Applied Computer Science: ITI 2100

DATA STRUCTURE AND ALGORITHMS

Harrison Njoroge
Foreword

The African Virtual University (AVU) is proud to participate in increasing access to education in African countries through the production of quality learning materials. We are also proud to contribute to global knowledge as our Open Educational Resources are mostly accessed from outside the African continent.

This module was developed as part of a diploma and degree program in Applied Computer Science, in collaboration with 18 African partner institutions from 16 countries. A total of 156 modules were developed or translated to ensure availability in English, French and Portuguese. These modules have also been made available as open education resources (OER) on oer.avu.org.

On behalf of the African Virtual University and our patron, our partner institutions, the African Development Bank, I invite you to use this module in your institution, for your own education, to share it as widely as possible and to participate actively in the AVU communities of practice of your interest. We are committed to be on the frontline of developing and sharing Open Educational Resources.

The African Virtual University (AVU) is a Pan African Intergovernmental Organization established by charter with the mandate of significantly increasing access to quality higher education and training through the innovative use of information communication technologies. A Charter, establishing the AVU as an Intergovernmental Organization, has been signed so far by nineteen (19) African Governments - Kenya, Senegal, Mauritania, Mali, Cote d’Ivoire, Tanzania, Mozambique, Democratic Republic of Congo, Benin, Ghana, Republic of Guinea, Burkina Faso, Niger, South Sudan, Sudan, The Gambia, Guinea-Bissau, Ethiopia and Cape Verde.

The following institutions participated in the Applied Computer Science Program: (1) Université d’Abomey Calavi in Benin; (2) Université de Ouagadougou in Burkina Faso; (3) Université Lumière de Bujumbura in Burundi; (4) Université de Douala in Cameroon; (5) Université de Nouakchott in Mauritania; (6) Université Gaston Berger in Senegal; (7) Université des Sciences, des Techniques et Technologies de Bamako in Mali (8) Ghana Institute of Management and Public Administration; (9) Kwame Nkrumah University of Science and Technology in Ghana; (10) Kenyatta University in Kenya; (11) Egerton University in Kenya; (12) Addis Ababa University in Ethiopia (13) University of Rwanda; (14) University of Dar es Salaam in Tanzania; (15) Université Abdou Moumouni de Niamey in Niger; (16) Université Cheikh Anta Diop in Senegal; (17) Universidade Pedagógica in Mozambique; and (18) The University of the Gambia in The Gambia.

Bakary Diallo

The Rector

African Virtual University
Production Credits

Author
Harrison Njoroge

Peer Reviewer
Victor Odumuyiwa

AVU - Academic Coordination
Dr. Marilena Cabral

Overall Coordinator Applied Computer Science Program
Prof Tim Mwololo Waema

Module Coordinator
Jules Degila

Instructional Designers
Elizabeth Mbasu
Benta Ochola
Diana Tuel

Media Team
Sidney McGregor     Michal Abigael Koyier
Barry Savala        Mercy Tabi Ojwang
Edwin Kiprono       Josiah Mutsogu
Kelvin Muriithi     Kefa Murimi
Victor Oluoch Otieno Gerisson Mulongo
# Table of Contents

- **Foreword**  
  Page 2
- **Production Credits**  
  Page 3
- **Copyright Notice**  
  Page 4
- **Course Overview**  
  Page 9
  - Welcome to Data Structures and Algorithm  9
  - Prerequisites  9
  - Materials  9
  - Course Goals  10
  - Course Goals  10
- **Units**  
  Page 10
  - Assessment  11
  - Schedule  12
- **Unit 0. Pre-Assessment**  
  Page 15
  - Unit Introduction  15
  - Unit Objectives  15
  - Key Terms  15
  - Class Assignment  16
  - Instructions  16
  - Grading Scheme  16
  - Unit Readings and Other Resources  18
- **Unit 1. Algorithmic problem solving**  
  Page 19
  - Unit Introduction  19
  - Unit Objectives  19
  - Learning Activities  19
  - Activity 1 - Introduction to Algorithms and problem solving  19
  - Key Terms  19
  - Algorithm  20
  - Algorithm:  21
<table>
<thead>
<tr>
<th>Conclusion</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity 2 - The characteristics of an algorithm</td>
<td>21</td>
</tr>
<tr>
<td>Introduction</td>
<td>21</td>
</tr>
<tr>
<td>Characteristics of an Algorithm</td>
<td>21</td>
</tr>
<tr>
<td>Assessment</td>
<td>21</td>
</tr>
<tr>
<td>Assessment</td>
<td>22</td>
</tr>
<tr>
<td>Conclusion</td>
<td>22</td>
</tr>
<tr>
<td>Activity 3 - Using pseudo-codes and flowcharts to represent algorithms</td>
<td>22</td>
</tr>
<tr>
<td>Introduction</td>
<td>22</td>
</tr>
<tr>
<td>Pseudo Code.</td>
<td>22</td>
</tr>
<tr>
<td>Example</td>
<td>23</td>
</tr>
<tr>
<td>Solution</td>
<td>23</td>
</tr>
<tr>
<td>Flow chart</td>
<td>23</td>
</tr>
<tr>
<td>Flowchart building blocks</td>
<td>23</td>
</tr>
<tr>
<td>PSEUDO CODE</td>
<td>25</td>
</tr>
<tr>
<td>Grading Scheme</td>
<td>27</td>
</tr>
<tr>
<td>Assessment.</td>
<td>27</td>
</tr>
</tbody>
</table>

### Unit Assessment

- Feedback                         | 28 |
- Unit Readings and Other Resources | 28 |

### Unit 2. RECURSION

- Unit Introduction                 | 29 |
- Unit Objectives                   | 29 |
- Learning Activities               | 29 |
- Activity 1 - Recursive Algorithm  | 29 |
- Key Terms                         | 29 |
- Laws of Recursion                 | 30 |
- Assessment                        | 32 |
Grading Scheme ........................................... 53
Assessment .............................................. 53
UNIT SUMMARY ........................................... 53
Unit Assessment ......................................... 53
Course Assessment ....................................... 54
Feedback .................................................. 54
Unit Readings and Other Resources .................... 54
Grading Scheme ........................................... 55
Feedback .................................................. 55
Final Examination ....................................... 56
Grading Scheme ........................................... 56
Feedback .................................................. 57
Insertion of elements is at the end of the queue ........ 58
Course References ....................................... 59
Course Overview

Welcome to Data Structures and Algorithm

“Every program depends on algorithms and data structures, but few programs depend on the invention of brand new ones.”! “I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”!

This course introduces students to data structures and algorithms, and how data structures can be created and used. The application of data structures will be presented on the basis of various algorithms. This course teaches learners the way information is organized in the computer and which have an impact on the performance of the computer. The purpose of this course is to provide the students with solid foundations in the basic concepts of programming: data structures and algorithms. The main objective of the course is to teach the students how to select and design data structures and algorithms that are appropriate for problems that they might encounter. This course is also about showing the correctness of algorithms and studying their computational complexities. This course offers the students a mixture of theoretical knowledge and practical experience. The study of data structures and algorithms is carried out within an object-oriented framework. When implementations are considered, the Java programming language is used.

Prerequisites

- Introduction to computer science
- Principles of programming

Materials

The materials required to complete this course are:

Textbooks on algorithms, including:

- SAMS. (1999). Teach Yourself Data Structures And Algorithms In 24 hours.
- Adam Drozdek. Data Structures and Algorithms in Java, SECOND EDITION.
- Internet
- Java
- Lecture notes
Course Goals

Upon completion of this course the learner should be able to:

- Identify the major classical and modern AI paradigms,
- Analyze the concepts and techniques of Artificial Intelligence.
- Learning AI by developing skills of using AI algorithms for solving practical problems. Analyze the structure of a given problem such that they can choose an appropriate method implement AI algorithms. To appreciate an intelligent machine which is capable of reasoning, planning, solving problems, thinking abstractly, comprehending complex ideas, learning quickly and learning from experience.

The notes are intended to enable the learner learn the data structure course. They are placed in the respective areas of each unit in the module. The notes may be supplemented by further reading of the provided materials or resources as indicated in the relevant sections. The web-links can also provide more information related to the indicated sections.

Course Goals

Upon completion of this course the learner should be able to;

- explain the fundamental of data structures
- discuss the importance of algorithms in problem solving
- determine which data structures are most effective for various scenarios and problems
- create, data structures
- apply the appropriate data structure for modeling a given problem

Units

Unit 0: introduction to basic computing and programming

This unit is aimed at students with little or no programming experience. It aims to provide students with an understanding of the role computation can play in solving problems. It also aims to help students, regardless of their major, to feel justifiably confident of their ability to write small programs that allow them to accomplish useful goals.

Unit 1: Algorithmic Problem Solving; introduction to algorithms and problem solving, characteristics of algorithm,

This unit provides an introduction to mathematical modeling of computational problems. It covers the common algorithms, algorithmic paradigms, and data structures used to solve these problems. The course emphasizes the relationship between algorithms and programming, and introduces basic performance measures and analysis techniques for these problems.

using pseudo code and flowcharts to represent algorithms
Unit 2: Recursion; Recursive algorithm

This unit entails solving problems with recursive algorithms, computing functions with recursive algorithms, and checking set membership with recursive algorithms.

Unit 3: Basic Data structures and Abstract data types; arrays, lists, linked lists; stack, queues, hashing and trees

Set of data values and associated operations that are precisely specified independent of any particular implementation.

Unit 4: Searching and sorting algorithms

Two for each of the following common tasks:

- sorting: ordering a list of values
- searching: finding the position of a value within a list

Assessment

Formative assessments, used to check learner progress, are included in each unit.

Summative assessments, such as final tests and assignments, are provided at the end of each module and cover knowledge and skills from the entire module.

Summative assessments are administered at the discretion of the institution offering the course. The suggested assessment plan is as follows:

<table>
<thead>
<tr>
<th></th>
<th>Class assignments</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>10%</td>
</tr>
<tr>
<td>2</td>
<td>CATS</td>
<td>20%</td>
</tr>
<tr>
<td>3</td>
<td>End of semester exams</td>
<td>70%</td>
</tr>
</tbody>
</table>
## Schedule

<table>
<thead>
<tr>
<th>Unit</th>
<th>Activities</th>
<th>Estimated time</th>
</tr>
</thead>
</table>
| 0    | Preview of the pre-requisite  
  1. Introduction to computer science  
  2. Principles of programming | 3 hours |
| 1    | Algorithmic problem solving  
  • Activity 1.1 - Introduction to Algorithms and problem solving  
  • Activity 1.2 - The characteristics of an algorithm  
  • Activity 1.3 - Using pseudo-codes and flowcharts to represent algorithms | 6 hours |
| 2    | Recursion  
  • Activity 2.1 - Recursive Algorithm | 6 hours |
| 3    | Basic Data structures and Abstract data types  
  • Activity 3.1 - Data Structures  
  • Activity 3.2 – Operations of Lists  
  • Activity 3.3 - Abstract Data Type | 15 hours |
Course Overview

4 Searching and Sorting Algorithms
• Activity 4.1 - Searching Algorithm
• Activity 4.2 - Sorting algorithm
9 hours

Readings and Other Resources

The readings and other resources in this course are:

Unit 0

Required readings and other resources:

• Paolo Coletti. (2014). Basic Computer course book, 8TH Edition:

Unit 1

Required readings and other resources:

• SAMS. (1999). Teach Yourself Data Structures And Algorithms In 24 hours.
• Christopher Fox. (2012). Concise Notes on Data Structures and Algorithms,
• Ruby Edition. Adam Drozdek, Data Structures and Algorithms in Java, SECOND EDITION

Unit 2

Required readings and other resources:

• SAMS (1999). Teach Yourself Data Structures And Algorithms In 24 hours.
• Adam Drozdek, Data Structures and Algorithms in Java, SECOND EDITION.

Unit 3

Required readings and other resources:

• SAMS (1999).Teach Yourself Data Structures And Algorithms In 24 hours.
• Adam Drozdek, Data Structures and Algorithms in Java, SECOND EDITION.
Unit 4

Required readings and other resources:


- Adam Drozdek, Data Structures and Algorithms in Java, SECOND EDITION.

Optional readings and other resources:

- Robert Sedgewick, (2012).  ALGORITHMS IN C : FUNDAMENTALS, DATA STRUCTURES, SORTING, SEARCHING, PARTS 1-4, ISBN 8131713059; Publisher: Pearson Education/AW Professional,

Unit 0. Pre-Assessment

Unit Introduction

The purpose of this unit is to determine your grasp of prior knowledge related to this course. The unit’s expectation is that the students have prior knowledge on basic computing and principles of programming. A recap of this course/s will prepare and enable the students to have a grasp of Data Structures and Algorithm.

Unit Objectives

Upon completion of this unit you should be able to:

- state the basic skills for computer data
- outline the basic principles of programming
- describe the problem-solving strategies in data structure and algorithm

Key Terms

**System unit:** is the part of a computer that houses the primary devices that perform operations and produce results for complex calculations

**Input:** is a device that feeds data into a computer, such as a keyboard or mouse, is called an input device

**Output:** A result produced by a computer that is internal to the system (from one program or process to another) or external to it (from a program or process to an output device) but internal to an output device (modem, monitor, printer, etc.).

**Storage devices:** device for recording/storing information/data

**Program:** set of instruction that tell the computer what to do or how to solve a given task

**Application Software:** are software products are designed to satisfy a particular need of a particular environment

**Operating systems:** a program that acts as an interface between the software and the computer hardware

**Utility programs:** are programs that perform a specific task related to the management of computer functions, resources, or files, as password protection, memory management, virus protection, and file compression.

**Computer generations:** is a change in technology a computer is/was being used.
Check your understanding!

Class Assignment

Students are to read on basic computers and introduction to programming which are the pre-requisite courses for this module. Students are to also attempt the following questions to test if they can recall what they know and can well prepare them for the data structure and algorithm course

Instructions

The following questions will help gauge how much you already know about basic computing and introduction to programming and which are the pre-requisites of data structure and algorithm

1. Define the following terms
   a. computer,
   b. programming
   c. syntax

2. Briefly describe the following terms
   a. storage devices
   b. software
   c. hardware

3. Distinguish between application and operating system software

Grading Scheme

The marks will be awarded as shown below

<table>
<thead>
<tr>
<th>question</th>
<th>sub-question</th>
<th>marks awarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>(a).</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>(b).</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>(c).</td>
<td>2</td>
</tr>
<tr>
<td>02</td>
<td>a).</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>b).</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>c).</td>
<td>2</td>
</tr>
<tr>
<td>03</td>
<td>each term will be awarded not more than 3 marks</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>18</td>
</tr>
</tbody>
</table>
Feedback

1. Definition of the terms

   (a). computer; an electronic data processing device which
   • accepts and stores data input,
   • processes the data input, and
   • generates the output in a required format.

   (b). programming; is the writing of a set of instructions for completing some specific task

   (c). syntax; the structure and rules (grammar) of a programming language

2. Briefly describe the following

   a). storage devices;
   i. devices for recording (storing) information (data).
   ii. may hold information, process information, or both
   iii. Electronic data storage requires electrical power to store and retrieve that data.
   iv. Electromagnetic data may be stored in either an analog data or digital data format on a variety of media.

   b). software;
   i. is any set of machine-readable instructions that directs a computer’s processor to perform specific operations.
   ii. includes computer programs, libraries and their associated documentation
   iii. is stored in computer memory and cannot be touched i.e. it is intangible
   iv. its categorized as system and application software

   c). hardware
   i. is the physical component of computers
   ii. refers to the physical parts or components of a computer such as the monitor, mouse, keyboard, computer data storage, hard drive disk (HDD), system unit (graphic cards, sound cards, memory, motherboard and chips), etc. all of which are physical objects that can be touched (known as tangible)

3. Distinguishing between application and operating system software
Application system software

- are also called the end-user software or productivity programs as they enable the user to create tasks
- is also a set of one or more programs designed to carry out operations for a specific problem
- examples include MS Word, MS Excel, a library system etc
- Operating system software
- is software that manages computer hardware and software resources and provides common services for computer programs
- It performs basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers
- examples DOS, Windows MAC OS, Unix etc

Unit Readings and Other Resources

The readings in this unit are to be found at the course-level section “Readings and Other Resources.”
Unit 1. Algorithmic problem solving

Unit Introduction

This unit introduces learners to data structures and algorithm course. The unit is on the different data structures and their algorithms that can help implement the different data structures in the computer. The application of the different data structures is presented by using examples of algorithms and which are not confined to a particular computer programming language.

Unit Objectives

Upon completion of this unit the learner should be able to:

• describe an algorithm
• explain the relationship between data and algorithm
• outline the characteristics of algorithms
• apply pseudo codes and flowcharts to represent algorithms

Key Terms

Data: the structural representation of logical relationships between elements of data

Algorithm: finite sequence of steps for accomplishing some computational task

Pseudo code: an informal high-level description of the operating principle of a computer program or other algorithm.

Flow chart: diagrammatic representation illustrates a solution model to a given problem.

Learning Activities

Activity 1 - Introduction to Algorithms and problem solving

Introduction

In this learning activity section, the learner will be introduced to algorithms and how to write algorithms to solve tasks faced by learners or everyday problems. Examples of the algorithm are also provided with a specific application to everyday problems that the learner is familiar with. The learners will particularly learn what is an algorithm, the process of developing a solution for a given task, and finally examples of application of the algorithms are given.
Algorithm

An algorithm is a finite sequence of steps for accomplishing some computational task. It must

- Have steps that are simple and definite enough to be done by a computer, and
- Terminate after finitely many steps.

An algorithm can be considered as a computational procedure that consists of a set of instructions, that takes some value or set of values, as input, and produces some value or set of values, as output, as illustrated in the fig 1 below. It can also be described as a procedure that accepts data, manipulate them following the prescribed steps, so as to eventually fill the required unknown with the desired value(s). The concept of an algorithm is best illustrated by the example of a recipe, although many algorithms are much more complex; algorithms often have steps that repeat (iterate) or require decisions (such as logic or comparison) until the task is completed. Correctly performing an algorithm will not solve a problem if the algorithm is flawed or not appropriate to the problem. A recipe is a set of instructions that show how to prepare or make something, especially a culinary dish.

Different algorithms may complete the same task with a different set of instructions in more or less time, space, or effort than others. Algorithms are essential to the way computers process information, because a computer program is essentially an algorithm that tells the computer what specific steps to perform (in what specific order) in order to carry out a specified task.

Algorithmic problem solving comes in two phases. These includes:

1. derivation of an algorithm that solves the problem, and
2. conversion of the algorithm into code.

It is worth noting that;

1. an algorithm is a sequence of steps, not a program.
2. same algorithm can be used in different programs, or the same algorithm can be expressed in different languages, because an algorithm is an entity that is abstracted from implementation details.

An algorithm can be expressed in following ways;

- a) human language
- b) pseudo code
- c) flow chart

EXAMPLE

Problem: Given a list of positive numbers, return the largest number on the list.

Inputs: A list L of positive numbers. This list must contain at least one number.

Outputs: A number n, which will be the largest number of the list.
Algorithm:

1. Set max to 0.
2. For each number $x$ in the list $L$, compare it to max. If $x$ is larger, set max to $x$.
3. max is now set to the largest number in the list.

Conclusion

The learner was introduced to the concept of algorithm and the various ways he/she can develop a solution to a task. In particular, the learner was introduced to the definition/s of an algorithm, the three main ways of developing or expressing an algorithm which are the human language, Pseudo code and the flow chart. Examples was also given to reinforce the concept of the algorithm.

Assessment

1. Outline the algorithmic steps that can be used to add two given numbers
2. By using an example, describe how the concept of algorithms can be well presented to a group of students being introduced to it.

Activity 2 - The characteristics of an algorithm

Introduction

This section introduces the learners to the characteristics of algorithms. These characteristics makes the learner to become aware of what to ensure is basic, present and mandatory for any algorithm to qualify to be one. It also exposes the learner to what to expect from an algorithm to achieve or indicate. Key expectations are; the fact that an algorithm must be exact, terminate, effective, general among others.

Characteristics of an Algorithm

There following are some key characteristics of an algorithm

1. Each step of an algorithm must be exact; this means that an algorithm must be precise and unambiguously described. This eliminates any uncertainty. it can also be said to the characteristic of precision, i.e. the steps are precisely stated(defined).
2. Algorithm must terminate; since the ultimate aim of an algorithm is to solve a problem, then it must terminate otherwise there wont be a solution for the problem. This leads to the fact that an algorithm must have a finite number of steps in its execution. the presence of endless (infinite) loops must be avoided
3. An algorithm must be effective; this means that an algorithm must provide the correct answers at all times
4. An algorithm must be general; this means that an algorithm must solve every instance of a problem
5. Uniqueness: results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
6. Finiteness: the algorithm stops after a finite number of instructions are executed.
7. Output: the algorithm produces output

Conclusion

This section has highlighted the properties of algorithms to the learner who is new to the concept of data structure and algorithm. It also alerts the learner on what to do to ensure the expected outcomes can be obtained when the algorithm is implemented in any computer programming language. This will ensure the learner comes up with very correct and accurate algorithms for solving any task at hand.

Assessment

1. With the aid of an appropriate example, explain the characteristics of algorithms.

Activity 3 - Using pseudo-codes and flowcharts to represent algorithms

Introduction

The student will learn how to design an algorithm using either a pseudo code or flowchart. Pseudo code is a mixture of English like statements, some mathematical notations and a selected keywords from a programming language. It is one of the tools used to design and develop the solution to a task or problem. Pseudo codes have different ways of representing the same thing and emphasis is on the clarity and not style.

Pseudo Code

Is an informal high-level description of the operating principle of a computer program or other algorithm. It is a procedure for solving a problem in terms of the actions to be executed and the order in which those actions are to be executed.

Pseudo code uses the structural conventions of a programming language, but is intended for
human reading rather than machine reading. Typically omits details that are not essential for human understanding of the algorithm, such as variable declarations, system-specific code and some subroutines. The programming language is augmented with natural language description details, where convenient, or with compact mathematical notation.

The purpose of using pseudo code is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm. It is commonly used in textbooks and scientific publications that are documenting various algorithms, and also in planning of computer program development, for sketching out the structure of the program before the actual coding takes place.

**Example**

In the following example, the pseudo code is on program that can add 2 numbers together then display the result.

**Solution**

```
Start
Enter two numbers A, B
Add the two numbers together
Print sum
End
```

**Flowchart**

is a type of diagram that represents an algorithm, workflow or process. It shows the steps in the form of boxes of various kinds and their order by connecting them with arrows. The diagrammatic representation illustrates a solution model to a given problem.

Flowcharts can be used in the analysis, design, documenting or managing a process or program in various fields. They are also used in designing and documenting complex processes or programs.

**Flowchart building blocks**

1. **Start and end symbols.**

   they are represented as circles, ovals or rounded (fillet) rectangles. They contain the word “Start” or “End”, or another phrase signaling the start or end of a process, such as “submit inquiry” or “receive product”.

23
2. **Arrows.**

   They show “flow of control”. For example an arrow coming from one symbol and ending at another symbol represents that control passes to the symbol the arrow points to. The line for the arrow can be solid or dashed. The meaning of the arrow with dashed line may differ from one flowchart to another and can be defined in the legend.

3. **Generic processing steps**

   These are represented as rectangles.

4. **Subroutines**

   Subroutines are represented as rectangles with double-struck vertical edges; they are used to show complex processing steps which may be detailed in a separate flowchart.

5. **Input/Output**

   These are represented as a parallelogram.

6. **Prepare conditional**

   These are represented as a hexagon. They normally show operations which have no effect other than preparing a value for a subsequent conditional or decision step (see below).

7. **Conditional or decision**

   These are normally represented as a diamond (rhombus) showing where a decision is necessary. They commonly test a Yes/No question or True/False condition. It has two arrows coming out of it, usually from the bottom point and right point, one corresponding to Yes or True, and one corresponding to No or False.

8. **Junction symbol**

   It is generally represented with a black blob, showing where multiple control flows converge in a single exit flow. A junction symbol will have more than one arrow coming into it, but only one going out.

9. **Labeled connectors**

   They are represented by an identifying label inside a circle. Normally used in complex or multi-sheet diagrams to substitute for arrows. For each label, the “outflow” connector must always be unique, but there may be any number of “inflow” connectors. In this case, a junction in control flow is implied.

10. **Concurrency symbol**

    It is normally represented by a double transverse line with any number of entry and exit arrows.
They can be used whenever two or more control flows must operate simultaneously. The exit flows are activated concurrently when all of the entry flows have reached the concurrency symbol. A concurrency symbol with a single entry flow is a fork; one with a single exit flow is a join.

**NB**

All processes should flow from top to bottom and left to right

**EXAMPLE**

Problem: Calculate and report the grade-point average for a class

Discussion: The average grade equals the sum of all grades divided by the number of students

**Output:** Average grade

**Input:** Student grades

**Processing:** Find the sum of the grades; count the number of students; calculate average

**PSEUDO CODE**

**Program:** Determine the average grade of a class

```
Initialize Counter and Sum to 0
Do While there are more data
  Get the next Grade
  Add the Grade to the Sum
  Increment the Counter
Loop
Compute Average = Sum / Counter
Display Average
```
Conclusion

This section covered the algorithm development tools, that is, the pseudo codes, flowcharts and how to design and develop them. This tools are not hinged to any programming language but can be implemented in any language of choice. Basically this are the initial steps of designing a program for any given task. Examples for both are given to try and show the learner how they look like. On the flow charts, it has further included the building blocks of a flow chart.

Figure 1: The Average Grade of a Class Flowchart
Assessment

1. Give two important differences between the flowchart and the pseudocode.
2. Give two examples of selection control structures. Use flowchart.
3. Draw the different types of symbols used in the flowchart. Explain the role of each.

UNIT SUMMARY

In this unit, you have seen what an algorithm is. Based on this knowledge, you should now be able to characterize an algorithm by stating its properties. We have explored the different ways of representing an algorithm such as using human language, pseudo codes and flow chart. You should now be able to present solutions to problems in form of an algorithm.

Unit Assessment

The following section will test the learners understanding of this unit.

Unit Revision

Instructions

Answer the following questions

1. Design an algorithm to add two numbers and display result.
2. Write pseudocode to input the dimensions of a rectangle and print area and perimeter.
3. Give an algorithm to find the maximum number.

Grading Scheme

The marks will be awarded as shown below:

<table>
<thead>
<tr>
<th>question</th>
<th>sub-question</th>
<th>marks awarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Any property listed award 1 mark, and explanation another 1 mark</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Each term correctly stated to be awarded 1 mark (maximum 6 statements)</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>14</td>
</tr>
</tbody>
</table>
Feedback

1. can be written as below;

<table>
<thead>
<tr>
<th>step</th>
<th>1</th>
<th>−</th>
<th>START</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>step</td>
<td>2</td>
<td>−</td>
<td>get values of a &amp; b</td>
<td></td>
</tr>
<tr>
<td>step</td>
<td>3</td>
<td>−</td>
<td>c ← a + b</td>
<td></td>
</tr>
<tr>
<td>step</td>
<td>4</td>
<td>−</td>
<td>display c</td>
<td></td>
</tr>
</tbody>
</table>

step 5 − STOP

2. the pseudocode is read length, breadth

\[
\text{area} = \text{length} \times \text{breadth}
\]

\[
\text{perimeter} = 2(\text{length} + \text{breadth})
\]

display area, perimeter

3. the solution is

Pick up the first number and suppose it is maximum.

Pick up a number from the remaining numbers.

Compare the picked number and current maximum.

Throw out the smaller and suppose the larger one as the new maximum.

Repeat 2~4 until there's no number remaining.

Output the current maximum.

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.
**Unit 2. RECURSION**

**Unit Introduction**

This unit introduces the learner to the processes that repeat themselves in a self-similar way. The processes can either call themselves directly or indirectly during their execution. The unit will enable the learner to understand the recursion process and solve problems that are recursive in nature.

**Unit Objectives**

Upon completion of this unit the learner should be able to:

- develop algorithms for recursive programs
- implement recursive formulation of a problem
- explain recursion as a form of iteration

**Key Terms**

Recursion: This is a process whereby a method calls itself so as to achieve repetitious

Base case: Is a boolean test at which the recursion ceases to “wind up” and starts to “unwind”.

Recursive step: Is a method call which causes the recursion to “repeat” on each successive iteration (unless the base case is reached).

**Learning Activities**

**Activity 1 - Recursive Algorithm**

**Introduction**

The learners are introduced to recursive processes learn how a method calls itself in order to achieve repetitious behaviour. By using recursive algorithms, the learners will be in a position to solve problems of this nature.

**Recursive Algorithm**

Recursion is defined as a method of solving problems that involves breaking a problem down into smaller and smaller sub problems until you get to a small enough problem that it can be solved trivially. simply this is done by the program calling itself to repeat the same steps.

Recursive problems are solved by developing a recursive algorithm. A recursive algorithm is defined as an algorithm which can call itself with smaller (or simpler) input values, and which
obtains the result for the current input by applying simple operations to the returned value for the smaller (or simpler) input. Thus if a problem can be solved by utilizing solutions to smaller versions of the same problem, and the smaller versions reduce to easily solvable cases, then it means that one can use a recursive algorithm to solve that problem.

The recursive method comprises the following;

1. A base case where the recursion stops and begins to unwind, and
2. A recursive step where the method calls itself

A point worth noting is that a recursive algorithm is said to wind up until the base case is reached and then unwinds. During the unwinding process, further statement programs “pending” may be called.

Recursion is a mechanism whereby repetition can be achieved using a method that continues to call itself until some terminating value is reached, i.e. without using some specific repetition construct.

**Example**

The factorial function “!”.

Its definition is:

- \(0! = 1\)
- \(n! = n \times (n-1)!\) for all \(n > 0\)

Thus, by repeatedly using the definition, we can work out that

\[
6! = 6 \times 5! = 6 \times 5 \times 4! = 6 \times 5 \times 4 \times 3! = 6 \times 5 \times 4 \times 3 \times 2! = 6 \times 5 \times 4 \times 3 \times 2 \times 1! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 \times 1 = 720
\]

Again, notice that the definition of “!” includes both a base case (the definition of 0!) and a recursive part.

**Laws of Recursion**

Recursive algorithms must obey the following laws;

1. A recursive algorithm must have a base case.
2. A recursive algorithm must change its state and move toward the base case.
3. A recursive algorithm must call itself, recursively

**An explanation of the laws is as follows;**

First law; -The base case allows the algorithm to stop recursing, i.e a problem small enough to solve directly.

Second law; -A change of state occurs moving the algorithm towards the base case. This occurs when a change of state takes place and which is best illustrated when a data modification happens.
Finally; -third law;- the algorithm must call itself

**Example 2**

Find the maximum value in a list of numbers;

The solution to the problem would look like the following

```plaintext
Function find_max(list)
possible_max_1 = first_value in list
possible_max_2 = find_max(rest of the list);
if (possible_max_1 > possible_max_2)
    answer is possible_max_1
else
    answer is possible_max_2
end
end
```

**Example 3**

adding three numbers;

The solution will be something like

```plaintext
Function result = add_numbers(a, b, c)
if (nargin() == 2)
    result = a + b;
else if (nargin() == 3)
    result = add_numbers(a+b, c);
else
    error('oops');
end
end
```

É a ç
Conclusion

This unit introduced the learner to the recursive algorithms. Examples were used to teach the learner how to write recursive algorithms that can solve self-repeating functions. The examples used in the teaching of recursion are also not hinged on any computing language and can thus apply across all the languages. Finally, the learners were introduced to the laws of recursion.

Assessment

1. List the three key features of a recursive algorithm.
2. Write an algorithm that can multiply two numbers using recursion

UNIT SUMMARY

In this unit, recursion was introduced. You should now be in a position to describe recursion and where it can be applied. The laws that a recursing algorithm must obey were also introduced as well as condition necessary for a process to be said to have recursed. Examples accompanied these explanations were provided where necessary.

Unit Assessment

Check your understanding!

Class exercise

Instructions

Answer the following questions on recursion

1. Write an algorithm for finding the k-th even natural number
2. Explain how a recursive algorithm works

Grading Scheme

The marks will be awarded as shown below

<table>
<thead>
<tr>
<th>Question</th>
<th>Scores (marks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Each line of explanation 2 marks; maximum 8</td>
</tr>
<tr>
<td>Total</td>
<td>10</td>
</tr>
</tbody>
</table>
Feedback

1. the solution

Algorithm:

if k = 1, then return 0;
else return Even(k-1) + 2

2. The result of one recursion is the input for the next recursion. The repetition is in the self-similar fashion. The algorithm calls itself with smaller input values and obtains the results by simply performing the operations on these smaller values. Generation of factorial, Fibonacci number series are the examples of recursive algorithms

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.
Unit 3. Basic Data structures and Abstract data types

Unit Introduction

This section introduces the learner to the basic types of data structures that are used in computer systems. Data structures also called data types are the particular ways of organizing data in a computer so that it can be used efficiently. Abstract data types (ADT) are set of values (the carrier set), and operations on these values. Also arrays, lists, linked lists, stacks, queues, hashing and trees are included in this section.

Unit Objectives

Upon completion of this unit you should be able to:

- identify data structures used to organize data in a computer
- describe abstract data types used in the organization of data in the computer
- explain the data structures used to organize data for the efficient use of the computer.

Key Terms

Data structure: are the programmatic way of storing data so that data can be used efficiently.

Abstract data type: is the carrier set together with their operations

Array: A fixed length, ordered collection of values of the same type stored in contiguous memory locations

Lists: an abstract data type that represents a sequence of values, where the same value may occur more than once

Linked Lists: a data structure consisting of a group of nodes which together represent a sequence

Stack: is a particular kind of abstract data type or collection in which the principal operations on the collection are the push and pop.

Queues: is an abstract data type or collection in which the entities in the collection are kept in order and the only operations on the collection are the addition of entities to the rear terminal position, and removal of entities from the front terminal position.

Hashing: is a method for storing and retrieving records from a database.

Trees: A tree is a data structure made up of nodes or vertices and edges and is not cyclic.
Learning Activities

Activity 1 - Data Structures

Introduction

This section introduces the learner to the different data structures that are used to organize data in the computer. Better data organization ensures efficient utilization of the computer. These include arrays, lists, linked lists, stacks, queues, hashing and trees. The different data structures offer advantages to algorithm designers in that they are able to know which one would enable efficient use of the computer.

Definition of Data Structures

Data structures is a particular way of organizing data in the computer in order to be used efficiently. It is defined as an arrangement of data in memory locations to represent values of the carrier set of an abstract data type. Different structures are best suited to different kinds of applications in the computer. Others are even highly specialized to specific tasks. A good example is like in databases which use the B-tree indexes for small percentages of data retrieval and compilers. Another example in databases is the use of dynamic hash tables as look up tables. Data structures also provide a means to manage large amounts of data efficiently for uses such as in large databases and internet indexing services. Efficient data structures are key to designing efficient algorithms.

Data structures are based on the ability of a computer to fetch and store data at any place in its memory, which is normally specified by a pointer – a bit string, representing a memory address that can be itself stored in memory and manipulated by the program.

Examples of data structures consists of

1. Array

An array is said to be a fixed length, ordered collection of values of the same type stored in contiguous memory locations. The collection may be ordered in several dimensions. An array consists of a collection of elements (values or variables), each identified by at least one array index or key. An array can be considered as the simplest type of data structure, and can either a one-dimensional array or a two-dimensional arrays (matrices). Arrays are used to implement tables, especially lookup tables and also lists and strings used by almost every program in the computer. They effectively exploit the addressing logic of computers. In most modern computers and many external storage devices, the memory is a one-dimensional array of words, whose indices are their addresses. Processors, especially vector processors, are often optimized for array operations.

The ability to compute the element indices at run time makes arrays important and useful. This feature allows a single iterative statement to process arbitrarily many elements of an array. This is because, the elements of an array data structure are required to have the same size and should use the same data representation.
2. Lists

A list is defined as an abstract data type that represents a sequence of values, where the same value may occur more than once. An instance of a list is a computer representation of the mathematical concept of a finite sequence; the (potentially) infinite analog of a list is a stream. They are a basic example of containers, as they contain other values. There are two families of lists. Those that primarily expose an infrastructure for the first element, and the rest of the elements, and one whose primary interface is random access through an indexer.

![Figure 2.1: A list representation](image1)

Many programming languages provide support for list data types, and have special syntax and semantics for lists and list operations. A list can often be constructed by writing the items in sequence, separated by commas, semicolons, or spaces, within a pair of delimiters such as parentheses ‘(’), brackets ‘[]’, braces ‘{}’, or angle brackets ‘<>’.

3. Linked Lists

A linked list is a data structure that consists of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data and a reference (a link) to the next node in the sequence; more complex variants add additional links. This structure allows for efficient insertion or removal of elements from any position in the sequence.

![Figure 2.2: A linked list](image2)

They are among the simplest and most common data structures and which can be used to implement several other common abstract data types. They have the principal benefit over a conventional array in that the list elements can easily be inserted or removed. This can be done without reallocation or reorganization of the entire structure as the data items need not be stored contiguously in memory or on disk. On the other hand, an array has to be declared in the source code, before compiling and running the program. They allow insertion and removal of nodes at any point in the list, and can do so with a constant number of operations if the link previous to the link being added or removed is maintained during list traversal. It has the following advantages and disadvantages;
Advantages:

- It is a dynamic data structure, allocating the needed memory while the program is running.
- Insertion and deletion node operations are easily implemented in a linked list.
- Linear data structures such as stacks and queues are easily executed with a linked list.
- They can reduce access time and may expand in real time without memory overhead.

Disadvantages:

- They have a tendency to waste memory due to pointers requiring extra storage space.
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.
- Nodes are stored incontiguously, greatly increasing the time required to access individual elements within the list.
- Difficulties arise in linked lists when it comes to reverse traversing. Singly linked lists are extremely difficult to navigate backwards, and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back pointer.

4. Stack

This is a particular kind of abstract data type or collection in which the principal operations on the collection are the addition of an entity to the collection (push) and removal of an entity (pop).

The stack is a Last-In-First-Out (LIFO) data structure, i.e. the last element added to the structure must be the first one to be removed. The push and pop operations occur only at one end of the structure, referred to as the top of the stack. A peek or top operation can also be implemented, returning the value of the top element without removing it.
A stack is said to be full if it does not contain enough space to accept an entity to be pushed, and it is then considered to be in an overflow state. A pop operation removes an item from the top of the stack. A stack is a restricted data structure, because only a small number of operations are performed on it. The nature of the pop and push operations also mean that stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition.

5. Queues

A queue is a kind of abstract data type or collection in which the entities in the collection are kept in order and the principal operations on the collection are the addition of entities to the rear terminal position, and removal of entities from the front terminal position. It is a First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. It is an example of a linear data structure.

![Queue Diagram](image)

Figure 2.4: A diagram of a queue

6. Hashing

Is a method for storing and retrieving records from a database. Hashing allows one to insert, delete, and search for records based on a search key value. A hash system stores records in an array called a hash table. It works by performing a computation on a search key K in a way that is intended to identify the position in hash table that contains the record with key K.

The calculation are done by a function called the hash function. Hash function is any function that can be used to map digital data of arbitrary size to digital data of fixed size, with slight differences in input data producing very big differences in output data. Since hashing schemes place records in the table in whatever order satisfies the needs of the address calculation, records are not ordered by value. A position in the hash table is known as a slot. The number of slots in hash table will be denoted by the variable M with slots numbered from 0 to M - 1.
Hashing is used to index and retrieve items in a database, it is faster to find the item using the shorter hashed key than to find it using the original value. It is also used in many encryption algorithms, and also to encrypt and decrypt digital signatures.

8. Trees

A tree is a data structure made up of nodes or vertices and edges without having any cycle. A tree with no nodes is called the null or empty tree. A tree that is not empty consists of a root node and potentially many levels of additional nodes that form a hierarchy.

In computer science, it is a widely used abstract data type (ADT) or data structure implementing this ADT that simulates a hierarchical tree structure, with a root value and sub-trees of children, represented as a set of linked nodes.

A tree data structure can be defined recursively as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the “children”), with the constraints that no reference is duplicated, and none points to the root.

![Diagram of a tree](image)

Figure 2.5: Diagram of a tree

Conclusion

This activity section has introduced the learner to the different data structures used to organize data in the memory of the computer, to make it efficient. It has also highlighted a few examples where they apply. Cited cases include the use of B-trees indexing in the databases and use of hashing to index and retrieve data in a database among others. The data structures looked at in this unit include; Arrays, Lists, Linked lists, Stack, Queues, hashing and Trees.
Assessment

1. Differentiate between a queue and a stack data structures
2. Define linear data structures and give an example
3. List out the basic operations that can be performed on a stack
4. Briefly state the difference between queues and linked lists

Activity 2 - Abstract Data Type

Introduction

This section introduces the learner to Abstract data type (ADT). The ADT section describes the different data types behaviour from the point of view of a user of the data, specifically in terms of possible values, possible operations on data of this type, and the behavior of these operations. All in all different data structures have different operations performed on them.

Definition of ADT

In computer science, an abstract data type (ADT) is defined as a mathematical model for a certain class of data structures that have similar behavior; or for certain data types of one or more programming languages that have similar semantics. An abstract data type is defined only by the operations that may be performed on it and by mathematical pre-conditions and constraints on the effects (and possibly cost) of those operations.

For example,

An abstract array is defined by the following operations;

- Adding elements
- Sorting elements
- Searching elements
- Re-arranging the elements
- Performing matrix operations
- Pre-fix and post-fix operations

An abstract list is defined by the following;

- inserting
- searching
- deletion
An abstract linked list has the following operations

- checking whether the list is empty;
- accessing a node to modify it or to obtain the information in it;
- traversing the list to access all elements (e.g., to print them, or to find some specific element);
- determining the size (i.e., the number of elements) of the list;
- inserting or removing a specific element (e.g., the first one, the last one, or one with a certain value);
- creating a list by reading the elements from an input stream;
- converting a list to and from an array, string, etc.

An abstract stack, which is a last-in-first-out structure, could be defined by three operations:

- push, that inserts some data item onto the structure,
- pop, that extracts an item from it, and
- Peek or top, which allows data on top of the structure to be examined without removal.

An abstract queue data structure, which is a first-in-first-out structure, would also have three operations,

- enqueue to join the queue;
- dequeue, to remove the first element from the queue; and
- Front, in order to access and serve the first element in the queue.

An abstract hashing; has the following operations

- Add (insert)
- Delete(remove)

Abstract tree has the following;

- Searching
- Insertion
- Deletion
- Traversal
- Sort

**Conclusion**

The section introduced the learner to abstract data types, which is about logical description of how data is viewed and the operations that are allowed without regard to how they will be implemented.
Assessment

1. What is the definition of a data type?
2. Describe briefly two ADTs that we studied in this section? In each case specifying the necessary operations for each

UNIT SUMMARY

This unit covered data structures and abstract data types. In the data structures section, different structures were discussed. These included arrays, links, Linked lists, Stacks, Queues, Hashing and Trees. These structures are used to organize data in the computer to make it work efficiently. In the abstract data types section, the operations performed on the data structures are discussed.

Unit Assessment

Check your understanding!

Question and answer session

Instructions

Answer all the questions and test your grasp of the unit

1. Differentiate between Abstract Data Type, Data Type and Data Structure
2. what is a tree as learned in computer science?
3. Give two importance of hashing
4. State the difference between arrays and linked lists
5. In tree construction which is the suitable efficient data structure? (Array, Linked list, Stack, Queue)
6. What is a spanning Tree?
Grading Scheme

The marks will be awarded as shown below

<table>
<thead>
<tr>
<th>Question</th>
<th>Scores (marks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>16</td>
</tr>
</tbody>
</table>

Feedback

1. An Abstract data type is the specification of the data type which specifies the logical and mathematical model of the data type.
   
   • A data type is the implementation of an abstract data type.
   • Data structure refers to the collection of computer variables that are connected in some specific manner. i.e) Data type has its root in the abstract data type and a data structure comprises a set of computer variables of same or different data types.

2. A tree is a collection of nodes. The collection can be empty; otherwise, a tree consists of a distinguished node, called the root, and zero or more non empty (sub) trees T1, T2, ..., Tk, each of whose roots are connected by a directed edge from r.

3. The importance of hashing.
   
   • Maps key with the corresponding value using hash function.
   • Hash tables support the efficient addition of new entries and the time spent on searching for the required data is independent of the number of items stored.

4. the difference between arrays and linked lists
   
   • Size of an array is fixed
   • Size of a list is variable in a linked list
   • It is necessary to specify the number of elements during declaration in an array
   • It is not necessary to specify the number of elements during declaration in a linked list
5. Linked list is the suitable efficient data structure.

6. A spanning tree is a tree associated with a network. All the nodes of the graph appear on the tree once. A minimum spanning tree is a spanning tree organized so that the total edge weight between nodes is minimized.

**Unit Readings and Other Resources**

The readings in this unit are to be found at [course level readings and other resources](#).
Unit Introduction

In this unit, you will learn some common processes that arise in computing; these are searching and sorting. Some common processes are those that involve finding a particular item in a given list of items called searching. Another involves ordering items in a particular way of some given data called sorting. The two represent a common application in computers where data has to be searched and retrieved. An example of an everyday application of this is in a telephone directory. It stores information such as the names, addresses, and phone numbers of its customers. As the amount of information to be stored and accessed becomes very large, the computer comes in handy to assist in this task. To access the stored data, searching and sorting are carried out.

Unit Objectives

Upon completion of this unit you should be able to:

- describe the process of searching and sorting
- design algorithms for searching and sorting
- implement algorithms for searching and sorting

Key Terms

- searching: Is the algorithmic finding of a particular item in a collection of given items
- Sorting: Is the ordering of items in a given list

Learning Activities

Activity 1 - SEARCHING ALGORITHM

Introduction

This activity involves studying the searching process and how to write algorithms that can find particular given item from a list of given items. The activity of searching will dedicate itself to providing the answer of a presence or no presence of the searched item. The section will dedicate itself to introducing the sequential and binary types of search.
Searching algorithm

In computer science, a search algorithm is an algorithm for finding an item with specified properties among a collection of items.

Finding a Data Item - The Search Problem

Consider a collection of data items of same type e.g. integers, and imagine we wish to check if a particular data item is in the collection. Call the particular data item of interest to us, the key. The task is then to search among the records in the database to find out which one “matches” the key. This can first be done by arranging the data using a structure that can hold a collection of data items of the same type, that is, the arrays or linked lists. The searching can be done using either the sequential or binary search, where the former involves looking at each item in turn while the later involves items that are in order. The example below, is of an array consisting of seven elements long, containing numeric values. That is, the maximum number of comparisons is 7, and occurs when the key we are searching for is in A[6].

Both sequential and binary search algorithm can be used to find the data item:

1. For a sequential search the following will do;

```
for (each item in list ) {
compare search item to current item
if match,
save index of matching item
break
}
return index of matching item, or -1 if item not found
```

2. For a binary search, the following algorithm illustrates how it is carried out;

```
set first = 1, last = N, mid = N/2
while (item not found and first < last ) {
compare search term to item at mid
if match,
break
}
return index of matching item, or -1 if item not found
```

Array:

<table>
<thead>
<tr>
<th>3</th>
<th>8</th>
<th>1</th>
<th>0</th>
<th>5</th>
<th>-2</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 4.1: an array of numbers
Conclusion

In this activity, you learned two different search algorithms, the sequential and binary searches. Examples illustrating them was also given. Its application to an array was also introduced i.e. how a search can be applied in an array to search for a particular item in an array.

Assessment

Provide a short description of binary search algorithm.

Activity 2 - SORTING ALGORITHM

Introduction

This activity involves studying the sorting process and how to write algorithms that can order particular items in a list to be in given order. The activity of sorting ensures items in a given list are arranged in a desired order say ascending or descending. The section involves studying insertion sort, shell sort, quicksort algorithms.

The sorting algorithm

A sorting algorithm is defined as an algorithm that puts elements of a list in a certain order. The most-used order is the alphabetical order, also known as the lexicographical order. Efficient sorting has the importance for optimizing the use of other algorithms (such as search and merge algorithms) which require input data to be in sorted lists. Examples of sorting algorithms includes; insertion sort, shell sort and quick sort.

Insertion sort is the simplest method, and does not require any additional storage. Shell sort is a simple modification that improves performance significantly. Quicksort is the most efficient and popular method, and is the method of choice for large arrays.
**Insertion Sort**

Is a simple sorting algorithm, and which is relatively efficient for small lists and mostly sorted lists. It also be used as part of more sophisticated algorithms. It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list. In an array structure, the new list and the remaining elements can share the array's space, but insertion is expensive, requiring shifting all following elements over by one.

The array is searched sequentially and unsorted items are moved and inserted into sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of $\Omega(n^2)$ where $n$ are no. of items.

How insertion sort works?

Consider the following an unsorted array for our example.

Insertion sort compares the first two elements

It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.

Insertion sort moves ahead and compares 33 with 27.

It swaps 33 with 27. Also it checks with all the elements of sorted sublist. Here we see that sorted sub-list has only one element 14 and 27 is greater than 14. Hence sorted sub-list remain sorted after swapping.
By now we have 14 and 27 in the sorted sublist. Next it compares 33 with 10.

These values are not in sorted order.

So we swap them

But swapping makes 27 and 10 unsorted.

Again we find 14 and 10 in unsorted order.

And we swap them. By the end of third iteration we have a sorted sublist of 4 items.

This process goes until all the unsorted values are covered in sorted sublist. And now we shall see some programming aspects of insertion sort.

**Example of the insertion algorithm**

The following are simple steps by which we can achieve insertion sort.

**Step 1** – If it is the first element, it is already sorted. return 1;

**Step 2** – Pick next element
Step 3 – Compare with all elements in the sorted sub-list

Step 4 – Shift all the elements in the sorted sub-list that is greater than the value to be sorted

Step 5 – Insert the value

Step 6 – Repeat until list is sorted

Shell Sort

Shell sort improves on the efficiency of insertion sort by quickly shifting values to their destination. It improves upon bubble sort and insertion sort by moving out of order elements more than one position at a time. One implementation can be described as arranging the data sequence in a two-dimensional array and then sorting the columns of the array using insertion sort.

Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort if smaller value is very far right and have to move to far left. This algorithm uses insertion sort on widely spread elements first to sort them and then sorts the less widely spaced elements. This spacing is termed as interval. This algorithm is quite efficient for medium sized data sets as its average and worst case complexity are of $O(n)$ where $n$ are no. of items.

How shell sort works

Consider example below to have an idea, how shell sort works? We take the same array we have used in our previous examples. For our example and ease of understanding we take the interval of 4. And make a virtual sublist of all values located at the interval of 4 positions. Here these values are {35, 14}, {33, 19}, {42, 27} and {10, 14}
We compare values in each sub-list and swap them (if necessary) in the original array. After this step, new array should look like this –

![Figure 4.2.2](image)

Then we take interval of 2 and this gap generates two sublists - {14, 27, 35, 42}, {19, 10, 33, 44}

![Figure 4.2.2](image)

We compare and swap the values, if required, in the original array. After this step, this array should look like this –

![Figure 4.2.2](image)

And finally, we sort the rest of the array using interval of value 1. Shell sort uses insertion sort to sort the array. The step by step depiction is shown below –

![Figure 4.2.3](image)
It can be seen that it required only four swaps to sort the rest of the array.

The following is the algorithm for shell sort.

- **Step 1** – Initialize the value of h
- **Step 2** – Divide the list into smaller sub-list of equal interval h
- **Step 3** – Sort these sub-lists using insertion sort
- **Step 4** – Repeat until complete list is sorted

### Quicksort

It is one of the most popular sorting algorithms. It is significantly a better algorithm than the insertion sort.

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than specified value say pivot based on which the partition is made and another array holds values greater than pivot value.

The quick sort partitions an array and then calls itself recursively twice to sort the resulting two subarray. This algorithm is quite efficient for large sized data sets as its average and worst case complexity are of $O(n\log n)$ where $n$ are no. of items.

**Below is longer version of an algorithm for a quicksort:**

- **Step 1** – Choose the highest index value has pivot
- **Step 2** – Take two variables to point left and right of the list excluding pivot
- **Step 3** – left points to the low index
- **Step 4** – right points to the high
- **Step 5** – while value at left is less than pivot move right
- **Step 6** – while value at right is greater than pivot move left
- **Step 7** – if both step 5 and step 6 does not match swap left and right
- **Step 8** – if left ≥ right, the point where they met is new pivot

It can be summarized into the following fewer steps using pivot algorithm recursively we end-up with smaller possible partitions. Each partition then processed for quick sort. We define recursive algorithm for quicksort as below –

- **Step 1** – Make the right-most index value pivot
- **Step 2** – partition the array using pivot value
- **Step 3** – quicksort left partition recursively
- **Step 4** – quicksort right partition recursively
Conclusion

This activity introduced the learner to three different sort algorithms, the insertion sort, the shell sort and quicksort. A brief description of each was given.

Assessment

1. Describe briefly an insertion sorting algorithm.

2. What is the difference between selection and insertion sorting?

UNIT SUMMARY

This unit discussed the searching and sorting algorithms together with their examples. Searching which is about finding a particular item in a given list of items and Sorting on the other hand which is the ordering in a particular way some given data were explained. For the searching, the sequential and binary searches were discussed. While for the sorting algorithm, insertion sort, shell sort and quicksort were discussed. Where possible algorithms that can be used to illustrate the two concepts were also introduced.

Unit Assessment

Check your understanding!

Question and answer

Instructions

Answer all the questions below;

1. Describe the quicksort algorithm.

2. Describe an insertion sorting algorithm.

Grading Scheme

The marks will be awarded as shown below

<table>
<thead>
<tr>
<th>Question</th>
<th>Scores (marks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Each 2 mark total mark is 6</td>
</tr>
<tr>
<td>2</td>
<td>Each line of explanation 2 marks; maximum 12</td>
</tr>
<tr>
<td>Total</td>
<td>18</td>
</tr>
</tbody>
</table>
Feedback

1. In quicksort, the steps performed are the following:
   a) pick an element, called a pivot, from the list
   b) reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way)
   c) recursively sort the sub-list of lesser elements and the sub-list of greater elements

2. An algorithm that sorts by insertion takes the initial, unsorted sequence and computes a series of sorted sequences using the following rules:
   a) the first sequence in the series is the empty sequence
   b) given a sequence S(i) in the series, for 0<=i<n, the next sequence in the series, S(i+1), is obtained by inserting the (i+1)th element of the unsorted sequence S(i+1) into the correct position in S(i).

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Course Assessment

Continuous Assessment

Instructions

Answer the following question and be as detailed as possible

1. What is sequential search? What is the average number of comparisons in a sequential search?

2. How would one sort a linked list?

3. What is a data structure? Give three types of data structures and briefly explain each

4. Which data structure is used to perform recursion?
Grading Scheme

The marks will be awarded as shown below

<table>
<thead>
<tr>
<th>Question</th>
<th>Scores (marks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Each correct answer award 2 marks each total mark is 6 (maximum)</td>
</tr>
<tr>
<td>2</td>
<td>Each correct answer award 2 marks; maximum 6</td>
</tr>
<tr>
<td>3</td>
<td>Each correct answer award 2 marks; maximum 6</td>
</tr>
<tr>
<td>4</td>
<td>Each correct answer award 2 marks; maximum 2</td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
</tr>
</tbody>
</table>

Feedback

1. Sequential search: Searching an element in an array, the search starts from the first element till the last element.

The average number of comparisons in a sequential search is \((N+1)/2\) where \(N\) is the size of the array. If the element is in the 1st position, the number of comparisons will be 1 and if the element is in the last position, the number of comparisons will be \(N\).

2. **Step 1:** Compare the current node in the unsorted list with every element in the rest of the list. If the current element is greater than any other element go to step 2 otherwise go to step 3.

   **Step 2:** Position the element with higher value after the position of the current element. Compare the next element. Go to step 1 if an element exists, else stop the process.

   **Step 3:** If the list is already in sorted order, insert the current node at the end of the list. Compare the next element, if any and go to step 1 or quit.

3. The scheme of organizing related information is known as ‘data structure’. Three types of data structure are:

   Lists: A group of similar items with connectivity to the previous or/and next data items.
Arrays: A set of homogeneous values

Records: A set of fields, where each field consists of data belonging to one data type.

Trees: A data structure where the data is organized in a hierarchical structure. This type of data structure follows the sorted order of insertion, deletion and modification of data items.

4. The data structure used for recursion is Stack.
   • Its LIFO property helps it remembers its ‘caller’. This helps it know the data which is to be returned when the function has to return.
   • System stack is used for storing the return addresses of the function calls.

Final Examination

Instructions

Answer all the questions that follow below

1. Briefly describe;
   a. three different application of a stack. (9 marks)
   b. the hashing technique. (8 marks)
   c. the binary search algorithm (8 marks)

2. Differentiate between;
   a) PUSH and POP as used in a stack. (8 marks)
   b) a stack and a Queue? (8 marks)

3. Explain the following;
   a) the types operations of an array (9 marks)
   b) three properties of an algorithm (6 marks)

4. How do you insert a new item in a binary search tree? (14 marks)

Grading Scheme

The marks will be awarded as shown below

<table>
<thead>
<tr>
<th>Question</th>
<th>Scores (marks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1(a)</td>
<td>Each 3 mark total mark is 9</td>
</tr>
<tr>
<td>1(b)</td>
<td>Each line of explanation 2 marks; maximum 8</td>
</tr>
</tbody>
</table>
### Feedback

1. **(a).** Any three among the following applications of stack are as follows:
   - Function calls.
   - Reversal of a string.
   - Checking validity of an expression containing nested parenthesis.
   - Conversion of infix expression to postfix.

   **(b).** In general, in all searching techniques, search time is dependent on the number of items. Hashing is a technique where search time is independent of the number of items or elements. In this technique a hash function is used to generate an address from a key. The hash function takes a key as input and returns the hash value of that key which is used as an address index in the array.

   **(c).** Binary search algorithm always chooses the middle of the remaining search space, discarding one half or the other, again depending on the comparison between the key value found at the estimated position and the key value sought. The remaining search space is reduced to the part before or after the estimated position.

2. **(a).** Pushing and popping refers to the way data is stored into and retrieved from a stack.
   - PUSH – Data being pushed/ added to the stack.
   - POP - Data being retrieved from the stack, particularly the topmost data.

   **(b).** Stack – Represents the collection of elements in Last In First Out order.
Operations includes testing null stack, finding the top element in the stack, removal of top most element and adding elements on the top of the stack.

**Queue** - Represents the collection of elements in First In First Out order. Operations include testing null queue, finding the next element, removal of elements and inserting the elements from the queue.

**Insertion of elements is at the end of the queue**

Deletion of elements is from the beginning of the queue.

3. (a). Arrays are used for storing the data until the application expires in the main memory of the computer system. So that, the elements can be accessed at any time. The operations are:

- Any 3 operations
- Adding elements
- Sorting elements
- Searching elements
- Re-arranging the elements
- Performing matrix operations
- Pre-fix and post-fix operations

(b). Properties of an algorithm:

- Any 3 properties
  - Should be written in simple English
  - Should be unambiguous, precise and lucid
  - Should provide the correct solutions
  - Should have an end point
  - The output statements should follow input, process instructions
  - The initial statements should be of input statements
  - Should have finite number of steps
  - Every statement should be definitive

4. Assuming that the data to be inserted is a unique value (that is, not an existing entry in the tree), check first if the tree is empty. If it’s empty, just insert the new item in the root node. If it’s not empty, refer to the new item’s key. If it’s smaller than the root’s key, insert it into the root’s left subtree, otherwise, insert it into the root’s right subtree.
MODULE SUMMARY

In this module, in unit 1 you learnt about algorithms, its properties and the different ways it can be represented. In unit 2, recursion was introduced where it can used and the laws that a recurring algorithm must obey examples accompanied these explanations were provided where necessary. In the unit 3, data structures and abstract data types were learnt. In the data structures section, different structures were discussed. These included arrays, links, Linked lists, Stacks, Queues, Hashing and Trees. In the abstract data types section, the operations performed on the data structures were discussed. In the final unit, the searching and sorting algorithms together with their examples were learnt. For the searching, the sequential and binary searches were discussed. While for the sorting algorithm, insertion sort, shell sort and quicksort were discussed.

Course References

- Paolo Coletti, (2014). Basic Computer course book, 8TH Edition:
- SAMS. (1999. Teach Yourself Data Structures And Algorithms In 24 hours.
- Adam Drozdek, Data Structures and Algorithms in Java, SECOND EDITION
- Harry H Chaudhary. (2014). Teach Yourself Data Structures and Algorithms in 15 Days.: Beginner s Guide. ISBN 1500137251; Publisher: Createspace, United States,
The African Virtual University
Headquarters
Cape Office Park
Ring Road Kilimani
PO Box 25405-00603
Nairobi, Kenya
Tel: +254 20 25283333
contact@avu.org
oer@avu.org

The African Virtual University Regional Office in Dakar
Université Virtuelle Africaine
Bureau Régional de l’Afrique de l’Ouest
Sicap Liberté VI Extension
Villa No.8 VDN
B.P. 50609 Dakar, Sénégal
Tel: +221 338670324
bureauregional@avu.org