Name - Briyanshu Gupta
Section - S11

Enrollment No. - 04115611922
B. Tech {AI & DS}

Data Structures

Assignment - 3

Q1 (a) Difference between Data Structure and Algorithms with example.

Ans -

Aspect	Data Structures	Algorithms
Defination	The way data is organised and retrieved and stored.	A set of instructions used to solve a specific problem or perform a specific task.
Purpose	Provides an efficient way to organise and store data for easy retrieval and modification	Provides a systematic approach to solving problems by breaking them down into smaller, more managable steps.
Operations	Insertion, Deletion, Search Update, Traverse, etc.	Sorting, Searching, Optimization, Path finding etc.
Examples	Array, Linked List, Stack, Queue, Tree, Graph, Hash Table, etc.	Graph Transversal, Sorting, Searching, Dynamic programming, Divide and Conquer, etc.

(b) Explain two disadvantages of Linked List over array?

Ans - Disadvantage of Linked List over Array

(1.) Linked List uses more memory because it stores both data and the address of the next node as compared to Array

(2.) Elements accessing requires the traversal of whole Linked list and hence not easily accessed as compared to Array.

(3.) Implementing a Linked List is more complex than implementing an array because it requires managing pointers and dynamically allocating memory.

(c) Explain Concept of algorithm complexity for binary Search.

Ans The time Complexity of an algorithm is a measure of how the running time of the algorithm increases as the size of the input increases. Binary Search is a divide and Conquer algorithm that works by repeatedly dividing the search in half until the target element is found.

The time Complexity of binary search is 'O(logn)', where n is the number of elements in the search space. This means that the running time of binary search increases logarithmically with the size of the search space.

(d) Explain Concept and application of Circular Linked List.

Ans Circular Linked List –

A circular linked List is a type of Linked List in which the last node points back to the first node, forming a loop or circle. This means that there is no head or tail node, and the list can be transversed in either direction.

Application of Circular Linked List –

1) Used in Circular Buffers, which are used to store data that is being processed in a circular fashion.
2) Multiplayer games uses a circular list to swap between the players in loop.
3) Music or Media player: Circular linked list can be used to create a playlist for a music, where each song is node and next node points to next song. This allows for continuous playback.

(e) Explain two operations of Strings with example.

Ans- Two operations of Strings -

① Concatenation: This operations joins two or more strings together. to form a new string.

For eg "Hello" and "Word" forms "Hello World"

```
char str1[ ] = "Hello";
char str2[ ] = "World";
char str3[50]; = strcat(str1, str2);
printf(Str3);
```

Output → "Hello World"

② String length: The strlen() Function calculates the length of a given string.

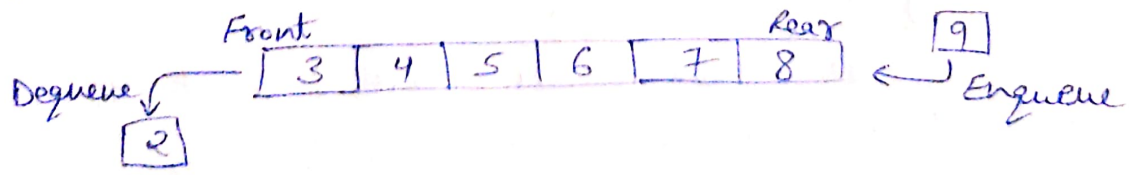
For eg. `str[ ] = "Hello";`
`length = strlen(str);`
`printf(length);`

Output → 5

Q2 (a) What is Queue Data Structure explain with example.

Ans A Queue is defined a linear data structure that is open at both ends and the operations are performed is First in First Out (FIFO) order.

This means that the element is added to the queue first is also the element that is removed first.



For eg. We can take the example of a row of Students in School's prayer line, where queue start adding from first and also removed from the first.

2(b) Explain features of list, set, tuple, and dictionary data structures.

Ans - List -
- It is a non-homogeneous data structure that
- Stores elements in row or multiple rows.
- It is represented by [ ]
- It allows duplicate elements
- For e.g [1, 2, 3, 4, 5]
- A list can be created using list() function
- A list is mutable i.e we can make any changes.
- It is ordered.

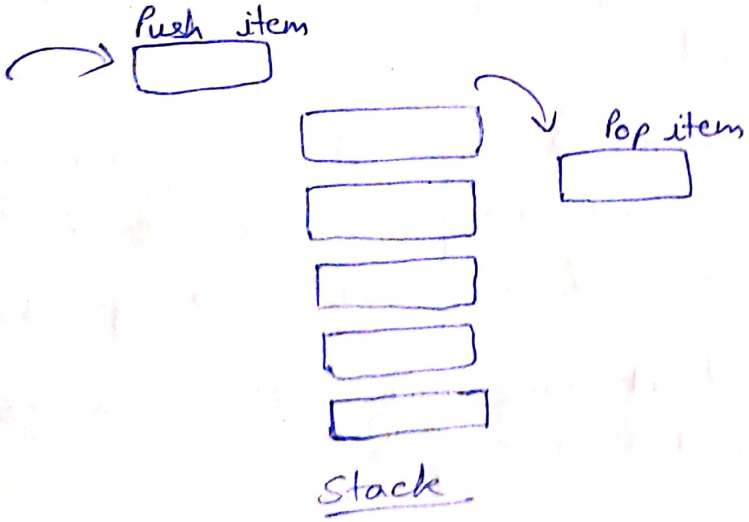
Tuple -
- It is non-homogeneous data structure that
- Stores elements in row or multiple rows.
- It is represented by ( ).
- It allows duplicate elements
- For eg (1, 2, 3, 4, 5)
- Tuple can be created using tuple() function.
- A Tuple is immutable ie we cannot make any changes.
- It is ordered.

Set -
- The Set data Structure is non-homogeneous collection but stores the data in a single row.
- It is represented by { }
- The Set will not allow duplicate elements.
- For eg {1, 2, 3, 4, 5}
- It can be created using set() function.
- A Set is mutable i.e we can make any changes.
- Elements are not duplicated in set.
- It is Unordered.

Dictionary – • It is non-homogeneous data structure that stores key-value pairs.
• It is represented by { }
• It doesn't allow duplicate keys.
• Example: {1:"a", 2:"b", 3:"c", 4:"d"}
• It can be created using dict() Function
• It is mutable.
• Dictionary is ordered.

2(c) What is Stack Data Structure? Considering Array representation of Stack write and explain algorithm for Push and Pop Operations?

Ans It is a type of Linear Data Structure. The Last in First Out (LIFO) concept is used by Stacks. It has only one pointer (top pointer), which points to the Stack's topmost member.
A Stack is a container that allows insertion and deletion from the end known as the stack's top.



Algorithm for the Push operation
Adds an item to the stack. If stack is full it is said to be an Overflow condition.

(1) Check if the stack is full. If it is full then return an error and exit.
(2) Increment the stack pointer if not full.
(3) Store the new element at the top of the stack.

Algorithm for Pop Operation

Removes an item from the stack. The item are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow Condition.

(1) Check if the stack is empty. If it is empty, then return error and exit.
(2) Decrement the stack pointer if not empty
(3) Return the element at the top of the stack.

Question-3

(a) Explain two advantages of using linked list over array.

Ans Advantages of Linked List -

(1) Linked List is Dynamic in size. It can grow or shrink dynamically, while array have a fixed size that is determined at the time of creation.

(2) Memory is allocated at run time while in array memory is allocated at compile time.

(3) Insertion and Deletion operation is efficient and faster in Linked list. This is because only the pointers of the affected node to be changed, regardless of the size of the list.

3(b) Write the process of Radix sort with an example?

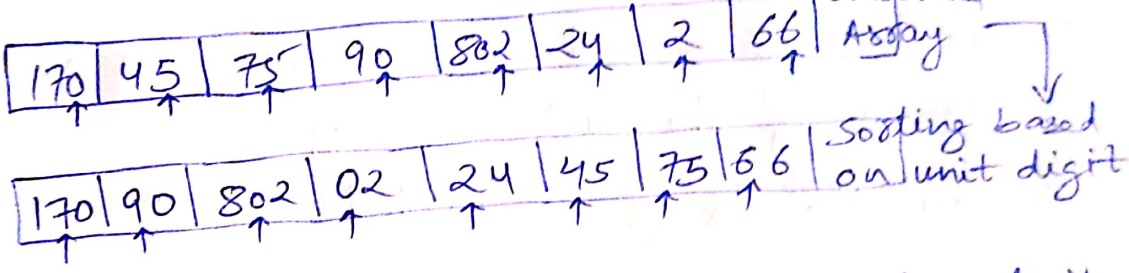
Ans - Algorithm of Radix Sort

Step 1: Find the largest element in the array, which is 802. It has three digit, so we iterate three times, once for each significant place.

170	45	75	90	802	24	2	66

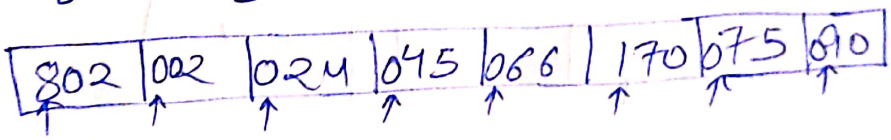
Unsorted Array

Step 2: Sort the element based on unit place by performing counting sort technique on the array.



Step 3: Now sort the element based on tens place digit by using Counting sorting technique.

Sorting on 10's digit →



Step 4: Sort the elements base on Hundreds place using Counting Sort on previous obtained array

2	24	45	66	75	90	170	802

Sorting till 100's digit

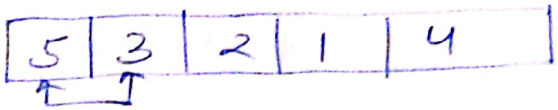
Step 5: The array is now sorted in ascending orders The final array after performing Radix sort for all digits.

2	24	45	66	75	90	170	802

3(c) Write algorithm for Insertion Sort and Quick Sort. Compare their Complexities.

Ans - * Algorithm for Insertion Sort

① Start with the second element in the Array and Compare.

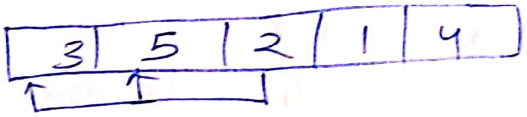


② Compare the current element with the elements before it. Here, 3 is smaller than 5

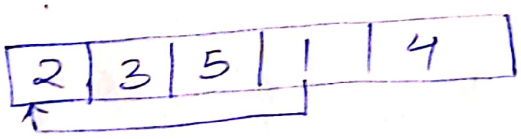
③ If the current element is smaller than the element before it, swap them.

④ Repeat steps 2 and 3 until you reach the begining of the list or Array

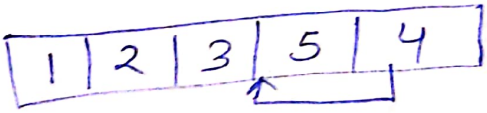
⑤ Now, the Array is Sorted



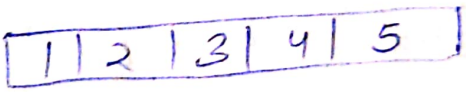
2 is smaller than 5 and 3, so swap



1 is smaller than 5, 3 and 2, so swap



4 is smaller than 5, Swap it



→ final Sorted Array

* Algorithm for Quick Sort

Step ① → Choose a pivot element from the array (say most right)

Step ② → Partition the array using pivot value

Step ③ → Take two variables: 'left' points to the low index
'right' points to the high index

Step ④ → If value of array at left is less than pivot move right or If value at right is greater than pivot move left.

Step(5.) → If step 4, does not match swap left and Right.

Step(6.) → if left ≥ right the point where they met is new pivot.

Step(7.) → Recursively Quick sort the left and Right partition.

Step(8.) → Combined the sorted subarray to form a Sorted array.

Algorithm	Time Complexities			Space Complexities
	Best	Average	Worst	Worst
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Quick Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(1)$

Ques 4

(a) Write Algorithm for infix to postfix conversion? Explain the Conversion of expression ((A*B)+(C/D)) using the same?

Ans **Algorithm for Infix to postfix**

Step 1 → Start Scanning from left to Right and repeat Step 2 to 6 for each element of the expression until stack is empty.

Step2 → Start scanning from left to right, If operand is encountered add it to the target string

Step3 → If left parenthesis is encountered, push it onto stack.

Step4 → If an operator is encountered then,

a) Add the operator to the stack

b) If new operator has less or equal precedence, then Pop it and add it to the target string and put new operator on to the top of stack.

Step (5) → If right parenthesis is encountered, then
a) Pop from stack and add to target string till a left parenthesis encountered.
b) Remove the left parenthesis

Step (6) → Exit

Expression ((A*B)+(C/D))

character	Stack	Expression String
(	(	
(	((	
A	((	A
*	((*	A
B	((*	AB
)	(	AB*
+	(+	AB*
(	(+(	AB*
C	(+(	AB*C
/	(+(/	AB*C
D	(+(/	AB*CD
)	(+	AB*CD/
)	empty	AB*CD/+

4(b) Write algorithm for inserting and deleting a given node in the middle of a doubly linked list?

Add a node between two nodes in a Doubly Linked List

We are given a pointer to a node as 'prev_node' and the new node is inserted after a given node.
This can be done using these steps:

Step 1 → Firstly create a new node (say new_node)

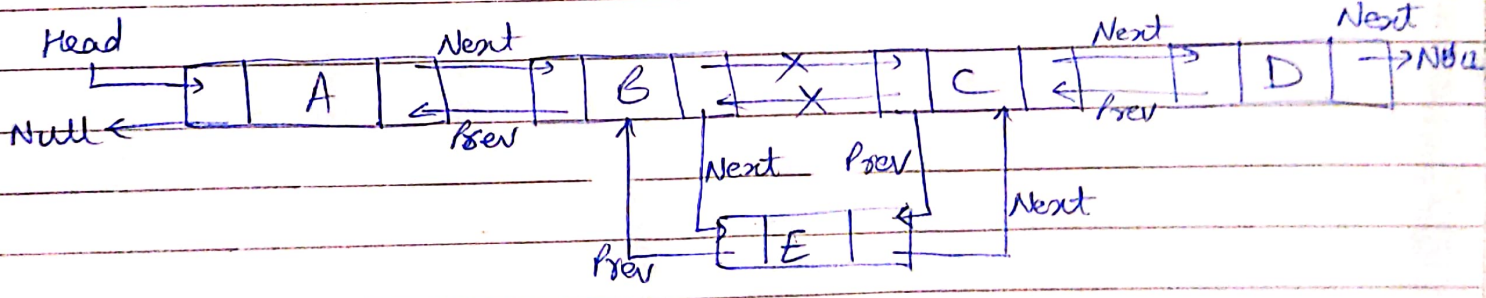
Step 2 → Now, Insert the data in a new node

Step 3 → Point the next of 'new_node' to the next of 'prev_node'

Step 4 → Point the next of 'prev_node' to 'new_node'

Step 5 → Point the previous of 'new_node' to 'prev_node'

Step 6 → Change the pointer of the new node's previous pointer to 'new_node'



Deletion at middle of Doubly Linked List

Step 1 → Traverse till the target node

Step 2 → Create a node called the previous storing previous node of the target node.

Step3 → Assign previous node's next pointer to the next node of the target node.

Step4 → For the next node of target node, its previous pointer is assigned to the targets node's previous node's address.

Step5 → Free memory of target node.