7. Computational Thinking and Problem-solving

19.1 Algorithms

Binary Search

- **Necessary condition – the elements in the list being searching must be ordered/sorted in ascending/descending order**
- **The time to search a list increases with an increasing number of items in the list**
- **Starts in the middle of the array/list**
- **Works by finding the mid-point of an array/list and determines which side contains the item to be found – it discards the half of the array/list not containing the search item**

Process

- **Find middle index/item**
- **Check the value of the middle index in the list**
- **The item searched for has been found if the index value is equal to it**
- **Otherwise, discard half the list that does not contain the searched item**
- **Repeat the previous steps until the item searched for is found, or there is only one index left in the list and it is not the item searched for**

Linear Search

- **sequentially checks each element of the array/list until the matching element is found, or the end of the array/list is reached.**
- **does not require the elements to be sorted.**
- **will usually do more comparisons of records/iterations against the target (before finding it) than a binary search**
- **starts at the beginning of the array/list.**

Binary VS. Linear Searches

- **time to search increases linearly in relation to the number of items in the list for a linear search and logarithmically for a Binary search**
- **time to search increases less rapidly for a binary search and time to search increases more rapidly for a linear search**

Big O Notation

- **is used to indicate the time/space complexity of an algorithm.**
- ***Linear search →O(n)***
- ***Binary search →O(log 2n) / O(Log n)***

- **O(log n) is a time complexity that uses logarithmic time – the time taken goes up linearly as the number of items rises exponentially**

Performance of a sorting algorithm is affected by…

- **Initial order of data**
- **Number of data items to be sorted**
- **The efficiency of the sorting algorithm**

Queue

- **Uses FIFO data structure – data is removed in the order it is received**
- **Is of varying length**
- **Data is 'enqueued' and 'dequeued' at different ends**
- **Has two movable pointers**

Stack

- **Uses FILO data structure – data is removed in the reverse order to which is it received**
- **Is of varying length**
- **Data is popped and pushed onto/off a stack from the same end**
- **Has one movable pointer**

Uses of a Stack

- **Recursion**
- **Implementation of ADTs**
- **Procedure calls**
- **Interrupt handling**
- **Evaluating an RPN expression**

Linked List Implementation

- **Is a dynamic data structure (not restricted in size)**
- **Has freedom to expand or contract – by adding/removing nodes as necessary**
- **Allows more efficient editing using pointers than an array**

Array Implementation

- **Is a static data structure, generally fixed in size**
- **When array is full, stack cannot be extended any further**

19.2 Recursion

Recursion

- **A process using a function/procedure that calls itself**
- **Must have a base case (stopping condition)**
- **Must have a general case – which calls itself recursively, changes its state and moves towards the base case**

Why Stacks are Effective for Implementation

- **Stacks have a LIFO data structure**
- **Each recursive call is pushed to the stack and is then popped as function ends**
- **Enables backtracking/unwinding to maintain the required order**

Winding and Unwinding

- ***Winding – the statements following a recursive function call are not executed until the general case has reached base case***
- ***Unwinding – occurs once the base case has been reached***

Benefits of Recursion

- ***Can require fewer programming statements than an iterative approach (shorter and more effective code)***
- ***Can solve complex problems in a simpler way than an iterative solution could / provides simpler solutions to problems***

Drawbacks

- ***Very repetitive and requires more storage space (demanding use of stack), meaning stack overflow can occur***
- ***Infinite recursion can occur***

Translating Recursive Programming Code

- **The compiler must produce object code to**
  - **➔ Push return addresses/values of local variables onto a stack with each recursive call**
  - **➔ Pop return addressed/values of local variables off the stack after the base case is reached**