance of problem into two or more smaller instances

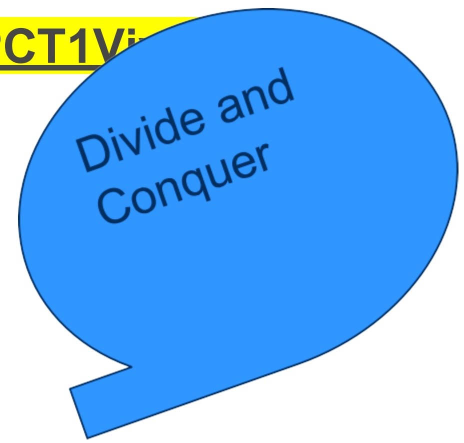
2. Solve smaller instances recursively

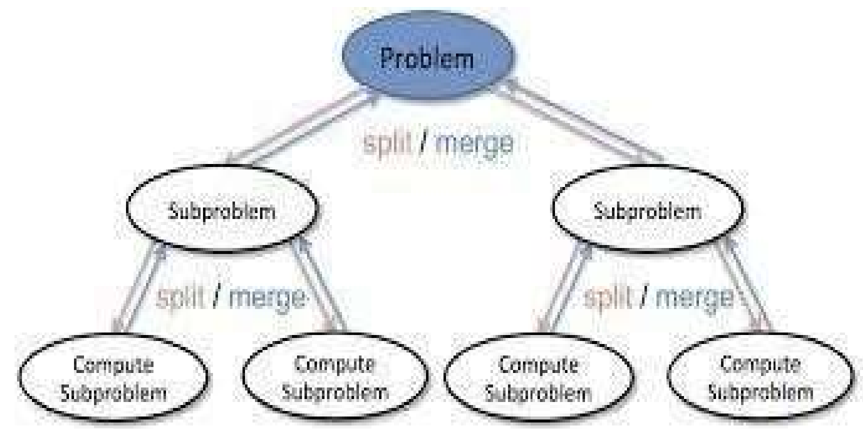
3. Obtain solution to original (larger) instance by combining thesesolutions

Example:

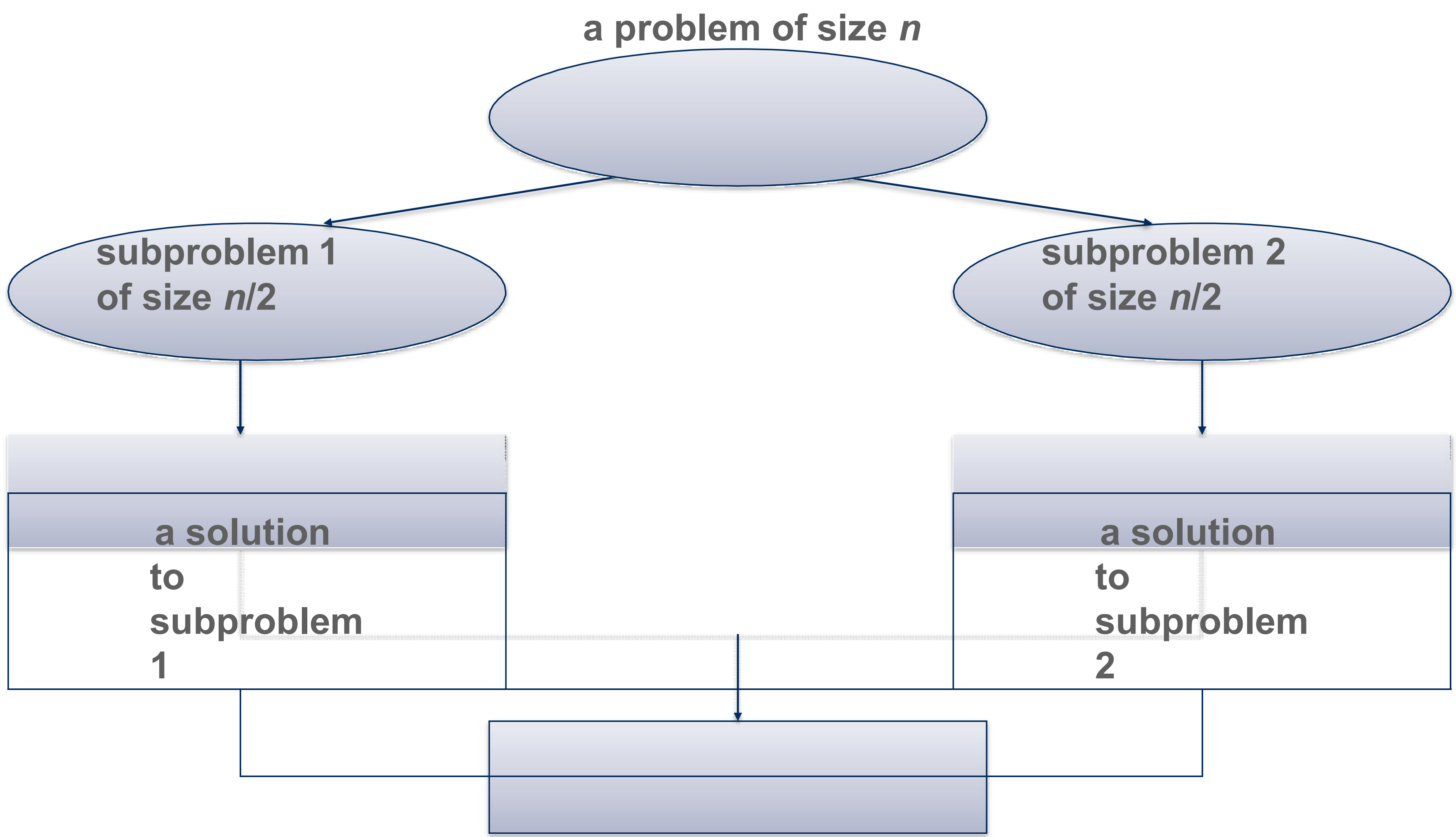
– Binary Search

https://www.youtube.com/watch?v=wVPCT1Vi





Divide-and-Conquer Technique (cont.)



a solution to
the original problem

What's the difference?

Consider the problem of exponentiation: Compute a^n

- Brute Force
- Divide and conquer

Dynamic Programming

- Dynamic Programming is a design principle which is used to solve problems with overlappingsub problems
- It solves the problem by combining the solutions for the sub problems
- “Programming” here means “planning”
- Main idea:
    - set up a recurrence relating a solution to a larger instance to solutions of somesmaller instances
    - solve smaller instances once
    - record solutions in a table
    - extract solution to the initial instance from that table
- The difference between Dynamic Programming and Divide and Conquer is that the sub problems in Divide and Conquer are considered to be disjoint and distinct whereas in DynamicProgramming they are overlapping

Dynamic Programming-Example

You have three jugs, which we will call A, B, and C. Jug A can hold exactly 8 cups of water, B can hold exactly 5 cups, and C can hold exactly 3 cups. A is filled to capacity with 8 cups of water. B and C are empty. We want you to find a way of dividing the contents of A equally between A and B so that both have 4cups. You are allowed to pour water from jug to jug.

Solution

Step 1: First fill the 8L bucket full.

Step 2: Pour the water from 8L bucket to 5L bucket. Water remaining in 8L bucket is 3L. Step 3: Pour the water from 5L bucket to 3L bucket. Water remaining in 5L bucket is 2L.

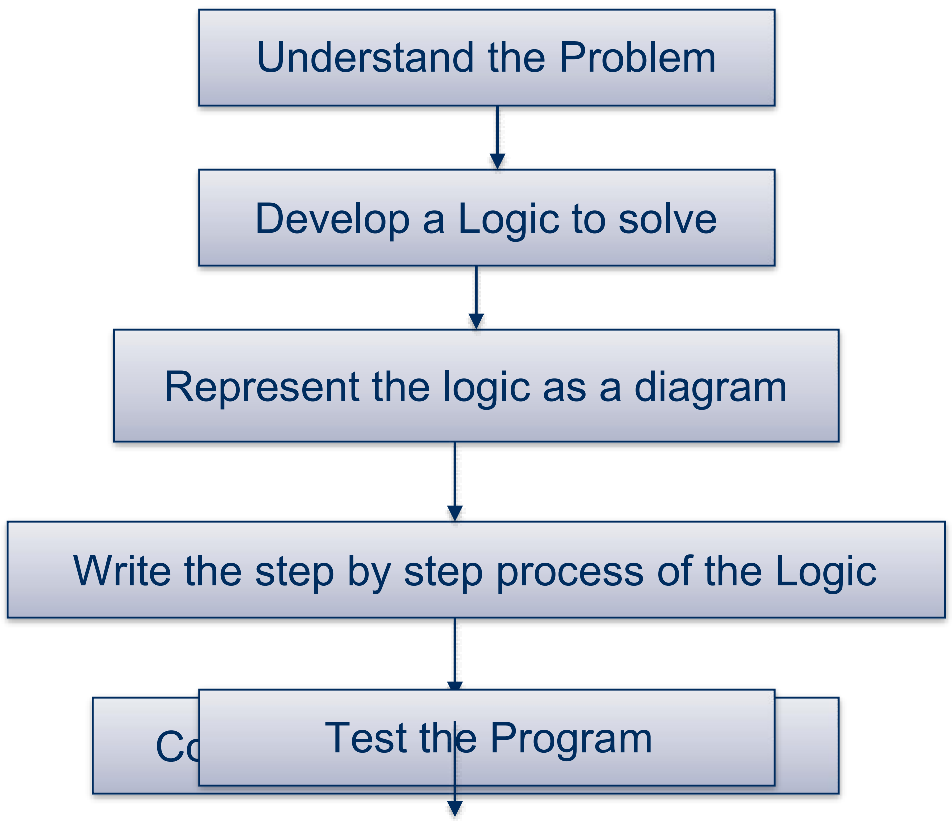
Step 4: Pour the water from 3L bucket to 8L bucket. Water in 8L bucket is 6L now and 3Lbucket gets empty.

Step 5: Pour the water from 5L bucket to 3L bucket. Water in 3L bucket is 2L now and 5Lbucket gets empty.

Step 6: Pour the water from 8L bucket to 5L bucket. Water remaining in 8L bucket is 1L 5Lbucket gets full.

Step 7: Pour the water from 5L bucket to 3L bucket. Water remaining in 5L bucket is now 4L as 3L bucket already had 2L of water and when we poured water from 5l bucket to 3L bucket we poured 1L of water from 5L bucket and thus the remaining water in 5L bucket isnow 4L.

Computer Based Problem Solving -Steps



- Analysis of the Problem
- Selecting a solution method
- Draw Flowcharts
- Develop Algorithms using Pseudo codes
- Develop Program using Programming language
- Test the program

Modeling Tools

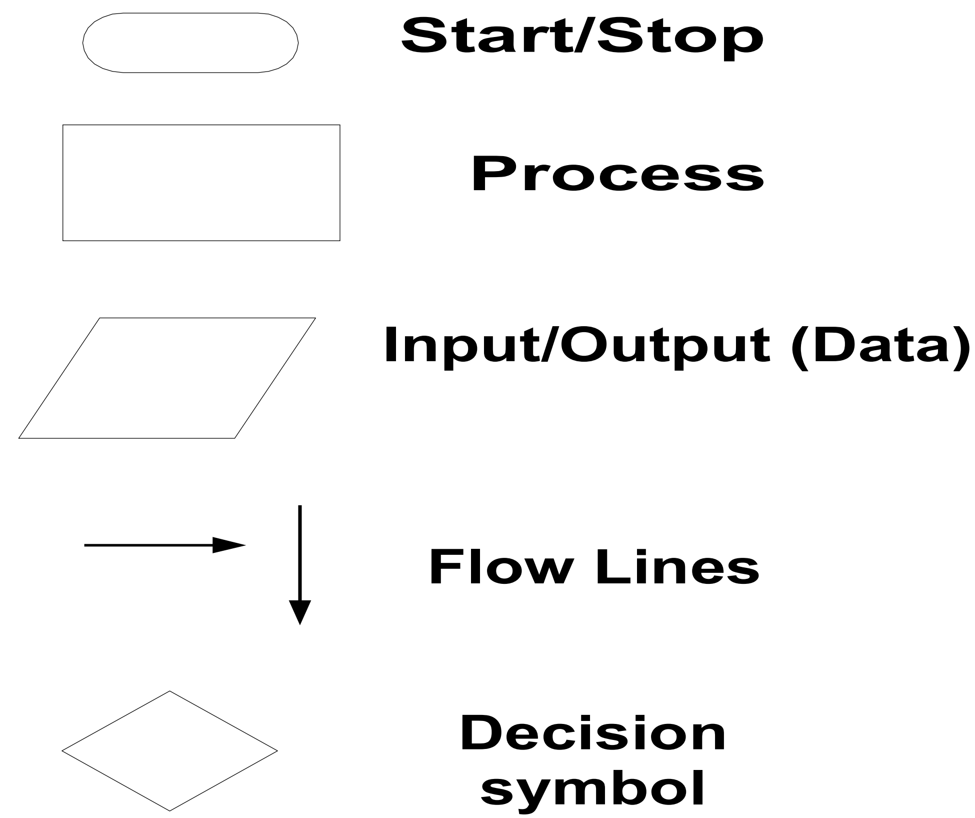
- Diagrammatic Representation of Logic
- Different Types:
  - Flow Charts
  - Data flow Diagrams
  - Entity Relationship diagram
  - Unified Modeling Language

Flow Charts

Flow Charts

- A flowchart is a diagrammatic representation of an algorithm
- A flow chart is an organized combination of shapes, lines and text
  that graphically illustrates a process or structure

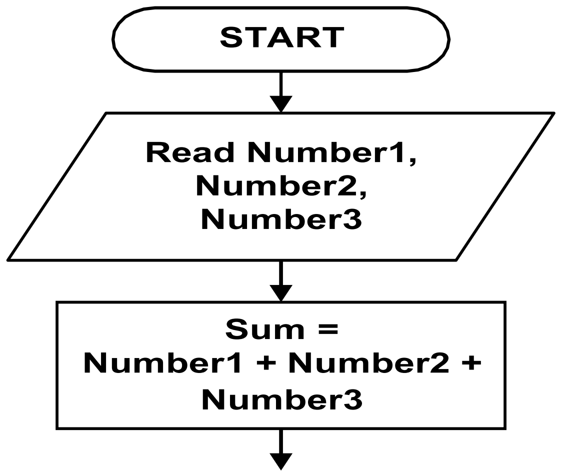
Symbols used

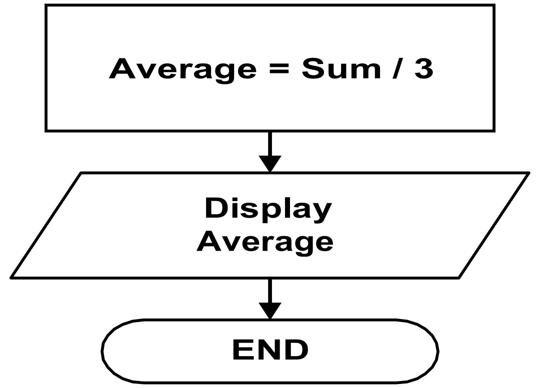


Connector

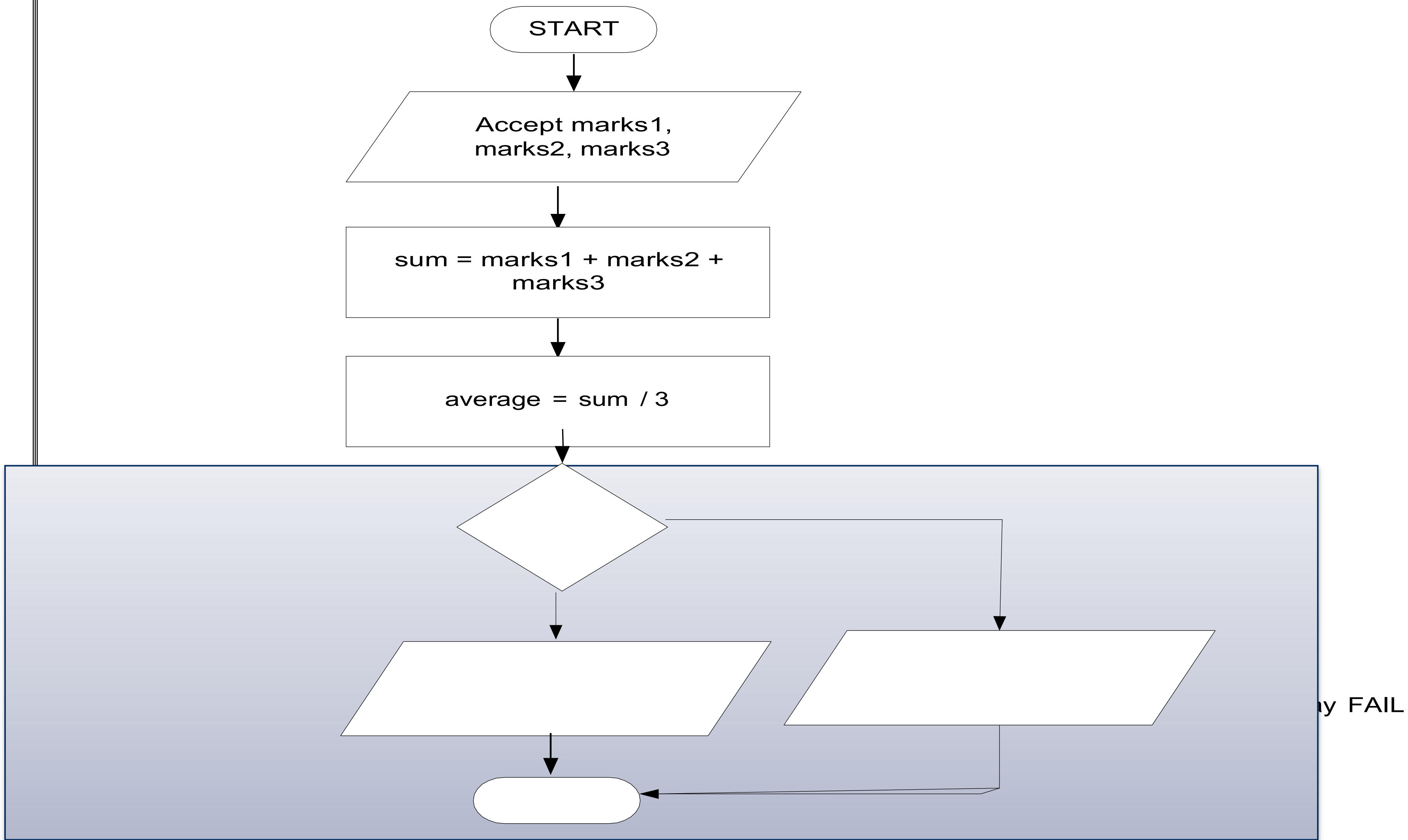
Example: Flow Chart (Sequential)Find the

average of three numbers




Average = Sum / 3
Display
Average
END

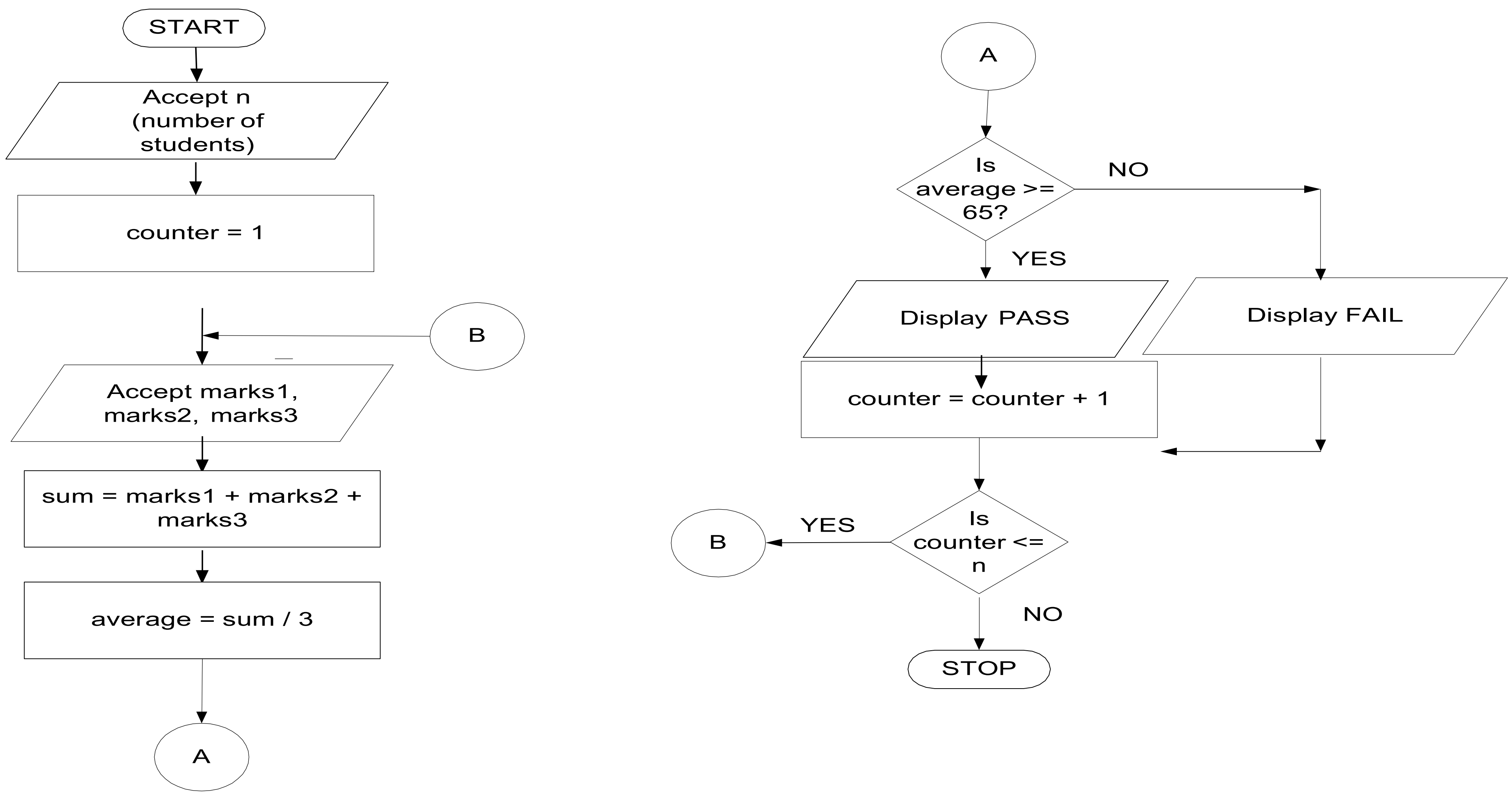
Flow Chart - Selectional



Example (Iterational)

- Do the following for N input values. Read N from user
  - Write a program to find the average of a student given the marks heobtained in three subjects.
  - Then test whether he passed or failed.
  - For a student to pass, average should not be less than 65.

Flow Chart – Example (Iterational)



Algorithm

Algorithm

- An *algorithm* is a sequence of unambiguous instructions for solvinga problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.
- Recipe, process, method, technique, procedure, routine,… withfollowing requirements:
- The properties of an algorithm are as follows:

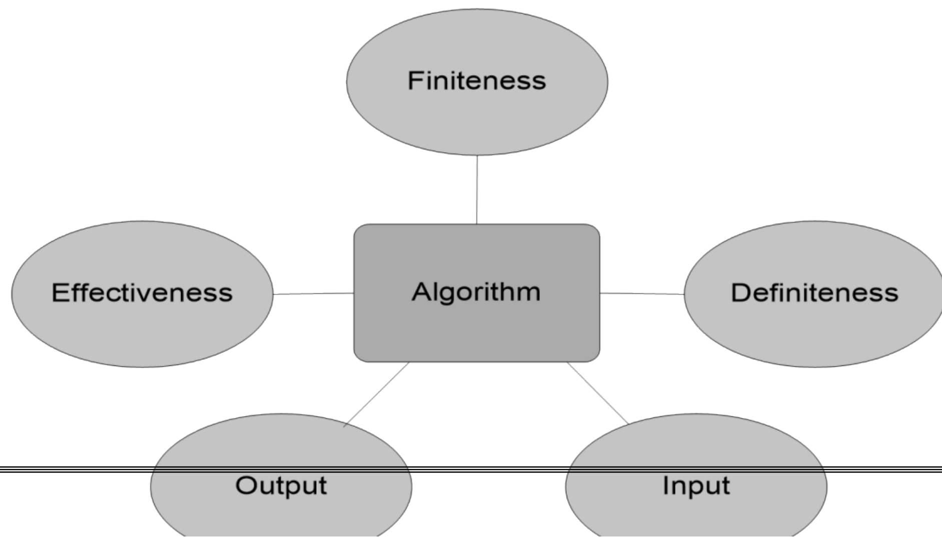
✓ Finiteness
  - terminates after a finite number of steps

✓ Definiteness
  - rigorously and unambiguously specified

✓ Input
  - valid inputs are clearly specified

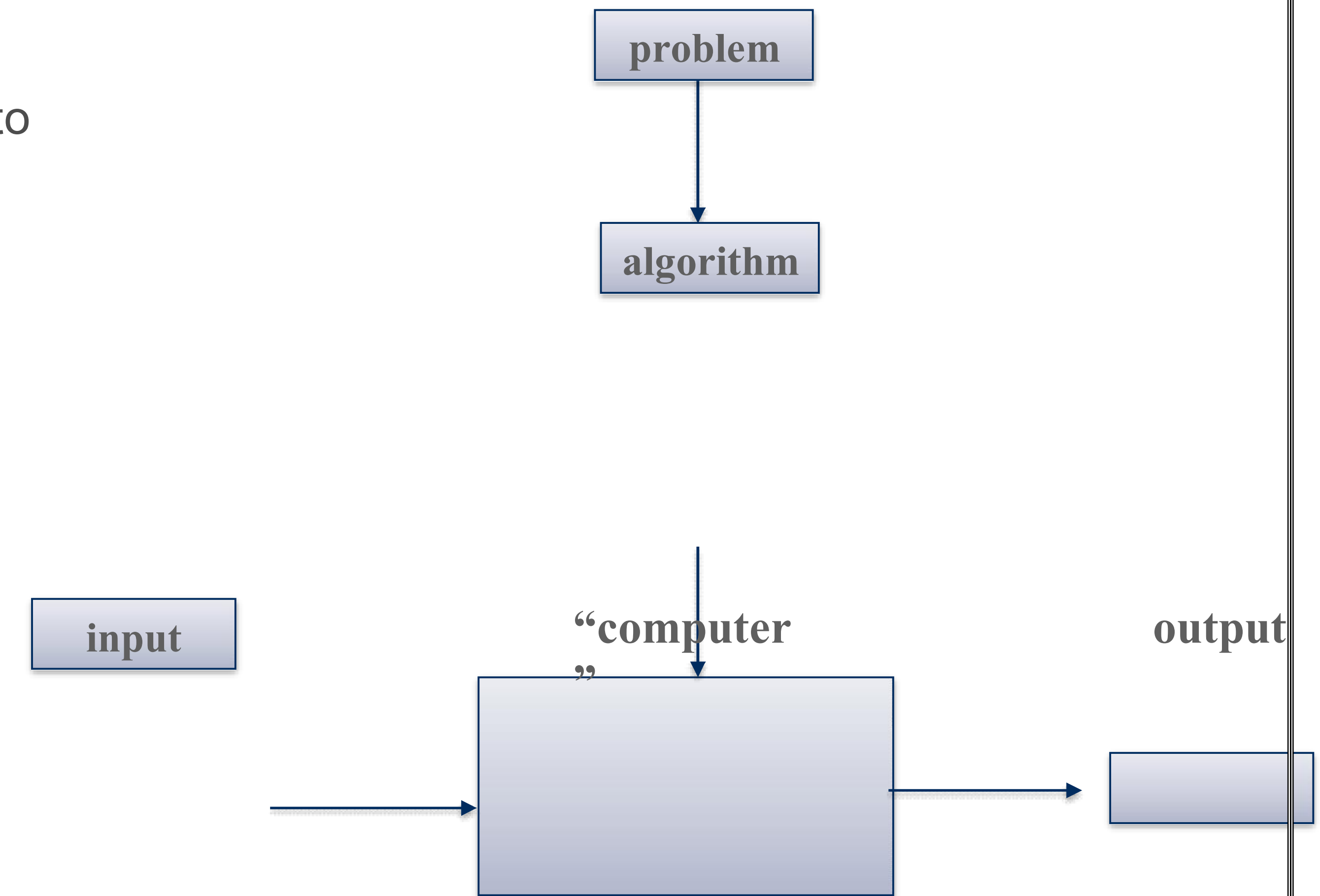
✓ Output
  - can be proved to produce the correct outputgiven a valid input

✓ Effectiveness
  - steps are sufficiently simple and basic



Steps to develop Algorithm

- Identify the Inputs and Outputs
- Identify any other data and constantsrequired to solve the problem
- Identify what needs to be computed
- Write an algorithm



Algorithm – Example (1 of 2)

Find the average marks scored by a student in 3 subjects:

BEGIN

Step 1 : Accept 3 marks say Marks1, Marks2, Marks3 scored by the student

Step 2 : Add Marks1, Marks2, Marks3 and store the result in Total

Step 3 : Divide Total by 3 and find the Average

Step 4 : Display Average

END

Algorithm-Example (2 of 2)

Find the average marks scored by a student in 3 subjects:

BEGIN

Step 1 : Read Marks1, Marks2, Marks3 Step 2 :

Sum = Marks1 + Marks2 + Marks3Step 3 :

Average = Sum / 3

Step 4 : Display Average

END

Different Patterns in Algorithms

- **Sequential**
  - Sequential constructs execute the program in the order in which they appear in theprogram
- **Selectional (Conditional)**
  - Selectional constructs control the flow of statement execution in order to achievethe required result
- **Iterational (Loops)**
  - Iterational constructs are used when a part of the program is to be executedseveral times

Example - Selectional

- Write an algorithm to find the average marks of a student. Also check whether thestudent has passed or failed.
- For a student to pass, average marks should not be less than 65.

BEGIN
Step 1 : Read Marks1, Marks2, Marks3 Step 2 :
Total = Marks1 + Marks2 + Marks3Step 3 :
Average = Total / 3
Step 4 : Set Output = “Student Passed”
Step 5 : if Average < 65 then Set Output = “Student Failed”Step 6 :
Display Output
END

Example – Iterational

Find the average marks scored by ‘N’ number of students

BEGIN

Step 1 : Read NumberOfStudents

Step 2 : Counter = 1

Step 3 : Read Marks1, Marks2, Marks3 Step 4

: Total = Marks1 + Marks2 + Marks3Step

5 : Average = Total / 3

Step 6 : Set Output = “Student Passed”

Step 7 : If (Average < 65) then Set Output = “Student Failed”Step 8 :

Display Output

Step 9 : Counter = Counter + 1

Step 10 : If (Counter <= NumberOfStudents) then goto step 3END

Pseudo Code

- An algorithm is independent of any language or machine whereas aprogram is dependent on a language and machine

- To fill the gap between these two, we need pseudo codes

- *Pseudo-code* is a way to represent the step by step methods in finding thesolution to the given problem

Pseudo code - Example

Here's pseudo-code to add the two numbers:

```
Begin
int a, b, c; input
a, b Let c= a +
b;output c; End
```

Pseudo codes (sequential)

```
BEGIN
Int a,b,c,avg;
Input a,b,c;
Let avg = (a+b+c)/3;
Output avg;
END
```

Pseudo codes (conditional)

Determine the largest number of A, B, C

```
Read A, B and C
If A is greater than B Then
    If A is greater than C Then
            Display A
   Else
            Display C
   End
IfElse
   If B is greater than C Then
            Display B
    Else                End IfEnd If
```

Pseudo codes (conditional)

Display C

Pseudo codes (iteration)

Pseudo code:

```
For emp # 1000 to 1500
Salary = salary + 10000 End
for;
```

Recap

- Skills of a software developer
- Problem classification
- Problem solving approaches
- Flow Chart
- Algorithm patterns
- Pseudo codes

Algorithm Design

Searching and Sorting

- Searching refers to finding whether a data item is present in the set of items or not
- Sorting refers to the arrangement of data in a particular order. That is, arranging items ina particular way
- Sorting and searching have many applications in the area of computers

Searching Algorithms

- The time required to search depends on the following factors:
  - Whether the data is arranged in a particular order or not
  - The location of the data to be searched
  - The total number of searches to be done
- When the data is arranged in a particular order then, the time taken to search for the itemis less.
- Searching algorithms
  - Linear Search
  - Binary Search

Searching Algorithm

Linear Search: A Simple Search

- A search traverses the collection until
  - The desired element is found
  - Or the collection is exhausted
- If the collection is ordered, I might not have tolook at all elements
  - I can stop looking when I know the elementcannot be in the collection.

The Scenario

- We have a sorted array
- We want to determine if a particular element is in thearray
  - Once found, print or return (index, boolean, etc.)
  - If not found, indicate the element is not in thecollection

7	12	42	59	71	86	104	212

A Better Search Algorithm

Of course we could use our simpler search andtraverse the array.

But we can use the fact that the array is sorted to ouradvantage.

This will allow us to reduce the number ofcomparisons.

Binary Search

- Requires a sorted array or a *binary search tree*.
- Cuts the “search space” in half each time.
- Keeps cutting the search space in half until the target is found or has exhausted the all possiblelocations.

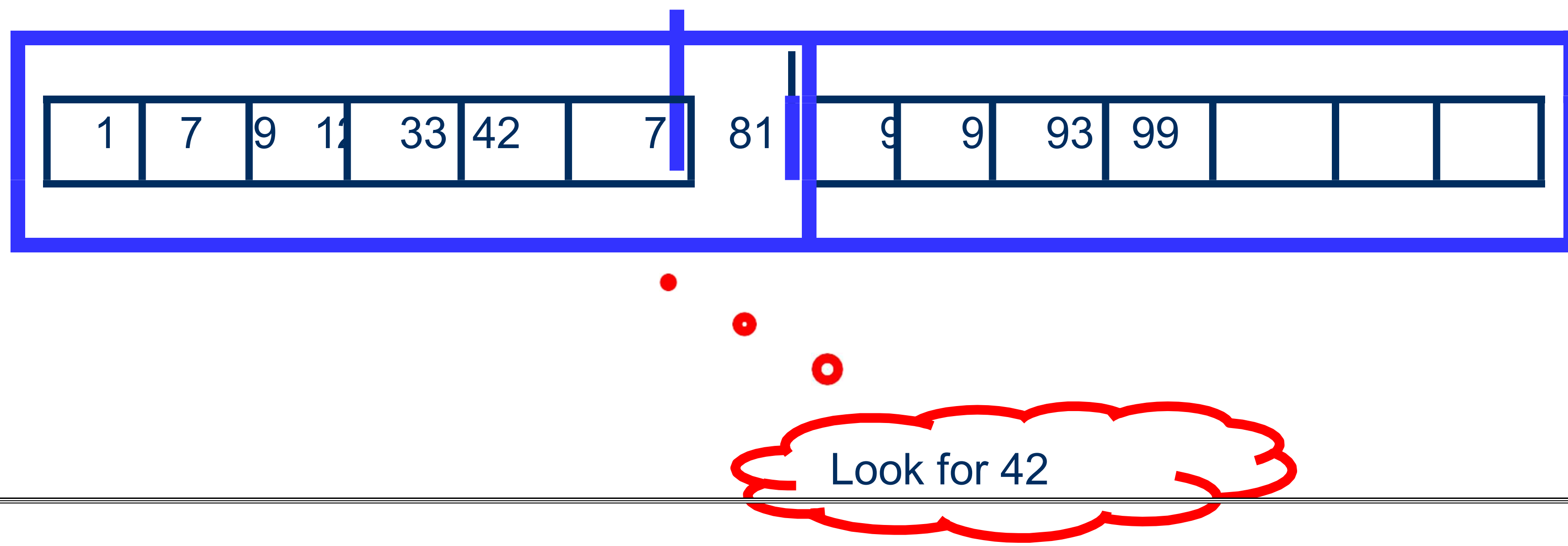
Binary Search Algorithm

look at "middle" element

if no match then

look *left* (if need smaller) or

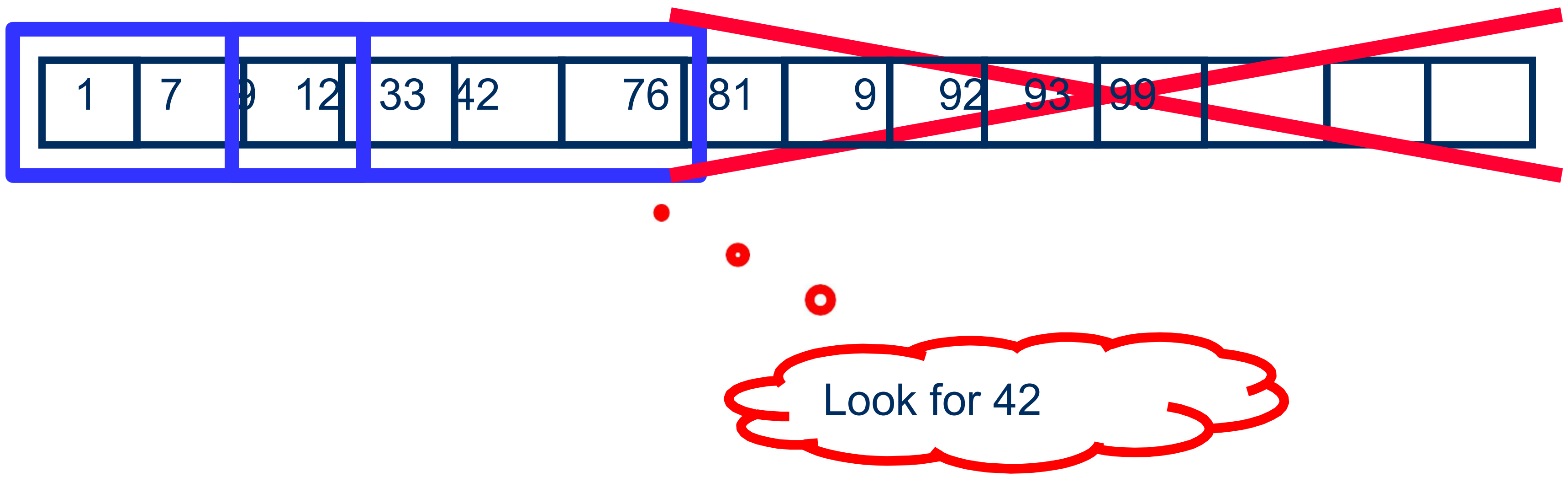
***right* (if need larger)**



Binary Search Algorithm

The Algorithm

look at "middle" elementif
no match then
look left or right



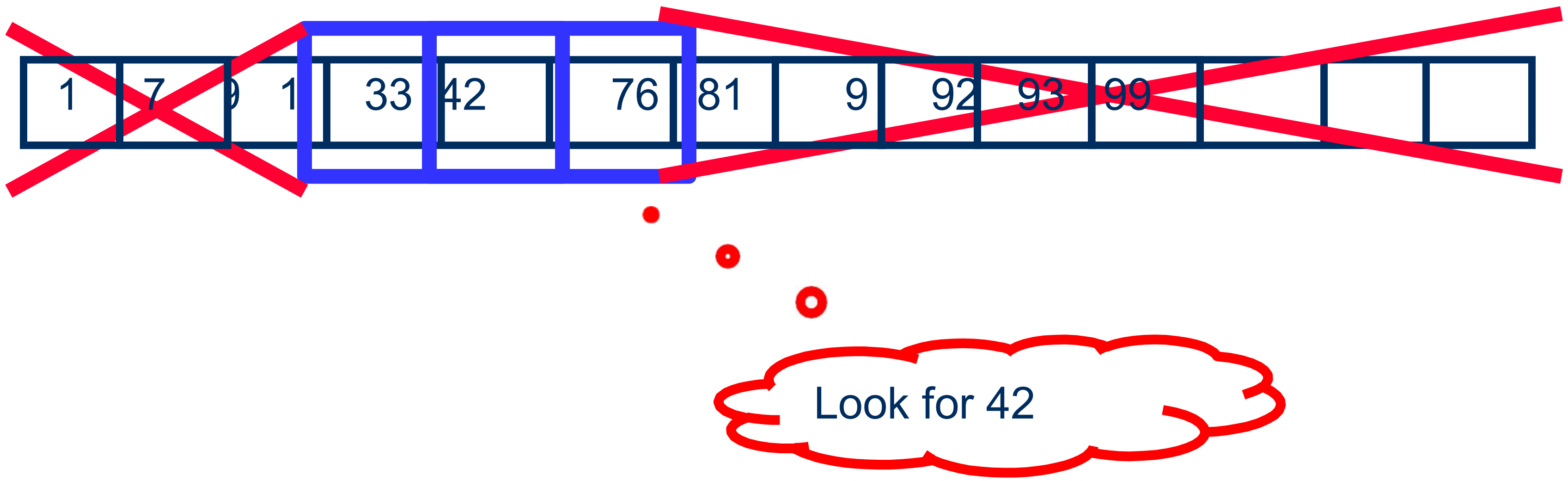
The Algorithm

The Algorithm

look at "middle" elementif

no match then

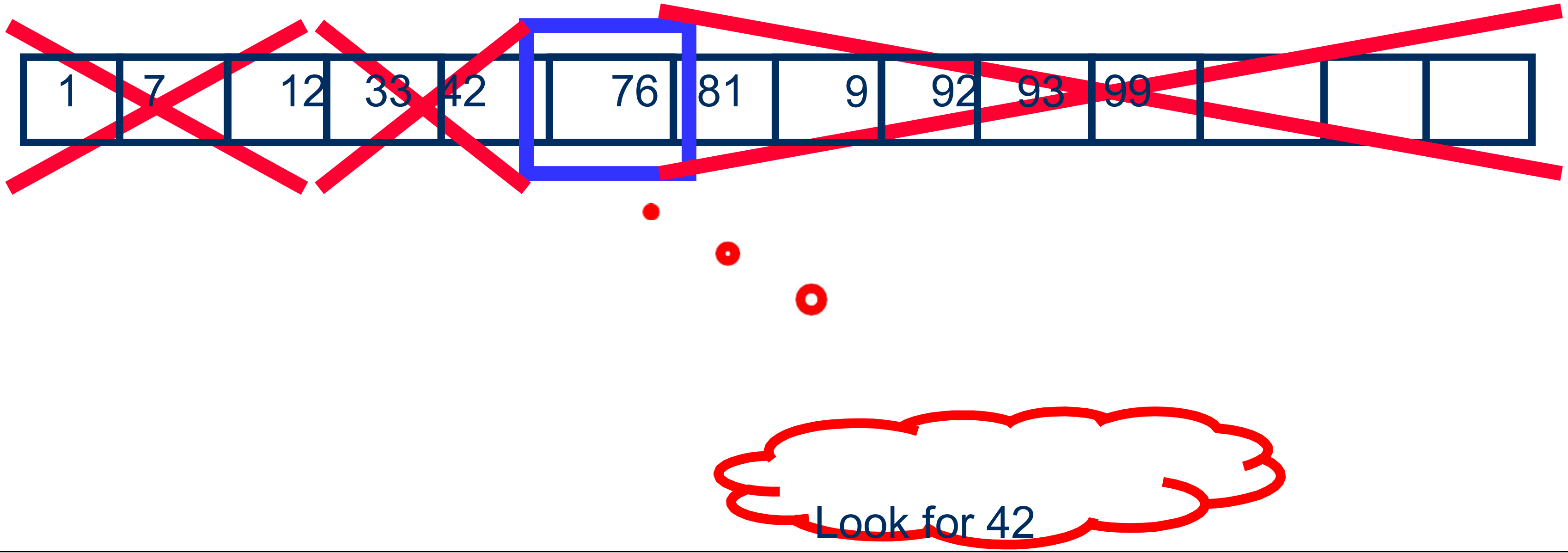
look left or right



The Algorithm

The Algorithm

look at “middle” element if no match then
look left or right



The Binary Search Algorithm

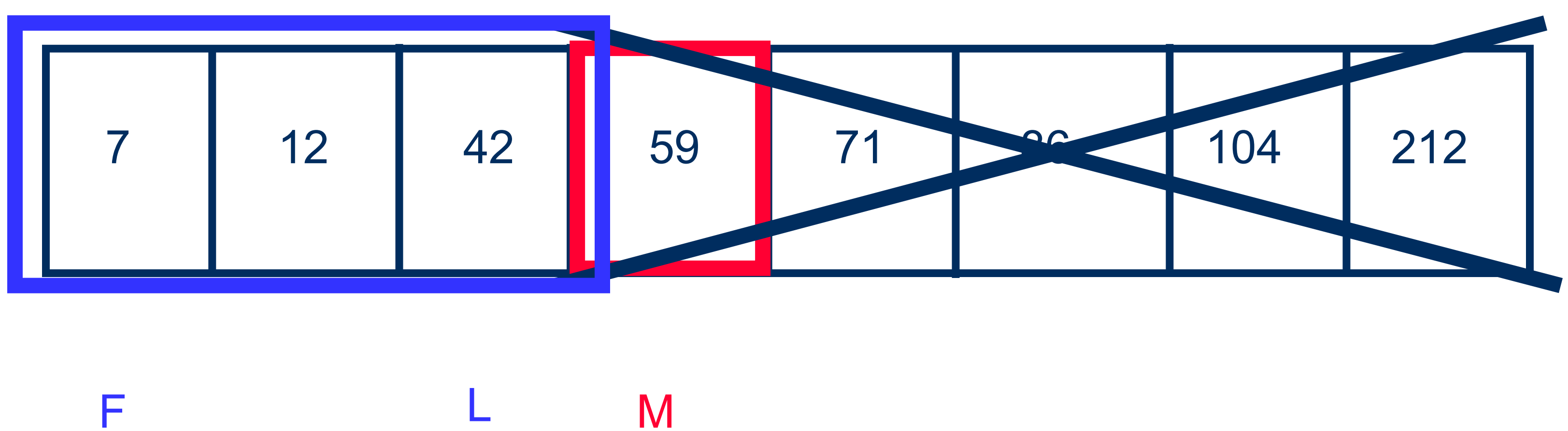
- Return found or not found (true or false), soit should be a function.
- When move *left* or *right*, change the arrayboundaries
  - We need a first and last index value.

The Binary Search Algorithm

```
calculate middle position

if (first and last have “crossed”) then
   “Item not found”

elseif (element at middle = to_find) then
   “Item Found”



 elseif to_find < element   at middle  then
    Look to the left


else
   Look to the right
```

Looking Left

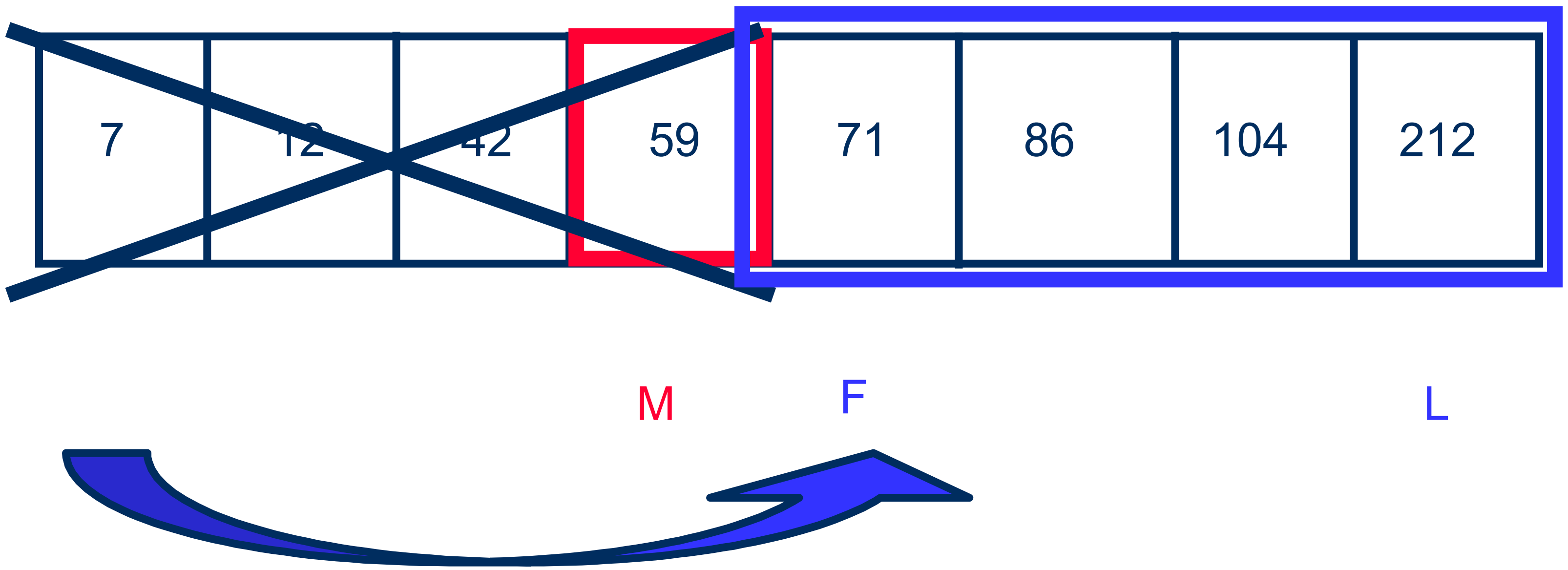
- Use indices “**first**” and “**last**” to keep track of where we are looking
- Move **left** by setting **last = middle – 1**



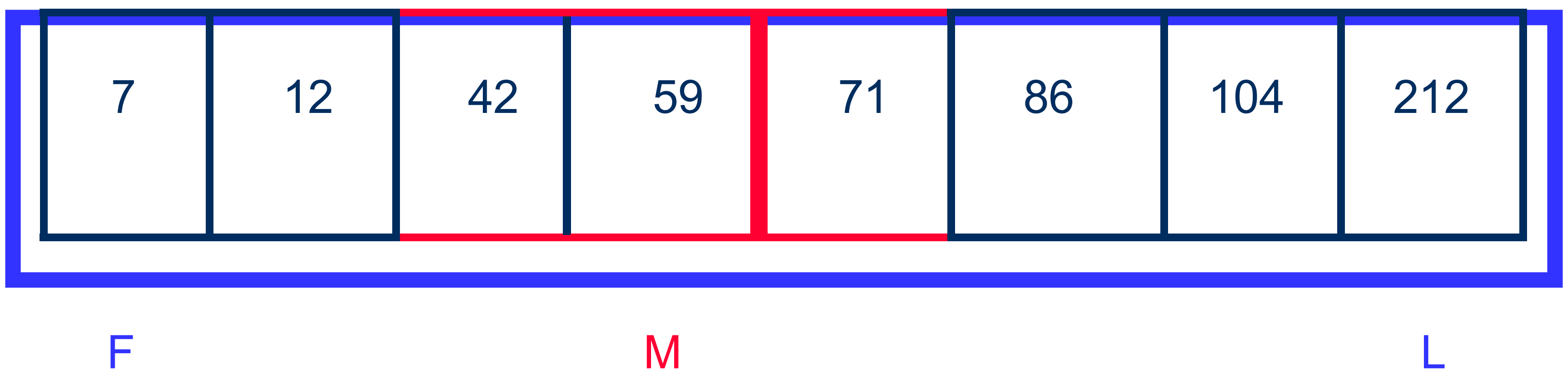
Looking Left

Looking Right

- Use indices "**first**" and "**last**" to keep track of where we are looking
- Move **right** by setting **first = middle + 1**

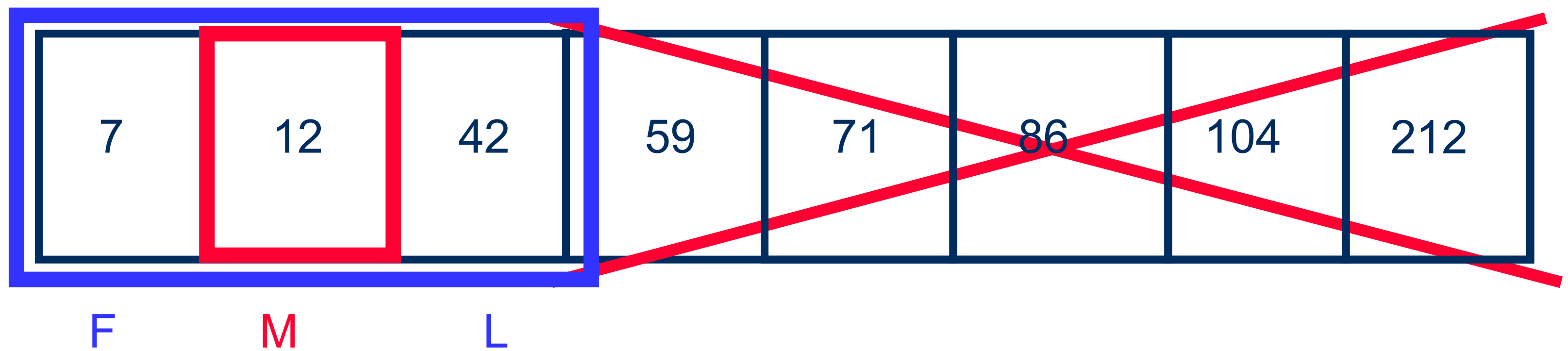


Binary Search Example – Found



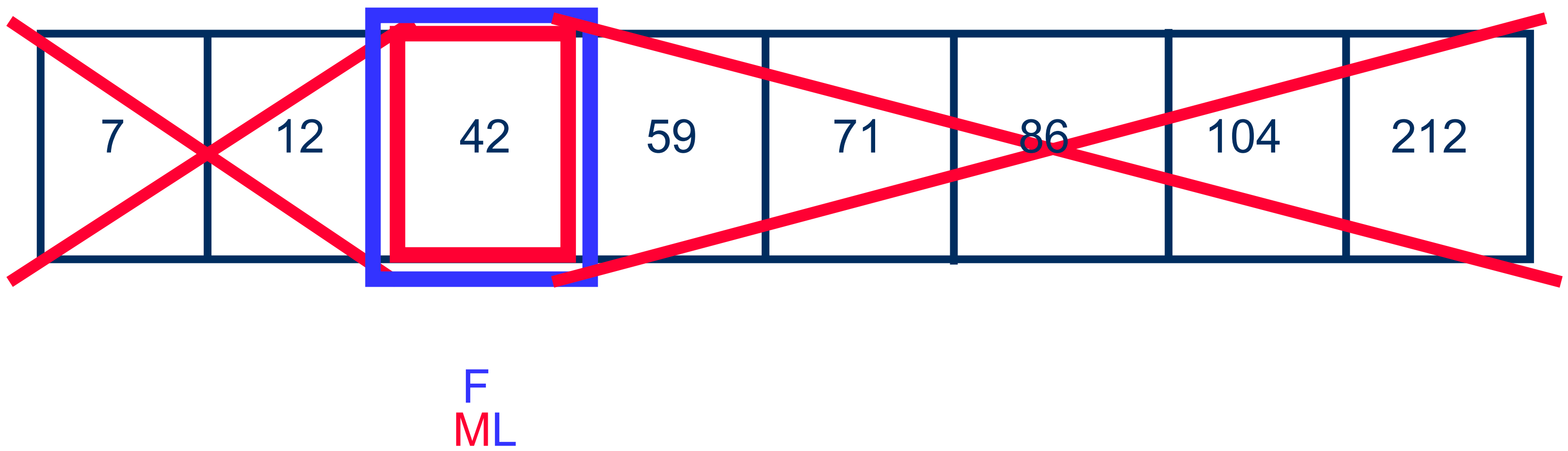
Looking for 42

Binary Search Example – Found



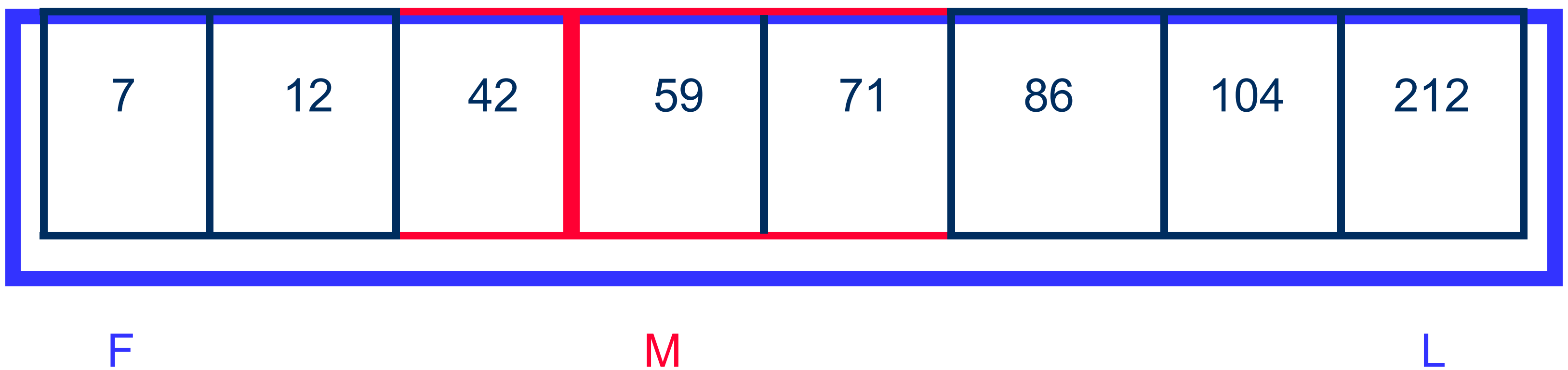
Looking for 42

Binary Search Example – Found



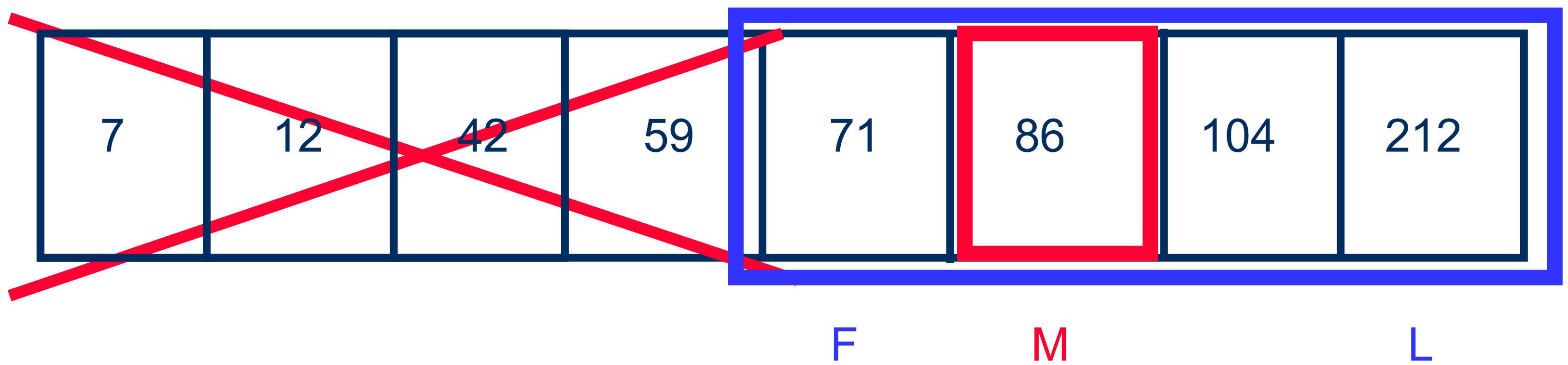
42 found – in 3 comparisons

Binary Search Example – Not Found



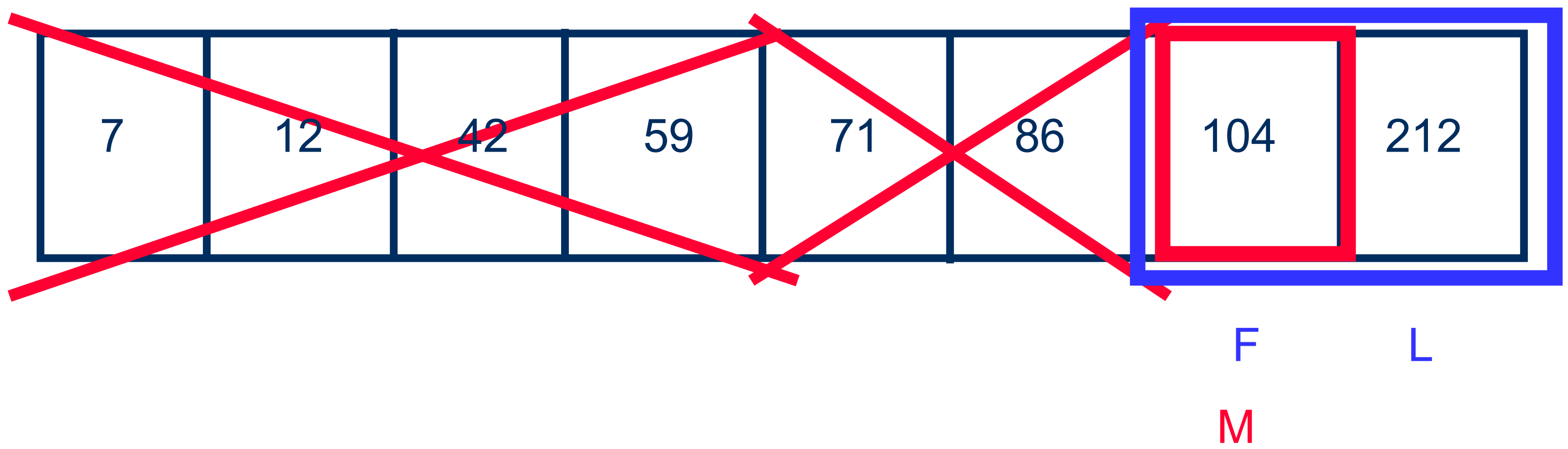
Looking for 89

Binary Search Example – Not Found



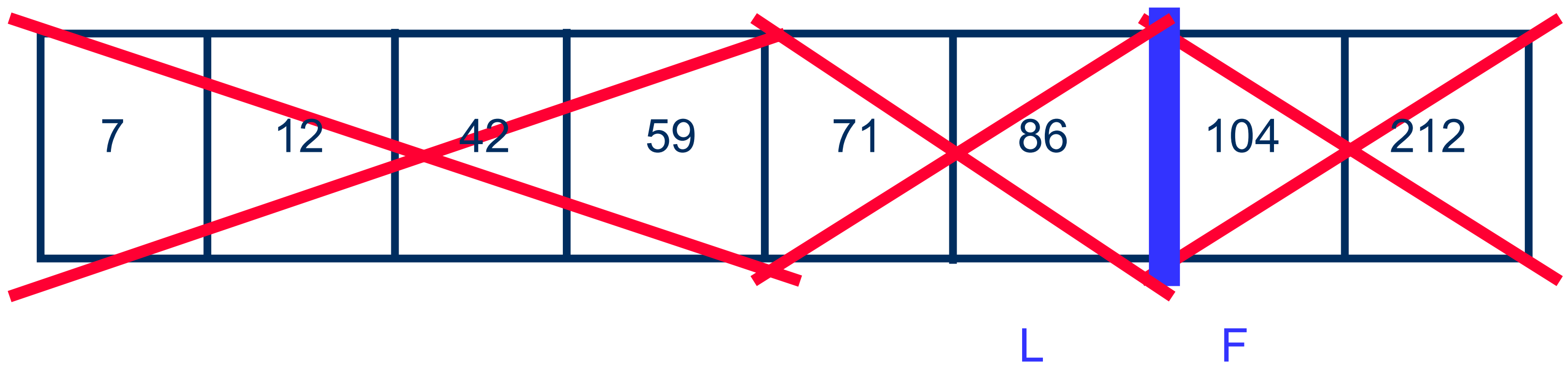
Looking for 89

Binary Search Example – Not Found



Looking for 89

Binary Search Example – Not Found



89 not found – 3 comparisons

Sorting Techniques

Sorting

Sorting is any process of arranging items systematically, and has two common, yet distinct meanings: ordering: arranging itemsin a sequence ordered by some criterion; categorizing: grouping items with similar properties.

Sorting

- Sorting takes an unordered collection andmakes it an ordered one.

1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
5	12	35	42	77	101

Complexity of sorting Algorithm

The complexity of sorting algorithm calculates the running time of a function inwhich 'n' number of items are to be sorted. The choice for which sorting method is suitable for a problem depends on several dependency configurations for different problems. The most noteworthy of these considerations are:The length of time spent by the programmer in programming a specific sorting program

- Amount of machine time necessary for running the program
- The amount of memory necessary for running the program

Types of Sorting Techniques

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

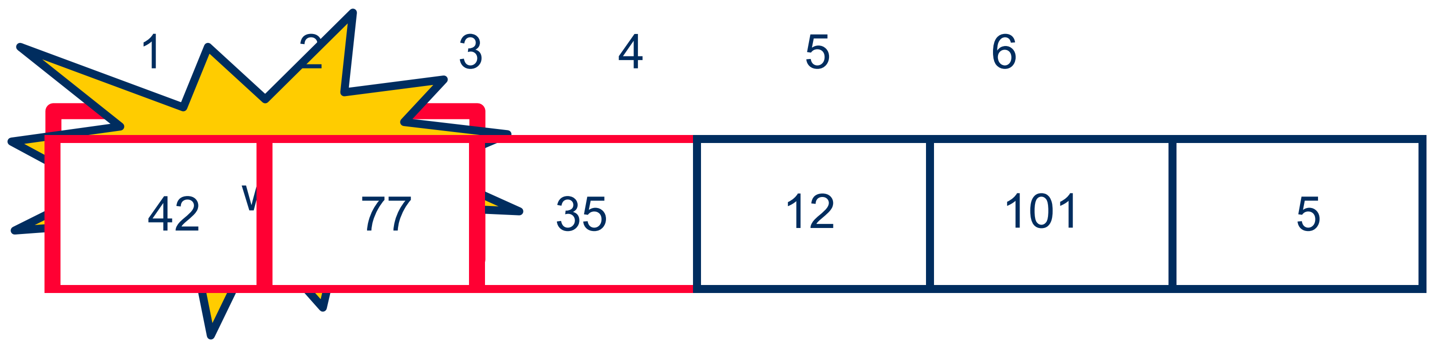
Bubble sort - "Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons andswapping

1	2	3	4	5	6
77	42	35	12	101	5

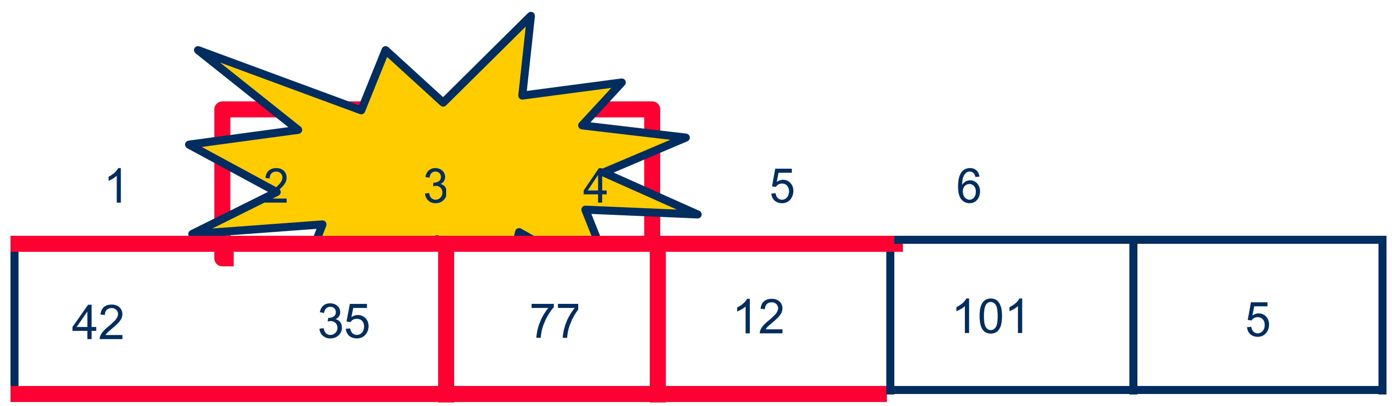
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons andswapping



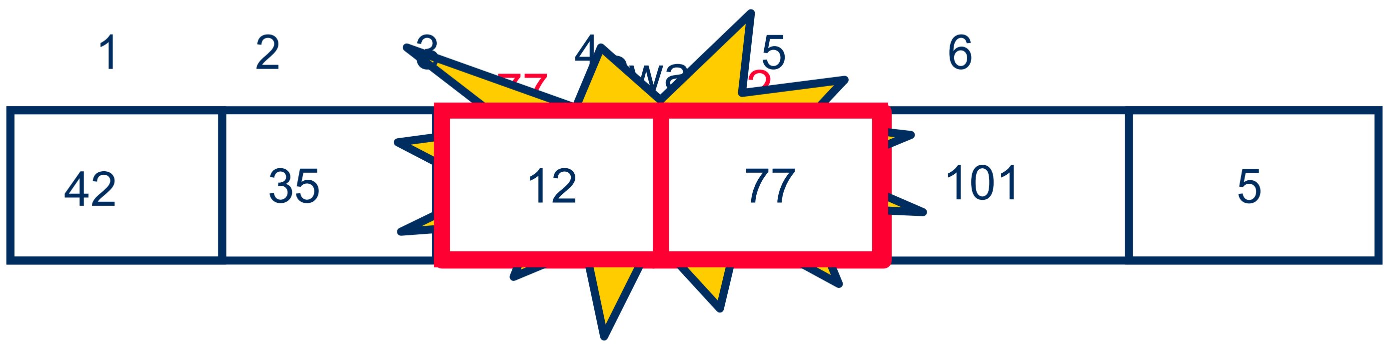
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons andswapping



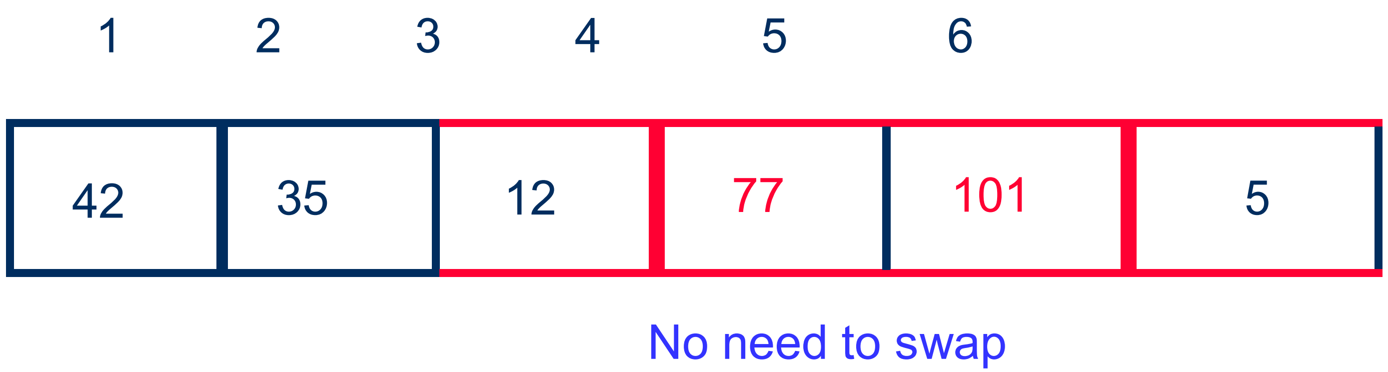
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons andswapping



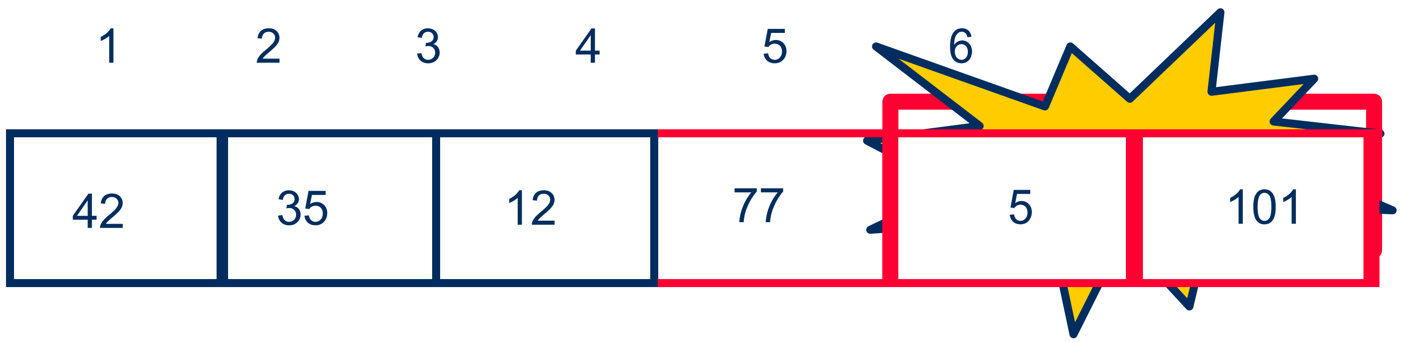
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons andswapping



"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons andswapping



"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons andswapping

1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

Items of Interest

- **Notice that only the largest value iscorrectly placed**
- **All other values are still out of order**
- **So we need to repeat this process**

1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

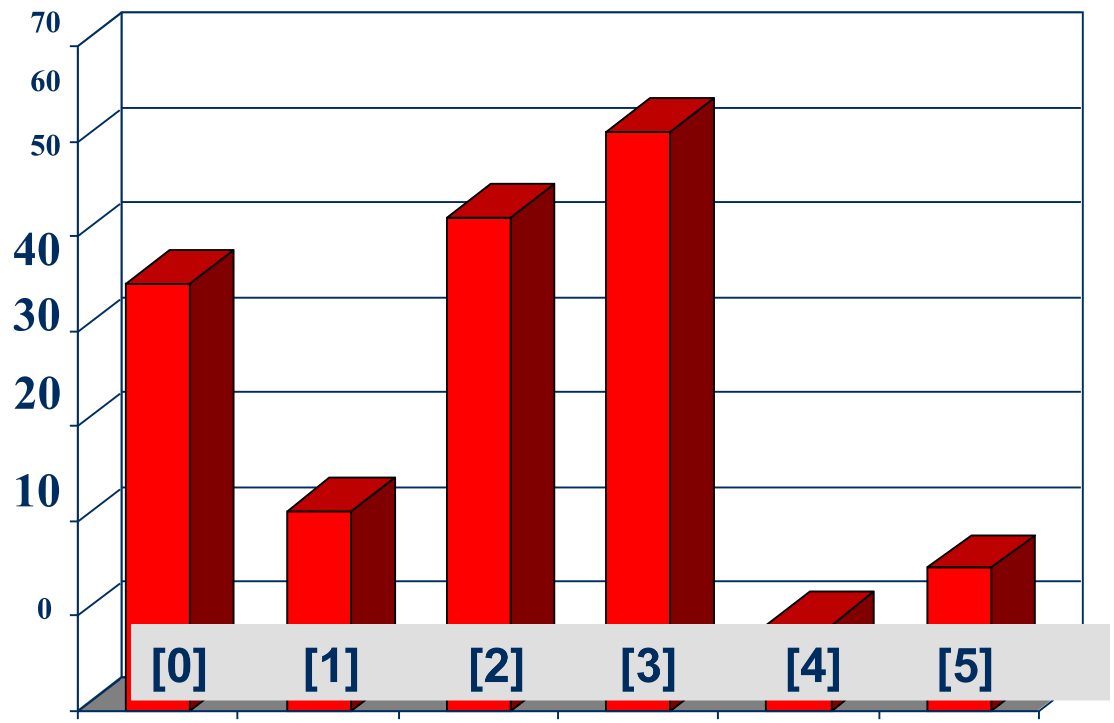
Selection sort

Selection Sort

- Idea:
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with theelement in the second position
  - Continue until the array is sorted
- Disadvantage:
  - Running time depends only slightly on the amount of orderin the file

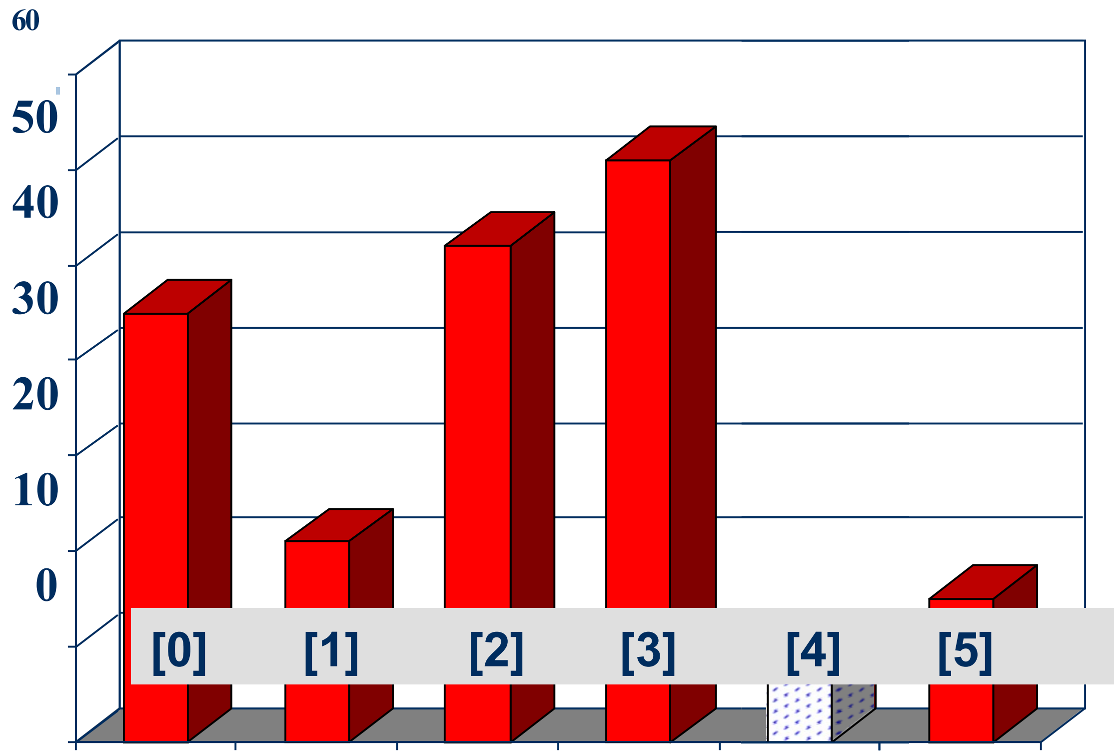
Selection Sort

- Example: we are given an array of six integers that we want to sort from smallest to largest



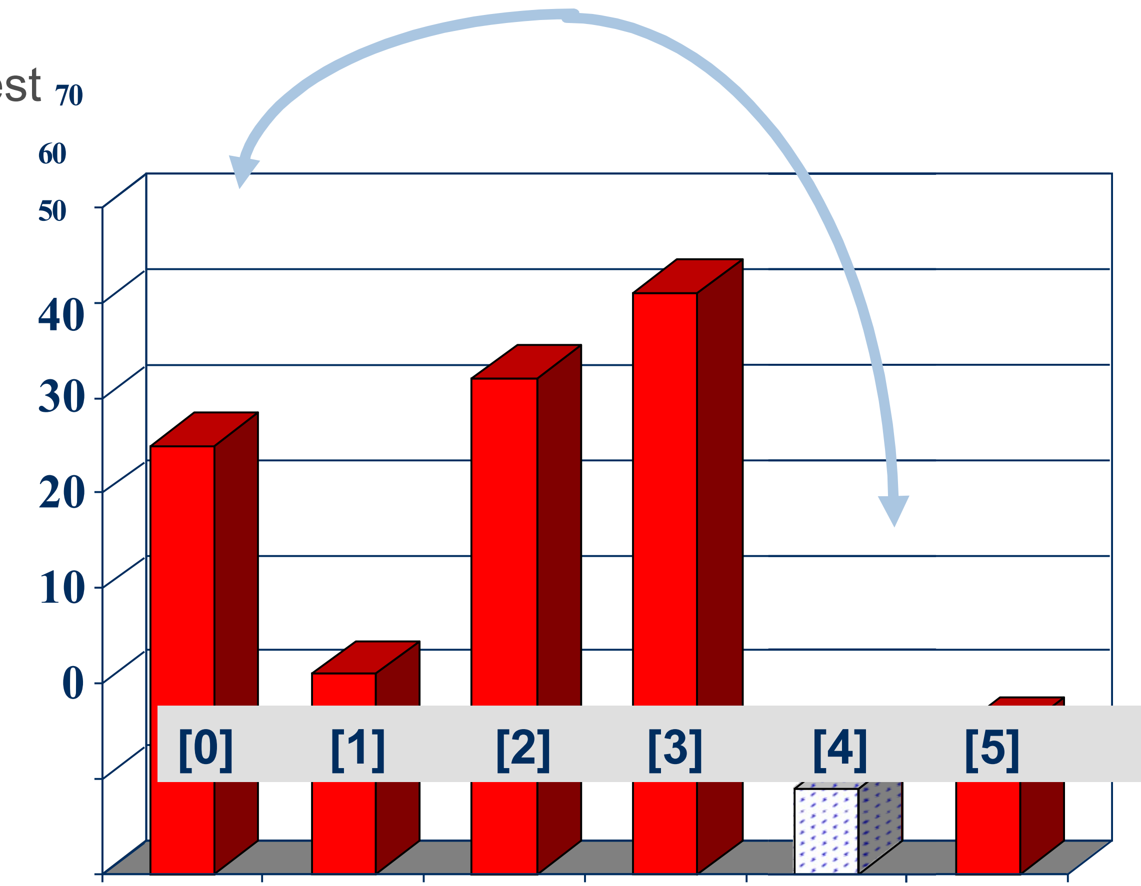
Selection Sort

- Start by finding the smallest entry.



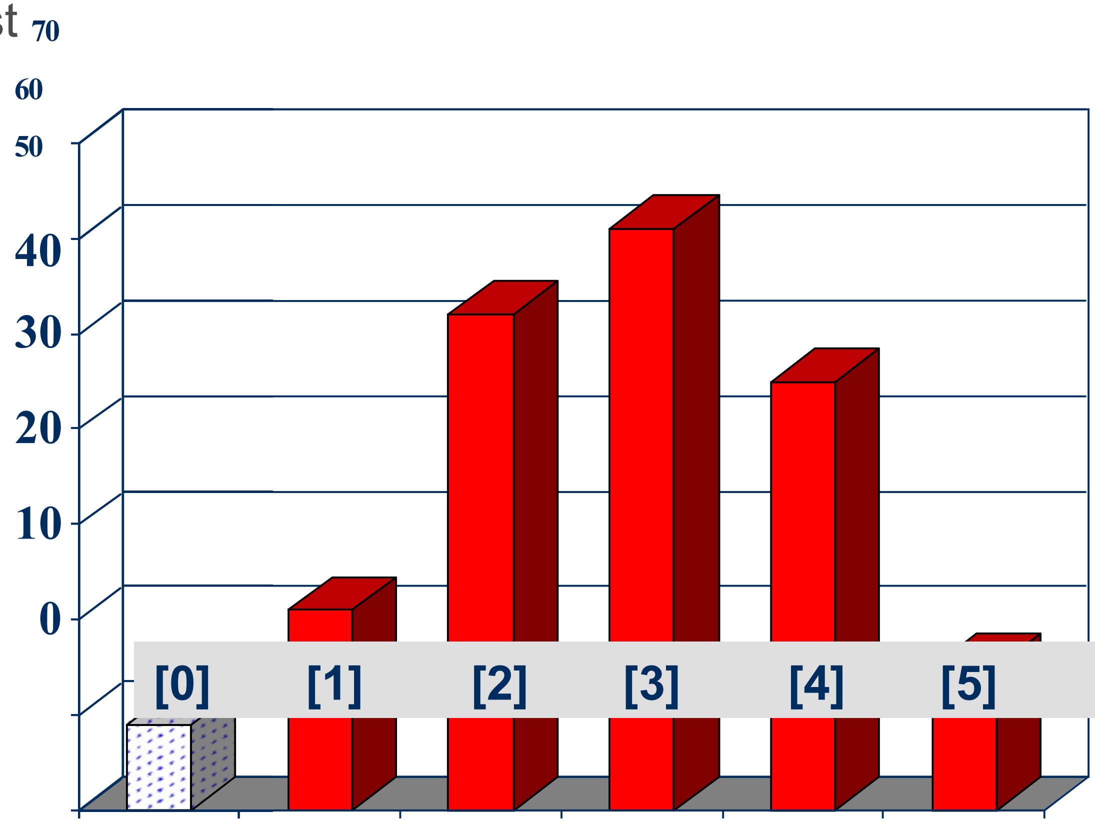
Selection Sort

- Swap the smallest entry with the [first entry]().



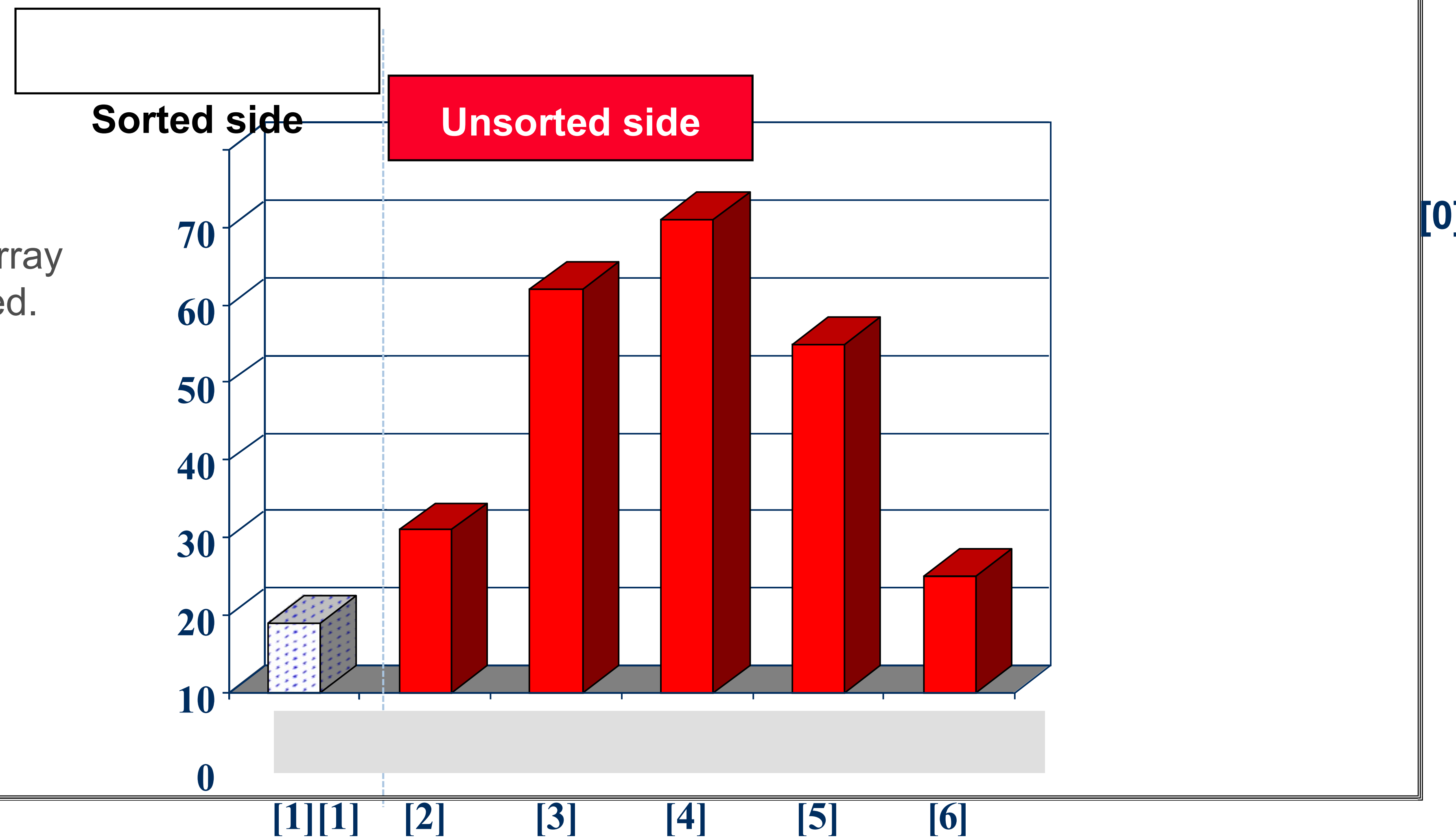
Selection Sort

- Swap the smallest entry with the first entry.



Selection Sort

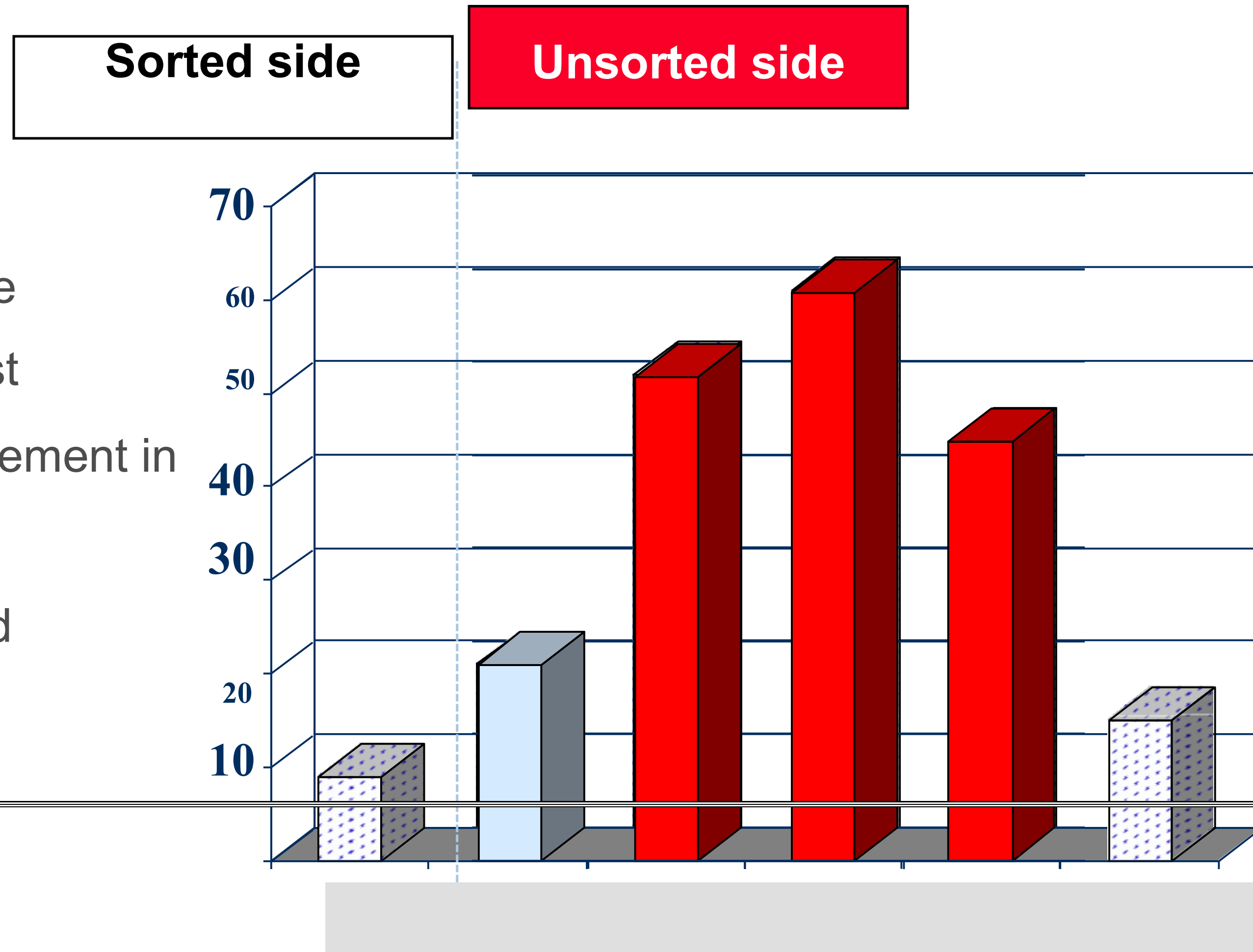
- Part of thearray is now sorted.



Selection Sort

[1] [2] [3] [4] [5]

Selection Sort



- Find the smallest element in the unsorted side.

Selection Sort

0

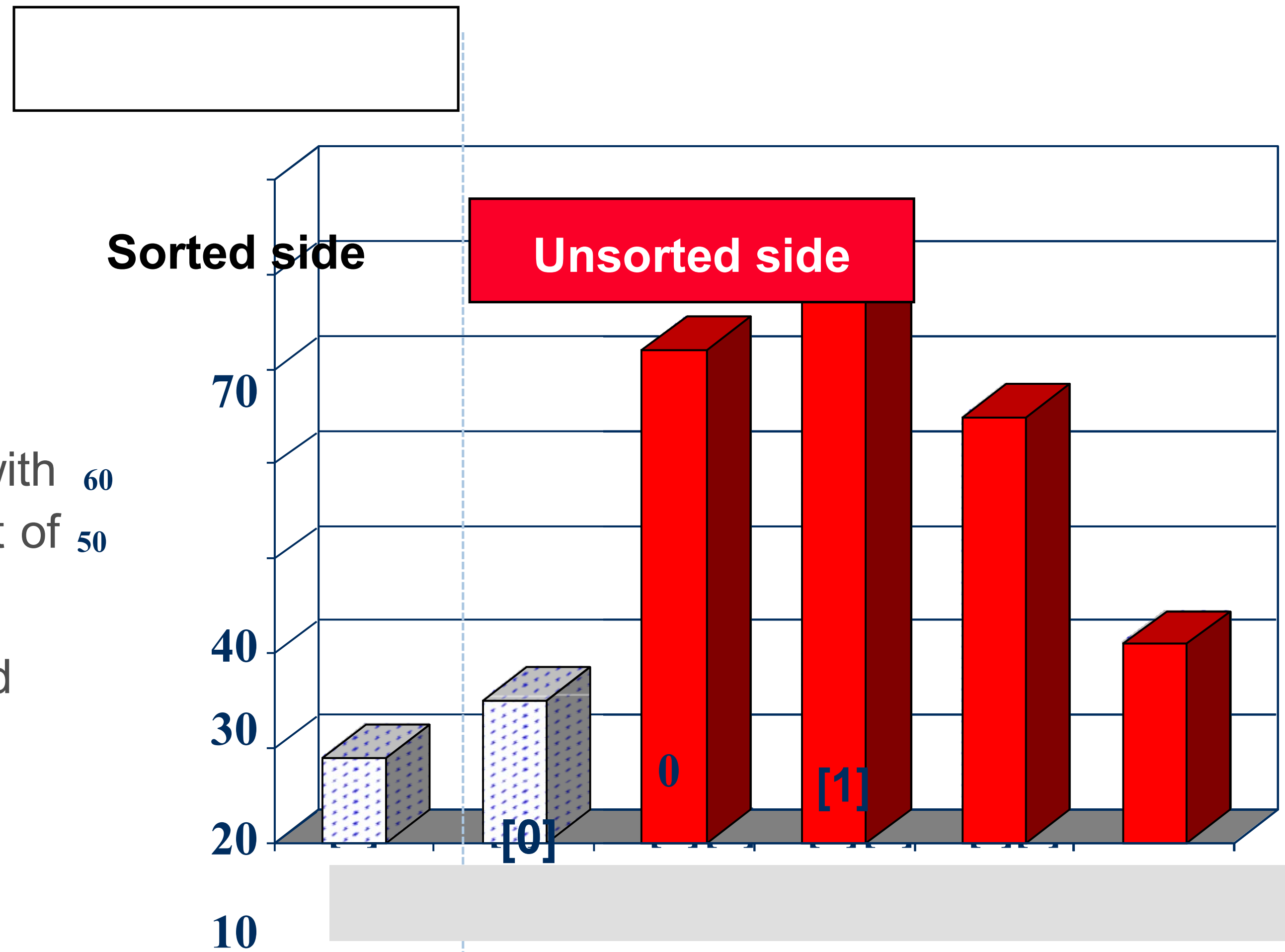
[1] [2] [3] [4] [5] [6]

[0] [1] [2] [3] [4] [5]

[2] [3] [4] [5]

Selection Sort

- Swap with the front of the unsorted side.



Selection Sort

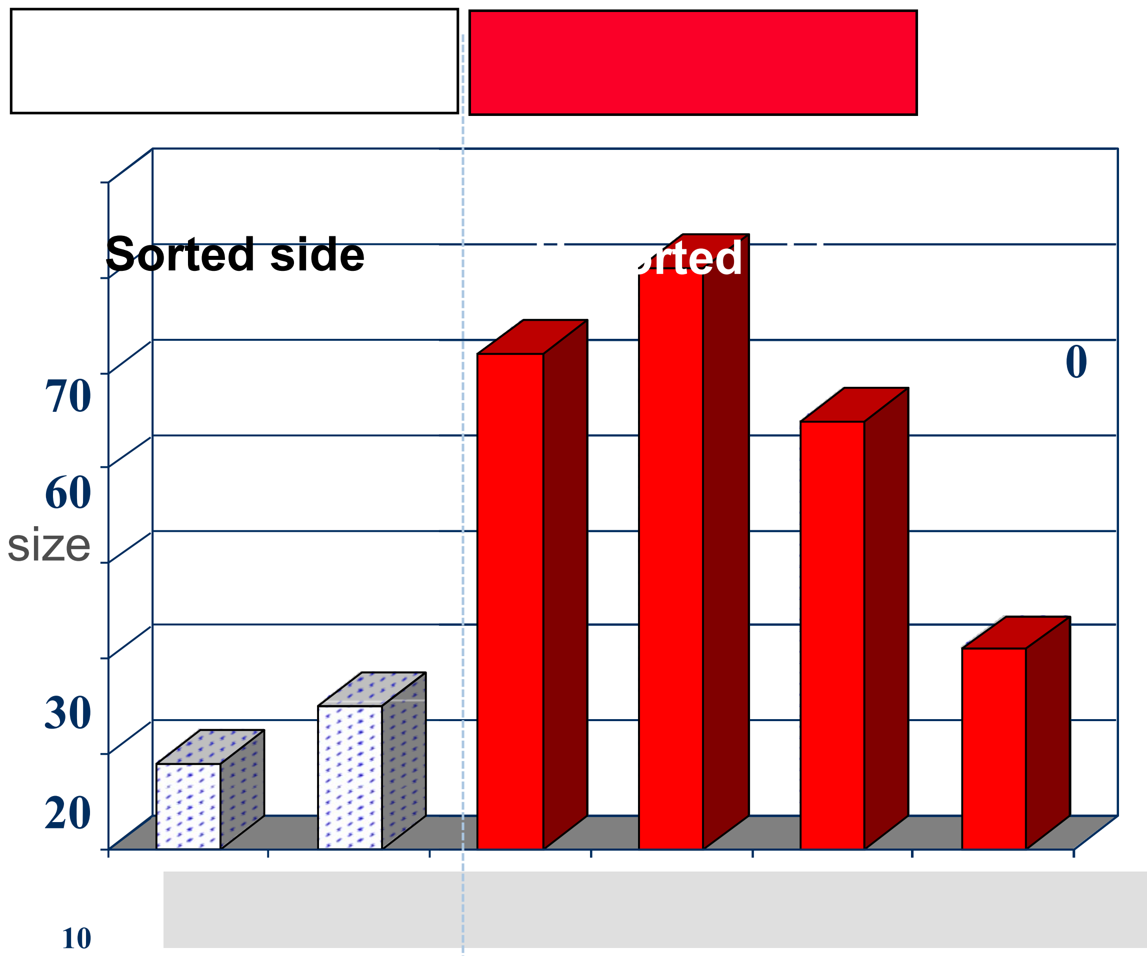
[2]

[3]

[4
]

[5]

Selection Sort



- We have increased the size of the sorted side by one element.

Selection Sort

[0] [1]

[1] [2] [3] [3] [4] [4] [5] [5]

[2]

[3

]

[4

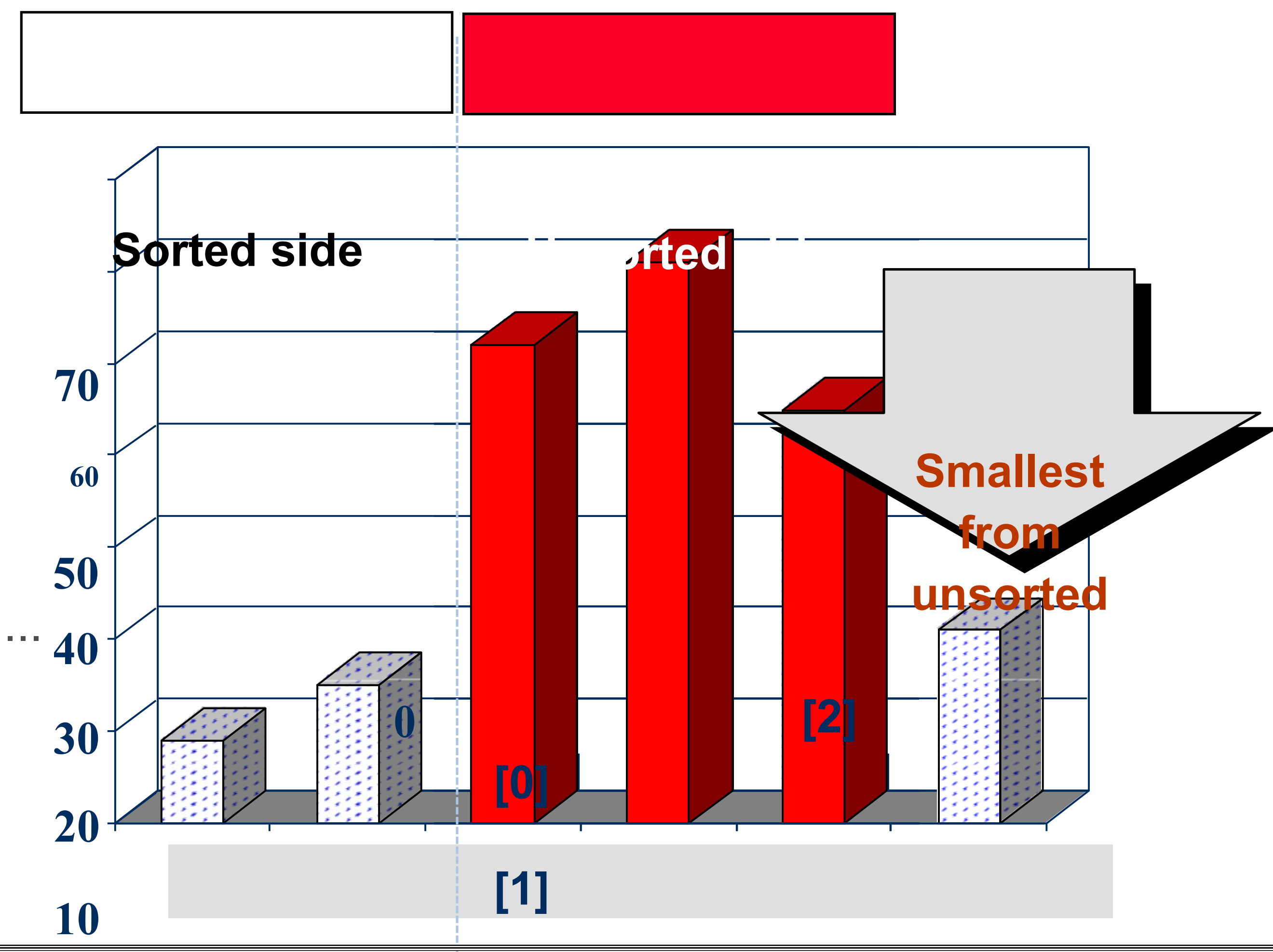
]

[5

]

Selection Sort

- The process continues ....

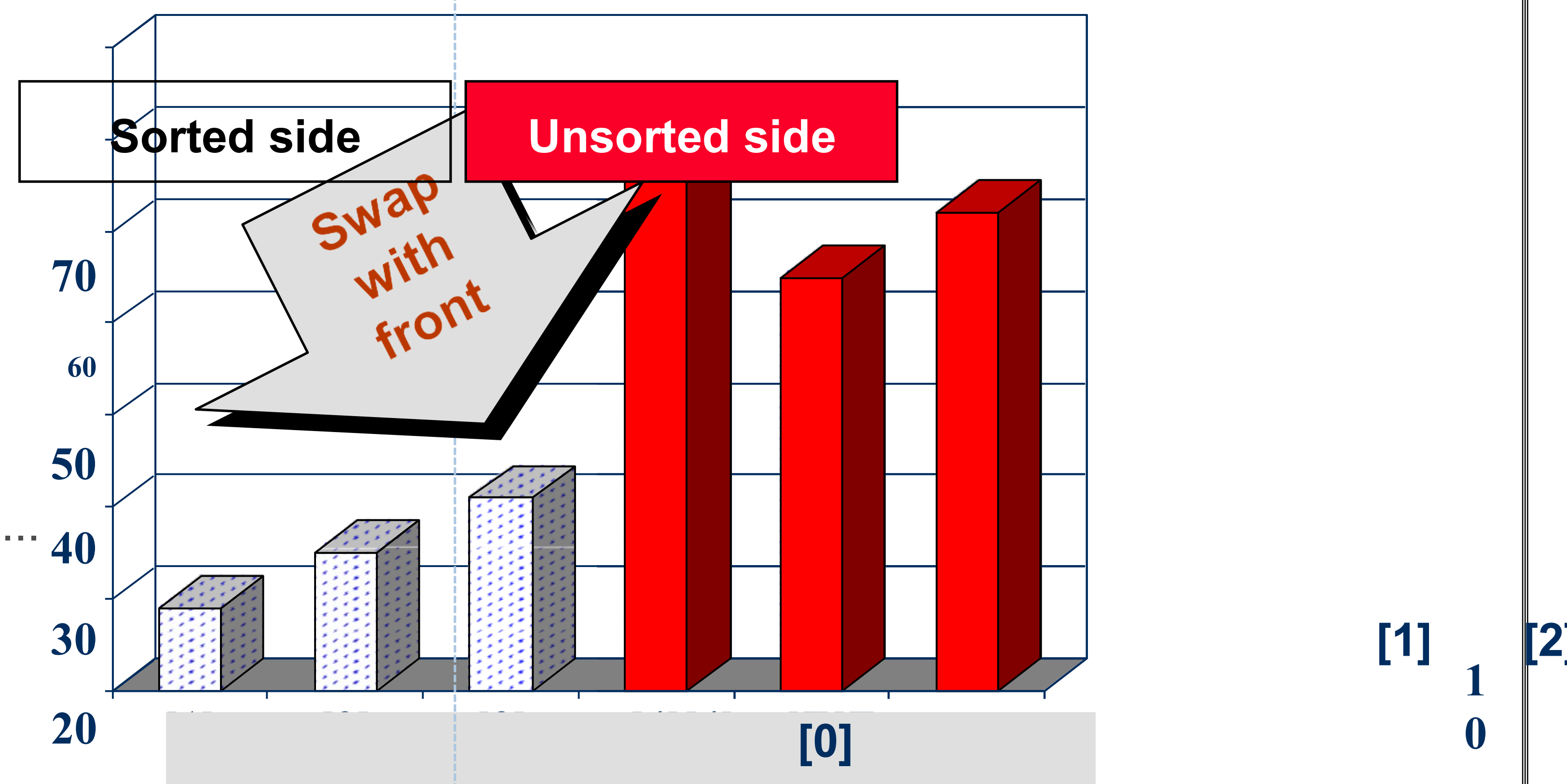


Selection Sort

[3] [4] [5]

Selection Sort

- The process continues ....



Selection Sort

[3]

[4
]

[5]

Selection Sort

Sorted side is bigger

Sorted side

70

60

50

40

30

20

10

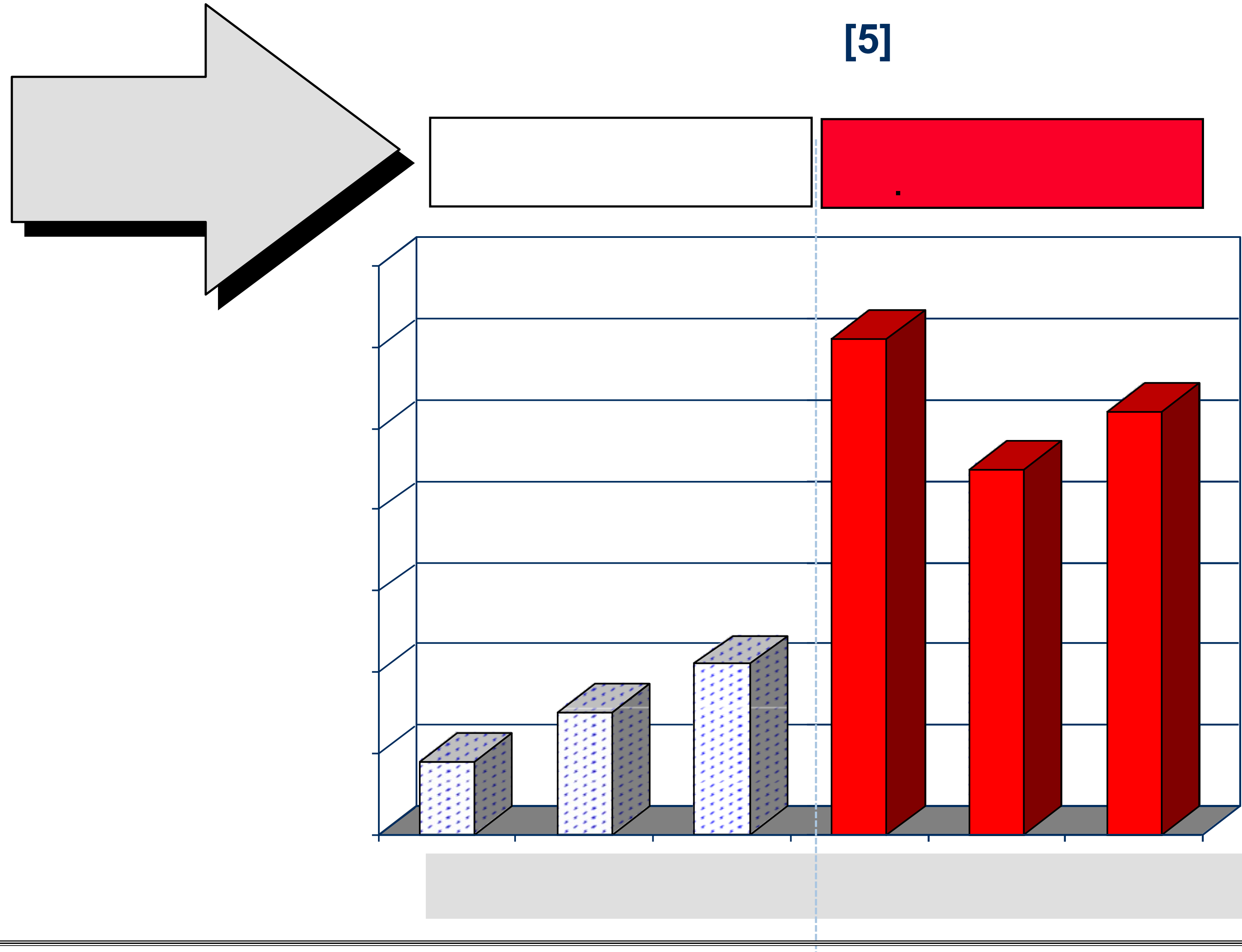
0

[0] [1] [2]

[1] [2] [3] [4] [4] [3] [5] [5] [4]

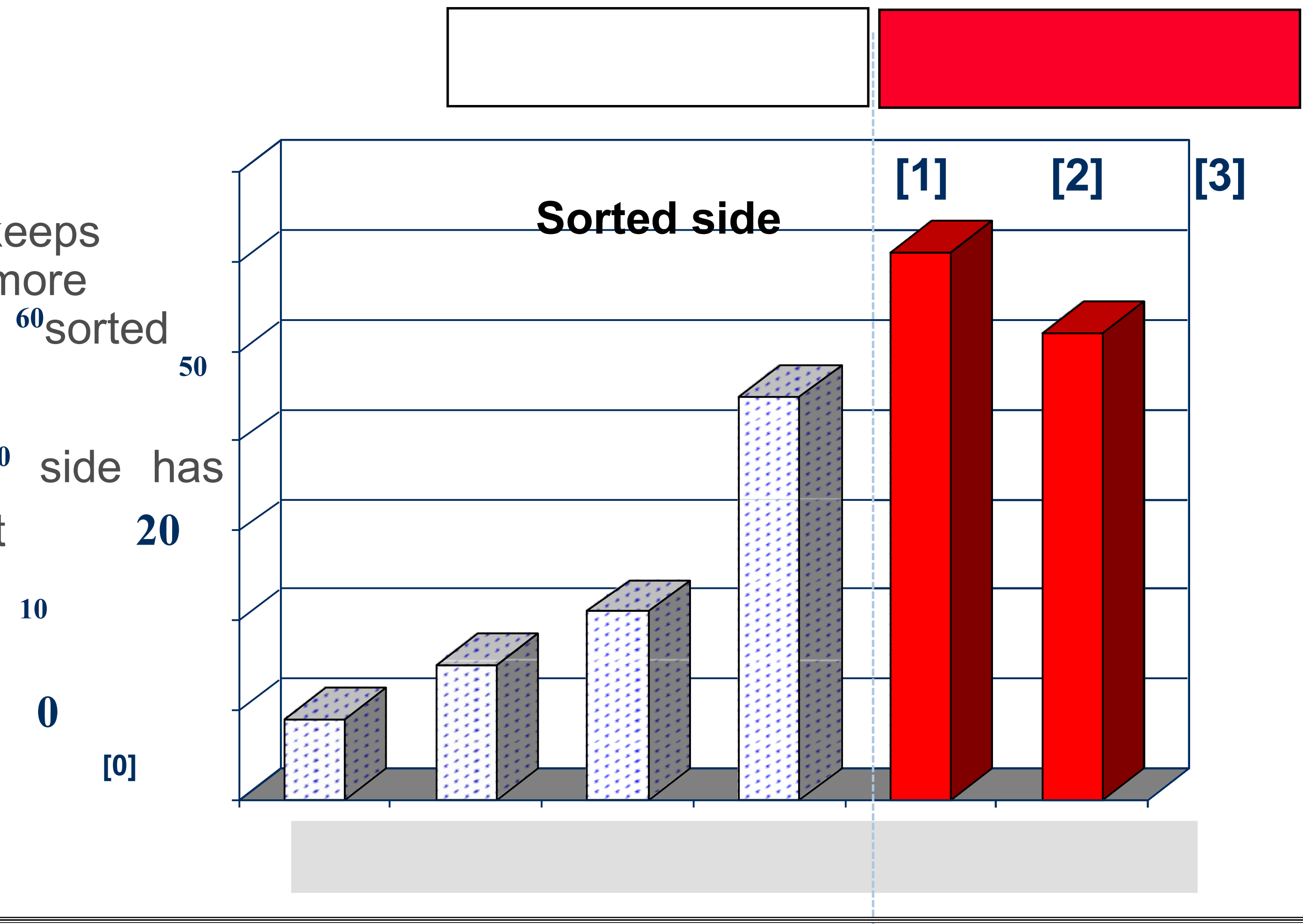
- The process continues...

Selection Sort



Selection Sort

- The process keeps adding one more number to the sorted side.
- The sorted side has the smallest numbers, arranged from small to large.

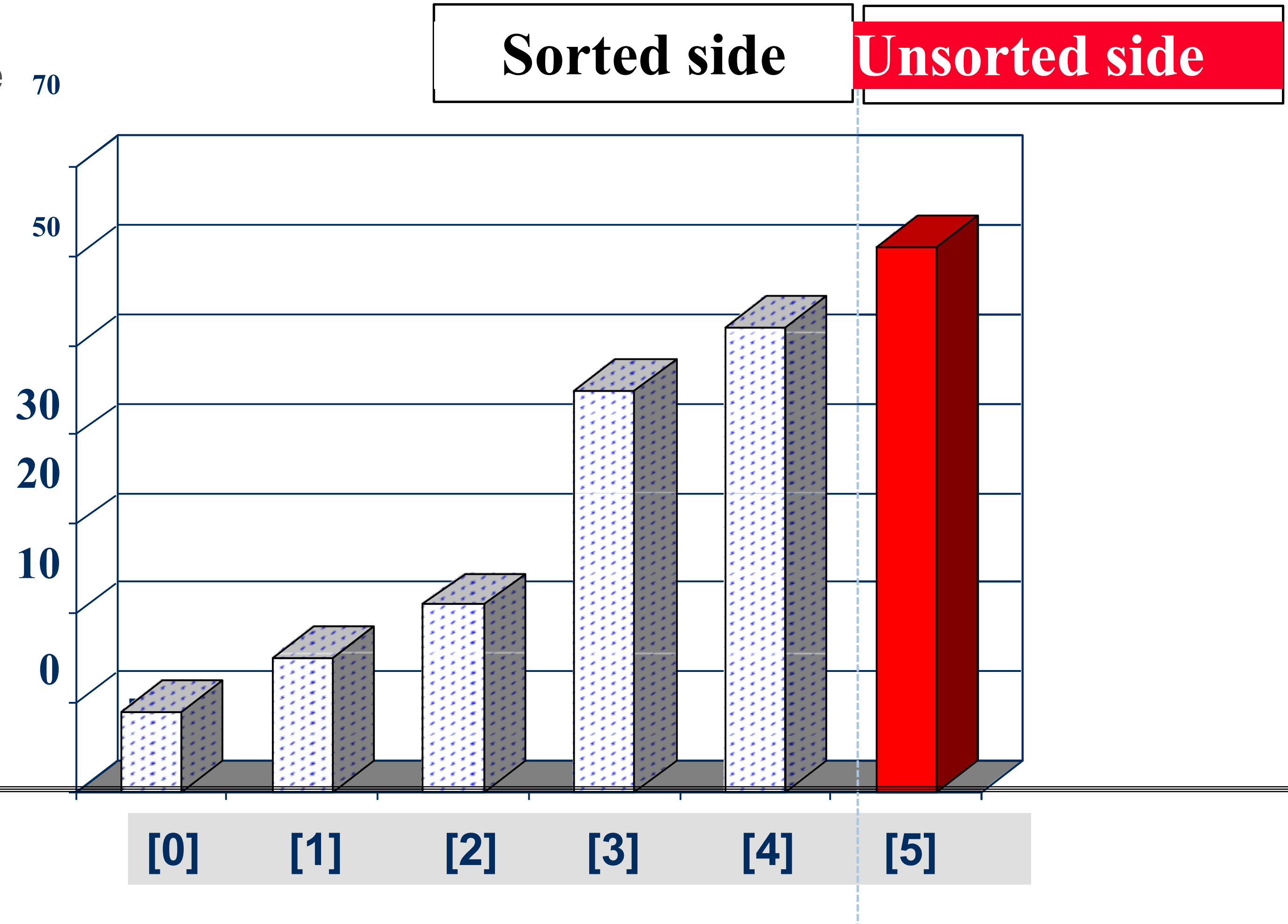


Selection Sort

[4] [5]

Selection Sort

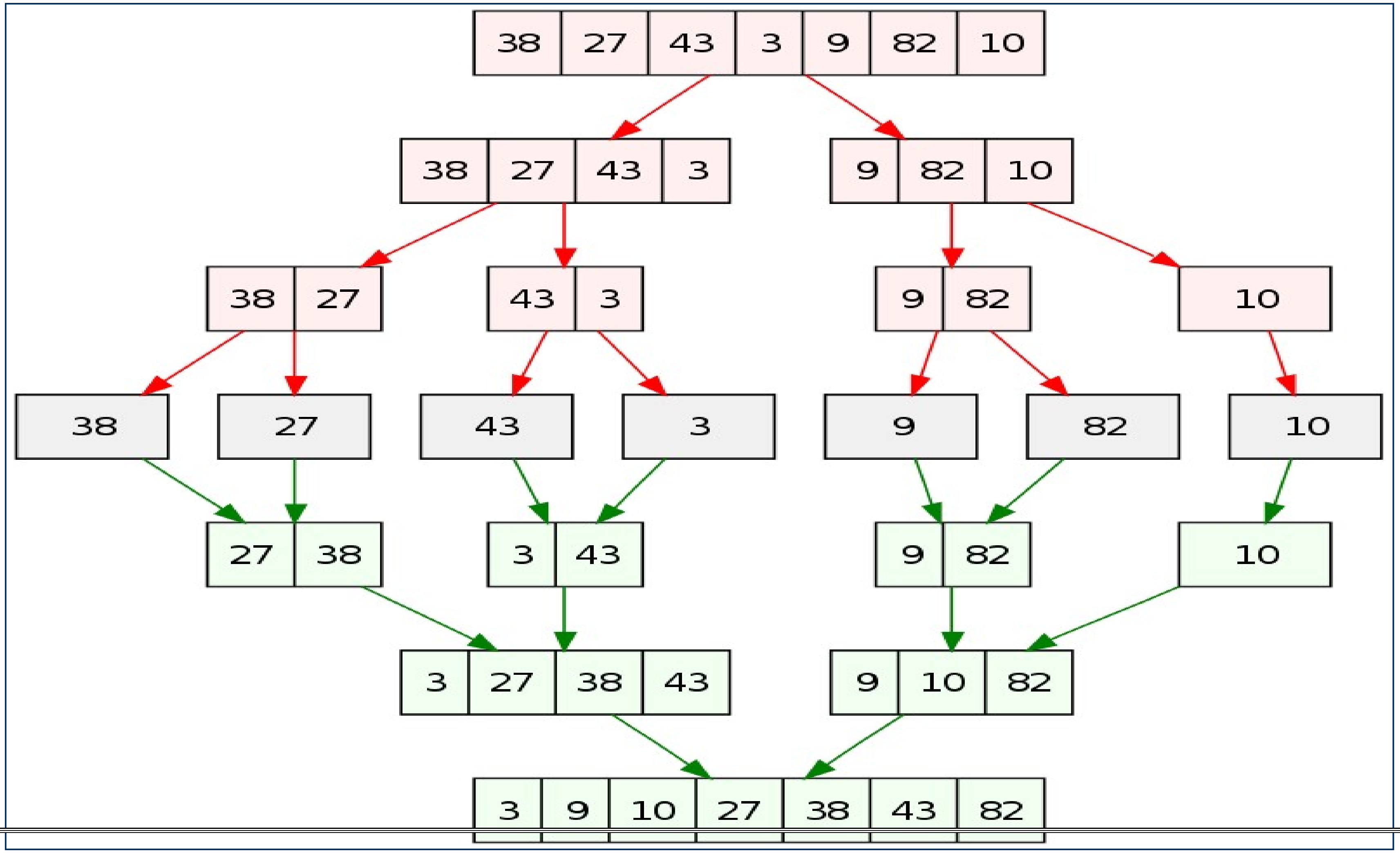
- We can stop when the unsorted side has just one number, since that number must be the largest number.



“Divide and Conquer”

- Very important strategy in computer science:
  - Divide problem into smaller parts
  - Independently solve the parts
  - Combine these solutions to get overall solution
- **Idea 1**: Divide array into two halves, *recursively* sort left and righthalves, then *merge* two halves ➐ Mergesort
- **Idea 2 :** Partition array into items that are “small” and items that are“large”, then recursively sort the two sets ➐ Quicksort

Mergesort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Mergesort

Quick Sort

- Quick Sort is based on the **Divide and Conquer rule.**
- It is also called **partition-exchange sort**. This algorithm divides the list intothree main parts:
- Elements less than the **Pivot** element
- Pivot element(Central element)
- Elements greater than the pivot element

Quick Sort

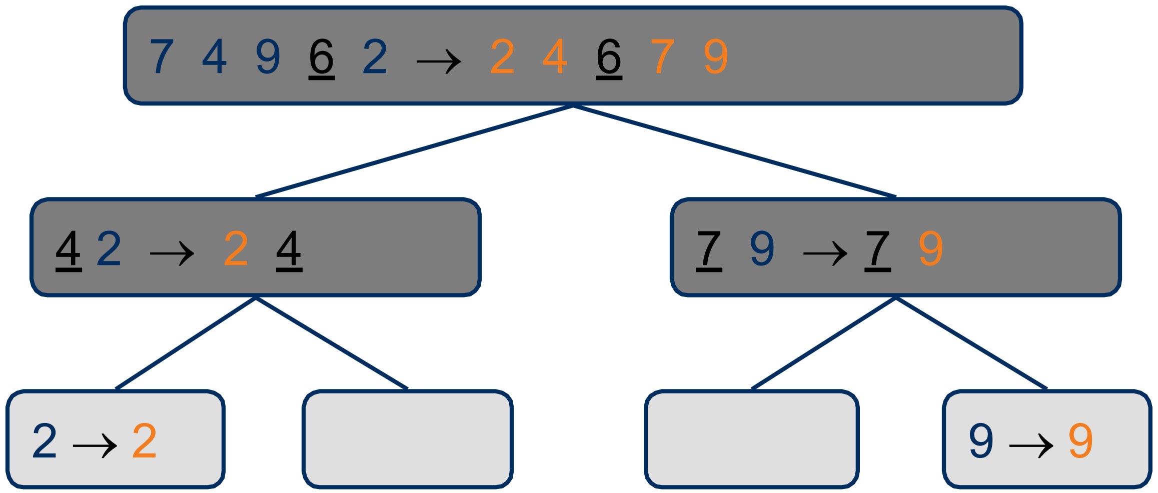
Pivot element can be any element from the array, it can be the first element, thelast element or any random element. In this tutorial, we will take the rightmost element or the last element as pivot.

For example: In the array {52, 37, 63, 14, 17, 8, 6, 25}, we take 25 as pivot. Soafter the first pass, the list will be changed like this.

{6 8 17 14 25 63 37 52}

Quick-Sort Tree

- An execution of quick-sort is depicted by a binary tree
  - Each node represents a recursive call of quick-sort andstores
    - Unsorted sequence before the execution and its pivot
    - Sorted sequence at the end of the execution
  - The root is the initial call
  - The leaves are calls on subsequences of size 0 or 1



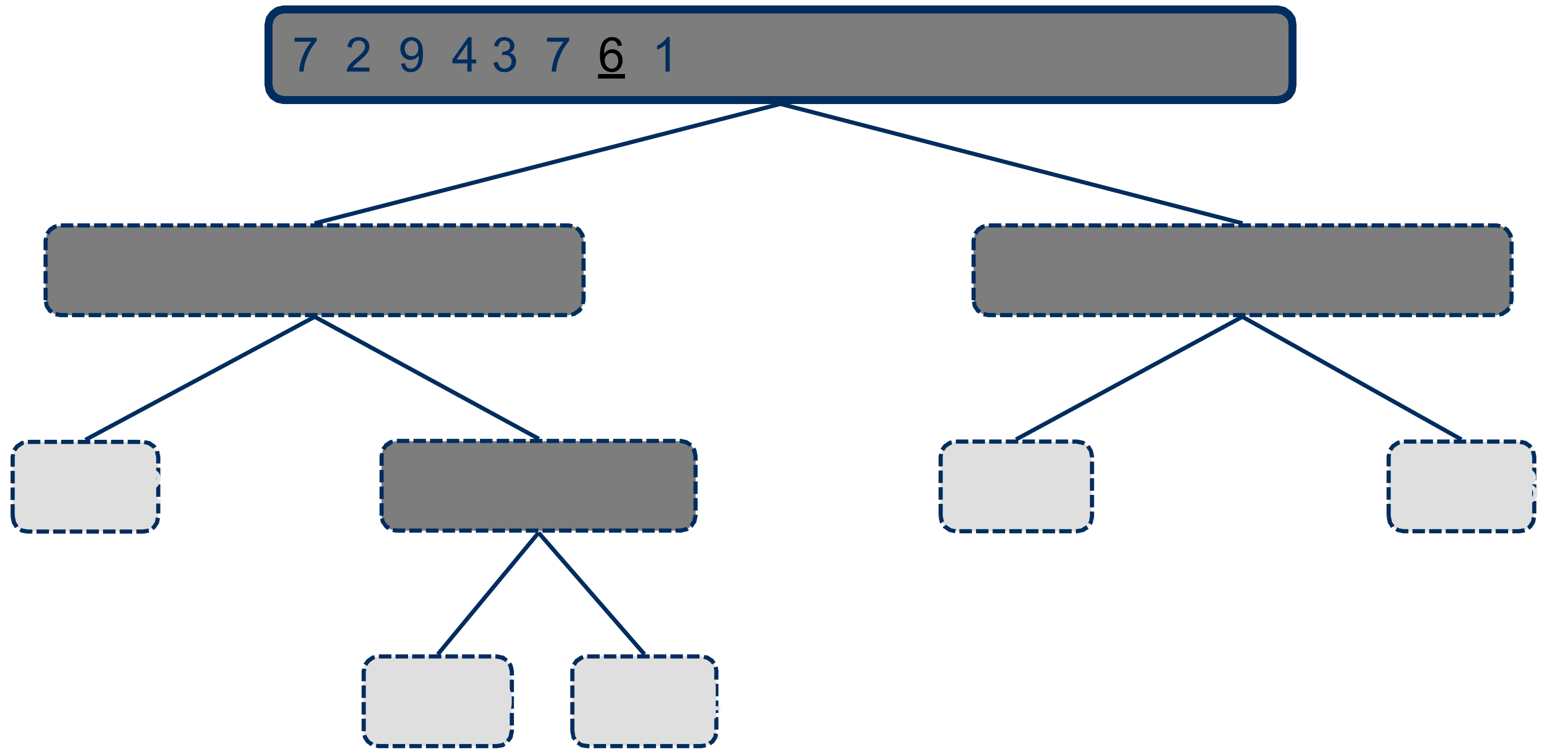
Quick-Sort Tree

98

Execution Example

- Pivot selection

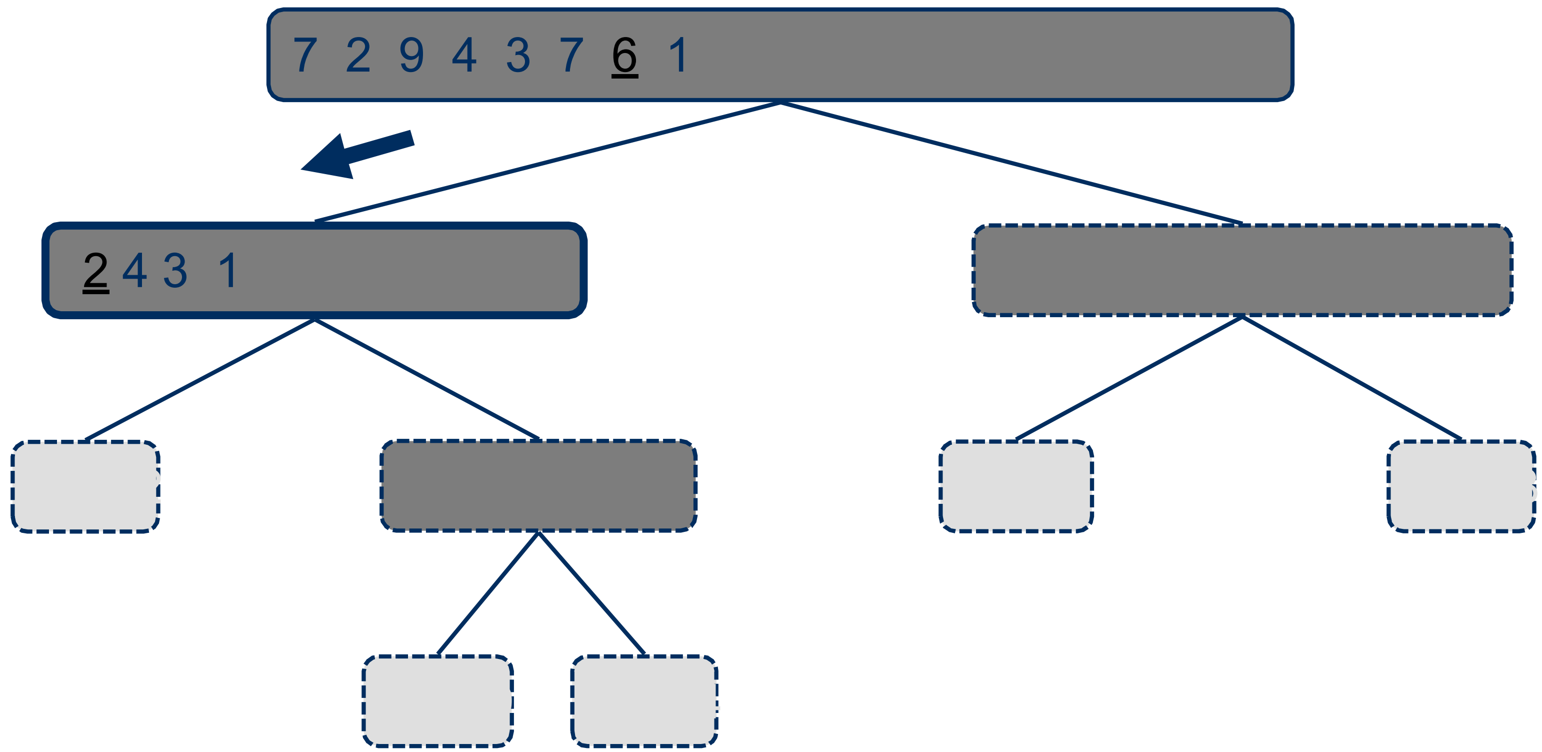
Execution Example



Execution Example (cont.)

- Partition, recursive call, pivot selection

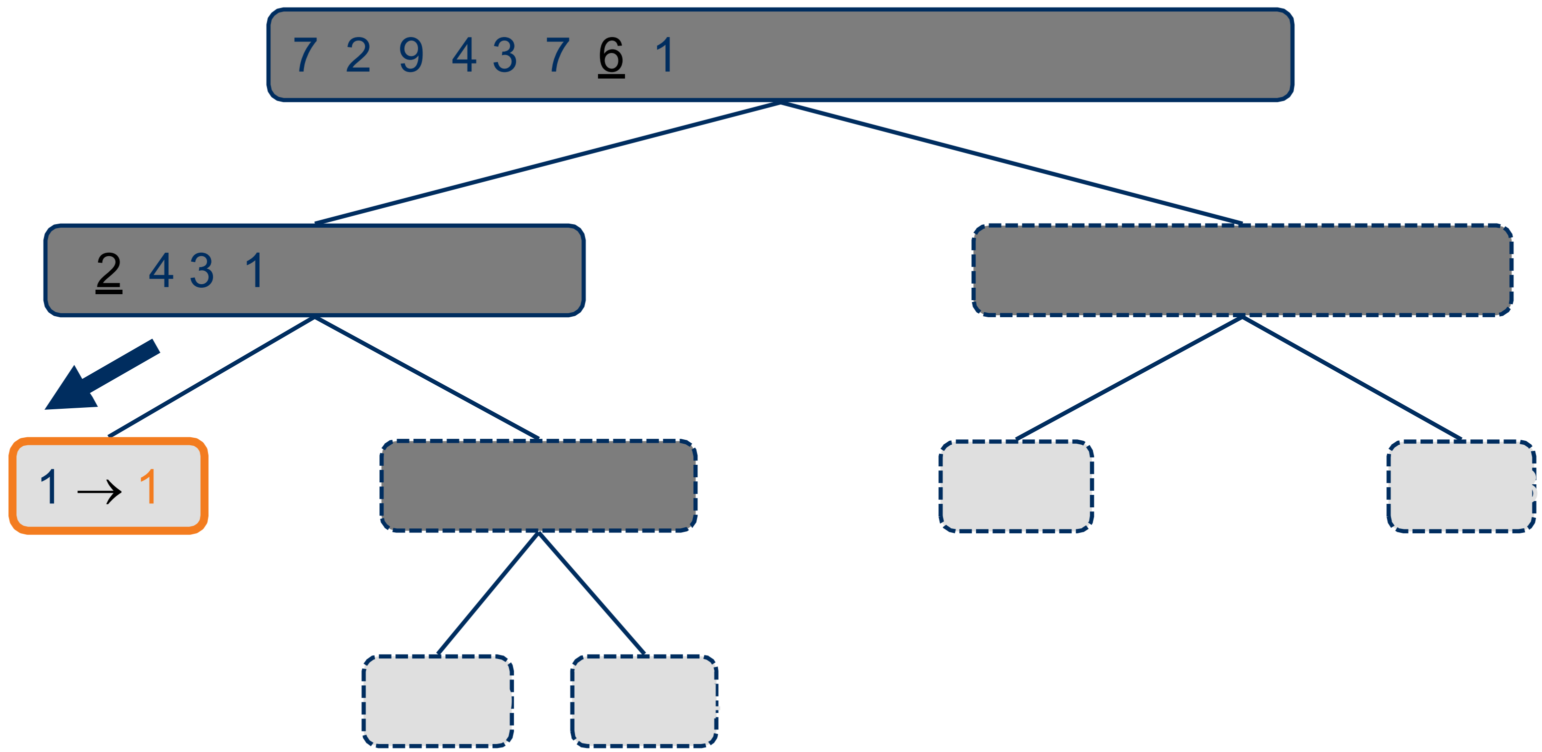
Execution Example (cont.)



Execution Example (cont.)

- Partition, recursive call, base case

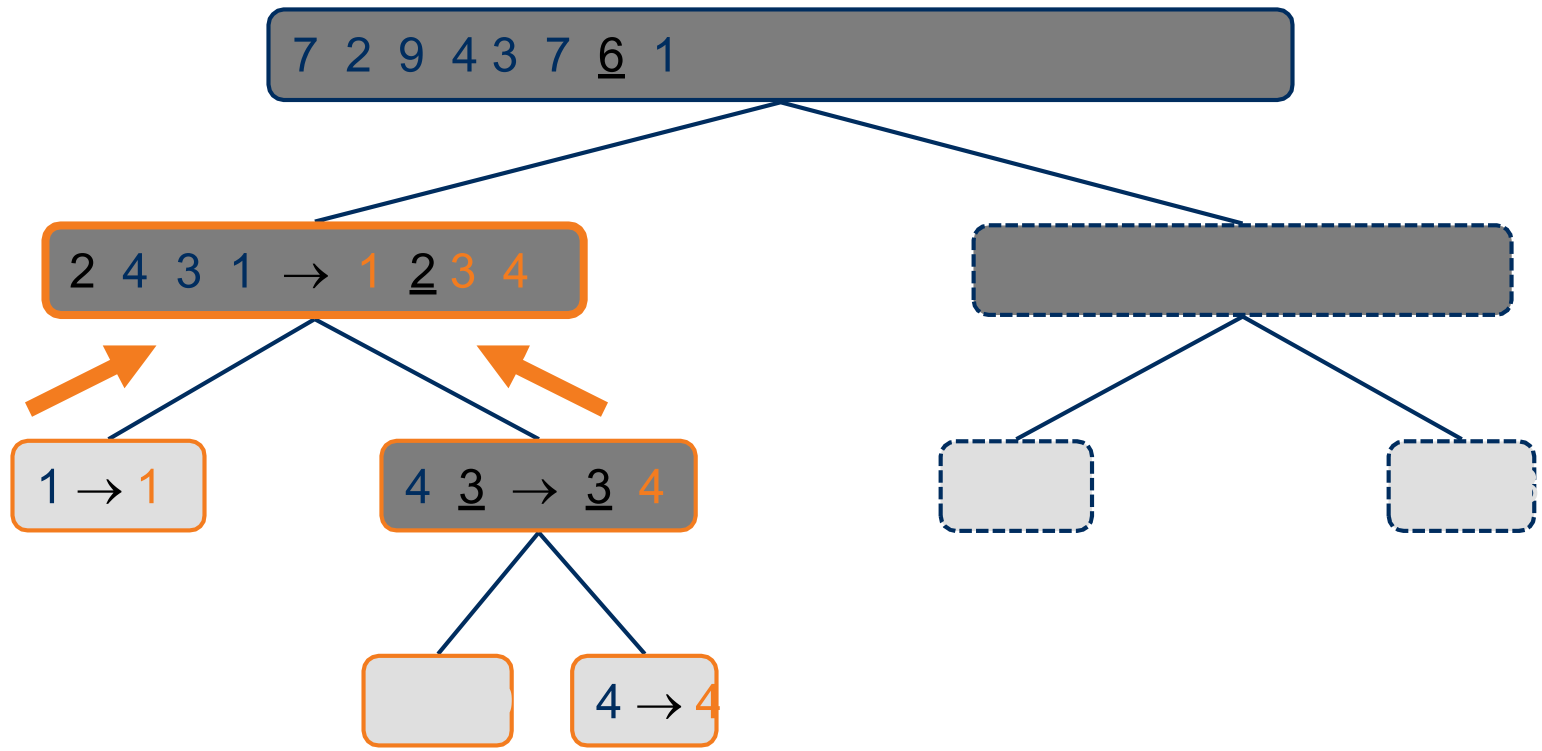
Execution Example (cont.)



Execution Example (cont.)

- Recursive call, …, base case, join

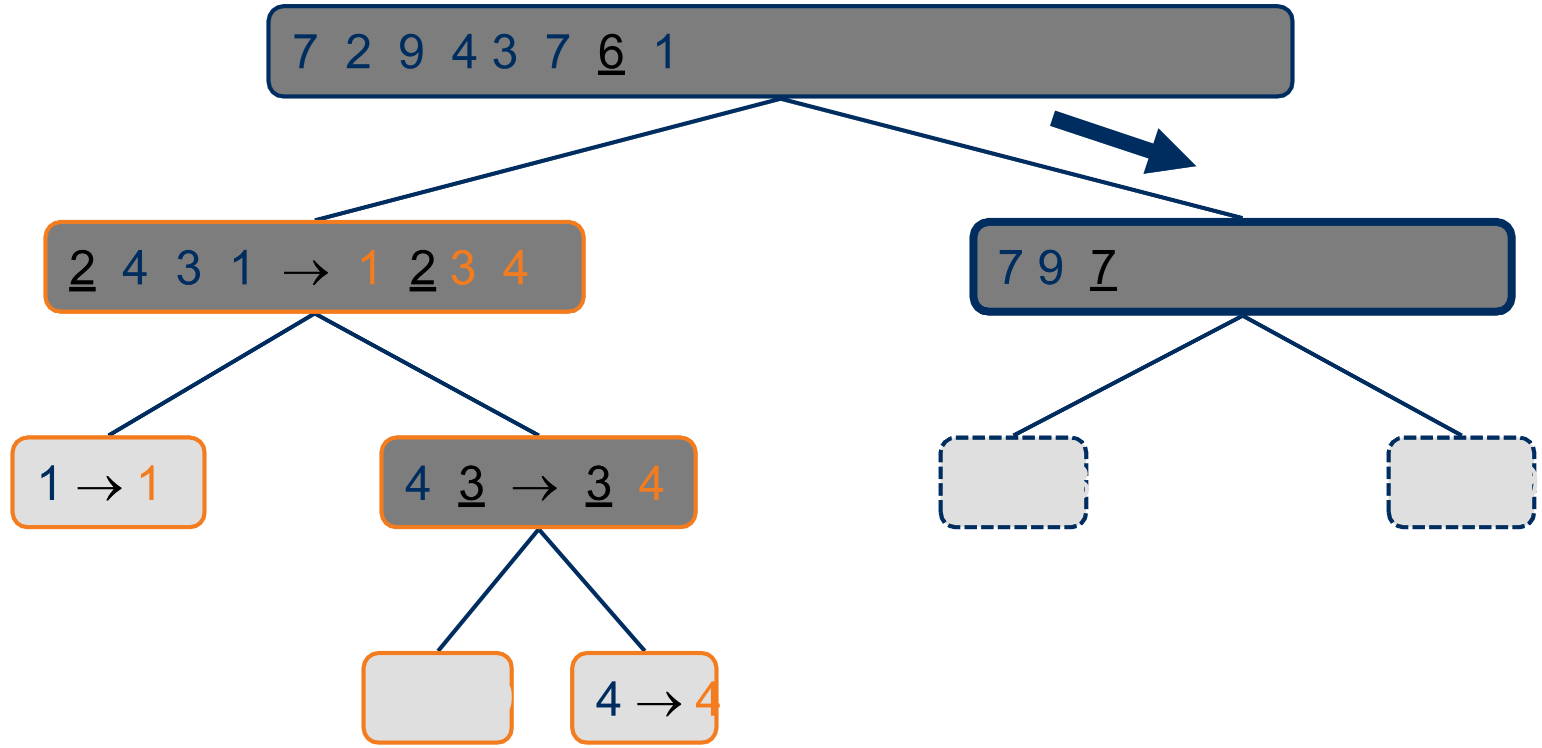
Execution Example (cont.)



Execution Example (cont.)

- Recursive call, pivot selection

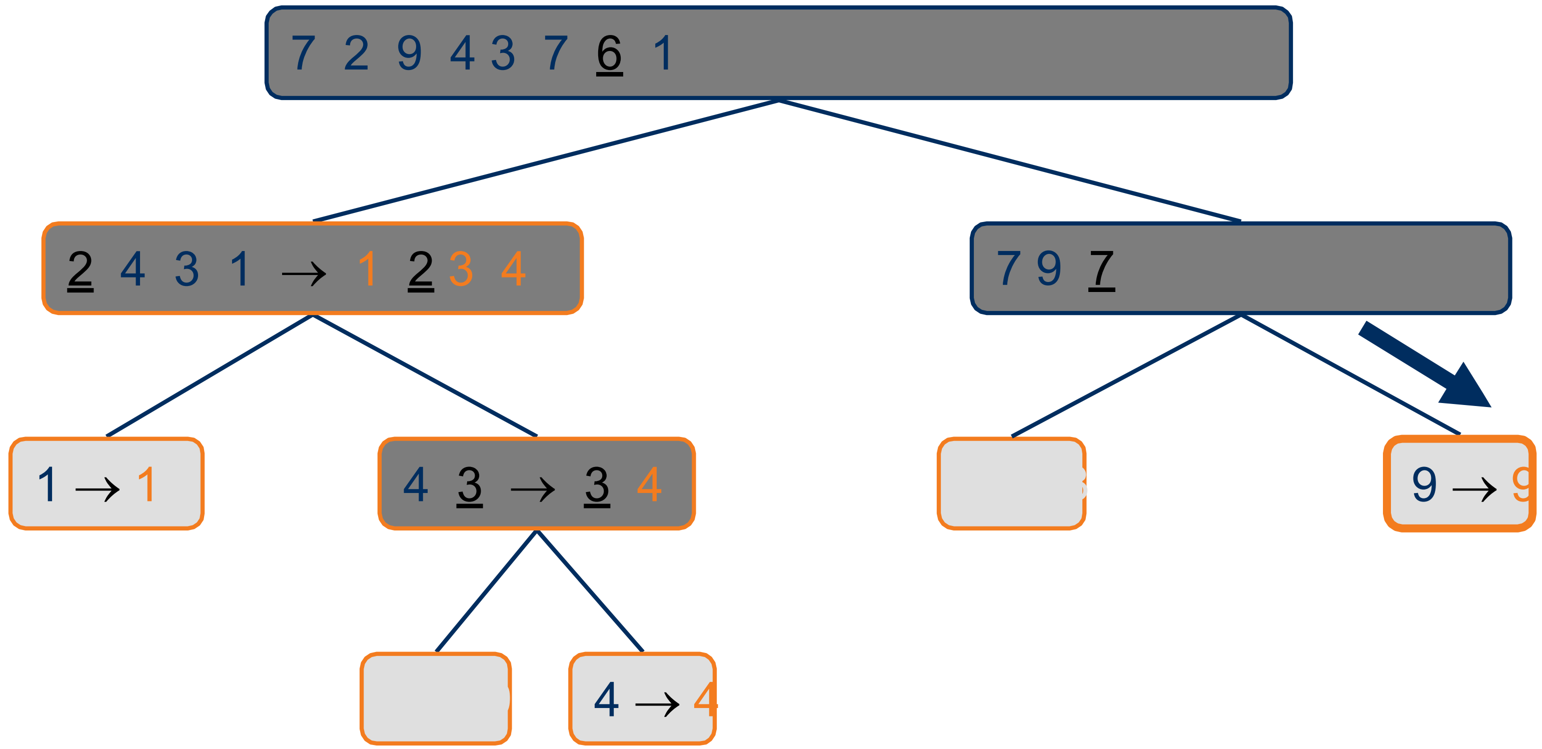
Execution Example (cont.)



Execution Example (cont.)

- Partition, …, recursive call, base case

Execution Example (cont.)



Execution Example (cont.)

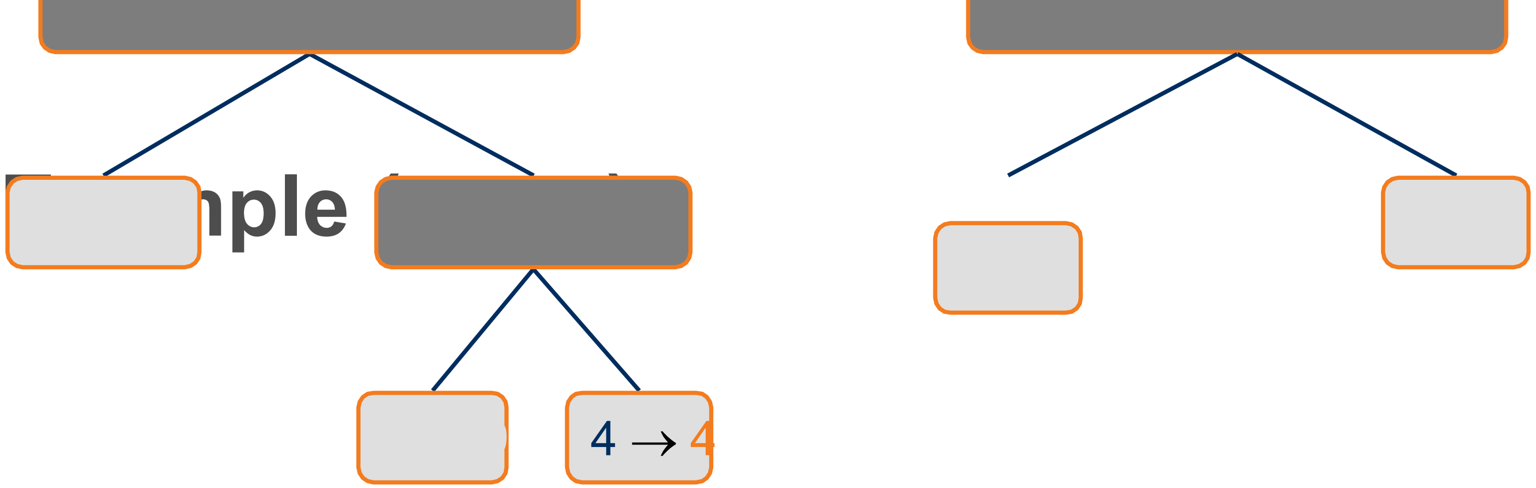
- Join, join

$7\ 2\ 9\ 4\ 3\ 7\ \underline{6}\ 1 \rightarrow 1\ 2\ 3\ 4\ \underline{6}\ 7\ 7\ 9$

$\underline{2}\ 4\ 3\ 1 \rightarrow 1\ \underline{2}\ 3\ 4$ $\qquad$ $7\ 9\ \underline{7} \rightarrow 7\ \underline{7}\ 9$

$1 \rightarrow 1$ $\qquad$ $4\ \underline{3} \rightarrow \underline{3}\ 4$ $\qquad$ 8 $\qquad$ $9 \rightarrow 9$

Execution Example



Heap Sort

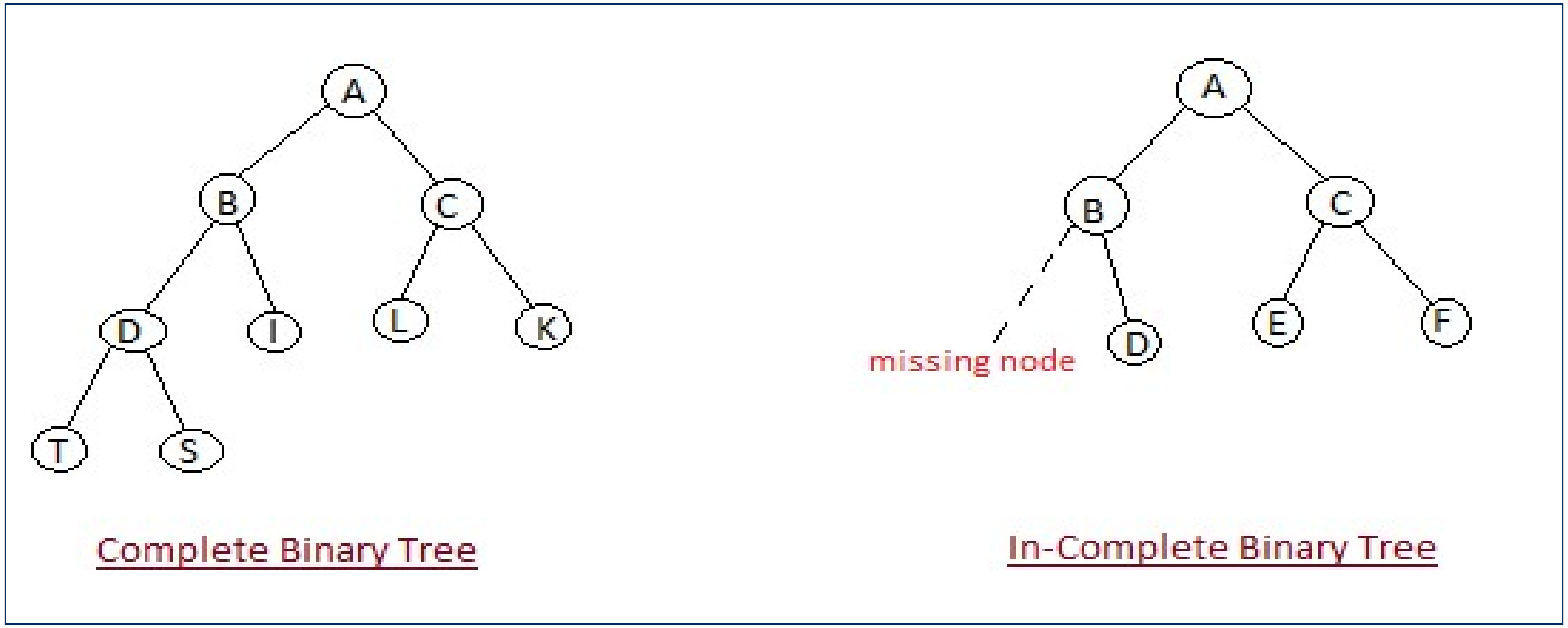
Heap sort involves building a Heap data structure from the given array and thenutilizing the Heap to sort the array.

- Heap is a special tree-based data structure, that satisfies the following specialheap properties:

Heap Sort

- Shape Property: Heap data structure is always a Complete Binary Tree, whichmeans all levels of the tree are fully filled.

Heap Sort


A
B
C
D
I
L
K
T
S
Complete Binary Tree
A
B
C
missing node
D
E
F
In-Complete Binary Tree

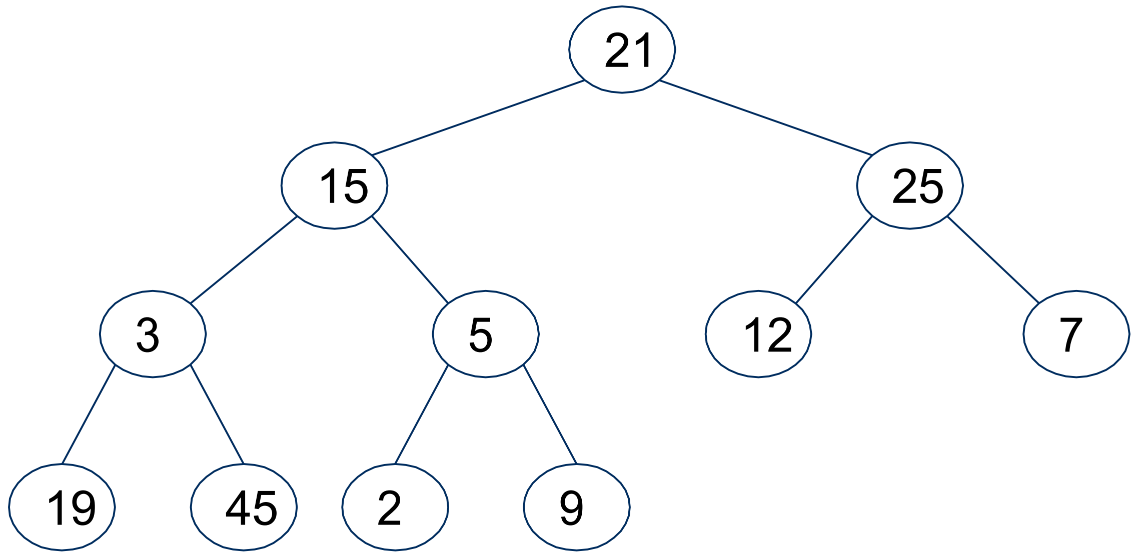
Sample Run

- Start with unordered array of data

Array representation:

	21	15	25	3	5	12	7	19	45	2	9

Binary tree representation:


21
15
25
3
5
12
7
19
45
2
9

Heap Sort - Video link

Heap Sort

https://www.youtube.com/watch?v=H5kAcmGOn4Q

Merge Sort Vs Quick Sort - Video link

Merge Sort Vs Quick Sort

https://www.youtube.com/watch?v=es2T6KY45cA

Additional References

- https://www.youtube.com/watch?v=WaNLJf8xzC4

Data Structures

Data Structures

Why Data structure

Watch the Video

https://www.youtube.com/watch?v=-q-3b_093do

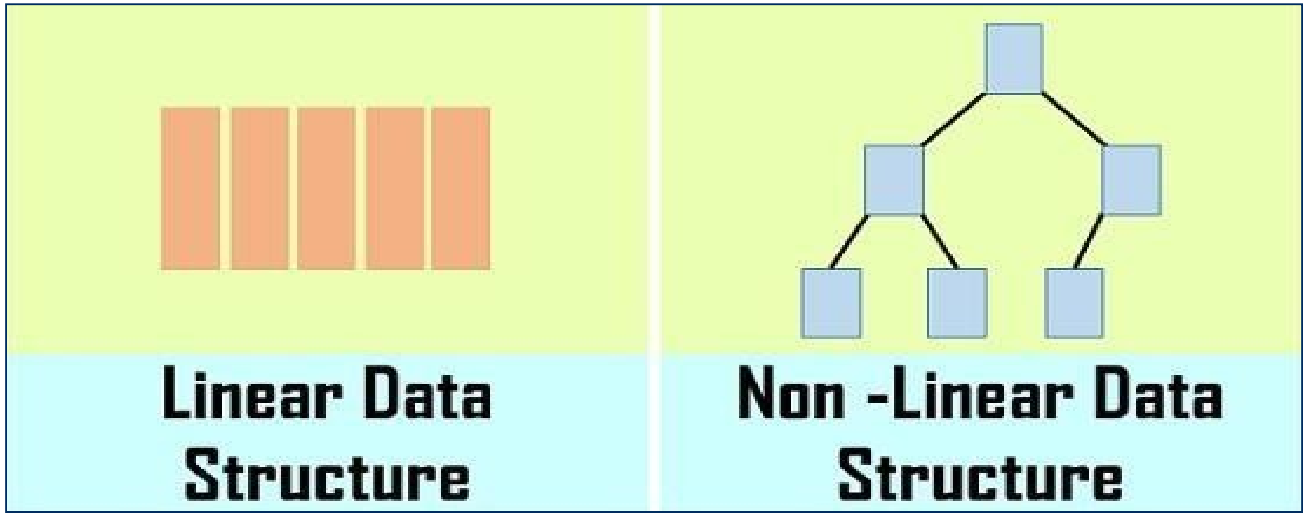
Data Structures

- Data structures is concerned with the representation and manipulation of data
- All programs manipulate data
- So, all programs represent data in some way
- Data manipulation requires an algorithm
- The study of Data Structure is fundamental to computer programming

Types of Data Structure

There are basically two types of data structure

1. Linear Data Structure
2. Non-Linear Data Structure.



Basic data structures: *data collections*

- Linear structures
  - Array: Fixed-size
  - Linked List: Add to top, bottom or in the middle
  - Stack: Add to top and remove from top
  - Queue: Add to back and remove from front
  - Priority queue: Add anywhere, remove the highestpriority

- Non- Linear Data Structure
  - Tree: A branching structure with no loops
  - Graph: A more general branching structure, with lessstringent connection conditions than for a tree

Static vs. Dynamic Structures

- A static data structure has a fixed size

  This meaning is different from the meaning of the static modifier (variableshared among all instances of a class)

- Arrays are static; once you define the number of elements it can hold, thenumber doesn't change

- A dynamic data structure grows and shrinks at execution time as required byits contents

- A dynamic data structure is implemented using links

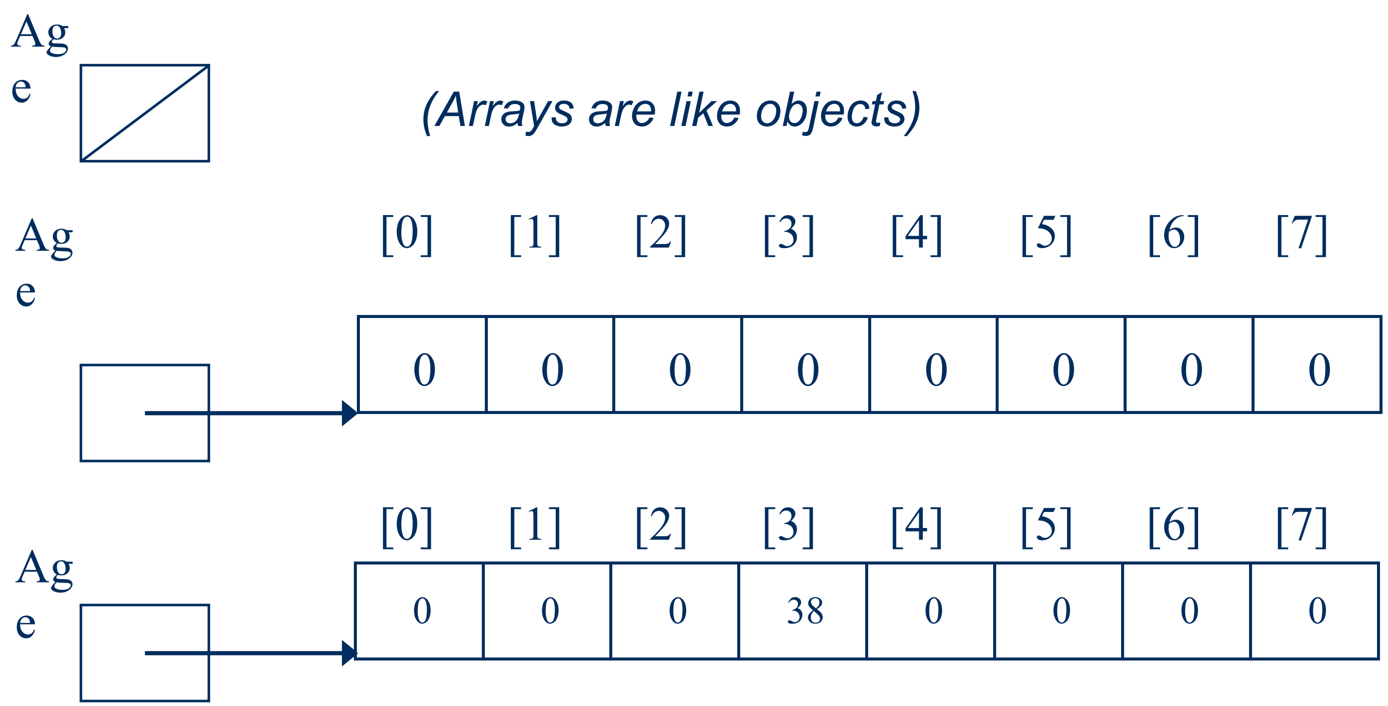
Array

An array of integers

1 int [ ] Age;

2 Age= new int[8];

3 Age [3] = 38;



Declaration ___Allocation ___Initialization

Linked List

- a **linked list** is a linear collection of data elements, in whichlinear order is not given by their physical placement in memory.
- Elements may be added in front, end of list as well asmiddle of list.
- Linked List may be use for dynamic implementation of stackand queue.

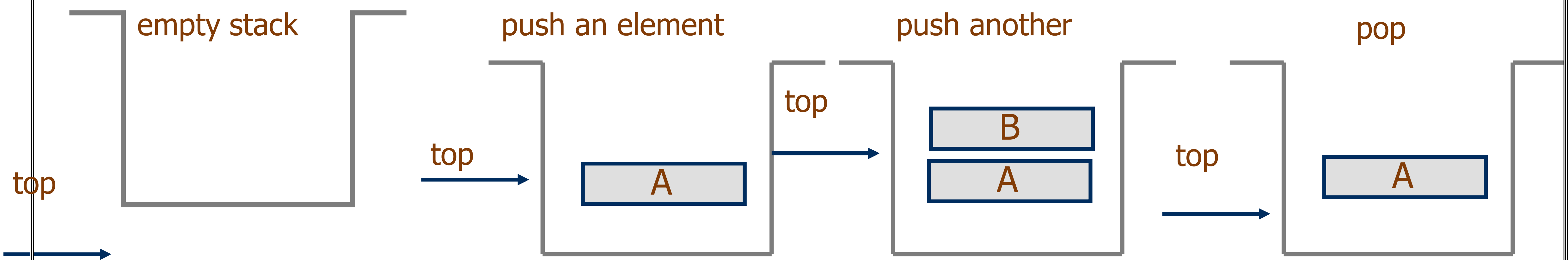
Stack

- Stack is a linear data structure which works on LIFOorder. So that Last In First Out .
- In stack element is always added at top of stack andalso removed from top of the stack.
- Stack is useful in recursive function, function calling,mathematical expression calculation, reversing the string etc.

Data Structure -- Stacks

- LIFO (Last In, First Out) in Stack:

  The last element inserted will be the first to be retrieved, using Push and Pop
- Push
  - Add an element to the *top* of the stack
- Pop
  - Remove the element at the *top* of the stack



Data Structures -- Stacks

Attributes of `Stack`

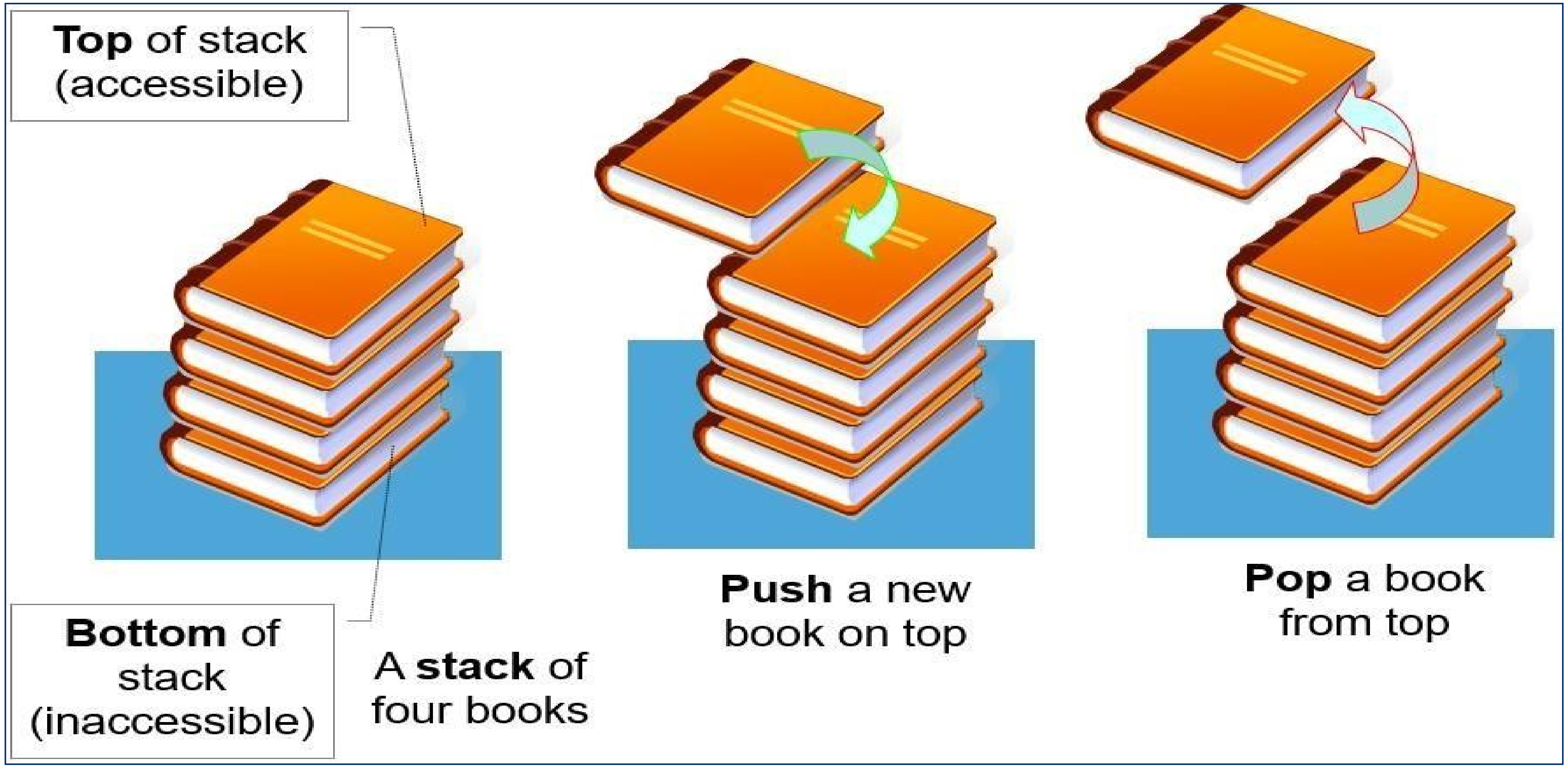
- `maxTop`: the max size of stack
- `top`: the index of the top element of stack

- Operations of `Stack`
  - `empty`: return true if stack is empty, return false otherwise
  - `full`: return true if stack is full, return false otherwise
  - `top`: return the element at the top of stack
  - `push`: add an element to the top of stack
  - `pop`: delete the element at the top of stack
  - `displayStack`: print all the data in the stack

Data Structure -- Stacks

- Real life analogy:
  - Elevator
    - Dish holders (stacks)
  - Typical uses of stacks:
  - Prefix-/Postfix- calculators
- Any list implementation could be used to implement a stack
  - Arrays (static: the size of stack is given initially)
  - Linked lists (dynamic: never becomes full)

Data Structure -- Stacks



Data Structure -- Queues

- Like a stack, a *queue* is also a list. However, with a queue, insertion is done atone end, while deletion is performed at the other end
  - The insertion end is called *rear*
  - The deletion end is called *front*

Data Structure -- Queues

Remove

(Dequeue)

front

rear

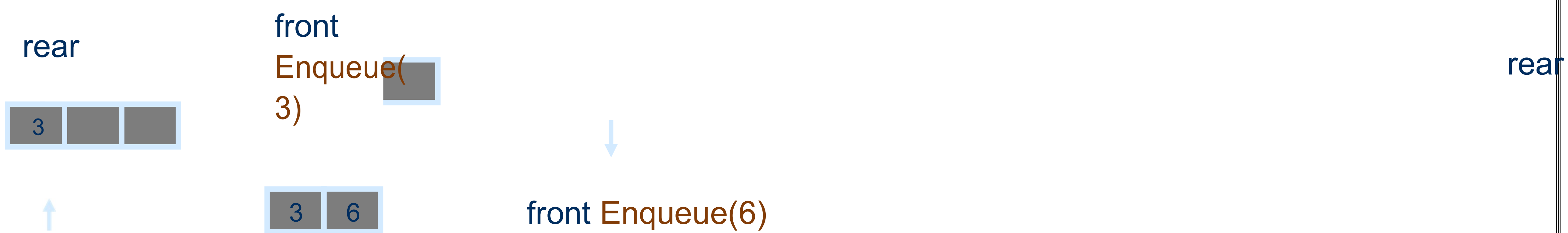
Insert
(Enqueue)

Data Structure -- Queues

- Attributes of `Queue`
  - `front/rear`: front/rear index
  - `counter`: number of elements in the queue
  - `maxSize`: capacity of the queue
- Operations of `Queue`
  - `IsEmpty`: return true if queue is empty, return false otherwise
  - `IsFull`: return true if queue is full, return false otherwise
  - `Enqueue`: add an element to the rear of queue
  - `Dequeue`: delete the element at the front of queue
  - `DisplayQueue`: print all the data

Data Structure -- Queues

- Accessing the elements of queues follows a FIFO (First In, First Out) order
  The first element inserted will be the first to be retrieved, using Enqueue and Dequeue
  - Enqueue
    - Add an element after the *rear* of the queue
  - Dequeue
    - Remove the element at the *front* of the queue



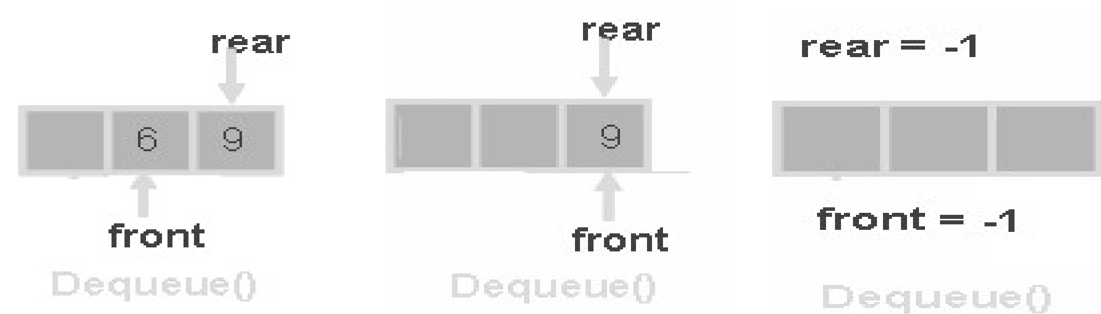
Data Structure -- Queues

(9)

rear

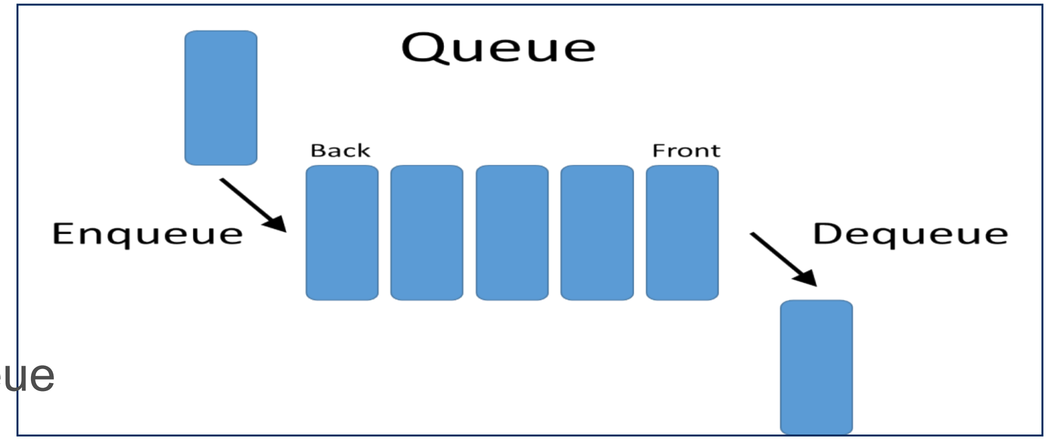
front

Enqueue



Data Structure -- Queues

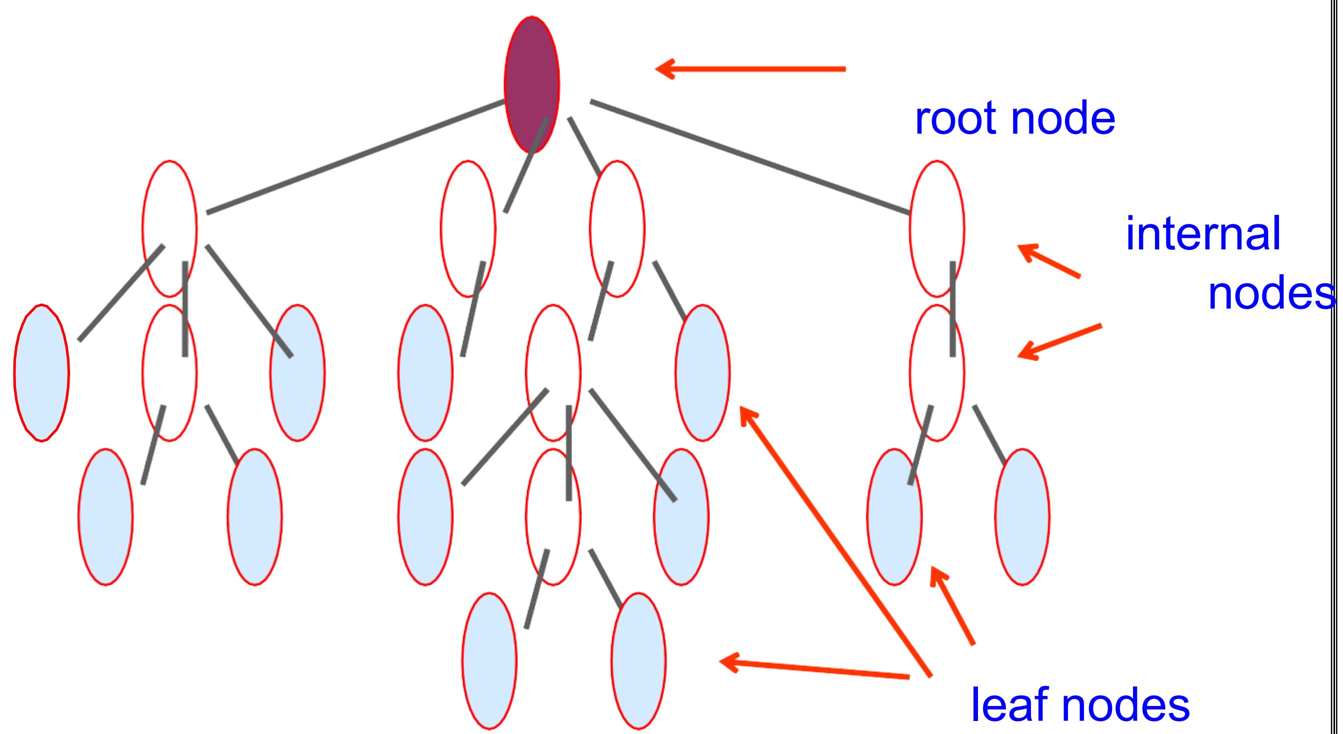
- Real life analogy:
  - Check-out lines in a store (queuing up)
- Typical uses of queues:
  - Waiting lists of course registration
  - Simple scheduling in routers
- Any list implementation could be used to implement a queue
  - Arrays (static: the size of queue is given initially)
  - Linked lists (dynamic: never becomes full)



Trees

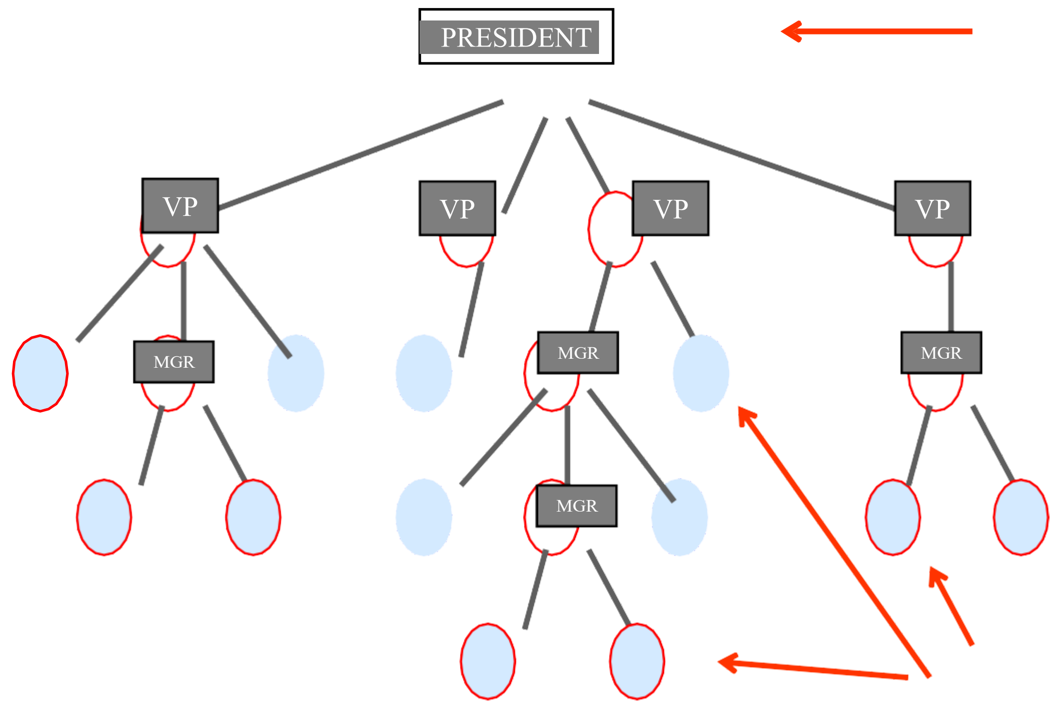
- A tree is a non-linear data structure that consists of a root node and potentiallymany levels of additional nodes that form a hierarchy
- Nodes that have no children are called leaf nodes
- Non-root and non-leaf nodes are called internal nodes

imagine an upside
down tree



A tree data structure

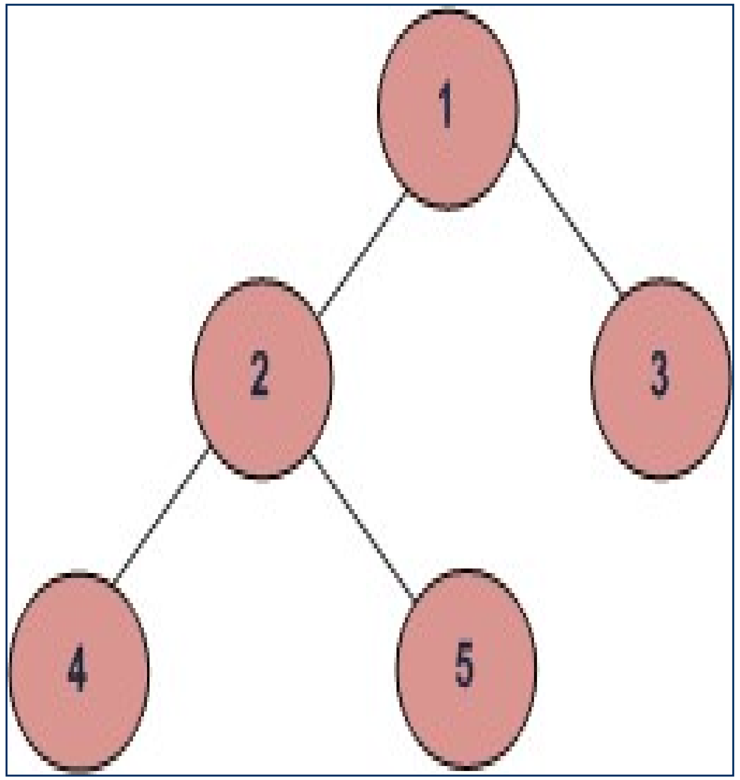
Organization chart represented via a tree data structure



Tree Traversal

- Two main methods:
  - Inorder
  - Preorder
  - Postorder
- Recursive definition
- Inorder
- Preorder:
  - visit the root
  - traverse in preorder the children (subtrees)
- Postorder
  - traverse in postorder the children (subtrees)
  - visit the root

Tree traversal (cont..)



BFS and DFSs of the Tree

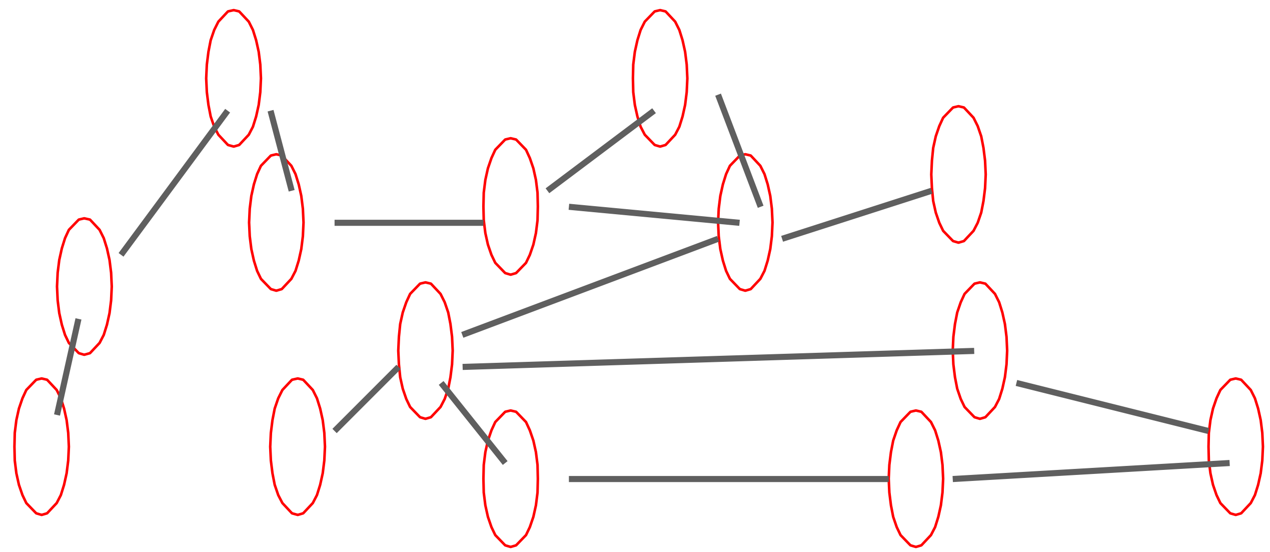
– Breadth First Traversal : 1 2 3 4 5

– Depth First Traversals:
– Preorder Traversal : 1 2 4 5 3
– Inorder Traversal : 4 2 5 1 3
– Postorder Traversal : 4 5 2 3 1

Graph

- A graph is a non-linear structure (also called a network)
- Unlike a tree or binary tree, a graph does not have a root – no primary entry point.
- Any node can be connected to any other node by an edge
- Can have any number of edges and nodes
- Analogy: the highway system connecting cities on a map

a graph data structure



Summary

- Qualities of a software Developer
- Problem solving approaches
- Problem classification
- Flow chart design
- Algorithms-Pseudo codes
- Algorithm Patterns
- Data Structures

Summary

Learning material references

- Books
  - “Introduction to Algorithms”, Thomas H Cormen, Charles Leiserson, Ronald Rivestand Clifford Stein, 3rd edition, MIT, July 2009
  - “Problem Solving Using C: Structured Programming Techniques”, Yuksel Uckan ,McGraw-Hill Inc.,1998
  - "Data Structures and Algorithms Made Easy in Java: Data Structure and AlgorithmicPuzzles", Narasimha Karumanchi, areerMonk Publications, 2014
- Web
  - http://www.slideshare.net/dokka/program-design-and-problem-solving-techniques

Learning material references