Catanduanes State University
College of Information and Communications Technology
Virac, Catanduanes

LEARNING MATERIALS and COMPILATION OF LECTURES/ACTIVITIES

Lesson 1	**Algorithm Design Techniques**
Module 2	Algorithm
Course	Data Structure and Algorithms

DISCLAIMER

This learning material is used in compliance with the flexible teaching-learning approach espoused by CHED in response to the pandemic that has globally affected educational institutions. Authors and publishers of the contents are well acknowledged. As such the college and its faculty do not claim ownership of all sourced information. This learning material will solely be used for instructional purposes not for commercialization

CatSU – College of Information and Communications Technology

Overview

This learning module of “Algorithm” has been written according to the approved syllabus approved by the CICT Dean and CSU.

Data Structures is a middle module in the curriculum of almost every computer science program. This subject makes the students learn the art of analyzing algorithms as well as recognizing between the detail of a data structure and it understands within an available programming language.

This learning module endeavors in a basic and clear language with neat and self-explanatory diagrams, which could be simply understood by an average student.

The concise content of this learning module is as follows:

Lesson 1 introduces what is an Algorithm

Lesson 2 deals with Algorithm Design Techniques

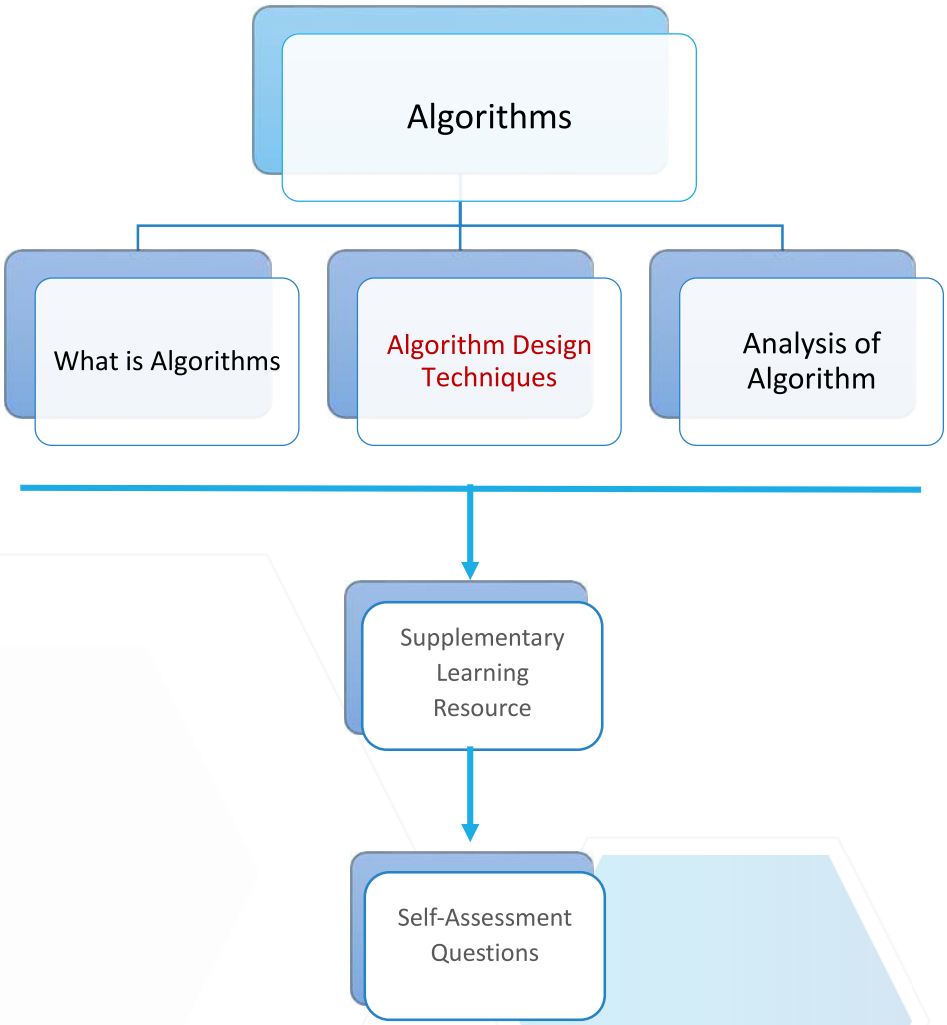
Lesson 3 covers Analysis of Algorithm

Learning Outcomes

At the end of the course, the students shall be able to:

1. Explain the concept of algorithms and design and techniques
2. Discuss the different types, space and time of algorithms

Module Map


Algorithms
What is Algorithms
Algorithm Design Techniques
Analysis of Algorithm
Supplementary Learning Resource
Self-Assessment Questions

Definition of Key Terms/Unlocking of Difficulties

Algorithm	An algorithm is the step-by-step unambiguous instructions to solve a given problem
Stepwise Refinement Techniques	Is the most effective way to solve a complex problem is to break it down into successively simpler sub-problems.
Modular Programming	is the act of designing and writing programs as functions that each one performs a single well-defined function, and which have minimal interaction between them

Modules 2 Algorithms

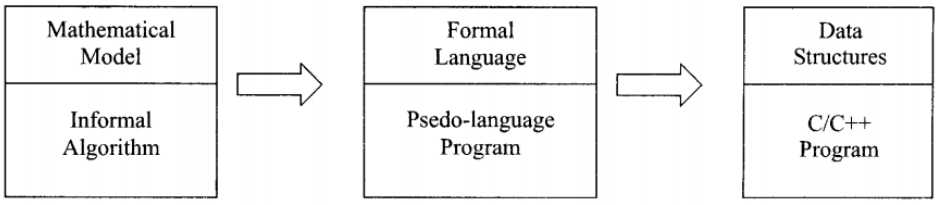
Lesson 2 Algorithm Design Technique

Data structure is a particular way of storing and organizing data so that it can be used efficiently. Arrays, trees, linked lists, stacks, graphs, etc. are all data structures and each of them allows us to perform different operations on data.

No matter which programming language you use for programming, it is important to learn algorithm design techniques in data structures in order to be able to build scalable systems.

Stepwise Refinement Techniques

We can write an informal algorithm, if we have an appropriate mathematical model for a problem. The initial version of the algorithm will contain general statements, i.e., informal instructions. Then we convert this informal algorithm to formal algorithm, that is, more definite instructions by applying any programming language syntax and semantics partially. Finally a program can be developed by converting the formal algorithm by a programming language manual. From the above discussion we have understood that there are several steps to reach a program from a mathematical model. In every step there is a refinement (or conversion).That is to convert an informal algorithm to a program, we must go through several stages of formalization until we arrive at a program — whose meaning is formally defined by a programming language manual — is called stepwise refinement techniques. There are three steps in refinement process, which is illustrated in Figure below



1. In the first stage, modeling, we try to represent the problem using an appropriate mathematical model such as a graph, tree etc. At this stage, the solution to the problem is an algorithm expressed very informally.
2. At the next stage, the algorithm is written in pseudo-language (or formal algorithm) that is, a mixture of any programming language constructs and less formal English statements. The operations to be performed on the various types of data become fixed.
3. In the final stage we choose an implementation for each abstract data type and write the procedures for the various operations on that type. The remaining informal

statements in the pseudo-language algorithm are replaced by (or any programming language) C/C++ code.

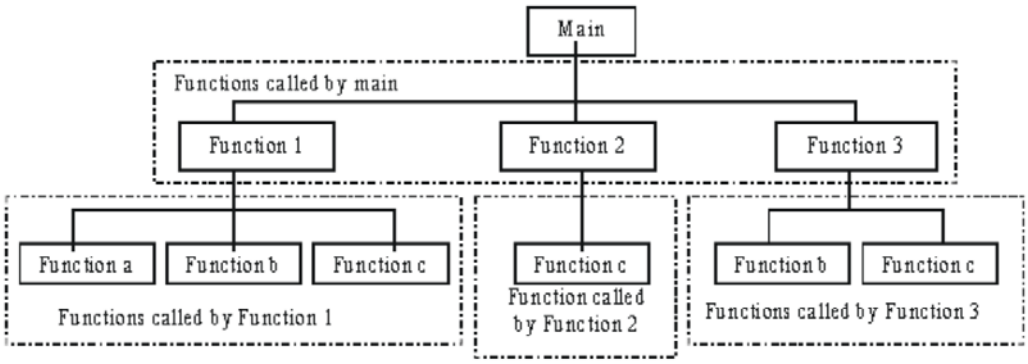
Modular Programming

Modular Programming is heavily procedural. The focus is entirely on writing code (functions). Any code may access the contents of any data structure passed to it. Modular Programming is the act of designing and writing programs as functions that each one performs a single well-defined function, and which have minimal interaction between them. That is, the content of each function is cohesive, and there is low coupling between functions

Two methods may be used for modular programming. They are known as top-down and bottom-up, regardless of whether the top-down or bottom-up method is used, and the end result is a modular program. This end result is important, because not all errors may be detected at the time of the initial testing.

1. Top-down Algorithm Design

The principles of top-down design dictates that a program should be divided into a main module and its related modules. Each module should also be divided into sub modules according to software engineering and programming style. The division of modules processes until the module consists only of elementary process that are intrinsically understood and cannot be further subdivided.



2. Bottom-up Algorithm Design

Bottom-up algorithm design is the **opposite** of top-down design. It refers to a style of programming where an application is constructed starting with existing primitives of the programming language, and constructing gradually more and more complicated features, until the all of the application has been written. That is, **starting the design with specific modules and build them into more complex structures, ending at the top.**

The bottom-up method is widely used for testing, because each of the lowest-level functions is written and tested first. This testing is done by special test functions that call the low-level functions, providing them with different parameters and examining the

results for correctness. Once lowest-level functions have been tested and verified to be correct, the next level of functions may be tested. Since the lowest-level functions already have been tested, any detected errors are probably due to the higher-level functions. This process continues, moving up the levels, until finally the main function is tested.

Other Design Algorithm

There are many ways to design algorithms for it, e.g. insertion sort is an incremental algorithm, merge sort is a divide and conquer approach. The following is the list of several popular design approaches

1. **Divide and Conquer Approach:** It is a top-down approach, and the examples are Merge sort and Quicksort. The algorithms which follow the divide & conquer techniques involve three steps:

    - Divide the original problem into a set of sub-problems.
    - Solve every sub-problem individually, recursively.
    - Combine the solution of the sub-problems (top level) into a solution of the whole original problem.

2. **Greedy Technique:** Greedy method is used to solve the optimization problem. An optimization problem is one in which we are given a set of input values, which are required either to be maximized or minimized (known as objective), i.e. some constraints or conditions. **Example:** Fractional Knapsack, Activity Selection.

    - Greedy Algorithm always makes the choice (greedy criteria) looks best at the moment, to optimize a given objective.
    - The greedy algorithm doesn't always guarantee the optimal solution however it generally produces a solution that is very close in value to the optimal.

3. **Dynamic Programming:** Dynamic Programming is a bottom-up approach we solve all possible small problems and then combine them to obtain solutions for bigger problems. This is particularly helpful when the number of copying sub-problems is exponentially large. Dynamic Programming is frequently related to Optimization Problems. **Example**: 0-1 Knapsack, subset-sum problem.

4. **Branch and Bound:** In Branch & Bound algorithm a given sub-problem, which cannot be bounded, has to be divided into at least two new restricted sub-problems. Branch and Bound algorithm are methods for global optimization in non-convex problems. Branch and Bound algorithms can be slow, however in the worst case they require effort that grows exponentially with problem size, but in some cases we are lucky, and the method coverage with much less effort.

5. **Randomized Algorithms:** A randomized algorithm is defined as an algorithm that is allowed to access a source of independent, unbiased random bits, and it is then

allowed to use these random bits to influence its computation. One example is Randomized Quicksort Algorithm.

6. **Backtracking Algorithm:** Backtracking Algorithm tries each possibility until they find the right one. It is a depth-first search of the set of possible solution. During the search, if an alternative doesn't work, then backtrack to the choice point, the place which presented different alternatives, and tries the next alternative.

7. **Randomized Algorithm:** A randomized algorithm uses a random number at least once during the computation make a decision.

A given problem can be solved in various different approaches and some approaches deliver much more efficient results than others. Algorithm analysis is a technique used to measure the effectiveness and performance of the algorithms. It helps to determine the quality of an algorithm based on several parameters such as user-friendliness, maintainability, security, space usage and usage of other resources.

Structured Programming

It is a programming style; and this style of programming is known by several names: Procedural decomposition, Structured programming, etc. Structured programming is not programming with structures.

Assignment 2.2

- Differentiate top-down and bottom-up algorithm design

Supplementary Learning Resource:

- Read Jyoti Signh (2019, January 11) article, “Top-Down and Bottom-Up Design Approach”, https://www.embhack.com/top-down-and-bottom-up-design-approach/

Synthesis

Algorithm is a set of instructions used to solve a specific problem, it is a finite sequence of well-defined, computer implementable instructions, typically to solve a class of problems or to perform a computation. An algorithm must have the following characteristics it must be; unambiguous, input, output, finiteness, feasibility, independent. Keep in mind that there is no well-defined standard in writing an algorithm for it is a problem and resource dependent.

Before writing an algorithm make sure that the problem domain is well defined. In writing the algorithm you may two column table on which have steps and procedure. Take note that we design algorithm to get a solution to a given problem. However, just keep in mind that a problem can be solve in more than one way.

No matter which programming language you use for programming, it is important to learn algorithm design techniques in data structures in order to be able to build scalable systems.

In theoretical analysis of algorithms, you’ve learned that there is computational complexity in algorithms. Such as time and space complexity, this provides the theoretical estimation required resources of an algorithm to solve a specific computational problem. The efficiency of algorithm depends on its running time. In time complexity it affects the number of steps of algorithm before it completes its execution. In space complexity this relates to the volume or memory resources needed by the algorithms to complete its execution

References

Agrawal, U. (2018). Data Structures Using C. S.K. Kataria and Sons, New Delhi.

Cormen, Thomas H., et al. (2015) Introduction to Algorithms 3rd edition

Karumanchi, Narasimba, (2017). Algorithms Made Easy – Data Structure and Algorithms Puzzles 5th Edition, Career Monk Publishing

Lafore, Robert, Data Structures and Algorithms2nd edition, Sams Publishing, USA

Waite, M., Lafore, R. (2017) Data Structures and Algorithms 2nd edition, Sams Publishing, USA