In this course, we are primarily interested in the design of good algorithms and data structures.

Algorithms : An algorithm is defined as the step-by-step procedure for solving a problem in finite number of well-defined steps.

* An algorithm is designed independent of any programming language.

* Expressed in natural language or graphically using flowchart or using pseudo code.

Program ⟷ Analogue ⟷ Algorithm

Programming Language ⟷ Analogue ⟷ Pseudo Code

(Ref:- MIT OCW 6.006)

Properties of an Algorithm

- Finiteness (an algorithm must terminate in finite no. of steps)

- Unambiguity or Definiteness (each step and order of steps must be defined unambiguously)

- Input (zero or more inputs)

- Output (one or more output)

- Effectiveness (each step must be sufficiently basic and doable in finite time)

Desirable Characteristics

1) Generality : Algorithm should be applicable to several cases.

2) Algorithms should use resources efficiently

* Time → (more important)
* Space

So, when I talk about "good" algorithms

I mean efficient algorithm.

↓

Consumes less resources.

Efficiency of an Algorithm
(Algorithmic Efficiency)

An algorithm must be analyzed to determine its resource requirements.

Efficiency of an algorithm depends upon its use of resources such as

* Time needed to produce desired output
* Space (memory) requirements.
* Network traffic it generates (if applicable)
* etc.

In this course, for the sake of brevity we concentrate only on two factors Time and Space. In fact, we are more concerned about the "time".

Let us suppose you want to find the sum of first 10 natural numbers

Stage 1 :- Describing the Problem

You have to find the sum of

$1+2+3+\cdots+10 = ?$ $\left(\sum_{i=1}^{10} i = ?\right)$

Stage 2 :- Identifying a suitable technique.

Way 1 :- Take the cumulative sum up to 10 starting from 1.

Way 2 :- Use the formula

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Stage 3 :- Design of an Algorithm

Way 1 :-

```
Sum := 0
for i = 1 to 10
    sum := sum + i
endfor
return sum
```

Way 2 :

```
n := 10
return n(n+1)/2
```

Stage 4 :- Proof of Correctness of an Algorithm

Correctness of an Algorithm

Qus. How do you assert that the algorithm you designed is actually correct?

Naive Ways

1. Getting some confidence by doing some logical analysis.

2. Implementing the algorithm (in some programming language), and carrying out check for different sets of input values.

But, this testing can never be exhaustive.

So,

We always attempt to provide a formal mathematical proof to assert that the algorithm you designed, indeed give desirable output for valid input values.

Hence, an Algorithm must be supported with the valid correctness Proof.

Way 1 :- Trivial

Way 2 :- Proof.

$$a,\ a+d,\ a+2d,\ \ldots,\ a+(n-1)d$$

$$S = a+(a+d)+(a+2d)+\ldots+(a+(n-1)d)$$

$$S = \frac{n}{2}\left(a+a+(n-1)d\right) = \frac{n}{2}\left(2a+(n-1)d\right)$$

Your sequence of Interest

$$1,\ 2,\ \ldots,\ n$$

$$a = 1,\ d = 1$$

$$S = \frac{n}{2}\left(2(1)+(n-1)1\right)$$

$$S = \frac{n}{2}\left(2+n-1\right)$$

$$S = \frac{n(n+1)}{2}$$

One can use other proof techniques like Principle of Mathematical Induction.

Once, correctness is established, we talk about the performance of the algorithm or Algorithmic efficiency.

However, it should be noted we have not made any assumptions or requirement of any specific hardware while designing the algorithm.

So, our performance evaluation must be done independent of any programming language and must not assume availability of any specific hardware requirement.

5. Performance Evaluation

Way 1 :- 21 Assignments (10 for i = i++) + 20 Additions (10 for i)

Roughly, $20C_1 + 21C_2$ (Cost/Time)

Way 2 :-

1	multiplication	C_3
1	addition	C_1
1	division	C_4
1	Assignment	C_2

Important :-

If the problem is changed to "finding the sum of first 10000 natural numbers"

Way 1 :- $2 \times 10000 (C_1 + C_2) + C_2$ → depends on I/p size

Way 2 :-

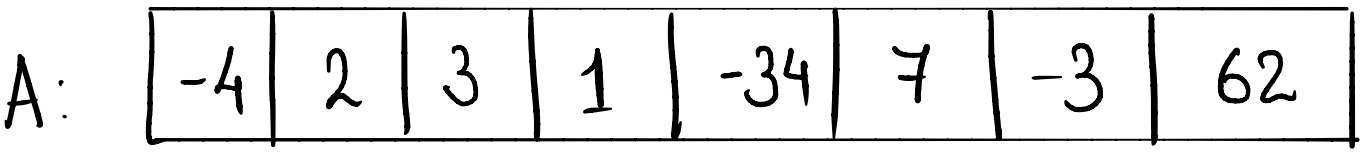
1	multiplication	
1	addition	
1	division	
1	Assignment	

→ $C_1 + C_2 + C_3 + C_4$ Independent on I/p size.

Problem 2:

Finding the smallest element in the given array → (data Structure)

→ Ordered Collection of items accessible by an index



Given, an Array A of integers of size n.

We have to find minimum element in an array.

(Stage 1 : Describing the Problem)

Stage 2:- Identifying a suitable technique.

We have to compare each element of the array to find minimum element.

Stage 3:- Design an Algorithm

(Step1) Given an array A[1...n] of integers.

(Step2) min := infinity

(Step3) for i = 1 to n

```
        if A[i] < min        → Comparison
            min := A[i]
        endif
    endfor
```

Stage 4 : Proof of Correctness

Proof By Induction :-

Base Case :- If $n = 1$

Clearly $A[1] < \text{infinity}$

Hence, algorithm will return $A[1]$ which is a single finite quantity in an array and therefore it is the minimum element.

Hypothesis :- Let the algorithm returns the minimum element (min_k) in first k entries of the array. ($P(k)$)

Then,

$$min_k = \text{minimum}\{A[1], \ldots, A[k]\}$$

Now,
Let's include one more element $A[k+1]$ then,

(See Step 3 of Algorithm)

$$\text{minimum}\{\min_k, A[k+1]\} = \begin{cases} \min_k, & \text{if } A[k+1] \geq \min_k \\ A[k+1], & \text{if } A[k+1] < \min_k \end{cases}$$

Hence, $P(k) \Rightarrow P(k+1)$

Hence Proved.

(Proof By Induction ↑)

More Simpler.

Proof by Contradiction

(next page)

(By Contradiction)

Let the algorithm returns "m" which is not the minimum among the all elements in the array.

This means

There exists ($\exists$) an index i ($i \in \{1, \ldots n\}$)

such that

$$A[i] < m$$

However, there is a contradiction as in step 3 after each iteration, we are checking the above condition and if it satisfies, we are assigning $min := A[i]$.

Hence proved //

Stage 5 : Performance Evaluation

(Homework)

Worst-Case Time Complexity

Best-Case Time Complexity.