FACULTY OF ENGINEERING & TECHNOLOGY
PARUL INSTITUTE OF ENGINEERING & TECHNOLOGY (DIPLOMA STUDIES)
COMPUTER ENGINEERING DEPARTMENT

3RD SEMESTER
QUESTION BANK WITH SOLUTION
DATA STRUCTURES
03606201

UNIT – 1: INTRODUCTION TO DATA STRUCTURES

1. What is Data structure?
2. Classification of Data Structures.
3. What is Primitive and non-primitive data structures?
4. Explain linear and non-linear data structures?
5. What is an Algorithm? And its Key feature?
6. What is Time-Complexity and Space-complexity?
7. Define Worst-Case, Average-Case, and Best-Case Analysis?

UNIT – 2: LINEAR DATA STRUCTURE-STACK

8. What is Stack? List out the basic operations of Stack.
9. Write down the algorithm of peek(), isFull(), and isEmpty().
10. Write down the algorithm of push() and pop().
11. Write the application of stack.
12. Explain Infix, Prefix, and Postfix expression?
13. What is a parsing expression?
14. What is precedence and associativity?
15. Explain the Tower of Hanoi?
16. Explain asymptotic notation?

UNIT – 3: LINEAR DATA STRUCTURE – QUEUE

17. What is Queue?
18. Explain basic operations of Queue?
19. Give the ways of implementing Queue?
20. Algorithm for Enqueue and Dequeue?
21. What is a linear Queue? Explain its limitations.
22. What is a circular queue?
23. Application of Queue?
24. Difference between a linear queue and a circular queue?

UNIT – 4: LINKED LIST

25. What is Pointer? How we can use it?
26. Define Structure.
27. Explain Dynamic memory operations.
28. What is Linked list? List types of linked list.
29. Explain Insertion operation in singly linked list.
30. Explain Deletion operation of singly linked list.
31. Exaplin Circular Linked list concepts.

UNIT – 5: SEARCHING AND SORTING

32. What is sorting and type of sorting?
33. Explain bubble sort with example and write an algorithm of bubble sort.
34. Explain selection sort with example and write an algorithm of selection sort.
35. Explain quick sort with example and write an algorithm of quick sort.
36. Explain merge sort with example and write an algorithm of merge sort.
37. Explain insertion sort with example and write an algorithm of insertion sort.

UNIT – 6: NON-LINEAR DATA STRUCTURES – TREE AND GRAPH

38. What is a non-linear datastructure?
39. What is a tree? And how many types of trees?
40. What is a Binary search tree?
41. What is a tree traversal?
42. What are Inorder, Preorder, and Postorder Traversal?
43. Write down application of Binary Tree?
44. What is conversion of the general tree to binary tree?

CHAPTER – 1 INTRODUCTION TO DATA STRUCTURE

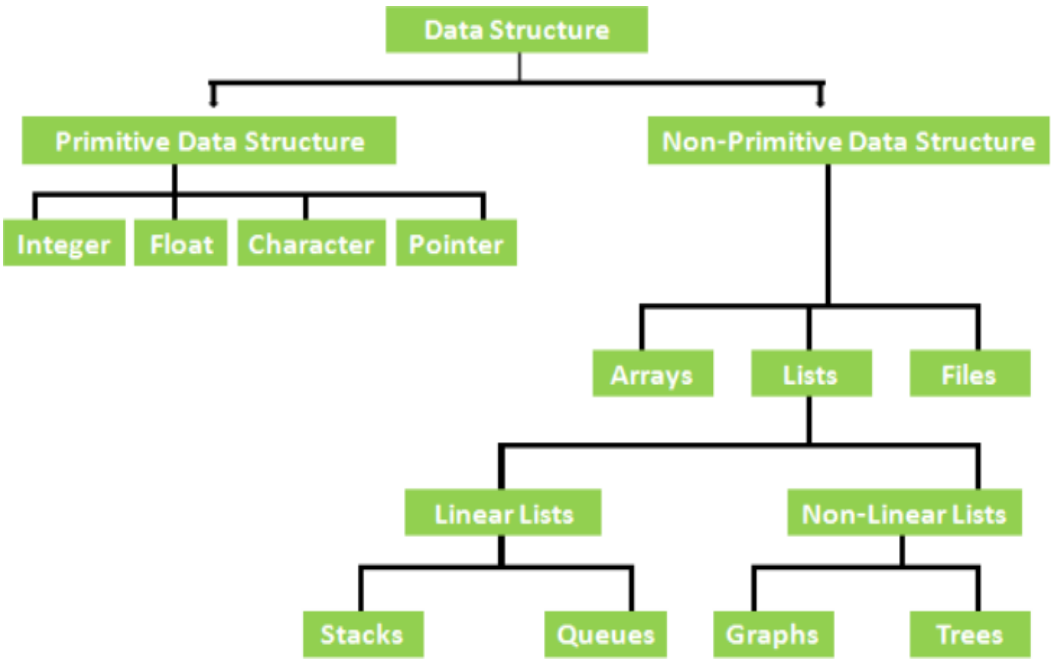
Q-1. What is Data structure?[2 Marks]

Ans:- Data Structure is a systematic way to organize Data.

- It helps arrange data elements in computer memory in a way that they could be retrieved faster, easier, and more efficiently.

Q-2. Classification of Data Structures? [2 Marks]

Ans:-



Q-3. What are Primitive and non-primitive data structures? [3 Marks]

Ans:- Primitive Data structures are Basic data structures and are directly operated by Machine instruction.

- EX:- Integer, Float, Character, andPointer.

- Non-primitive data structures are more complicated data structures and are derived from primitive data structures. They emphasize the grouping of the same or different data items with the relationship between each item.
  - Ex:- Array, list, and Files.

Q-4. Explain linear and non-linear data structures? [4 Marks]

Ans:-

- A data structure is said to be linear if its elements are connected in a linear fashion by means of logically or in sequence memory locations.
    - Ex:- Stack and Queue

- In Non – Linear Data Structures, the data items are not in sequence.
    - Ex:- Tree and Graph

Q-5. What is an Algorithm? And its Key feature? [2 Marks]

Ans:- An Algorithm is a step-by-step procedure for solving a particular problem.

- **Key Features:-**
    - Unambiguous
    - Input
    - Output
    - Finiteness
    - Feasibility
    - Independent

Q-6. What is Time-Complexity and Space-complexity? [4 Marks]

Ans:-

- The Time Complexity of an algorithm represents the amount of time required by the algorithm to run to completion.

- The space complexity of an algorithm represents the amount of space or memory an algorithm takes in terms of the amount of input to the algorithm.

Q-7. Define Worst-Case, Average-Case, and Best-Case Analysis? [3 Marks]

Ans:-

- **Worst-Case Analysis**
    - The maximum time required for program execution.
- **Average-Case Analysis**
    - Average time required for program execution.
- **Best-Case Analysis**

Minimum time required for program execution.

CHAPTER – 2 – LINEAR DATA STRUCTURE - STACK

Q-8. What is Stack? List out the basic operations of Stack. [4 Marks]

Ans:- Stack is an Abstract Data Type(ADT), Commonly used in most programming languages. It follows LIFO(Last In First Out), Here, the element which is placed last, is accessed first.

- Basic Operations
  - **PUSH()** – pushing(storing) an element on the stack.
  - **POP()** – Removing(accessing) an element from the stack.
  - **Peek()** – get the top data element of the stack, without removing it.
  - **isFull()** – Check if stack is full.
  - **isEmpty()** – Check if stack is empty.

Q-9. Write down the algorithm of peek(), isFull(), and isEmpty().[4 Marks]

Ans:-

- **peek()**

Algorithm

```
begin procedure peek
return stack[top]
end procedure
```

Implementation of peek() function in C programming language −

Example

```
int peek() {
   return stack[top];
}
```

- **isfull()**

Algorithm

```
begin procedure isfull

   if top equals to MAXSIZE
      return true
   else
      return false
```

```
  endif

end procedure
```

Implementation of isfull() function in C programming language −

Example

```
bool isfull() {
  if(top == MAXSIZE)
    return true;
  else
    return false;
}
```

- **isempty()**

Algorithm

```
begin procedure isempty

  if top less than 1
    return true
  else
    return false
  endif

end procedure
```

Implementation of isempty() function in C programming language is slightly different. We initialize top at -1, as the index in array starts from 0. So we check if the top is below zero or -1 to determine if the stack is empty. Here's the code −

Example

```
bool isempty() {
  if(top == -1)
    return true;
  else
    return false;
}
```

Q-10. Write down the algorithm of push() and pop()[4 Marks]

Ans:-

- **push()**

```
begin procedure push: stack, data

  if stack is full
    return null
  endif
```

```
  top ← top + 1
  stack[top] ← data

end procedure
```

- **pop()**

```
begin procedure pop: stack

  if stack is empty
    return null
  endif

  data ← stack[top]
  top ← top - 1
  return data

end procedure
```

Q-11. Write the application of stack. [2 Marks]

Ans:-

- tower of Hanoi implement
- Expression conversion(Infix to Postfix, Postfix to Prefix, etc.)
- Parsing
- The simplest application of a stack is to reverse a word.

Q-12. Explain Infix, Prefix, and Postfix expression? [3 Marks]

Ans:-

- **Infix**
  - We write the expression in infix notation, e.g. a-b+c, where operators are used in-between operands.
- **Prefix**
  - In this expression, the operator is prefixed to operands, i.e., the operator is written ahead of operands. For example, +ab. This is equivalent to a+b infix notation.
- **Postfix**
  - In this, the operator is post fixed to the operands i.e. the operator is written after the operands. For example, ab+. This is equivalent to a+b infix notation.

Q-13. What is a parsing expression? [2 Marks]

Ans:- As we have discussed, it is not a very efficient way to design an algorithm or program to parse infix notations. Instead, these infix notations are first converted into either postfix or prefix notations and then Computed. To parse any arithmetic expression, we need to take care of operator precedence and associativity also.

Q-14. What is precedence and associativity? [2 Marks]

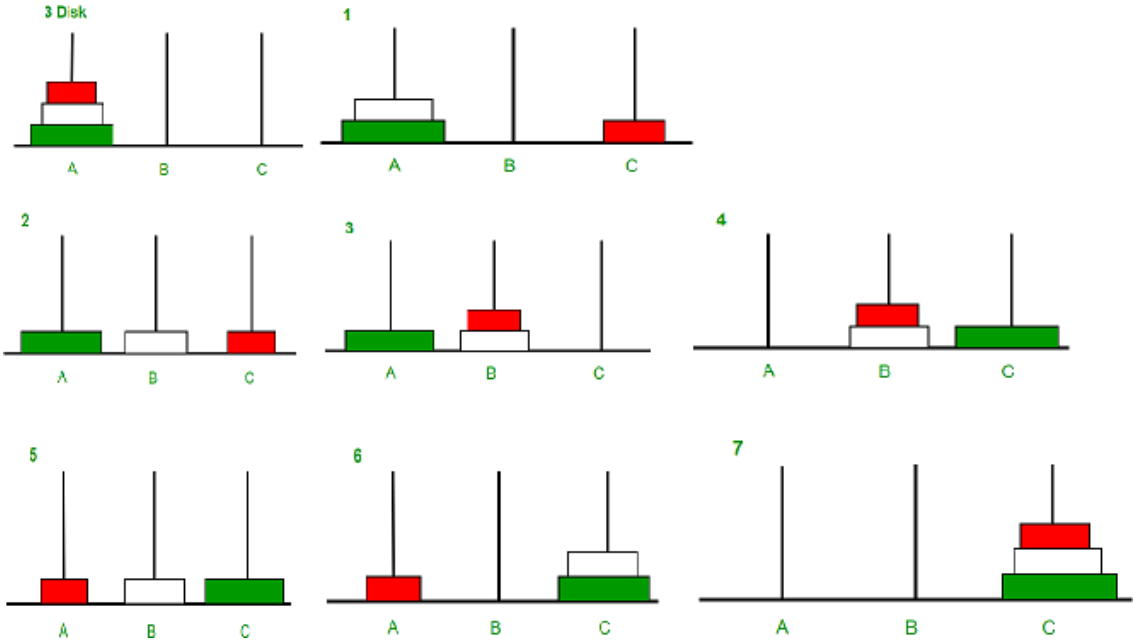
Ans:-

Operator precedence: It dictates the order of evaluation of operators in an expression.

Associativity: It defines the order in which operators of the same precedence are evaluated in an expression.

Q-15:- Explain the Tower of Hanoi? [4 Marks]

Ans:- Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. Each move consists of taking the upper disk from one of the stack sand
placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack. No disk may be placed on top of a smaller disk.



Q-16. Explain asymptotic notation? [2 Marks]

Ans:- Asymptotic Notations are the expressions that are used to represent the complexity of an algorithm.

Ex:- Worst-Case, Average-Case, and Best-case.

CHAPTER – 3 LINEAR DATA STRUCTURE - QUEUE

Q-17. What is Queue? [2 Marks]

Ans:- Queue is an Abstract Data Structure similar to Stack. The queue is open at both ends. One end is always used to insert data and another end is always used to remove data. It follows FIFO(First In First Out).

Q-18. Explain basic operations of Queue. [4 Marks]

Ans:-

- **Basic Operations**
  - **Enqueue() –** add(store) items in the queue.
  - **Dequeue() –** remove(access) an item from the queue..
  - **Peek() –** gets the element at the front of the queue without
  - removing it.
  - **Isfull() –** checks if the queue isfull.
  - **Isempty() –** checks if the queue isempty.

Q-19. Give the ways of implementing Queue? [2 Marks]

Ans:-

- **Sequential allocation:** The sequential allocation in a Queue can be implemented using an array.
- **Linked list allocation:** The linked list allocation in a Queue can be implemented using a linked list.

Q-18. Algorithm for Enqueue and Dequeue? [3 Marks]

Ans:-

- **Enqueue**
  - procedure enqueue(data)

    if queue is full

    return overflow

    Endif

    rear ← rear + 1 queue[rear] ← data return true

    end procedure

- **Dequeue**
  - procedure dequeue

    if queue is empty

    return underflow

    end if

    data = queue[front] front ← front + 1 return true

    end procedure

Q-19. What is a linear Queue? Explain its limitations. [3 Marks]

Ans:- A linear queue is a linear data structure that serves the request first, which has been arrived first. It consists of data elements that are connected in a linear fashion. It has two pointers, i.e., front and rear, where the insertion takes place from the front end, and deletion occurs from the front end.

In a linear queue, the traversal through the queue is possible only once ,i.e., once an element is deleted, we cannot insert another element in its position. This disadvantage of a linear queue is overcome by a circular queue, thus saving memory.

Q-20. What is a circular queue? [3 Marks]

Ans:- Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called 'Ring Buffer'. In a normal Queue, we can insert elements until the queue becomes full.

A circular Queue is also a linear data structure, which follows the principle of FIFO(First In First Out), but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behaves like a circular data structure.

Q-23:- Application of Queue? [2 Marks]

Ans:-

1) When are source is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2) When data is transferred asynchronously (data not necessarily received at the same rateassent)between two processes. Examples include IOBuffers, pipes, file IO, etc.
3) In Operating systems:
   a) Semaphores

b) FCFS (first come first serve) scheduling, for example, FIFO queue
c) Spooling in printers
d) Buffer for devices like keyboard

4) In Networks:
a) Queues in routers/switches Mail Queues

Q-24:- Difference between a linear queue and a circular queue? [3/4 Marks]

Ans:-

Key Points	Linear Queue	Circular Queue
Basic	Organizes data elements and instructions in a sequential order one after the other.	Arranges data in a circular pattern where the last element is connected with the first element.
Order of task execution	FIFO(First-In-First-Out)	The order of task execution may change.
Insertion and deletion	The new element is added from the rear end and removed from the front.	Insertion and deletion can be done at anyplace.
performance	inefficient	Works better than a linear queue.
Memory requirement	Requires more memory.	Requires less memory.

CHAPTER - 4 LINKED LIST

Q-25. What is Pointer? How we can use it? [3 Marks]

Ans:- A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address.

The general form of a pointer variable declaration is –

Type *var-name;

Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations –

```
Int *ip;          /* pointer to an integer */
Double *dp;   /* pointer to a double */
Float *fp;       /* pointer to a float */
Char *ch;      /* pointer to a character */
```

Q-26. Define Structure. [2 Marks]

Ans:- Arrays allow to define type of variables that can hold several data items of the same kind. Similarly structure is another user defined data type available in C that allows to combine data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- Subject
- Book ID

Defining a Structure

To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –

```
struct [structure tag] {

   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

Q-27. Explain Dynamic memory operations. [3 Marks]

Ans:- The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime. Dynamic memory allocation in c language is possible by 4

functions of stdlib.h header file.

- malloc()
- calloc()
- realloc()
- free()

- **Malloc() function**
  - The malloc() function allocates single block of requested memory.
  - It doesn't initialize memory at execution time, so it has garbage value initially.
  - It returns NULL if memory is not sufficient.
  - The syntax of malloc() function is given below:

    ptr=(cast-type*)malloc(byte-size)

- **Calloc() function**
  - The calloc() function allocates multiple block of requested memory.
  - It initially initialize all bytes to zero.
  - It returns NULL if memory is not sufficient.
  - The syntax of calloc() function is given below:

    ptr=(cast-type*)calloc(number, byte-size)

- **Realloc() function**
  - If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.
  - Let's see the syntax of realloc() function.

    ptr=realloc(ptr, **new**-size)

- **Free() function**
  - The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.
  - Let's see the syntax of free() function.

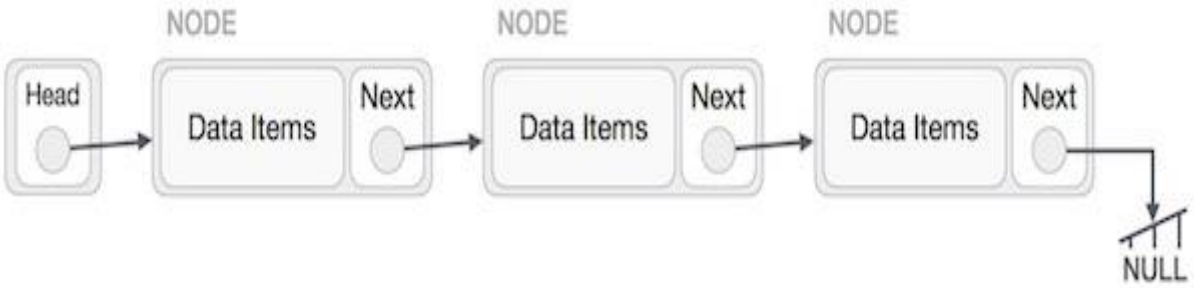
    free(ptr)

Q-28. What is Linked list? List types of linked list. [4 Marks]

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

- Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.
  - **Link** − Each link of a linked list can store a data called an element.
  - **Next** − Each link of a linked list contains a link to the next link called Next.

- **LinkedList** − A Linked List contains the connection link to the first link called First.



Types of Linked List

- **Simple Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

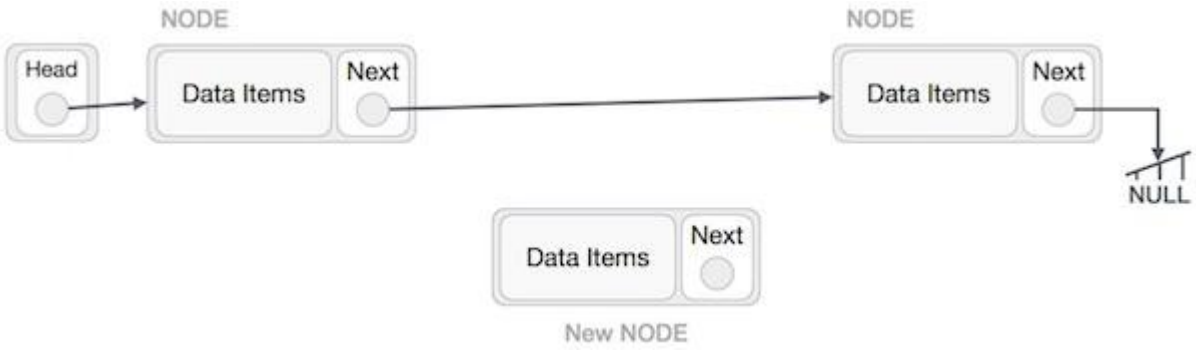
Q-29. Explain Insertion operation in singly linked list. [4 Marks]

Ans:- Insertion Operation

1. ADDING NEW NODE

Adding a new node in linked list is a more than one step activity.

First, create a node using the same structure and find the location where it has to be inserted.



- Allocate memory for new node
- Store data
- Change next of new node to point to head
- Change head to point to recently created node

```
struct node *newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;
```

```
newNode->next = head;

head = newNode;
```

2. ADD TO THE END

- Allocate memory for new node
- Store data
- Traverse to last node
- Change next of last node to recently created node

```
struct node *newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

newNode->next = NULL;

struct node *temp = head;

while(temp->next != NULL){

  temp = temp->next;

}

temp->next = newNode;
```

3. ADD TO THE MIDDLE

- Allocate memory and store data for new node
- Traverse to node just before the required position of new node
- Change next pointers to include new node in between

```
struct node *new Node;

new Node = malloc(sizeof(struct node));

new Node->data = 4;

struct node *temp = head;

for(int i=2; i < position; i++) {

   if(temp->next != NULL) {

      temp = temp->next;

   }
```

```
}

new Node->next = temp->next;

temp->next = new Node;
```

Q-30. Explain Deletion operation of singly linked list. [4 Marks]

Ans:- We can delete either from beginning, end or from a particular position.

Delete from beginning

- Point head to the second node

```
head = head->next;
```

Delete from end

- Traverse to second last element.
- Change its next pointer to null.

```
struct node* temp = head;

while(temp->next->next!=NULL){

  temp = temp->next;

}

temp->next = NULL;
```

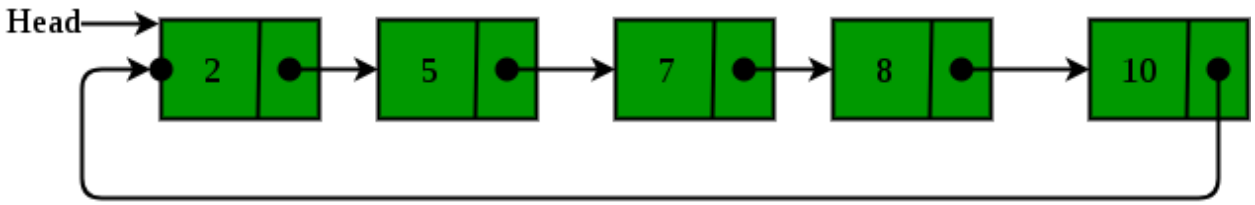
Delete from Middle

- Traverse to element before the element to be deleted
- Change next pointers to exclude the node from the chain

```
for(int i=2; i< position; i++) {

   if(temp->next!=NULL) {

      temp = temp->next;

   }

}

temp->next = temp->next->next;
```

Q-31. Explain Circular Linked list concepts. [4 Marks]

Ans:- Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.



- **Advantages of Circular Linked Lists:**

- Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.
- Useful for implementation of queue. Unlike this implementation, we don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last.
- Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.
- Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.

CHAPTER – 5 – SEARCHING AND SORTING

Q-32: what is sorting and type of sorting? [3 Marks]

- Sorting is nothing but arranging data in ascending or descending order. The term sorting came into picture, as humans realised the importance of searching quickly.
- There are so many things in our life that we need to search for, like a particular record in database, roll numbers in a merit list, a particular page in a book, a particular telephone number in a dictionary, etc.
- All these would have been a mess if the data was kept unordered and unsorted, but fortunately the concept of sorting came into existence, making it easier for everyone to arrange data in the order, hence making it easier to search.
- Sorting arranged data in sequence which makes searching easier.
- There are so many methods for sorting data elements which are as listed below:
  - Bubble Sort
  - Selection Sort
  - Quick Sort
  - Insertion Sort
  - Merge Sort
  - Radix sort

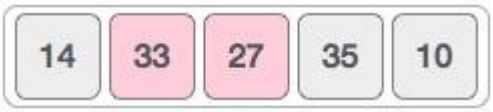
Q-33. Explain bubble sort with example and write an algorithm of bubble sort. [4 Marks]

- Bubble sort is a simple sorting algorithm.
- This sorting algorithm is comparison-based algorithm in which each pair of adjacent element is compared and the element are swapped if they are not in order.
- This algorithm is not suitable for large data sets as its average and worst case complexity are O (n^2) where n is the number of items.
- **Working of Bubble Sort**
  - We will take an unsorted array. Bubble sort takes O(n^2) time so we are taking a small example.

  - Bubble sort starts with very first two elements, comparing them to check which one is greater.

  - In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

  - We find that 27 is smaller than 33 and these two values must be swapped.



- The new array should look like this –

- Next we compare 33 and 35. We find that both are in already sorted positions.

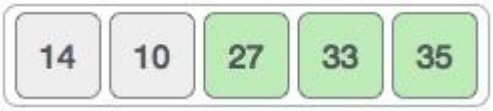
- Then we move to the next two values, 35 and 10.

- We know then that 10 is smaller 35. Hence they are not sorted.

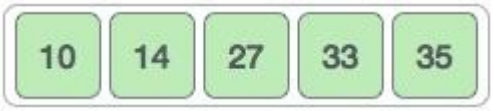
- We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this –

- To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this

- Notice that after each iteration, at least one value moves at the end.



- And when there's no swap required, bubble sorts learns that an array is completely sorted.



- **Algorithm**

Step 1 – Starting from the first index, compare the first and the second elements.
Step 2 − If the first element is greater than the second element, they are swapped.
Step 3 − Now, compare the second and the third elements. Swap them if they are not in order.
Step 4 − the above process goes on until the last element

Q-34. Explain selection sort with example and write an algorithm of selection sort. [4 Marks]

- In selection sort algorithm, list is divided into two parts, the sorted part at the left end and unsorted part at the right end.
- Initially, the sorted part is empty and the unsorted part is the entire list.
- The smallest element is selected from the list and swapped with the leftmost element, and that element becomes part of the sorted array. This process continue moving unsorted array boundary by one element to the right.
- This algorithm is not suitable for large data sets as its average and worst case complexities are of $O(n^2)$, where **n** is the number of items.
- **Working of selection sort**
  - Consider the following depicted array as an example.

  - For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

  - So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

  - For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.

- We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

- After two iterations, two least values are positioned at the beginning in a sorted manner.

- The same process is applied to the rest of the items in the array.

- **Algorithm:**

Step 1 − Set MIN to location 0
Step 2 − Search the minimum element in the list
Step 3 − Swap with value at location MIN
Step 4 − Increment MIN to point to next element
Step 5 − Repeat until list is sorted

Q-35. Explain quick sort with example and write an algorithm of quick sort. [4 Marks]

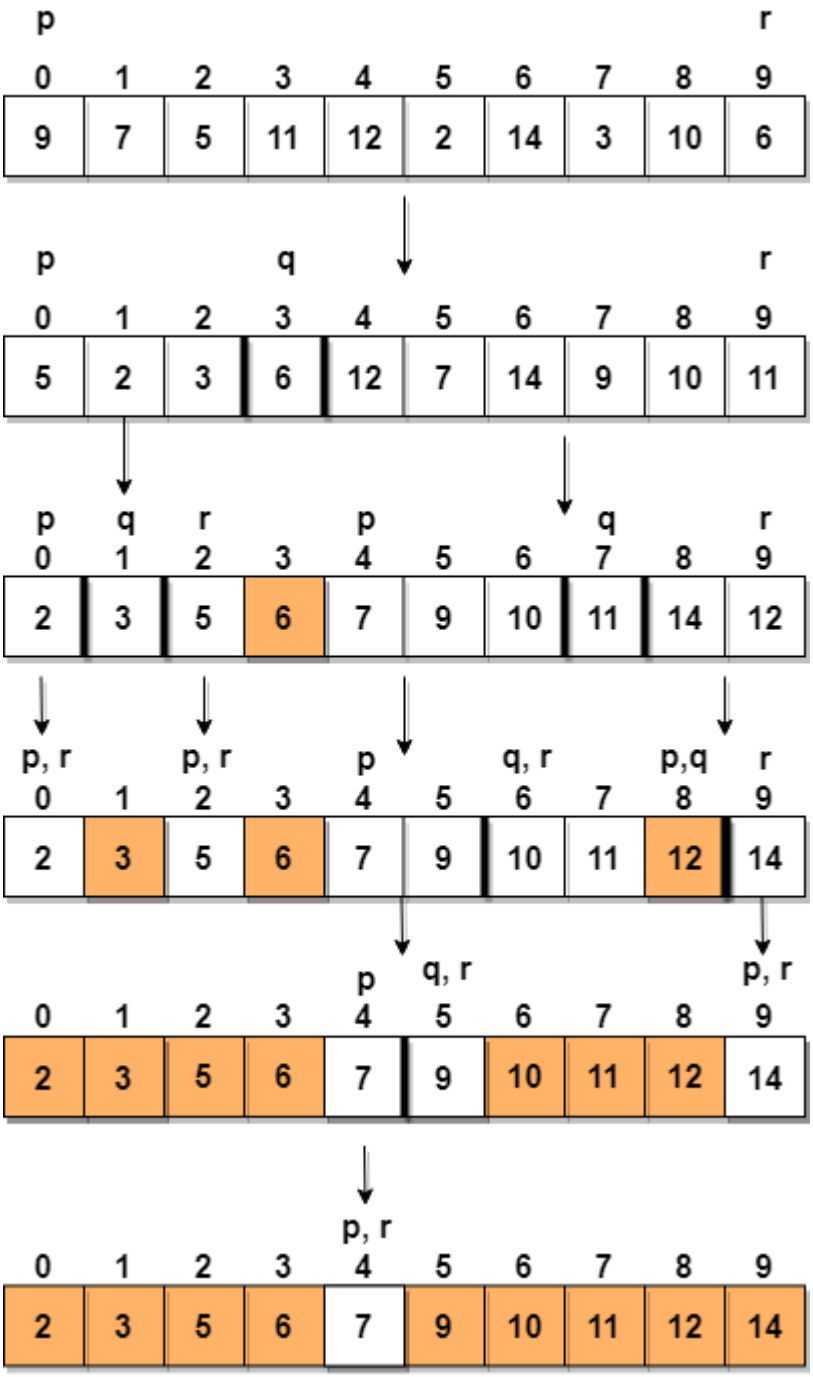
- Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays.
- A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.
- Quick sort partitions an array and then calls itself recursively twice to sort the two resulting sub arrays.
- This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are O(nLogn) and image.png(n^2), respectively.
- Let's consider an array with values {9, 7, 5, 11, 12, 2, 14, 3, 10, 6}
- Below, we have a pictorial representation of how quick sort will sort the given array.

- **Working of Quick Sort**

- After selecting an element as **pivot**, which is the last index of the array in our case, we divide the array for the first time.

- In quick sort, we call this **partitioning**. It is not simple breaking down of array into 2 subarrays, but in case of partitioning, the array elements are so positioned that all the elements smaller than the **pivot** will be on the left side of the pivot and all the elements greater than the pivot will be on the right side of it. And the **pivot** element will be at its final **sorted** position.

- The elements to the left and right, may not be sorted.
- Then we pick subarrays, elements on the left of **pivot** and elements on the right of **pivot**, and we perform **partitioning** on them by choosing a **pivot** in the subarrays.



- **Algorithm**

Quick Sort Pivot Algorithm

Step 1 − Choose the highest index value has 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

Quick Sort Algorithm

Step 1 − Make the right-most index value pivot

Step 2 − partition the array using pivot value

Step 3 − quicksort left partition recursively

Step 4 − quicksort right partition recursively

Q-36. Explain merge sort with example and write an algorithm of merge sort. [4 Marks]

- Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being O(n log n), it is one of the most respected algorithms.
- Merge sort first divides the array into equal halves and then combines them in a sorted manner.
- **Working of Merge Sort**
  - To understand merge sort, we take an unsorted array as the following

  - We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

  - This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

  - We further divide these arrays and we achieve atomic value which can no more be divided.

- Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.
- We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.

- In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

- After the final merging, the list should look like this –

- **Algorithm:**

Step 1 − if it is only one element in the list it is already sorted, return.

Step 2 − divide the list recursively into two halves until it can no more be divided.

Step 3 − merge the smaller lists into new list in sorted order.

Q-37. Explain insertion sort with example and write an algorithm of insertion sort. [4 Marks]

- In this, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted.
- An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, **insertion sort**.
- The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array).
- This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$, where **n** is the number of items.
- **Working of Insertion sort**

- We take an unsorted array for our example.

- Insertion sort compares the first two elements.

- It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.

- Insertion sort moves ahead and compares 33 with 27.

- And finds that 33 is not in the correct position.

- It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.

- By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.

- These values are not in a sorted order.

- So we swap them.

- However, swapping makes 27 and 10 unsorted.

- Hence, we swap them too.

- Again we find 14 and 10 in an unsorted order.

- We swap them again. By the end of third iteration, we have a sorted sub-list of 4 items.

- This process goes on until all the unsorted values are covered in a sorted sub-list. Now we shall see some programming aspects of insertion sort.

➢ **Algorithm:**

Step 1 − If it is the first element, it is already sorted. Return 1;

Step 2 − Pick next element

Step 3 − Compare with all elements in the sorted sub-list

Step 4 − Shift all the elements in the sorted sub-list that is greater than the

value to be sorted

Step 5 − Insert the value

Step 6 − Repeat until list is sorted.

CHAPTER – 6 NON-LINEAR DATA STRUCTURES – TREE AND GRAPH

Q-38. What is a non-linear data structure? [2 Marks]

Ans:- Data structures where data elements are not arranged sequentially or linearly are called non-linear data structures. In a non-linear data structure, a single level is not involved. It utilizes computer memory efficiently in comparison to a linear data structure. Its examples are trees and graphs.

Q-39. What is a tree? And how many types of trees? [3 Marks]

Ans:- A tree is a collection of nodes that creates parent-child relationships. Nodes in a tree are stored hierarchically where the topmost node is the root node of the tree.

There are many types of trees in the data structure. Some of the important types are as follows:

- GeneralTree
- BinaryTree
- Binary SearchTree
- AVLTree
- BTree
- B+Tree

Q-40. What is a Binary search tree? [2 Marks]

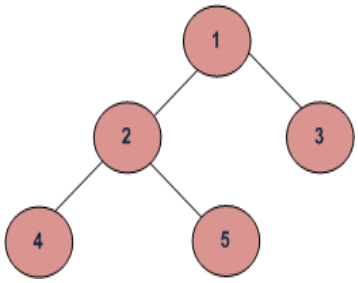
Ans:-

- A binary search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order.
- This is also called an ordered binary tree.
- In a binary search tree, the value of all the nodes in the left-sub-tree is less than the value of the root.
- Similarly, the value of all the nodes in the right sub-tree is greater than or equal to the value of the root.

Q-41. What is a tree traversal? [3 Marks]

Ans:- Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways.

Following are the generally used ways for traversing trees.



- **Depth First Traversals:**

(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

- **Breadth-First or Level Order Traversal: 1 2 3 45**

Q-42. What is Inorder, Preorder, and Postorder Traversal? [4 Marks]
Ans:-

Inorder Traversal:- For binary search trees (BST), Inorder Traversal specifies the nodes in non-descending order. In order to obtain nodes from BST in non-increasing order, a variation of inorder traversal may be used where inorder traversal is reversed.

Preorder Traversal:- Preorder traversal will create a copy of the tree. Preorder Traversal is also used to get the prefix expression of an expression.

Post order Traversal:- Post order traversal is used to get the postfix expression of an expression given.

Q-43. Write down application of Binary Tree? [2 Marks]

Ans:- Binary trees are used to represent a nonlinear data structure. There are various forms of Binary trees. Binary trees play a vital role in a software application. One of the most important applications of the Binary tree is in the searching algorithm. A general tree is defined as a nonempty finite set T of elements called nodes such that:

The tree contains the root element. The remaining elements of the tree form an ordered collection of zeros and more disjoint trees T1, T2, T3, T4 …. Tn which are called sub trees.

Q-44. What is conversion of the general tree to binary tree? [4 Marks]

Ans:-

Step 1: root node of the general tree becomes the root node of a binary tree.

Step 2: Now Root Node (A) has three child Nodes (B, H, E) in the general tree. Theleftmost node (B) of the root node (A) in the general tree becomes the leftmost node of the root node (A) in a binary tree.

Step 3: Now Node H becomes the right node of B and Node E becomes the right node of H.

Step 4: Now Node B has only one left child node, which is C in the general tree. So Node C becomes the left child of Node B in the binary tree.

Step 5: Now Node E has two child nodes (F, G). The leftmost node (F) of the node

(E) in the general tree becomes the leftmost node of node E in the binary tree.

Step 6: Now Node G becomes the right node of Node F in the binary tree.