Sr. Software Engineer (SSE)

ABSTRACT
This is one of the subject from my personal notes series named “Coding-With-Arqam” that I am developing from the start of my professional development career.

Subject
Data Structure

DATA STRUCTURE

--> Why to Learn Data Structure and Algorithms?

-> As applications are getting complex and data rich, there are three common problems that applications face now-a-days.

-> Data Search:

-> Consider an inventory of 1 million(106) items of a store.

-> Processor speed:

-> Processor speed although being very high, falls limited if the data grows to billion records.

-> Multiple requests:

-> As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

--> Applications of Data Structure and Algorithms:

-> Search , Sort , Insert , Update , Delete

--> Foundation terms:

-> Interface:

-> Each data structure has an interface.

-> Interface represents the set of operations that a data structure supports.

-> An interface only provides the list of supported operations, type of parameters they can accept and return type of these operations.

-> Implementation:

-> Implementation provides the internal representation of a data structure.

-> Implementation also provides the definition of the algorithms used in the operations of the data structure.

--> Characteristics of a Data Structure:

-> Correctness:

-> Data structure implementation should implement its interface correctly.

-> Time Complexity:

-> Running time or the execution time of operations of data structure must be as small as possible.

-> Space Complexity:

-> Memory usage of a data structure operation should be as little as possible.

--> Execution Time Cases:

-> Worst Case

-> Average Case

-> Best Case

--> Basic Terminology:

-> Data − Data are values or set of values.

-> Data Item − Data item refers to single unit of values.

-> Group Items − Data items that are divided into sub items are called as Group Items.

-> Elementary Items − Data items that cannot be divided are called as Elementary Items.

-> Attribute and Entity − An entity is that which contains certain attributes or properties, which may be assigned values

-> Entity Set − Entities of similar attributes form an entity set.

-> Field − Field is a single elementary unit of information representing an attribute of an entity.

-> *Record − Record is a collection of field values of a given entity.*

-> *File − File is a collection of records of the entities in a given entity set.*

--> *Asymptotic Notations:*

-> *Types:*

-> *O Notation:*

-> *Big Oh*

-> *Upper bound of an algorithm's running time*

-> *It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.*

-> *Ω Notation:*

-> *Omega*

-> *Lower bound of an algorithm's running time.*

-> *ϑ Notation:*

-> *Theta*

-> *Both the lower bound and the upper bound of an algorithm's running time.*

-> *Example:*

-> *constant = O(1)*

-> *logarithmic = O(log n)*

-> *linear = O(n)*

--> *Greedy Algorithms:*

-> *Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions.*

-> *Example (Counting):*

-> *We have 1, 7, 10 number and we have to give 15 as sum result.*

-> *Approach 1: 10+1+1+1+1+1 (6 numbers used)*

-> *Approach 2: 7+7+1 (3 numbers used)*

--> *Divide and conquer approach:*

-> *Divide/Break*

-> *Conquer/Solve*

-> *Merge/Combine*

-> *Example:*

-> *Merge Sort*

-> *Quick Sort*

-> *Binary Search*

-> *Strassen's Matrix Multiplication*

-> *Closest pair (points)*

--> *Data Type:*

-> *Built-in: Integers, Boolean , Floating , Character and Strings*

-> *Derived: List, Array, Stack, Queue*

--> *Basic Operations:*

-> *Traversing, Searching, Insertion, Deletion, Sorting, Merging*

--> *Array:*

-> *Can hold a fix number of items and these items should be of the same type.*

-> *Array Representation:*

-> Type Name[Size] = {Elements}

--> Linked List:

-> A linked list is a sequence of data structures, which are connected together via links.

-> Linked List Representation:

-> Head -> NODE(Data Items -> Next) -> NODE(Data Items -> Next) -> NODE(next of last node will be null)

-> Types:

-> Simple Linked List − Item navigation is forward only.

-> Doubly Linked List − Items can be navigated forward and backward.

-> Last item contains link of the first element as next

--> Stack:

-> LIFO, FILO

-> Basic Operations:

-> push()

-> pop()

-> peek() = get the top data element

-> isFull() − check if stack is full.

-> isEmpty() − check if stack is empty.

--> Expression Parsing:

-> Infix Notation:

-> Example:

-> a - b = a - b

-> Prefix Notation:

-> Example:

-> a + b = +ab

-> Postfix Notation:

-> Example:

-> a + b = ab+

--> Queue:

-> A queue can also be implemented using Arrays.

-> Basic Operations:

-> enqueue() − add (store) an item to the queue.

-> dequeue() − remove (access) an item from the queue.

-> peek(), isfull(), isempty()

--> Linear Search:

-> worst-case complexity of O(n)

-> Simple loop iteration N-times.

--> Binary Search:

-> worst-case complexity of O(log n)

-> For a binary search to work, it is mandatory for the target array to be sorted

-> mid = low + (high - low) / 2

-> Divide array in to two portions and then select one portion and do the same.

--> Interpolation Search:

-> O(????)

-> Similar to binary search for searching for a given target value in a sorted array.

--> Hash Table:

-> Hashing:

-> Hashing is a technique to convert a range of key values into a range of indexes of an array.

-> Linear Probing:

-> As we can see, it may happen that the hashing technique is used to create an already used index of the array. In such a case, we can search the next empty location in the array by looking into the next cell until we find an empty cell.

--> Sorting techniques:

-> Example (real time):

-> Telephone Directory

-> Dictionary

-> Stable sorting:

-> If a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear.

-> Example:

-> Input: 2, 3(first), 5, 3(second)

Outout: 2, 3(first), 3(second), 5

-> Un-Stable sorting:

-> changes the sequence of similar content in which they appear.

-> Example:

-> Input: 2, 3(first), 5, 3(second)

Outout: 2, 3(second), 3(first), 5

--> Bubble Sort Algorithm:

-> O(n log n)

-> Example (ascending sorting):

-> Input: 14, 33, 27, 35, 10

-> Iteration 1:

-> Step 1: Compare 14, 33 => already sorted => 14,33,27,35,10

-> Step 2: Compare 33, 27 => Swap => 14,27,33,35,10

-> Step 3: Compare 33, 35 => already sorted => 14,27,33,35,10

-> Step 4: Compare 35, 10 => Swap => 14,27,33,10,35

-> Iteration 2:

-> Final step: 14,27,10,33,35

-> Iteration 3:

-> Final step: 14,10,27,33,35

-> Iteration 4:

-> Final step: 10,14,27,33,35

-> Algorithm:

```
for all elements of list
    if list[i] > list[i+1]
        swap(list[i], list[i+1])
    end if
```

end for

--> Merge Sort:

-> O(n log n)

-> Divides the whole array iteratively into equal halves unless the atomic values are achieved.

-> Example:

-> Input: 14,33,27,,10,35,,19,42,44

-> Step 1: 14,33,27,10 | 35,19,42,44

-> Step 2: 14,33 | 27,10 | 35,19 | 42,44

-> Step 3: 14 | 33 | 27 | 10 | 35 | 19 | 42 | 44

-> Now, we combine them in exactly the same manner as they were broken down by arranging the pairs.

-> Step 4: 14,33 | 10,27 | 19,35 | 42,44

-> Step 5: 10,14,27,33 | 19,35,42,44

-> Step 6: 10,14,19,27,33,35,42,44

--> Quick Sort:

-> O(n logn)

-> Step 1 − Choose the highest index value as 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

-> Link: https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_algorithm.htm

--> Insertion Sort:

-> O(n^2)

-> It swaps the element and also checks with all the elements of sorted sub-list (array left to the current index).

-> Example:

-> Input: 14,33,27,10,35,19,42,44

-> Iteration 1:

-> Step 1: Compare 14, 27 (already sorted) => 14,33,27,10,35,19,42,44 => Sub-List = [14] (already sorted)

-> Iteration 2:

-> Step 1: Compare 33,27 => Swap => 14,27,33,10,35,19,42,44 => Sub-List = [14,27] (already sorted)

-> Iteration 3:

-> Step 1: Compare 33,10 => Swap => 14,27,10,33,35,19,42,44 => 14,10,27,33,35,19,42,44 => 10,14,27,33,35,19,42,44

-> Iteration 4:

-> This process goes on until all the unsorted values are covered in a sorted sub-list.

--> Selection Sort:

-> O(n^2)

-> List is divided into two parts, the sorted part at the left end and the unsorted part at the right end.

-> The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array.

-> Example:

-> Input: 14,33,27,10,35,19,42,44

-> Step 1: Compare 14 and 10 => Swap 10 and 14 => Sorted array: [10] => 10,33,27,14,35,19,42,44

-> Step 2: Compare 14 and 33 => Swap 14 and 33 => Sorted array: [10,14] => 10,14,27,33,35,19,42,44

-> Step 3: Compare 14 and 10 => Swap 10 and 14 => Sorted array: [10,14,33] => 10,33,27,14,35,19,42,44

-> Step 4: The same process is applied to the rest of the items in the array.

-> Step 5: Final Array: [10,14,19,27,33,35,42,44]

--> Shell Sort:

-> Worst case: Depends on gap sequence

*-> Average complexity: n*log(n)^2 or n^(3/2)*

-> Best casae: n

-> Based on insertion sort algorithm

-> Link: https://www.tutorialspoint.com/data_structures_algorithms/shell_sort_algorithm.htm

--> Graph Data Structure Traversal:

-> Depth First Search(DFS):

-> Uses Stack.

-> Rules/Steps:

-> Step 1 − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

-> Step 2 − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)

-> Step 3 − Repeat Rule 1 and Rule 2 until the stack is empty.

-> Applications:

-> Path finding algorithm is based on BFS or DFS.

->

-> Link: https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm

-> Breadth First Search(BFS):

-> Uses Queue.

-> Rules/Steps:

-> Step 1: Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.

-> Step 2: If no adjacent vertex is found, remove the first vertex from the queue.

-> Step 3: Repeat Rule 1 and Rule 2 until the queue is empty.

-> Link: https://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm

-> Applications:

-> Path finding algorithm is based on BFS or DFS.

-> In peer-to-peer network like bit-torrent, BFS is used to find all neighbor nodes

-> Search engine crawlers are used BFS to build index. Starting from source page, it finds all links in it to get new pages

-> Using GPS navigation system BFS is used to find neighboring places.

-> In networking, when we want to broadcast some packets, we use the BFS algorithm.

--> Tree:

-> Binary Tree:

-> Each node can have a maximum of two children.

-> Important Terms:

-> Path, Root , Parent , Child , Leaf , Subtree , Visiting , Traversing , Levels, keys

-> BST Basic Operations:

-> Insert − Inserts an element in a tree/create a tree.

-> Search − Searches an element in a tree.

-> Preorder Traversal − Traverses a tree in a pre-order manner.

-> Steps:

-> Step 1 – Visit root node.

-> Step 2 – Recursively traverse left subtree.

-> Step 3 – Recursively traverse right subtree.

-> Syntax: N-L*-R*

-> Inorder Traversal – Traverses a tree in an in-order manner

-> The left subtree is visited first, then the root and later the right sub-tree.

-> Steps:

-> Step 1 – Recursively traverse left subtree.

-> Step 2 – Visit root node.

-> Step 3 – Recursively traverse right subtree.

-> Syntax: L*-N-R*

-> Postorder Traversal – Traverses a tree in a post-order manner.

-> Steps:

-> Step 1 – Recursively traverse left subtree.

-> Step 2 – Recursively traverse right subtree.

-> Step 3 – Visit root node.

-> Syntax: L*-R*-N

--> Binary Search Tree:

-> A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties:

-> The value of the key of the left sub-tree is less than the value of its parent (root) node's key.

-> The value of the key of the right sub-tree is greater than or equal to the value of its parent (root) node's key.

-> Syntax: left_subtree (keys) < node (key) ≤ right_subtree (keys)

-> Order: (L-N-R)

--> AVL Trees:

->

->

--> Spanning Tree:

->

->

--> Heap:

-> Special case of balanced binary tree data structure

where the root-node key is compared with its children and arranged accordingly.

-> Min-Heap:

-> Where the value of the root node is less than or equal to either of its children.

-> Max-Heap:

-> Where the value of the root node is greater than or equal to either of its children.

-> Steps:

-> Step 1 – Create a new node at the end of heap.

-> Step 2 – Assign new value to the node.

-> Step 3 – Compare the value of this child node with its parent.

-> Step 4 – If value of parent is less than child, then swap them.

-> Step 5 – Repeat step 3 & 4 until Heap property holds.

-> Max Heap Deletion Algorithm:

-> Step 1 − Remove root node.

-> Step 2 − Move the last element of last level to root.

-> Step 3 − Compare the value of this child node with its parent.

-> Step 4 − If value of parent is less than child, then swap them.

-> Step 5 − Repeat step 3 & 4 until Heap property holds.

--> Recursion:

-> Properties:

-> Base criteria

-> Progressive approach

-> Uses Stack.

-> Time Complexity: O(n)

-> Example:

-> Tower of Hanoi:

->

->

-> Fibonacci Series:

-> F8 = "0 1 1 2 3 5 8 13" OR "1 1 2 3 5 8 13 21"

->

*--> Notations: (a + b * c + d) -> [a + (B * c) + d]*

*Prefix Notation −> * + a b + c d*

*Postfix Notation −> a b + c d + **

// Always follow DMAS rule.

Abbrevations:

MEAN = mongo, express, angular, node.

Reference Links

- *https://www.tutorialspoint.com/data_structures_algorithms/pdf/data_structures_algorithms_interview_questions.pdf*
- *https://www.toptal.com/algorithms/interview-questions*