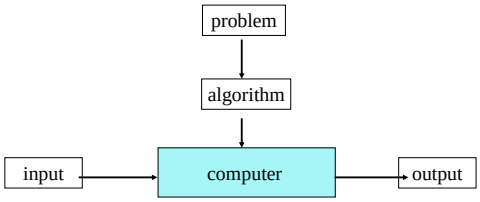
Algorithms and Problem Solving (15B11CI411)

Computer Era

- **Pre Computer Era**
  - Focus was on Computability Theory (What can be computed and what cannot be computed?)
- **Post Computer Era**
  - Focus is on Complexity Theory (How well can it be computed?)

What is an algorithm?

- An algorithm is a sequence of unambiguous instructions for solving a problem,
  i.e., for obtaining a required output for any legitimate input in a finite amount of time.



Algorithm Properties

An algorithm must have following properties:

1. **Finiteness:** terminates after a finite number of steps
2. **Definiteness:** thoroughly and unambiguously specified
3. **Input:** valid inputs are clearly specified
4. **Output:** can be proved to produce the correct output given a valid input
5. **Effectiveness:** steps are sufficiently simple and basic

Some Well-known Computational Problems

- Sorting
- Searching
- Shortest path
- Minimum spanning tree
- Traveling salesman problem
- Knapsack problem
- Chess
- Towers of Hanoi.

Why study algorithms and performance?

- **Algorithms** help us to understand **scalability**.
- **Algorithmic** mathematics provides a **language** for talking about **program behavior**.
- **Performance** is the ***currency of computing***
- **Performance** often **draws the line** between what is feasible and what is impossible.

Basic Issues Related to Algorithms

- How to design algorithms
- How to express algorithms
- How to analyze algorithm efficiency?
  - Theoretical analysis
  - Empirical (experimental) analysis
- Proving correctness
- Optimality

Analysis of Algorithms

- How good is the algorithm?
  - Correctness
  - **Time efficiency**
  - Space efficiency
- Does there exist a better algorithm?

Efficiency Measures

- Performance of a solution
- Most of the software problems do not have a single best solution.
- Then how do we judge these solutions?

Efficiency Measures (Space Time Tradeoff)

Example : Think of a GUI drop-down list box that displays a list of employees whose names begin with a specified sequence of characters. If the employee database is on a different machine, then there are two options:

- **Option a: fire a SQL and retrieve the relevant employee names each time the list is dropped down.**
- **Option b: keep the complete list of employees in memory and refer to it each time the list is dropped down.**
- **In your opinion which is the preferred option and why?**

Time Space Tradeoff

- This example does not have a unique solution. It depends on various parameters which include:
  - The number of employees
  - The transmission time from the database server to the client machine
  - The volume of data transmission each time
  - The frequency of such requests.
  - The network bandwidth
- Very difficult to **get a solution** which is **optimal in terms of both the time and space**.

Kinds of Analysis

- **Worst-case:** (usually)
  - T(n) = maximum time of algorithm on any input of size n.
- **Average-case:** (sometimes)
  - (n) = expected time of algorithm over all inputs of size n.
  - Need assumption of statistical distribution of inputs.
- **Best-case**
  - Cheat with a slow algorithm that works fast on some input.

Performance

More important than performance:

- modularity
- correctness
- maintainability
- functionality
- robustness
- user-friendliness
- programmer time
- simplicity
- extensibility
- reliability

Improving the performance of a solution

By improving the

- algorithm design,
- database design,
- transaction design,
- paying attention to the end-user psychology,
- continuous improvements in hardware and communication infrastructure.

Fundamental data structures

- List :array, linked list, string
- stack
- queue
- priority queue
- graph
- tree
- set and dictionary

Some of the Algorithm Design techniques

- Brute force
- Divide and conquer
- Greedy approach
- Dynamic programming
- Backtracking
- Space and time tradeoffs