Data Structure: Theoretical Approach

INTRODUCTION

- Data structures serve as the basis for abstract data types (ADT).
- Data structures manage large amounts of data efficiently for uses such as large databases and internet indexing services.
- Efficient data structures are key to designing efficient algorithms.
- Organizes the storage and retrieval of information, both in main memory and secondary memory
- Array and record data structures are based on computing the addresses of data items with arithmetic operations. Linked data structures are based on storing addresses of data items within the structure itself.
- The implementation of a data structure requires a set of procedures that create and manipulate instances of that structure.
- High-level programming languages and some higher-level assembly languages have special syntax or built-in support for certain data structures, like records and arrays.

SEQUENTIAL SEARCH

- Data items stored in a collection such as a list, have a linear or sequential relationship, where each data item is stored in a position relative to the others.
- These relative positions are index values of the individual items in order.
- To visit them in sequence, we have our first searching technique, the sequential search.
- Starting at the first item, we move from item to item, following the underlying ordering until we find the certain item or run out of items, i.e, the item is not present.
- If there are \(n\) items, then the sequential search requires \(n\) comparisons to discover that the item is not there.
- Best case; item found in the first place and worst case; item not found until the last comparison.

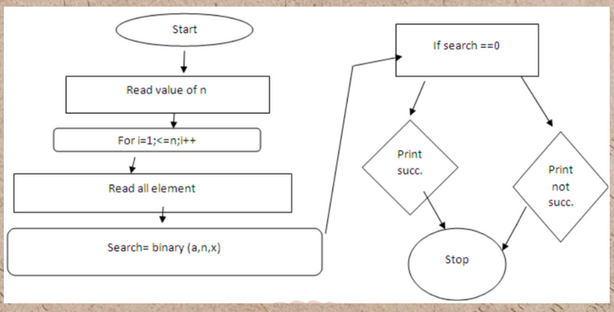
BINARY SEARCH

- Binary search is a fast search algorithm with run-time complexity of O(log n), this works on the principle; divide and conquer.
- The data collection should be in the sorted form.
- Binary search looks for an item by comparing to the middle item. If it's found, then index of item is returned.
- If middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub- array to the right of the middle item.
- This process continues on the sub-array as well until the size of the sub array reduces to zero.
- Due to the variable range of their node length, B-trees are optimized for systems that read large blocks of data. eg: usually used in databases.

Binary Search Tree is a node-based tree data structure whose properties are:
1) Node to be deleted is leaf (remove from tree)
2) Node to be deleted has only one child; copy child to node & delete child
3) Node to be deleted has two children; copy contents of order successor to the node & then delete it.

BINARY SEARCH TREE PERFORMANCE

- Adelson, Velski & Landis, (AVL) trees are height balancing BSTs that perform the following rotations:
  - Left-Right rotation Right-Left rotation
  - Left rotation Right rotation
  - Left Rotation
  - Left-Right Rotation Double rotations



Internal sorting algorithms include:
1) Bubble Sort
2) Insertion Sort
3) Quick Sort Heap
4) Sort Radix Sort
5) Selection sort

Comparison between linear search and binary search

Basis for comparison	Linear search	Binary search
Time complexity	O(N)	O($log2N$)
Best case time	First element 0(1)	Centre element 0(1)
Prerequisite for an array	Not required	Array must be sorted in order
Worst case for N number of elements	N comparisons are required	Can conclude after only ($logN$)
Can be implemented on	Array and linked list	Cannot be directly implemented to linked list
Algorithm type	Iterative in nature	Divide and conquer in nature
Insert operation	Easily inserted t end of list	Require processing to insert at its proper place to maintain a sorted list
Usefulness	Easy to use	Tricky algorithms
Lines of code	less	More

CONCLUSION

- This paper covered the basics of data structures.
- Each data structure has its pros & cons. It must be used according to the needs of the application.
- High level & OOP languages like C#, Java, Python come built in with many data structures. Thus, it is important to know how it works.
- Fundamentals of storage management are important as they have significant impact on the behaviour of programs.
- The concept of pointers or pointer variables underlies the use of these facilities, & complex algorithms are required for implementation.
- We can conclude by comparing the two searching techniques based on important factors.

C034 -Reneeka Nadkarni
C036 - Murtaza Hamid