- **Appreciate the importance of the concepts of data structure, their functionality and techniques.**

The data structure is a fundamental concept in computer science that refers to how data is organised, stored, and accessed in a computer program or system. Data structures are essential for efficient data manipulation and are a critical factor in the performance of algorithms and applications (Mehlhorn, 2013).

Hence, Mehlhorn (2013) stated that Data structures are designed to provide specific functionality to meet the requirements of various applications. For instance, a stack data structure is used for implementing undo/redo functionality in text editors. In contrast, databases use a hash table for fast key-value lookup.

Data structures can be broadly categorised into two types: *primitive data structures* and *composite data structures*. Primitive data structures include fundamental data types such as integers, floating-point numbers, and characters. In contrast, composite data structures have arrays, linked lists, trees, and graphs.

The functionality of a data structure depends on its properties, such as its access time, memory usage, and storage capacity. For example, an array provides constant-time access to elements but requires contiguous memory allocation. A linked list allows for efficient insertion and deletion but requires additional memory for maintaining the links between nodes.

Techniques for designing and implementing data structures include algorithm analysis, memory management, and abstraction. Algorithm analysis evaluates operations' time and space complexity on a data structure. At the same time, memory management allocates and deallocates memory for data storage. Abstraction hides a data structure's implementation details, providing a clean and simple interface for accessing and manipulating data (Thareja, 2014).

In conclusion, data structures are essential for efficient and effective data manipulation in computer programs and systems. Computer science professionals and students must understand the importance of data structures, their functionality, and the techniques for designing and implementing them.

- **Apply and demonstrate the applications of data structure in computer programming.**

Data structures are used extensively in computer programming for various applications, enabling programmers to create algorithms to solve complex problems. The heart of computer science is data structures, allowing computers to store and retrieve data efficiently (adservio.fr, 2022).

Therefore, McMillan R. (2014) explained with the following examples of how data structures can be used in programming:

Arrays: An array can store a list of integers and perform operations such as sorting, searching, or computing the average. Here are some examples:

- The [] operator is used to declare an array variable, which is the simplest way to create an array:

  ```
  var numbers = [];
  ```

- When you create an array in this manner, it has a length of 0. This can be confirmed by using the built-in length property:

  ```
  print(numbers.length); // displays 0
  ```

- Another method is to declare an array variable with a set of elements within the [] operator:

  ```
  var numbers = [1,2,3,4,5];
  print(numbers.length); // displays 5
  ```

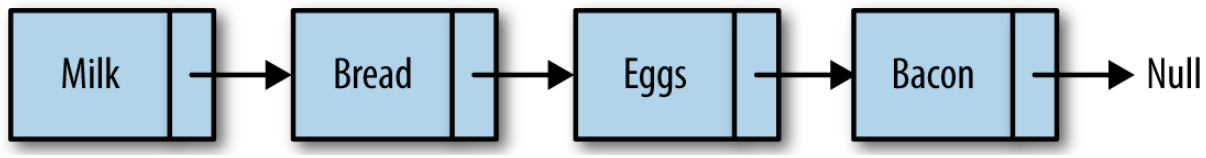
- You can also use the Array function Object() { [native code] } to create an array:

  ```
  var numbers = new Array();
  print(numbers.length); // displays 0
  ```

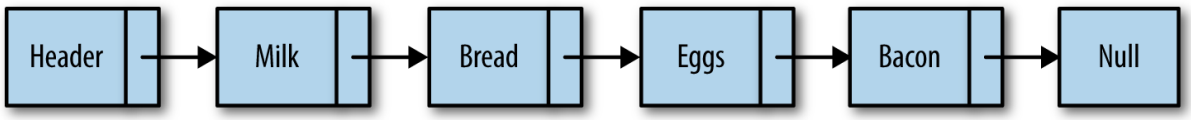
- You can invoke the Array function Object() { [native code] } with a set of elements as arguments:
  var numbers = new Array(1,2,3,4,5);
  print(numbers.length); *// displays 5*

- Finally, you can make an array by invoking the Array function Object() { [native code] } with a single argument specifying the array's length:
  var numbers = new Array(10);
  print(numbers.length); *// displays 10*

Linked Lists: A linked list is a set of objects known as nodes. Using an object reference, each node in the list is linked to a successor node, and a link is a reference to another node.



A linked list



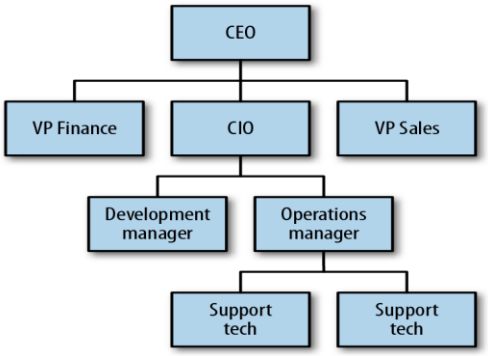
A linked list with a head node

The function Object() { [native code] } function is defined as follows:

```
function LList() {
        this.head = new Node("head");
        this.find = find;
        this.insert = insert;
        this.remove = remove;
        this.display = display;
}
```

Because we create a new Node element but do not modify its next property here in the function Object() { [native code] }, the head node starts with its next property set to null and is changed to point to the first element inserted into the list.

Trees: A tree is made up of nodes that are connected by edges.



Because nodes make up a binary search tree, the first object we need to create is a Node object, similar to the Node object we used with linked lists. The Node class is defined as follows:

```
function Node(data, left, right) {
        this.data = data;
        this.left = left;
        this.right = right;
        this.show = show;
}
function show() {
        return this.data;
}
```

The Node object stores data and links to other nodes (left and right). There is also a show() function for displaying the data stored in a node.

Graphs: A graph comprises two parts: vertices and edges.

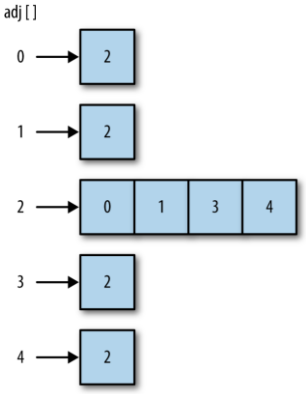
The function Object() { [native code] } function, which allows us to set the values for a vertex's data members, is the only function required for the class. The code is as follows:

```
function Vertex(label) {
this.label = label;
}
```

The list of vertices will be stored in an array and referenced in the Graph class by their position in the array.

Because the edges describe the structure of a graph, they contain accurate information about it.

An adjacency list, or an array of adjacency lists, is the method we will use to represent the edges of a graph. Edges are stored as a vertex-indexed array of lists (arrays) of vertices adjacent to each vertex.



An adjacency list

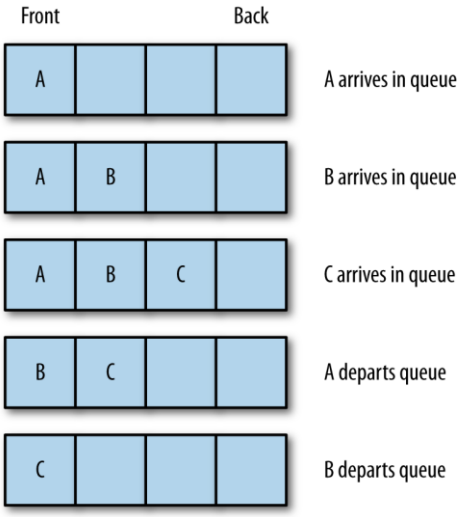
Hash Tables: Hash tables are used for fast key-value lookups in databases, caches, and other data structures. They provide constant time search, insertion, and deletion operations.

Here is the HashTable class's function Object() { [native code] } function:

```
function HashTable() {
        this.table = new Array(137);
        this.simpleHash = simpleHash;
        this.showDistro = showDistro;
        this.put = put;
        //this.get = get;
}
```

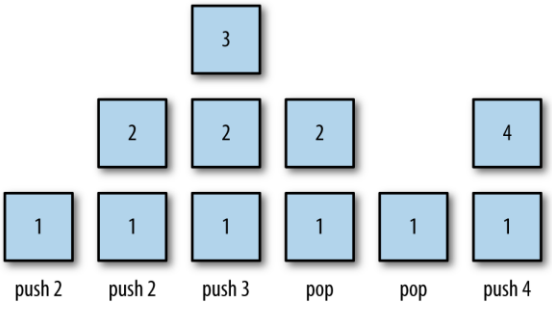
Queues: A queue is a type of list in which data is inserted at the end and removed from the beginning. Queues, as opposed to stacks, are used to store data in the order in which it is entered, rather than the last piece of data entered is the first element used for processing.

A queue is an example of a FIFO (first-in, first-out) data structure. Queues organise processes submitted to an operating system or a print spooler.



Inserting and removing elements from a queue

Stacks: A stack is a list of elements that are only accessible from one end of the list, known as the top. A stack of trays at a cafeteria is a typical real-world example of a stack. Trays are always removed from the top, and when re-stacked after being washed, they are re-positioned on top. The stack is a data structure that operates on a last-in, first-out (LIFO) basis.



Pushing and popping elements of a stack

(McMillan, 2014).

In summary, data structures are used extensively in computer programming for various applications such as storing, accessing, and manipulating data efficiently.

Understanding the applications of data structures is essential for software developers to design and implement efficient algorithms and applications.

- **Demonstrate a critical understanding of core data structures and programming concepts, including algorithm computability.**

The pursuit of programme efficiency should be kept separate from the pursuit of good design and clear coding. Creating efficient programmes has little to do with "programming tricks" but instead with good information organisation and algorithms. A programmer who has not mastered the fundamental principles of simple design is unlikely to write efficient programmes (Shaffer, 2011).

Data structures are fundamental concepts in computer science that allow programmers to store and organise data efficiently and effectively (Goodrich et al., 2014). For instance, here are some critical understandings of core data structures and programming concepts:

1. **The core data structures:**

For programmers, data structure refers to the organisation and storage of data in a specific manner to facilitate efficient operations such as searching, insertion, deletion, and traversal. Understanding data structures is critical for writing efficient and scalable programmes in computer programming. Different data structures have other properties and are appropriate for various problems. An array, for example, is a data structure that allows constant-time access to any element, but its size is fixed at creation. In contrast, a linked list allows efficient insertion and deletion at any position, but accessing a specific element takes linear time.

Programmers use data structures to represent real-world objects such as customers, products, or transactions and solve computational problems such as sorting,

searching, and graph traversal. Data structures are implemented in various programming languages, including C, C++, Java, Python, and others, and use multiple programming concepts such as pointers, classes, and interfaces (Beu, 2014).

The data structure requires programmers to write efficient, scalable code and solve computational problems. For that, programmers must understand the properties of various data structures, such as arrays, linked lists, stacks, queues, trees, and graphs. Know when to use a specific data structure based on the requirements of the problem, such as efficient access, insertion, deletion, or search, and how to implement and manipulate data structures in code.

2. **Programming Concepts:**

Writing instructions that direct a computer to perform tasks is known as computer programming, and all software programmes follow Certain programming principles and concepts (indeed, 2022). Programming concepts are the foundation for creating programmes, and they teach you to write efficient, maintainable, scalable code (McCool, 2012). For instance, here are some of the core programming concepts:

Variables: Variables are memory locations where values are stored, allowing programmers to assign and manipulate data (Engineering LibreTexts, 2019).

Control Structures: Control structures control the flow of execution in a program. Examples include if/else statements, loops, and switch statements (Engineering LibreTexts, 2019).

Functions: Functions are self-contained blocks of code that perform a specific task. They allow programmers to modularise and reuse code across the program (Engineering LibreTexts, 2019).

Object-Oriented Programming (OOP): OOP is a programming paradigm focusing on objects that encapsulate data and behaviour. OOP allows for code reuse, modularity, and abstraction (Pokkunuri, 1989).

Exception Handling: Exception handling is the process of handling errors or exceptions that occur during program execution, and it allows for graceful error recovery and program stability (jython.readthedocs.io, N.D.).

3. Algorithm Computability:

Algorithm computability studies the types of problems that algorithms can solve. Some problems are not computable, meaning no algorithm exists to solve them. The most famous example of an unsolvable problem is the Halting problem, which asks whether a given program will eventually halt or run forever.

The computability of an algorithm is determined by its time and space complexity. Time complexity measures how long an algorithm takes to solve a problem as a function of the input size. Space complexity measures how much memory an algorithm requires to solve a problem as a function of the input size.

(nus-cs1010.github.io, N.D.).

In summary, a critical understanding of core data structures and programming concepts, including algorithm computability, is essential for any computer scientist or programmer. These concepts allow for efficient and effective problem-solving and program development.

References:

Mehlhorn, K. (2013). *Data Structures and Algorithms 1: Sorting and Searching*. [online] *Google Books*. Springer Science & Business Media. Available at: https://www.google.co.uk/books/edition/Data_Structures_and_Algorithms_1/zdupCAAAQBAJ?hl=en&gbpv=1&dq=Data+structures+are+essential+for+efficient+data+manipulation+and+are+a+critical+factor+in+the+performance+of+algorithms+and+applications.&pg=PA1&printsec=frontcover [Accessed 9 Apr. 2023].

Thareja, R. (2014). Data structures using C. Available at: http://www.ir.juit.ac.in:8080/jspui/bitstream/123456789/5374/1/Data%20structures%20using%20C%2C%202nd%20Ed.%20by%20Thareja%2C%20Reema%20%282014%29.pdf [Accessed 26 Mar. 2023].

McMillan, M. (2014). *Data Structures and Algorithms with JavaScript: Bringing Classic Computing Approaches to the Web*. [online] *Google Books*. 'O'Reilly Media, Inc.' Available at: https://www.google.co.uk/books/edition/Data_Structures_and_Algorithms_with_Java/1ywEAwAAQBAJ?hl=en&gbpv=1&dq=Data+Structures+and+Algorithms+with+JavaScript%2Bo%27reilly&pg=PR2&printsec=frontcover [Accessed 9 Apr. 2023].

Shaffer, C. (2011). *A Practical Introduction to Data Structures and Algorithm Analysis Edition 3.2 (C++ Version)*. [online] Available at: https://people.cs.vt.edu/~shaffer/Book/C++3e20110520.pdf.

Goodrich, M.T., Tamassia, R. & Goldwasser, M.H. (2014). *Data Structures and Algorithms in Java*. [online] *Google Books*. John Wiley & Sons. Available at: https://www.google.co.uk/books/edition/Data_Structures_and_Algorithms_in_Java/UgmYAgAAQBAJ?hl=en&gbpv=1&dq=Data+structures+are+fundamental+concepts+in

+computer+science+that+allow+programmers+to+store+and+organise+data+efficiently+and+effectively&pg=PA2&printsec=frontcover [Accessed 9 Apr. 2023].

Beu, T.A. (2014). *Introduction to numerical programming: a practical guide for scientists and engineers using Python and C/C++*. CRC Press.

indeed.com (2022). *6 Fundamental Programming Concepts (With Tips to Improve)*. [online] Available at: https://ca.indeed.com/career-advice/career-development/fundamental-programming-concepts#:~:text=The%20basic%20programming%20concepts%20include,%2B%2B%2C%20C%2C%20and%20Java. [Accessed 9 Apr. 2023].

Pokkunuri, B.P. (1989). Object Oriented Programming. *ACM SIGPLAN Notices*, 24(11), pp.96–101. doi:https://doi.org/10.1145/71605.71612.

McCool, M., Reinders, J. & Robison, A. (2012). *Structured parallel programming: patterns for efficient computation*. Elsevier.

Engineering LibreTexts. (2019). *6: Control Structures*. [online] Available at: https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_Languages/Java_Java_Java_-_Object-Oriented_Programming_(Morelli_and_Walde)/06%3A_Control_Structures [Accessed 9 Apr. 2023].

jython.readthedocs.io. (n.d.). *Chapter 7: Exception Handling and Debugging — Definitive Guide to Jython latest documentation*. [online] Available at: https://jython.readthedocs.io/en/latest/ExceptionHandlingDebug/ [Accessed 9 Apr. 2023].

nus-cs1010.github.io. (N.D.). *2. Computational Problem & Algorithms - CS1010 Programming Methodology*. [online] Available at: https://nus-cs1010.github.io/2021-s1/02-algo.html [Accessed 9 Apr. 2023].