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.
  - *80 percent of problems should be solved at the moment when they come up, only 20% needs time, deliberation and research.*

- 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

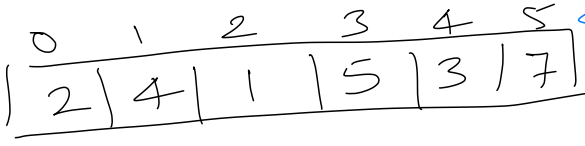
- A *data structure* is a way to **store** and **organize** data in order to facilitate **access** and **modifications**.

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

Array ⇒ Sequential / Linear data structure.

↓

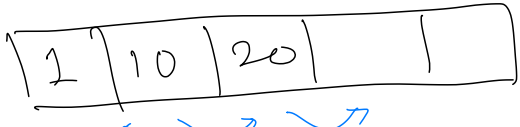
Stores data in a sequential



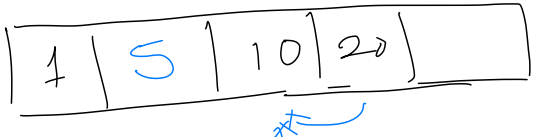
All element in an array are stored in a single memory block.

we can randomly access array element using index / subscript.

Inserting / deleting elements in array require shifting of elements.



Insert 5 : Shift all elements to right, from place where to put 5.



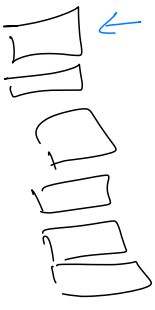
Delete 10 : Shift all elements right of 10 to left by 1 place.

Data Structures: Define how data can be accessed.

- Stack : Last In First Out (LIFO)
- Queue : First In First Out (FIFO)

Stack → Push : Adds the element at the top of stack.

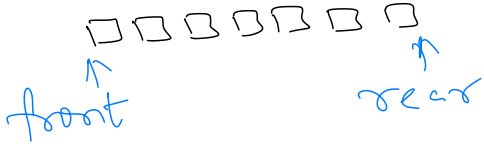
↳ Pop : Removes the element from top of stack.



Stack of Books

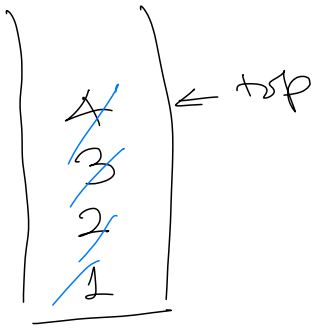
Queue → Add : Adds the element at the rear of queue

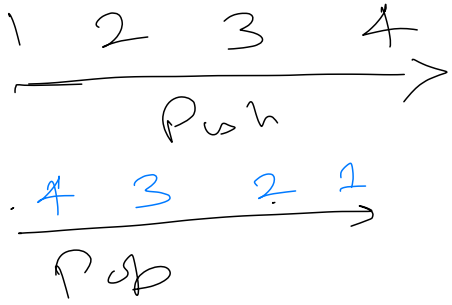
↳ Delete : Removes the element from front of queue.




```
public interface StackIntf {
    public void Push(int);
    public int Pop();
    public bool IsEmpty();
    public bol IsFull();
};
```

ADT





public class StackArray
implements StackInt{

};

---

void RevUsingStack(int[] elems,
StackInt stackObj){

}

StackInt s = new StackArray();
RevUsingStack(arr, 5);

```
void RevUsingStack(int[] elems,
                   StackInt stackObj) {

    for (int e: elems)
        stackObj.Push(e);

    int i = 0;
    while (!stackObj.IsEmpty()){
        elems[i] = stackObj.Pop();
        ++i;
    }
}
```

```
public interface QueueInt {
    public void AddQ(int);
    public int DeleteQ();
    public bool IsEmpty();
    public bool IsFull();

};
```