ALGORITHMS

INTRODUCTION

What is an algorithm?

- Set of rules required to perform calculations or problem-solving operations.

Characteristics:

- Input = Some input values
- Output = One or more output values
- Unambiguity = Instructions are clear and simple
- Finiteness = Contain limited number of instructions
- Effectiveness = Should be effective as each instruction affects the overall process
- Language independent = So it can be implemented in other languages with same output

Dataflow:

- Problem = Problem that needs solving
- Algorithm = Designed with a step-by-step procedure
- Input = Required inputs are provided
- Processing unit = Input given to processing unit and is processed
- Output = Output is outcome of the program

Factors to consider:

- Modularity = Can problem be broken down into small modules?
- Correctness = Does the algorithms give the desired output?
- Maintainability = Is It designed so when we redefine the algorithm no major change will occur.
- Functionality = Considers logical steps to solve the problem
- Robustness = How the algorithm clearly defines our problem
- User-friendly = Must be able to explain it easily and clearly
- Simplicity = Easy to understand
- Extensibility = Extensible for other designers to use

Approaches of Algorithms

- **Brute force**: General logic structure applied to design algorithm. Exhaustive search algorithm. Two types: Optimising (find all solutions and take the best one) and sacrificing (as soon as best solution is found it stops)
- **Divide and conquer**: Allows to design in step-by-step variation. Breaks down algorithm to solve problem in different methods. Valid output produced for valid input, and this is passed to some other function.
- **Greedy**: Makes optimal choice on each iteration with hope of getting best solution. Easy to implement and fast execution time, rarely provides optimal solution.
- **Dynamic programming**: 1. Breaks problem into subproblem and finds optimal solution. 2. After breaking down finds optimal solution for these subproblems. 3. Stores results of subproblems (memorisation). 4. Reuse results so they are not recomputed for same subproblems. 5. Computes result of complex problem.
- **Branch and Bound**: Applied to integer programming. Divides all sets of feasible solutions to subsets and subsets are evaluated to find best solution.
- **Randomised**: Randomised processes are added to the input to create an output that is random in nature.
- **Backtracking**: Solves problem recursively and removes solution if does not satisfy constraints of problem.

Complexity:

- Time complexity = Amount of time required to complete execution. Denoted by big O notation. Calculated by counting number of steps to finish execution.
- Space complexity = Amount of space in memory required to solve a problem and produce an output. Auxiliary space is extra space required excluding input size. Space complexity = Auxiliary space + Input size.

Types of Algorithms:

- Search: Linear vs Binary. Linear = iterates through array until element is found, can be implemented on unsorted list (O(n) complexity). Binary = Uses middle element to find element, list must be sorted (O(log(n) complexity)
- Sort: Rearrange elements in array or structure in ascending or descending order. Sorted structures are more efficient when applying other algorithms and more easily readable.