- Here, we will introduce a framework that can be used to discuss the design and analysis of algorithms.
- we will use "pseudocode" for specifying the algorithms.
  ⇒ algorithms are described as programs written in pseudocode.

Pseudocode

- similar to C/Pascal/Java (in many respects)
- uses most clear and concise way (such as english sentences or phrases)
- doesn't concern with issues of S/W engineering (Issues of data abstraction, modularity, error handling are often ignored)

Pseudocode Conventions

1. Indentation indicates block structure.
2. The looping constructs(while, for ..) and conditional constructs(if, else ..) have similar interpretations to similar to those in C/Pascal.
   - loop counter variable retains its value after exiting the loop.

     eg for $j \leftarrow 2$ to n

     so when loop terminats, the value of j will be $n+1$.
3. The symbol '▷' indicates that the remainder of the line is a comment.
4. A multiple assignment of the form $i \leftarrow j \leftarrow e$ assigns to both variables i and j the value of expression e;

   $\Rightarrow$ it is equivalent to $j \leftarrow e$ and
   $i \leftarrow j$
5. Variables are local to the given procedure. (We will not use global variable without explicit mention)
6. Array elements are accessed by specifying the array name followed by the index in square brackets.

   eg $A[i]$ = i^{th} element of the array A.

7- - Compound data are treated as objects (which have attributes or fields)

- A particular field (of an object) is accessed using field name followed by name of its object in square brackets

eg name[Emp]

eg an array is an object with attribute 'length' containing how many elements are present.

length[A]

- square brackets are used for both array indexing and object attributes (it will be clear from the context which interpretation is intended)

8- Parameters are passed by value. (call by reference is not supported)

9 The boolean operators "and" and "or" are short circuiting

Analyzing Algorithms

- Before analyzing an algorithm, we must have a model of computation that will be used to implement own algorithms

- Here, we will use a generic one processor, random-access machine (RAM) model as our implemention technology.

- In this RAM model, instructions are executed one after another with no concurrent operations.

- One should precisely define* all the instructions of the RAM model and their costs. The RAM model should be realistic², therefore, its design is influenced from real computers' working.

*
- tedious task
- allow small focus on ADA (while it must focus on ADA)

overall, we have to maintain the level of abstraction

- RAM model contains instructions commonly found in real computers:
  - Arithmetic (add, substract, multiply, divide, remainder, floor, ceiling)
  - Data Movement (load, store, copy)
  - Control (conditional & unconditional branch, subroutine call and return)

** sort in just one instruction

- Each instruction takes a constant amount of time.
- The datatypes in RAM model are integer and floating point (with fixed size)
- Real computers contain instructions* that are not available our RAM model) these instructions represent a gray area in the RAM model. We will try to avoid such gray areas in the RAM model.
*(such as exponentiation)
- RAM model neglected to consider the memory hierarchy which is commonly present in real computers.
  ⇒ it doesn't model caches/virtual memory.
  ⇒ to remove complexity

Note: 1. It should be noted that the analysis (of an algo) predicted by the RAM model, usually depicts nearly same performance on actual machines.

Scanned with CamScanner

2. Analyzing a simple algo in the RAM model can be a challenging task and may involve various mathematical tools such as
   - combinatorics
   - Probability theory
   - Algebra ..

Because the behaviour of an algo may be different for each possible input. But our motive is to summarize that behaviour in simple & easily understood formulas.

3.

Insertion Sort

- solves the sorting problem

  Input: A sequence of n numbers $\langle a_1, a_2, a_3 \cdots a_n \rangle$.

  Output: A permutation (reordering) of the input sequence

  i.e. $\langle a_1', a_2' \cdots a_n' \rangle$ such that

  $$a_1' \leq a_2' \leq \cdots a_n'$$

- it works the way many people sort a hand of playing cards.

- Analogy Description : Refer book.

INSERTION SORT(A)

```
for j ← 2 to length[A]
    do key ← A[j]
        ▷ Insert A[j] into the sorted sequence A[1····j-1]
        i ← j-1
        while i>0 and A[i] > key
            do A[i+1] ← A[i]
               i ← i-1
        A [i+1] ← key
```

Sorted in place

1) rearranged within the array and
2) at any time, at most a constant number of items are stored outside the array.

Loop invariants

j : current card being inserted into the hand.

At the beginning of each iteration of the outer for loop,
the subarray A[1..j-1] :- currently sorted hand
and A[j+1 .. n] :- pile of cards still on the table

⇒ In fact, elements A[1..j-1] are the elements originally in positions 1 through j-1, but now in sorted order.

⇒ these properties of A[1..j-1] may be used to define as a loop invariant

" At the start of each iteration of the for loop of lines 1-8, the subarray A[1..j-1] consists of the elements originally in A[1..j-1] but in sorted order. "

- Loop invariant is used to understand the correctness of an algorithm

- we must show 3 things about a loop invarient: