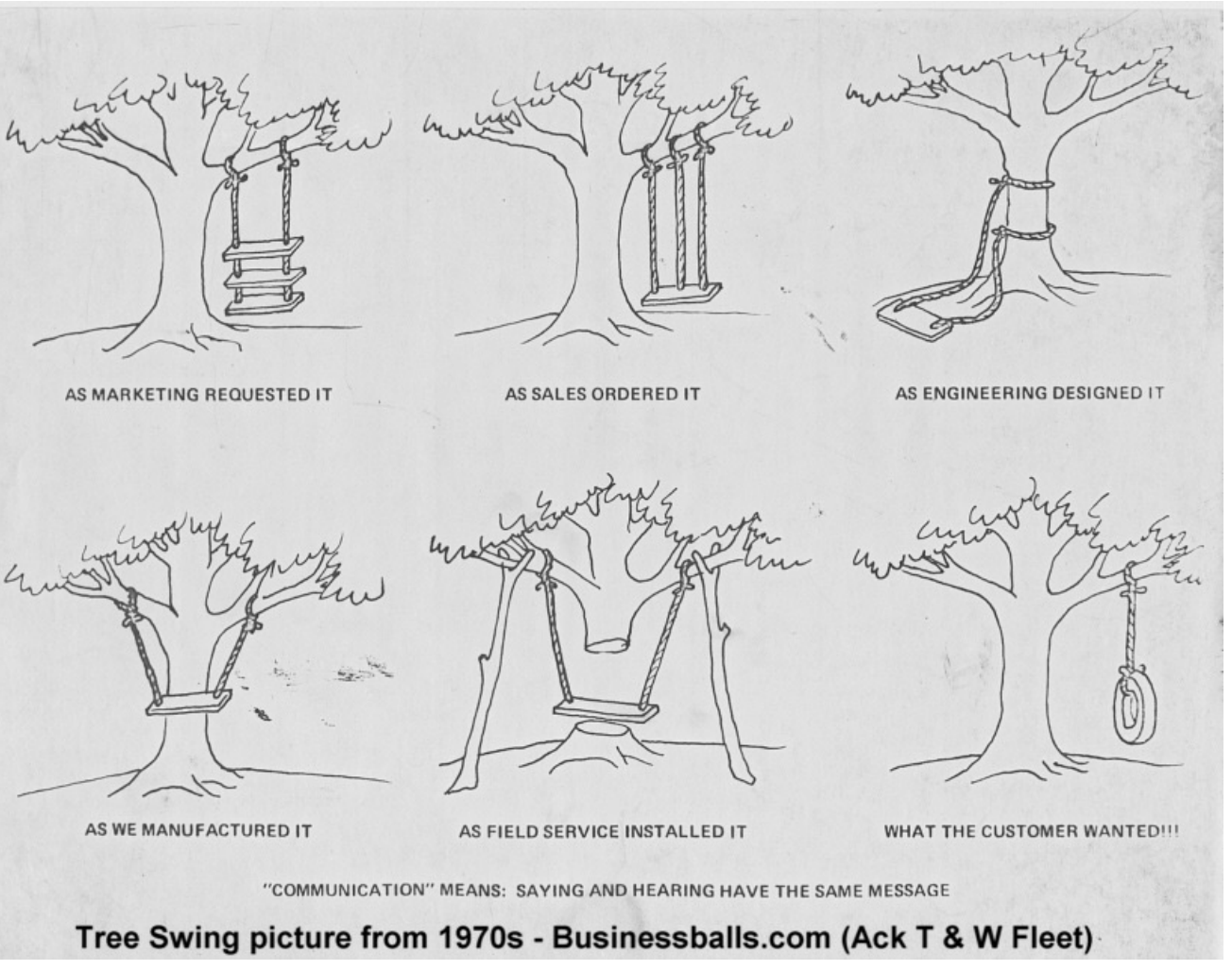
Problem Solving
and
Computational Thinking

“Problem solving is a skill that can be developed via practice”

- Define the Problem
  - What exactly is the problem that we are trying to solve?
- Identify the Problem
  - How and why did the problem happen?
- What are all the possible solutions?
  - The ideal solution could be one of the many possible solutions.
- A decision is to be made.
  - Any decision is usually better than no decision at all.



Tree Swing picture from 1970s - Businessballs.com (Ack T & W Fleet)

- Assign responsibility to carry out the decision.
  - If a team then who will do what and when.
  - If alone, still decide when are you going to do it

- Set a schedule.
  - Without schedule and deadline, its just a discussion.

- Task self/someone else to take definite action to implement the solution and resolve the problem.

Core Components of Computational Thinking

- Decomposition
  - Break down complex problems into smaller, simpler problems.
- Pattern recognition
  - Make connections between similar problems and experience.
- Abstraction
  - Identify important information while ignoring unrelated or irrelevant details.
- Algorithms
  - Creates sequential rules to follow in order to solve a problem.

Algorithm
and
Data Structures

Algorithm

- A “**finite sequence**” of “**well defined**” computational steps that transforms “**input**” into the “**output**”.

- Basic constructs of an algorithm.
  - Linear Sequence – statements that follow one after the other.
  - Conditional – “if then else”
  - Loop – sequence of statements that are repeated a number of times.

Data Structure

- A *data structure* is a way to **store** and **organize** data in order to facilitate **access** and **modifications**.

- No single data structure works well for all purposes, and so it is important to know the strengths and limitations of several of them.

Linear Data Structures

Array

- Need for an array?

If same processing is to be done on a set of values.

```
int sum = 0;
int age = 20;
int year = 2024;
```

```
int arr[3];
```

Sum of 3 numbers

```
int no1, no2, no3;
int sum = 0;
    :
sum = no1 + no2 + no3;
    :
```

Sum of 100 numbers

```
int nos[100];
int sum = 0;
    :
for (i = 0; i < 100; ++i)
    sum = sum + nos[i];
    :
```

Properties of Array

- Data Structure that stores multiple elements, all of the same type.

each element takes same amount of memory.

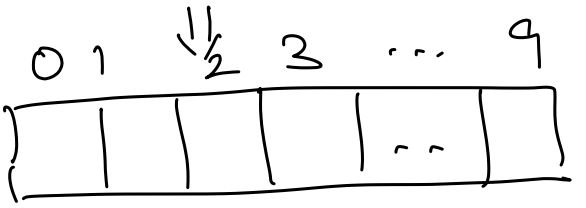
- All elements of an array are stored sequentially in memory, one after another.

Random Access

arr [i]

base address + i * memory required by one integer.

int arr [10];



Base address of array

Pros and Cons of Array

- Advantages
  - Efficient lookup OR Random access.
  - Efficient in adding and removing elements at the end of array

logical size

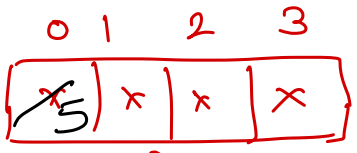
- Disadvantages
  - Fixed size. Resizing of array is inefficient.
  - Insertion and deletion of elements, in middle of array is inefficient.

physical size / capacity

How much memory is allocated to array. = 4

logical size

How many elements are stored in array.



garbage

int arr [4]

int count = 0

logical size.

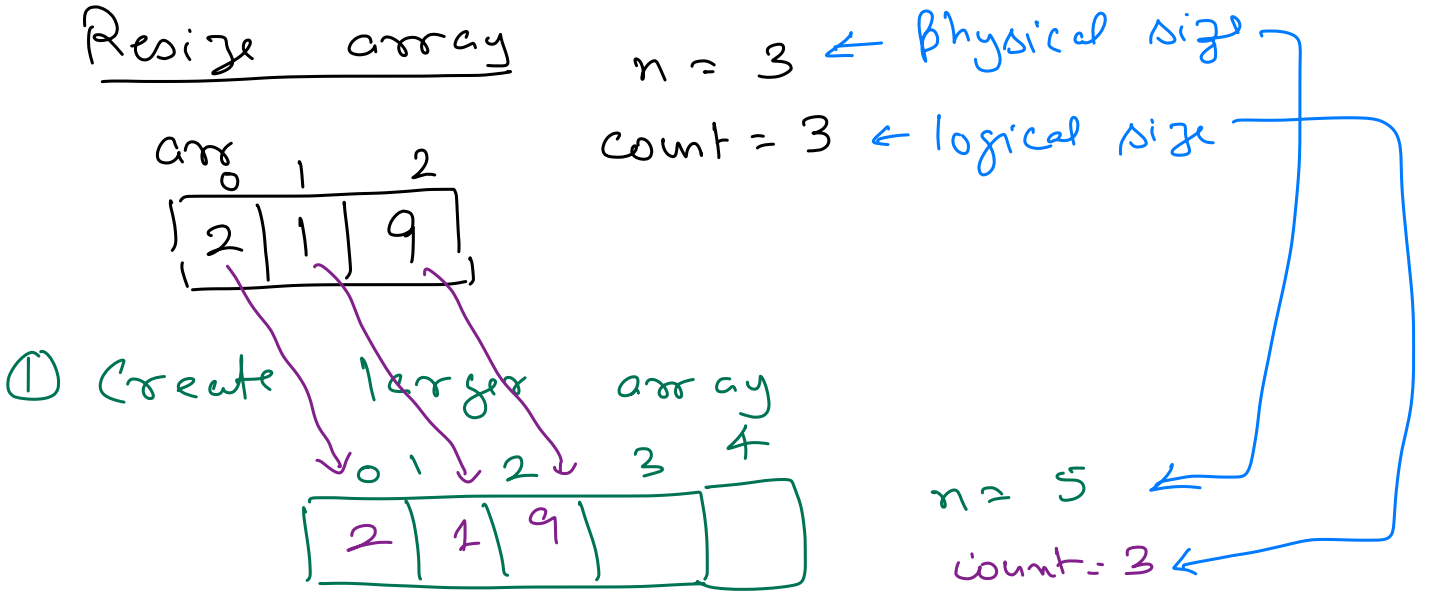
Append an element

```
arr [count] = 5;
count = count + 1;
```

Delete last element

```
count = count - 1;
```

Resize array



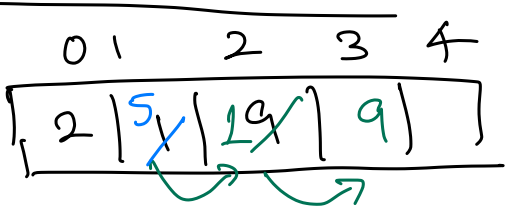
```java
public static int[] resizeArray(int[] arr, int newSize) {
    // Create a larger array.
    int[] newArray = new int[newSize];

    // Copy elements from old array to new.
    for (int i = 0; i < arr.length; ++i) {
        newArray[i] = arr[i];
    }

    return newArray;
}
```

Insert an element

Insert (5, 1)



Count = ~~3~~ 4

1. Shift elements to right by one place.
2. Store element at position.
3. Increment logical size by 1

```java
public static int insertInArray(int[] arr, int n, int pos, int value) {
    // Shift elements to right by one place.
    for (int i = n; i > pos; --i) {
        arr[i] = arr[i - 1];
    }

    // Store new element.
    arr[pos] = value;

    // Increment logical size by 1.
    return n + 1;
}
```

Assignment : Implement delete an element from array.

Stack

- Stack is a linear data structure.
- Stack is a container of objects.

Stack operations

- LIFO – Last In First Out
- Elements are added and removed according to LIFO principle.
- Operations are performed with respect to “**top**” of stack.

Stack as Abstract Data Type (ADT)

- push ← add element to stack
- pop ← remove element from stack
- peek ← get top most element
- isEmpty
- isFull

define what operations can be performed.

Client → what needs the functionality.

Interface → defines what operations can be performed.

Implementation.

```java
public interface Stack {
    void push(int element);
    int pop();
    int peek();
    boolean isEmpty();
    boolean isFull();
}
```

← ADT.