# CAPITAL UNIVERSITY - KODERMA

## DATA STRUCTURES ASSIGNMENT

Name : Arshad Nazir

Electrical and Electronics Engineering

Signature:

Date :

## 1.Define Data Structure?

A digital computer can manipulate only primitive data,that is,data in terms of 0’s and 1’s.Manipulation of primitive data is inherent with in the computer and doesnot require any extra effort from the user side.But in our real life applications,various kind of data other than the primitive data are involved.manipulation of real life data requires the following tasks,

Storage-user data should be stored in such a way that the computer can understand it.

Retrieval of stored data-Data stored in a computer should be retrieved in such a way that the user can understand it.

Transformation of user data-various operations which require to be performed on user data so that it can be transformed from one form to another.

## 2.List the operations performed in the linear data structure?

Creation-create a new data element in a data structure.

Insertion-add a new data element in a data structure.

Deletion-removal of a data element from a data structure.

Searching-searching for the specified data element in a data structure.

## 3.what is abstract data tyoe?

When an application requires a special kind of data which is not available as a built in data type then it is the programers responsibility to implement his own kind of data.Here,the programmer has to specify how to store a value for that data,what are the operations that can meaningfully manipulate variables of that kind of data,amount of memory required to store a variable.It is also called user-defined data type.

Example:-structure and union.

## 4.Define Array?

An array is a data structure that contains a group of elements.Typically these elements are homogeneous,such as intiger or string.Array are commonly used in computer programming to organize data so that a related set of values can be easily sorted or searched.

## 5.write the limitations of array?

Static memory allocations and can have only values of same data type and not different data type values.

## 6.what are the ways to represent two dimensional arrays in memory?

Representation of two dimensional array in memory is row-major and column-major.

## 8.what are structures in C?

C Structures. Structure is **a user-defined datatype in C language** which allows us to combine data of different types together. Structure helps to construct a complex data type which is more meaningful. ... In structure, data is stored in form of records.

## 9.Define Stack.

A stack is **an abstract data type that holds an ordered, linear sequence of items**. In contrast to a queue, a stack is a last in, first out (LIFO) structure. A real-life example is a stack of plates: you can only take a plate from the top of the stack, and you can only add a plate to the top of the stack.

## 10.what are the operations allowed in stack

- Push, which adds an element to the collection, and.
- Pop, which removes the most recently added element that was not yet removed.

## 13. Write any four applications of stack.

- Evaluation of Arithmetic Expressions.
- Backtracking.
- Delimiter Checking.
- Reverse a Data.
- Processing Function Calls.

## 14Define queue.

a queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence.

## 15. Mention some applications of queue.

**Managing requests on a single shared resource** such as CPU scheduling and disk scheduling. Handling hardware or real-time systems interrupts. Handling website traffic.

## 16.what is priority queue?

The priority queue in the data structure is **an extension of the “normal” queue**. It is an abstract data type that contains a group of items. It is like the “normal” queue except that the dequeuing elements follow a priority order. The priority order dequeues those items first that have the highest priority.

## 17.Define Circular Queue.

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**'.

PART1A

## 1.Explain in detail about Arrays?

An array is a data structure that contains a group of elements. Typically these elements are all of the same data type, such as an integer or string. Arrays are commonly used in computer programs to organize data so that a related set of values can be easily sorted or searched.

For example, a search engine may use an array to store Web pages found in a search performed by the user. When displaying the results, the program will output one element of the array at a time. This may be done for a specified number of values or until all the values stored in the array have been output. While the program could create a new variable for each result found, storing the results in an array is much more efficient way to manage memory.

The syntax for storing and displaying the values in an array typically looks something like this:

```
arrayname[0] = "This ";
arrayname[1] = "is ";
arrayname[2] = "pretty simple.";

print arrayname[0];
print arrayname[1];
print arrayname[2];
```

The above commands would print the first three values of the array, or "This is pretty simple." By using a "while" or "for" loop, the programmer can tell the program to output each value in the array until the last value has been reached. So not only do arrays help manage memory more efficiently, they make the programmer's job more efficient as well.

## 2.Discuss in detail about structures in c.

Structure is a user-defined datatype in C language which allows us to combine data of different types together. Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.

**For example:** If I have to write a program to store Student information, which will have Student's name, age, branch, permanent address, father's name etc, which included string values, integer values etc, how can I use arrays for this problem, I will require something which can hold data of different types together.

In structure, data is stored in form of **records**.s

Examples of structure.

```
struct Student

{

   char name[25];

   int age;

   char branch[10];

   // F for female and M for male

   char gender;

};
```

Here struct Student declares a structure to hold the details of a student which consists of 4 data fields, namely name, age, branch and gender. These fields are called **structure elements or members**.

Each member can have different datatype, like in this case, name is an array of char type and age is of int type etc. **Student** is the name of the structure and is called as the **structure tag**.

## 3. What is a Stack? Explain its operations with example.

So a stack supports two basic operations: **push and pop**. Some stacks also provide additional operations: size (the number of data elements currently on the stack) and peek (look at the top element without removing it). The primary stack operations. A new data element is stored by pushing it on the top of the stack.

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations –

- **push()** – Pushing (storing) an element on the stack.

- **pop()** – Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks –

- **peek()** – get the top data element of the stack, without removing it.
- **isFull()** – check if stack is full.
- **isEmpty()** – check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named **top**. The **top** pointer provides top value of the stack without actually removing it.

4. Write the algorithm for converting infix expression to postfix expression.

**Algorithm**

- **Step 1** : Scan the Infix Expression from left to right.
- **Step 2** : If the scanned character is an operand, append it with final Infix to Postfix string.
- **Step 3** : Else,
    - **Step 3.1** : If the precedence order of the scanned(incoming) operator is greater than the precedence order of the operator in the stack (or the stack is empty or the stack contains a '(' or '[' or '{'), push it on stack.
- **Step 3.2** : Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
- **Step 4** : If the scanned character is an '(' or '[' or '{', push it to the stack.
- **Step 5** : If the scanned character is an ')'or ']' or '}', pop the stack and and output it until a '(' or '[' or '{' respectively is encountered, and discard both the parenthesis.
- **Step 6** : Repeat steps 2-6 until infix expression is scanned.
- **Step 7** : Print the output
- **Step 8** : Pop and output from the stack until it is not empty.

  5. What is a Queue? Explain its operations with example

  Queue is an abstract abstract data structure, structure, somewhat somewhat similar similar to Stacks. Stacks. Unlike stacks, stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) (enqueue) and the other is used to remove data (dequeue). (dequeue). Queue follows follows First-In-First-Out First-In-First-Out methodology methodology, i.e., the data item stored first will be accessed first.

As we now understand understand that in queue, we access both ends for different reasons. reasons. The following following diagram given below tries to explain queue representation as data structure – As in stacks, a queue can also be implemented using Arrays, Linked-lists, Pointers and Structures. For the sake of simplicity, we shall implement queues using one-dimensional array. Basic Operations Queue operations erations may involve involve initializing initializing or defining defining the queue, utilizing it, and then completely completely erasing erasing it from the memory. Here we shall try to understand understand the basic operations operations associated associated with queues – enqueue() – add (store) an item to the queue. dequeue() – remove (access) an item from the queue. Few more functions are required to make the above-mentioned queue operation efficient. These are – peek() – Gets the element at the front of the queue without removing it. isfull() – Checks if the queue is full. isempty() – Checks if the queue is empty. In queue, we always dequeue dequeue (or access) access) data, pointed pointed by front pointer pointer and while enqueing enqueing (or storing) data in the queue we take help of rear pointer. Let's first learn about supportive functions of a queue

```
Example bool iissffuullll() {
if(rear == MMAAXXSSIIZZEE - 1)
returrn true;
else retturn false;
}
```

## 6.Explain any two applications of stack?

**Reverse a Data:**

To reverse a given set of data, we need to reorder the data so that the first and last elements are exchanged, the second and second last element are exchanged, and so on for all other elements.

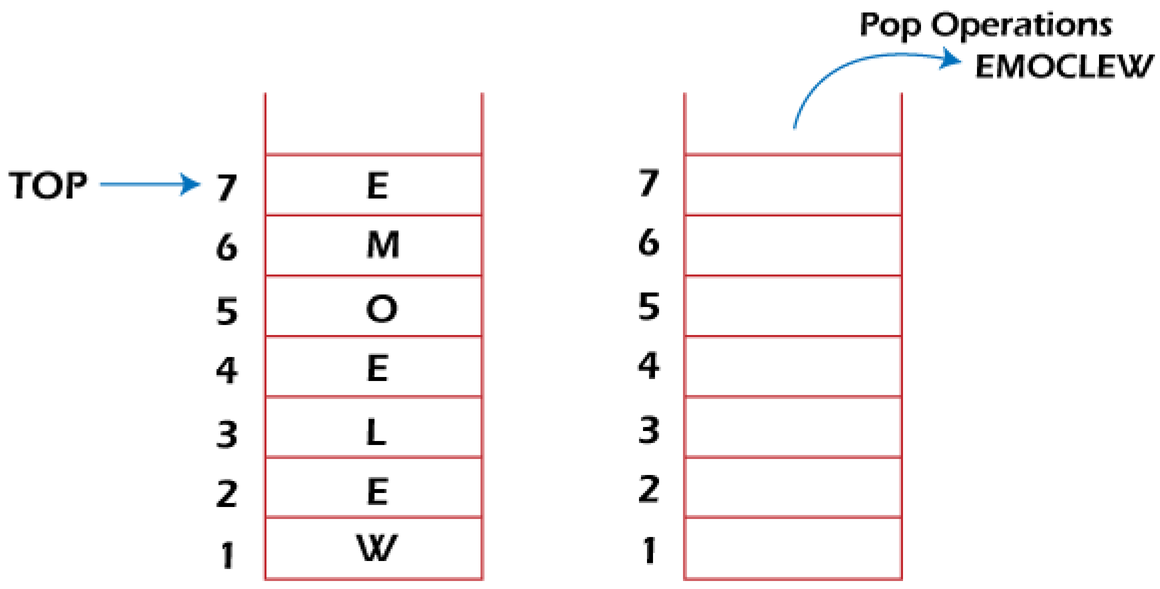
**Example:** Suppose we have a string Welcome, then on reversing it would be Emoclew.

**There are different reversing applications:**

- Reversing a string
- Converting Decimal to Binary

Reverse a String

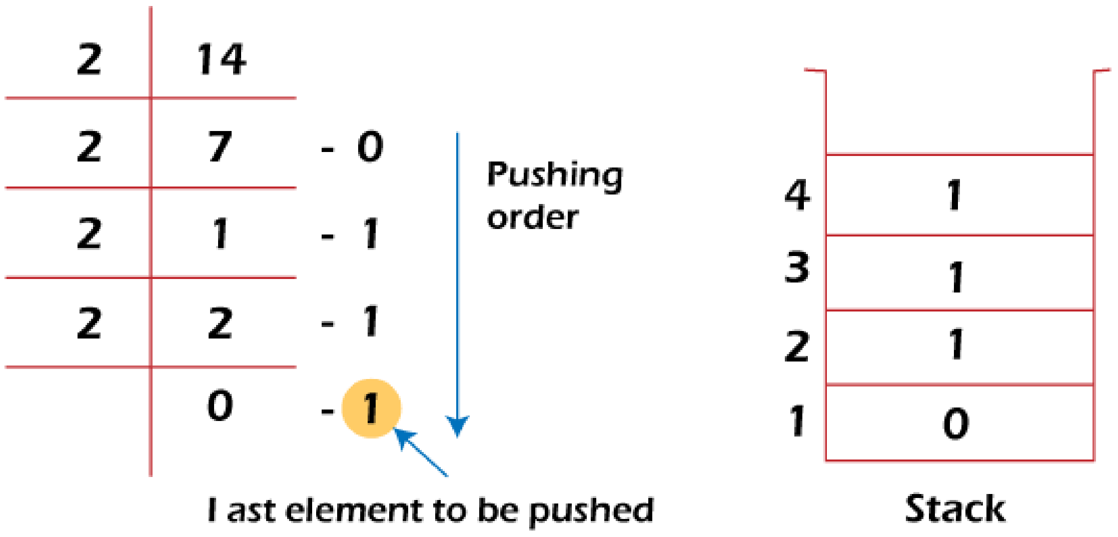
A Stack can be used to reverse the characters of a string. This can be achieved by simply pushing one by one each character onto the Stack, which later can be popped from the Stack one by one. Because of the **last in first out** property of the Stack, the first character of the Stack is on the bottom of the Stack and the last character of the String is on the Top of the Stack and after performing the pop operation in the Stack, the Stack returns the String in Reverse order.



Converting Decimal to Binary:

Although decimal numbers are used in most business applications, some scientific and technical applications require numbers in either binary, octal, or hexadecimal. A stack can be used to convert a number from decimal to binary/octal/hexadecimal form. For converting any decimal number to a binary number, we repeatedly divide the decimal number by two and push the remainder of each division onto the Stack until the number is reduced to 0. Then we pop the whole Stack and the result obtained is the binary equivalent of the given decimal number.

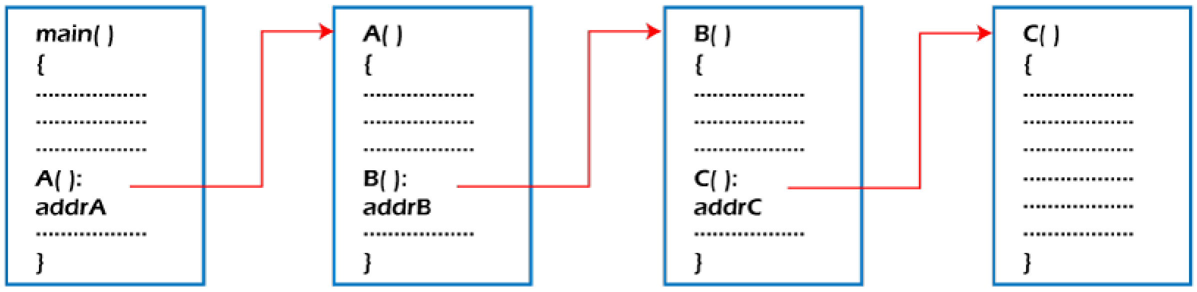
**Example: Converting 14 number Decimal to Binary:**



In the above example, on dividing 14 by 2, we get seven as a quotient and one as the reminder, which is pushed on the Stack. On again dividing seven by 2, we get three as quotient and 1 as the reminder, which is again pushed onto the Stack. This process continues until the given number is not reduced to 0. When we pop off the Stack completely, we get the equivalent binary number **1110.**

**5. Processing Function Calls:**

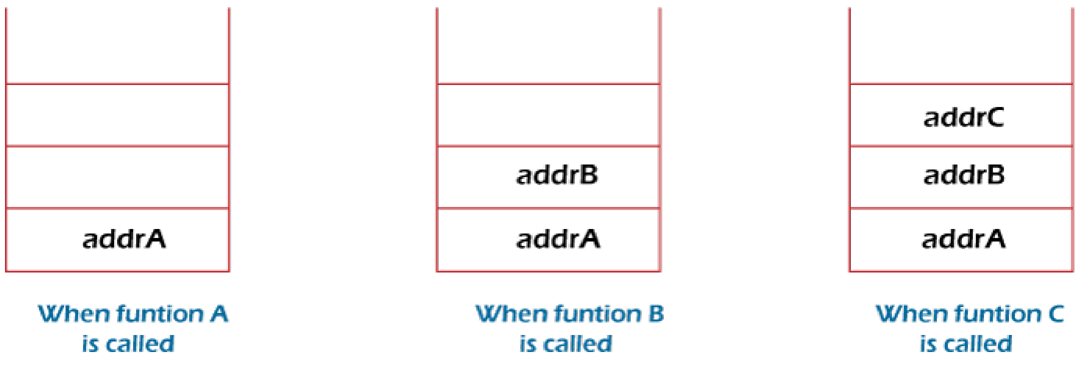
Stack plays an important role in programs that call several functions in succession. Suppose we have a program containing three functions: A, B, and C. function A invokes function B, which invokes the function C.



Function call

When we invoke function A, which contains a call to function B, then its processing will not be completed until function B has completed its execution and returned. Similarly for function B and C. So we observe that function A will only be completed after function B is completed and function B will only be completed after function C is completed. Therefore, function A is first to be started and last to be completed. To conclude, the above function activity matches the last in first out behavior and can easily be handled using Stack.

Consider addrA, addrB, addrC be the addresses of the statements to which control is returned after completing the function A, B, and C, respectively.



Different states of stack

The above figure shows that return addresses appear in the Stack in the reverse order in which the functions were called. After each function is completed, the pop operation is performed, and execution continues at the address removed from the Stack. Thus the program that calls several functions in succession can be handled optimally by the stack data structure. Control returns to each function at a correct place, which is the reverse order of the calling sequence.

## 7. Explain the Linear linked Implementation of Stack and Queue.

Stack can be implemented using both arrays and linked lists. The limitation, in the case of an array, is that we need to define the size at the beginning of the implementation. This makes our stack static. It can also result in "*stack overflow*" if we try to add elements after the array is full. So, to alleviate this problem, we use a linked list to implement the stack so that it can grow in real time.

First, we will create our Node class which will form our linked list. We will be using this same Node class to also implement the queue in the later part of this article.

```
internal class Node
{
   internal int data;
   internal Node next;

   // Constructor to create a new node.Next is by default initialized as null
   public Node(int d)
```

```
  {
    data = d;
    next = null;
  }
}
```

Now, we will create our stack class. We will define a pointer, top, and initialize it to null. So, our LinkedListStack class will be:

```
internal class LinkListStack
{
  Node top;

  public LinkListStack()
  {
    this.top = null;
  }
 }
```

## Push an Element Onto a Stack

Now, our stack and Node class is ready. So, we will proceed to push the operation onto the stack. We will add a new element at the top of the stack.

### Algorithm

- Create a new node with the value to be inserted.
- If the stack is empty, set the next of the new node to null.
- If the stack is not empty, set the next of the new node to top.
- Finally, increment the top to point to the new node.

The time complexity for the *Push* operation is O(1). The method for push will look like this:

```
internal void Push(int value)
{
   Node newNode = new Node(value);
   if (top == null)
   {
      newNode.next = null;
   }
   else
   {
      newNode.next = top;
   }
   top = newNode;
   Console.WriteLine("{0} pushed to stack", value);
}
```

## Pop an Element From the Stack

We will remove the top element from the stack.

### Algorithm

- If the stack is empty, terminate the method as it is stack underflow.
- If the stack is not empty, increment the top to point to the next node.
- Hence the element pointed to by the top earlier is now removed.

The time complexity for Pop operation is O(1). The method for pop will look like the following:

```
internal void Pop()
{
   if (top == null)
   {
      Console.WriteLine("Stack Underflow. Deletion not possible");
      return;
   }

   Console.WriteLine("Item popped is {0}", top.data);
   top = top.next;
}
```

Peek the Element From Stack

The peek operation will always return the top element of the stack without removing it from the stack.

Algorithm

- If the stack is empty, terminate the method as it is stack underflow.
- If the stack is not empty, return the element pointed to by the top.

The time complexity for the peek operation is O(1). The peek method will look like the following:

```
internal void Peek()
{
    if (top == null)
```

```
  {
    Console.WriteLine("Stack Underflow.");
    return;
  }

  Console.WriteLine("{0} is on the top of Stack", this.top.data);
}
```

## 8. Explain the Insertion and Deletion algorithm for Circular linked List.

Algorithm for Insertion in a circular queue

```
Insert CircularQueue ( )

1. If (FRONT == 1 and REAR == N) or (FRONT == REAR + 1) Then

2.          Print: Overflow

3. Else

4.          If (REAR == 0) Then [Check if QUEUE is empty]

                    (a) Set FRONT = 1

                    (b) Set REAR = 1

5. Else If (REAR == N) Then [If REAR reaches end if QUEUE]

6.                  Set REAR = 1

7. Else

8.                  Set REAR = REAR + 1 [Increment REAR by 1]

  [End of Step 4 If]

9. Set QUEUE[REAR] = ITEM

10. Print: ITEM inserted
```

```
[End of Step 1 If]

11. Exit
```

## Algorithm for Deletion in a circular queue

```
Delete CircularQueue ( )

1. If (FRONT == 0) Then [Check for Underflow]

2.          Print: Underflow

3. Else

4.          ITEM = QUEUE[FRONT]

5.                     If (FRONT == REAR) Then [If only element is left]

                                (a) Set FRONT = 0

                                (b) Set REAR = 0

6. Else If (FRONT == N) Then [If FRONT reaches end if QUEUE]

7.                              Set FRONT = 1

8. Else

9.                              Set FRONT = FRONT + 1 [Increment FRONT by 1]

                       [End of Step 5 If]

10. Print: ITEM deleted

                       [End of Step 1 If]

11. Exit
```

## 9. Explain the operations of Priority Queue.

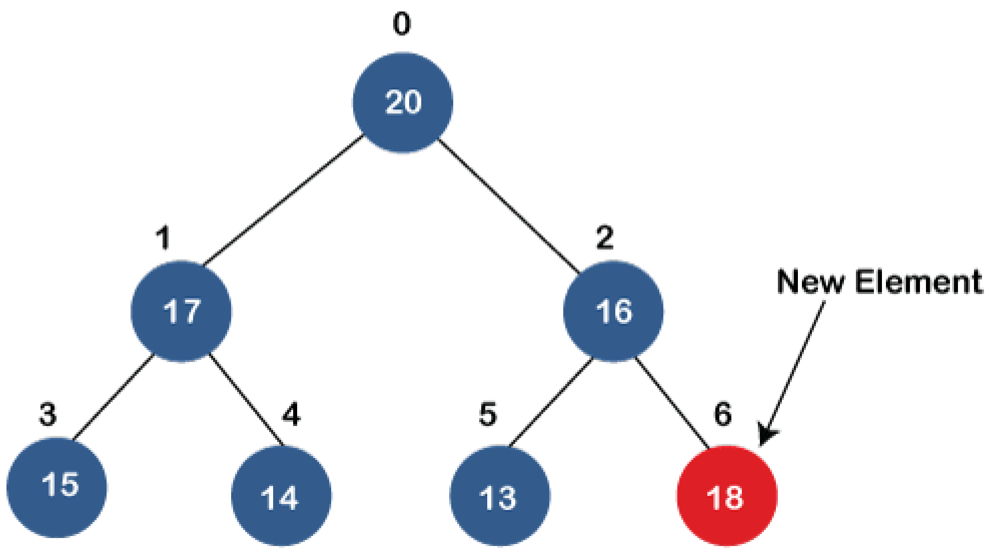
Priority Queue Operations

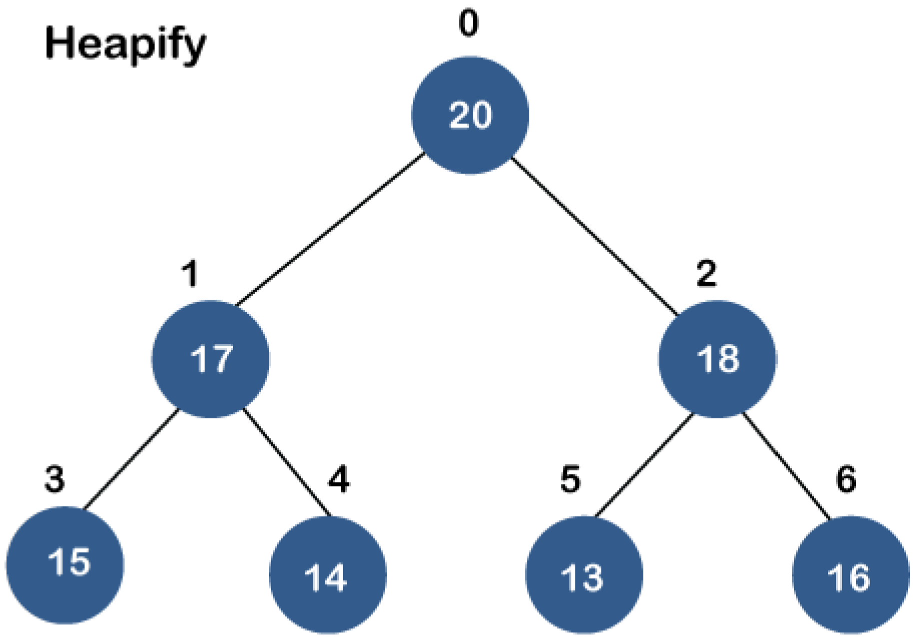
The common operations that we can perform on a priority queue are insertion, deletion and peek. Let's see how we can maintain the heap data structure.

## Inserting the element in a priority queue (max heap)

If we insert an element in a priority queue, it will move to the empty slot by looking from top to bottom and left to right.

If the element is not in a correct place then it is compared with the parent node; if it is found out of order, elements are swapped. This process continues until the element is placed in a correct position.





**Removing the minimum element from the priority queue**

As we know that in a max heap, the maximum element is the root node. When we remove the root node, it creates an empty slot. The last inserted element will be added in this empty slot. Then, this element is compared with the child nodes, i.e., left-child and right child, and swap with the smaller of the two. It keeps moving down the tree until the heap property is restored.

PART2

## 1.What is the purpose of non linear data structures?

Non-linear data structures are used in image processing and Artificial Intelligence.

## 2.What is binary tree?

A binary tree is **a tree-type non-linear data structure with a maximum of two children for each parent**. Every node in a binary tree has a left and right reference along with the data element. The node at the top of the hierarchy of a tree is called the root node.

## 3.What is mean by siblings?

**Nodes which belong to the same parent are** called as siblings. In other words, nodes with the same parent are sibling nodes.
4What are ancestors and descendants?

An ancestor of a node is **any other node on the path from the node to the root**. A descendant is the inverse relationship of ancestor: A node p is a descendant of a node q if and only if q is an ancestor of p. We can talk about a path from one node to another.

## 5.What is strictly binary tree (or) full binary tree?

A full binary tree (sometimes proper binary tree or 2-tree) is **a tree in which every node other than the leaves has two children**. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

## 6.What do you mean by complete binary tree?

A complete binary tree is **a binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left**. A complete binary tree is just like a full binary tree, but with two major differences. All the leaf elements must lean towards the left.

## 7.Define almost complete binary tree?

Almost complete binary trees are **not necessarily strictly binary** (although they can be), and are not complete. If the tree has a maximum level of d, then the subtree containing all the nodes from the root to level d-1 is a complete tree.

## 8. What are the different representations of nodes in a tree?

A node can be represented in a binary search tree with three fields, i.e., **data part, left-child, and right-child**. A node can be connected to the utmost two child nodes in a binary search tree, so the node contains two pointers (left child and right child pointer).

## 9. Give various implementations of binary tree?

- Full binary tree: Every node other than leaf nodes has 2 child nodes.
- Complete binary tree: All levels are filled except possibly the last one, and all nodes are filled in as far left as possible.
- Perfect binary tree: All nodes have two children and all leaves are at the same level.

  10. What are the advantages of binary tree linked representation over array representation?

It **has both dynamic size and ease in insertion and deletion** as advantages.
11. What do you mean by External nodes and internal nodes?

**An external node is one without child branches**, while an internal node has at least one child branch. ... The root is at level 0, the root's children are at level 1, and so on. The sum of the levels of each of the internal nodes in a tree is called the internal path length of that tree.

## 12. What is an expression tree?

A binary expression tree is a specific kind of a binary tree used to represent expressions. Two common types of expressions that a binary expression tree can represent are algebraic and boolean. These trees can represent expressions that contain both unary and binary operators.

## 13. What is a almost complete binary tree?

An almost complete binary tree is **a special kind of binary tree where insertion takes place level by level and from left to right order at each level and the last level is not filled fully always**. It also contains. nodes at each level except the last level.

PART2A

## 4. Write a non-recursive and recursive algorithm for in order tree traversal

```
1. POSTORD(INFO, LEFT, RIGHT, ROOT)
2. Step-1 [Push NULL onto STACK and initialize PTR]
3.   Set TOP=1, STACK[1]=NULL and PTR=ROOT.
4.
5. Step-2 [Push left-most path onto STACK] repeat step 3 to 5 while PTR not equal
     NULL.
6.
7. Step-3 set TOP=TOP+1 and STACK[TOP]=PTR. [Pushes PTR on STACK].
8.
```

```
Step-4 if RIGHT[PTR] not equal NULL then [push on STACK.]
   Set TOP=TOP+1 and STACK[TOP]= RIGHT[PTR]
  [End of if structure]

Step-5 set PTR = LEFT[PTR].[Update pointer PTR]
   [End of step 2 loop].

Step-6 set PTR= STACK[TOP] and TOP=TOP-1.
  [Pops node from STACK].

Step-7 Repeat while PTR>0;
  Apply PROCESS TO INFO[PTR].
Set PTR= STACK[TOP] and TOP= TOP-1.
[Pops node from STACK]
[End of loop]

Step-8 if PTR<0 then:
   Set PTR= -PTR
Go to step 2
[End of if structure]

Step-9 Exit.
```

## 5. Explain how arithmetic expressions are manipulated using binary trees. Illustrate with example.

1.One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a compuer.

———–

```
    /  <-- root
  /
...       home
     /
  ugrad       course
  /         /   |
 ...       cs101 cs112 cs113
```

1.If we organize keys in form of a tree (with some ordering e.g., BST), we can search for a given key in moderate time (quicker than Linked List and slower than arrays). Self-balancing search trees like AVL and Red-Black trees guarantee an upper bound of O(Logn) for search.

1.We can insert/delete keys in moderate time (quicker than Arrays and slower than Unordered Linked Lists). Self-balancing search trees like AVL and Red-Black trees guarantee an upper bound of O(Logn) for insertion/deletion.
Like Linked Lists and unlike Arrays, Pointer implementation of trees don't have an upper limit on number of nodes as nodes are linked using pointers.
**Other Applications :**
1.Heap is a tree data structure which is implemented using arrays and used to implement priority queues.
2.B-Tree and B+ Tree : They are used to implement indexing in databases.
3.Syntax Tree: Used in Compilers.
4.K-D Tree: A space partitioning tree used to organize points in K dimensional space.
5.Trie : Used to implement dictionaries with prefix lookup.
6.Suffix Tree : For quick pattern searching in a fixed text.following are the common uses of tree.
1. Manipulate hierarchical data.
2. Make information easy to search (see tree traversal).
3. Manipulate sorted lists of data.
4. As a workflow for compositing digital images for visual effects.
5. Router algorithms

Insert function is used to add a new element in a binary search tree at appropriate location. Insert function is to be designed in such a way that, it must node violate the property of binary search tree at each value.

1. Allocate the memory for tree.
2. Set the data part to the value and set the left and right pointer of tree, point to NULL.
3. If the item to be inserted, will be the first element of the tree, then the left and right of this node will point to NULL.
4. Else, check if the item is less than the root element of the tree, if this is true, then recursively perform this operation with the left of the root.
5. If this is false, then perform this operation recursively with the right sub-tree of the root.
6. **Step1:** IF TREE=NULL

```
  Allocate memory for TREE
 SET TREE -> DATA = ITEM
SET TREE -> LEFT = TREE -> RIGHT = NULL
ELSE
 IF ITEM < TREE -> DATA
  Insert(TREE -> LEFT, ITEM)
ELSE
```

```
 Insert(TREE -> RIGHT, ITEM)
[END OF IF]
[END OF IF]
```

**Step 2:** END

**PART3**

## 1.Define B tree.

a B-tree is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree generalizes the binary search tree, allowing for nodes with more than two children.

## 2.Define sentinel.

a sentinel node is **a specifically designated node used with linked lists and trees as a traversal path terminator**. This type of node does not hold or reference any data managed by the data structure.

## 3.Define a heap. How can it be used to represent a priority queue?

Priority queue is a type of queue in which every element has a key associated to it and the queue returns the element according to these keys, unlike the traditional queue which works on first come first serve basis. ... We use **a max-heap for a max-priority queue and a min-heap for a min-priority queue**.

## 4.Give any two applications of binary heaps.

Heap Sort: Heap Sort uses Binary Heap to sort an array in O(nLogn) time.

Priority Queue: Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in O(logn) time.

## 5.What are the methods for avoiding collision?

We can avoid collision by making **hash function random, chaining method and uniform hashing**.

## 6.Define priority queue.

a priority queue is **an abstract data type similar to a regular queue or** stack data structure in which each element additionally has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority.

## 7.Define AVL tree.

an AVL tree is a self-balancing binary search tree. It was the first such data structure to be invented. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property.

## 8.Define single rotation on AVL tree.

A single rotation applied **when a node is inserted in the right subtree of a right subtree**. In the given example, node A has a balance factor of 2 after the insertion of node C. By rotating the tree left, node B becomes the root resulting in a balanced tree. Left-Right Rotation.

## 9.What is the minimum number of nodes in an AVL tree of height 15?

If height of AVL tree is h, maximum number of nodes can be $2^{h+1} – 1$. Minimum number of nodes in a tree with height h can be represented as: **N(h) = N(h-1) + N(h-2) + 1 for n>2 where N(0) = 1 and N(1) = 2**. The complexity of searching, inserting and deletion in AVL tree is O(log n).

## 10.Define binary heap.

A binary heap is a heap data structure that takes the form of a binary tree. Binary heaps are a common way of implementing priority queues.

## 11.What are the methods used for implementing priority queue?

The priority queue can be implemented in four ways that include **arrays, linked list, heap data structure and binary search tree**.

## 12.Define hashing functions.

A hash function is **a function that takes a set of inputs of any arbitrary size and fits them into a table or other data structure that contains fixed-size elements**. ... The table or data structure generated is usually called a hash table.

## 13.List out the structural Properties of B Trees

A B-tree is a tree data structure that **keeps data sorted and allows searches, insertions, and deletions in logarithmic amortized time**. Unlike self-balancing binary search trees, it is optimized for systems that read and write large blocks of data. It is most commonly used in database and file systems.

## 15. What is a balance factor? 16. Difference between B tree and B+ tree.

The balance factor of a binary tree is **the difference in heights of its two subtrees (hR - hL)**. The balance factor (bf) of a height balanced binary tree may take on one of the values -1, 0, +1. An AVL node is "leftheavy" when bf = 1, "equalheight" when bf = 0, and "rightheavy" when bf = +1.

## 16. Difference between B tree and B+ tree.

B tree.

In the B tree, all the keys and records are stored in both internal as well as leaf nodes.

B+tree.

In the B+ tree, keys are the indexes stored in the internal nodes and records are stored in the leaf nodes.

## 17. List the different ways of implementing priority queue.

The priority queue can be implemented in four ways that include **arrays, linked list, heap data structure and binary search tree**.

## 18. What is Min heap and Max heap?

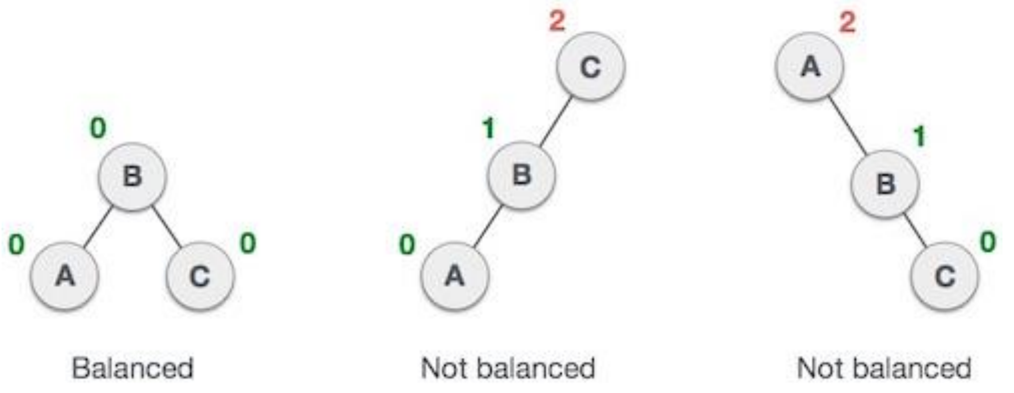
a min-max heap is a complete binary tree data structure which combines the usefulness of both a min-heap and a max-heap, that is, it provides constant time retrieval and logarithmic time removal of both the minimum and maximum elements in it.

PART3A

1. It is observed that BST's worst-case performance is closest to linear search algorithms, that is O(n). In real-time data, we cannot predict data pattern and their frequencies. So, a need arises to balance out the existing BST.

Named after their inventor **Adelson**, **Velski** & **Landis**, **AVL trees** are height balancing binary search tree. AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the **Balance Factor**.

Here we see that the first tree is balanced and the next two trees are not balanced –



In the second tree, the left subtree of **C** has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of **A** has height 2 and the left is missing, so it is 0, and the difference is 2 again. AVL tree permits difference (balance factor) to be only 1.

***BalanceFactor*** = height(left-sutree) – height(right-sutree)

If the difference in the height of left and right sub-trees is more than 1, the tree is balanced using some rotation techniques.

**AVL Rotations**

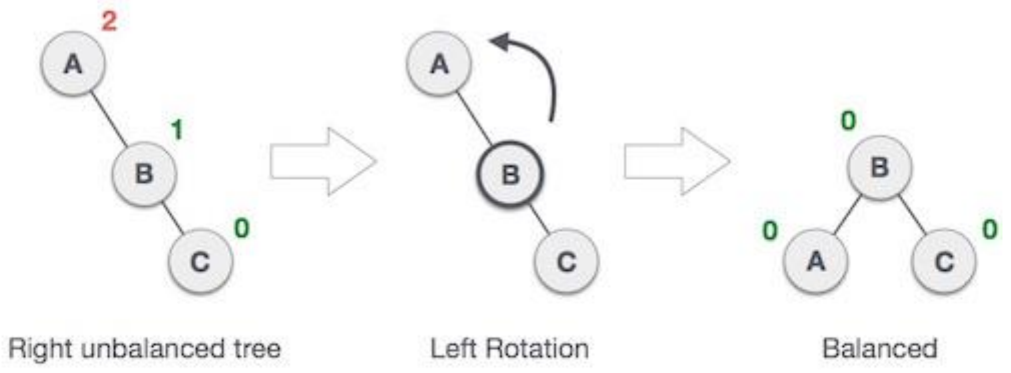
To balance itself, an AVL tree may perform the following four kinds of rotations –

- Left rotation
- Right rotation
- Left-Right rotation
- Right-Left rotation

The first two rotations are single rotations and the next two rotations are double rotations. To have an unbalanced tree, we at least need a tree of height 2. With this simple tree, let's understand them one by one.

## Left Rotation

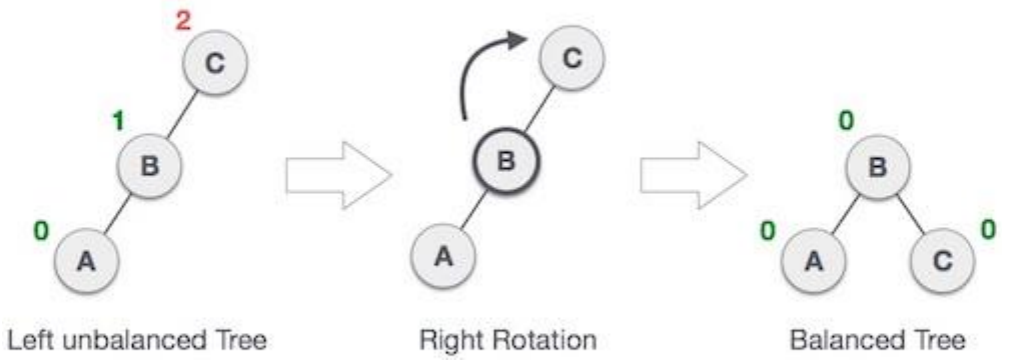
If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation –



In our example, node **A** has become unbalanced as a node is inserted in the right subtree of A's right subtree. We perform the left rotation by making **A** the left-subtree of B.

## **Right Rotation**

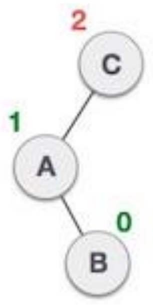
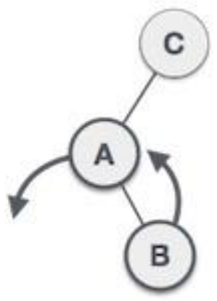
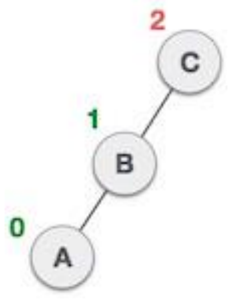
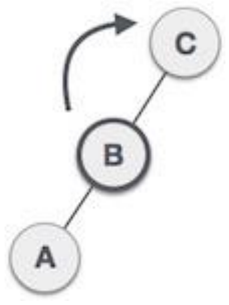
AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.

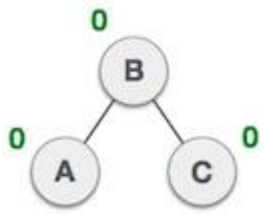


As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

## Left-Right Rotation

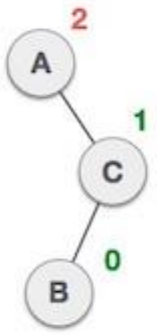
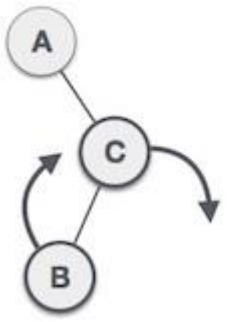
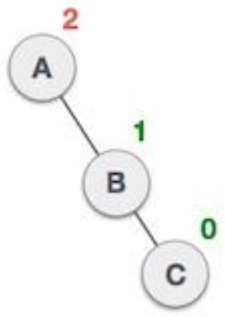
Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

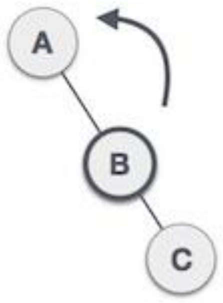
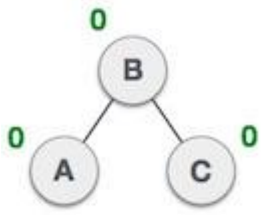
| State | Action |
|---|---|
|  | A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation. |
|  | We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**. |
|  | Node **C** is still unbalanced, however now, it is because of the left-subtree of the left-subtree. |
|  | We shall now right-rotate the tree, making **B** the new root node of this subtree. **C** now becomes the right subtree of its own left subtree. |

| | |
|---|---|
|  | The tree is now balanced. |

Right-Left Rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

| State | Action |
|---|---|
|  | A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2. |
|  | First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**. |
|  | Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation. |

| | |
|---|---|
|  | A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree **B**. |
|  | The tree is now balanced. |

**Approch for finding minimum element:**

- Traverse the node from root to left recursively until left is NULL.
- The node whose left is NULL is the node with minimum value.

**Approch for finding maximum element:**

- Traverse the node from root to right recursively until right is NULL.
- The node whose right is NULL is the node with maximum value.

Implementation of the above approches.

```
// C++ program to find maximum or minimum element in binary search tree
#include <bits/stdc++.h>
using namespace std;

struct node
{
   int key;
   struct node *left, *right;
};

// A utility function to create a new BST node
struct node *newNode(int item)
{
   struct node *temp =  (struct node *)malloc(sizeof(struct node));
   temp->key = item;
```

```
    temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with given key in BST */
struct node* insert(struct node* node, int key)
{
    struct node *newNode(int );
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}
/* Given a non-empty binary search tree,
return the minimum data value found in that
tree. Note that the entire tree does not need
to be searched. */
int minValue(struct node* node)
{
    struct node* current = node;

    /* loop down to find the leftmost leaf */
    while (current->left != NULL)
    {
        current = current->left;
    }
    return(current->key);
}
/* Given a non-empty binary search tree,
return the maximum data value found in that
tree. Note that the entire tree does not need
to be searched. */
int maxValue(struct node* node)
{
    struct node* current = node;

    /* loop down to find the leftmost leaf */
    while (current->right != NULL)
    {
        current = current->right;
    }
    return(current->key);
}
// Driver Program to test above functions
int main()
{
    int maxValue(struct node* );
    struct node* insert(struct node* , int );
    int minValue(struct node* );
    struct node *root = NULL;
    root = insert(root, 8);
    insert(root, 3);
    insert(root, 10);
    insert(root, 1);
    insert(root, 6);
    insert(root, 4);
    insert(root, 7);
    insert(root, 5);
    insert(root, 10);
    insert(root, 9);
    insert(root, 13);
    insert(root, 11);
    insert(root, 14);
    insert(root, 12);
    insert(root, 2);

    cout << "\n Minimum value in BST is " << minValue(root)<<endl;
    cout << "\n Maximum value in BST is " << maxValue(root);
    return 0;
}
C++
Copy
```

Output:

```
Minimum value in BST is 1

Maximum value in BST is 14
```

PART4

## 1.Define graph.

a graph is an abstract data type that is meant to implement the undirected graph and directed graph concepts from the field of graph theory within mathematics.

## 2.Define node edge, vertex, degree and cycle of a graph.

Degree: Degree of any vertex is **defined as the number of edge Incident on it**. ... The cycle graph with n vertices is called Cn. Properties of Cycle Graph:- It is a Connected Graph. A Cycle Graph or Circular Graph is a graph that consists of a single cycle.

A vertex (or node) of a graph is **one of the objects that are connected together**. The connections between the vertices are called edges or links. A graph with 10 vertices (or nodes) and 11 edges (links). For more information about graph vertices, see the network introduction.

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are **lines or arcs that connect any two nodes in the graph**. ... For example, in Facebook, each person is represented with a vertex(or node).

## 3.Define tree graph.

Tree is a **non-linear data structure in which elements are arranged in multiple levels**. A Graph is also a non-linear data structure. Structure. It is a collection of edges and nodes.

## 4.Define digraph, directed graph and undirected graph with an example

A directed graph is sometimes called a digraph or a directed network. In contrast, a graph where the edges are bidirectional is called an undirected graph. ... One can formally define a directed graph as **G=(N,E)**, consisting of the set N of nodes and the set E of edges, which are ordered pairs of elements of N.

## 5.Define connected graph. Say when a graph G is said to be complete.

connected graph **A graph in which there is a path joining each pair of vertices, the graph being undirected.** It is always possible to travel in a connected graph between one vertex and any other; no vertex is isolated.

A complete graph is **a graph with N vertices and an edge between every two vertices.** ▶ There are no loops. ▶ Every two vertices share exactly one edge. We use the symbol KN for a complete graph with N vertices.

## 6.Define strongly connected and weakly connected graph.

Strongly Connected: A graph is said to be strongly connected if every pair of vertices(u, v) in the graph contains a path between each other. ... Weakly Connected: A graph is said to be weakly connected **if there doesn't exist any path between any two pairs of vertices**.

## 7.Define adjacency matrix for graph G with example.

an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph. In the special case of a finite simple graph, the adjacency matrix is a-matrix with zeros on its diagonal.

## 8.Name the two standard ways of maintaining graph in memory.

There are two standard ways of maintaining a graph G in the memory of a computer. One way, called the sequential representation ofG is by means of its adjacency matrixA. The other way, **called the linked representation or adjacency structure of G, uses linked lists of neighbors**.

## 9.Define spanning tree.

A spanning tree is **a tree that connects all the vertices of a graph with the minimum possible number of edges**. Thus, a spanning tree is always connected.

## 10.Give the various types of graph traversal methods.

The graph has two types of traversal algorithms. These are called **the Breadth First Search and Depth First Search**.

## 11.What is NP –Hard problem?

A problem is NP-hard **if all problems in NP are polynomial time reducible to it**, even though it may not be in NP itself. If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable. These problems are called NP-complete.

## 12.Write short notes on NP complete problem.

A problem is called NP (nondeterministic polynomial) if its solution can be guessed and verified in polynomial time; nondeterministic means that no particular rule is followed to make the guess. If a problem is NP and all other NP problems are polynomial-time reducible to it, the problem is NP-**complete**.

### 13.Give the properties of NP Hard and NP completeness

A problem is NP-hard if all problems in **NP are polynomial time reducible to it**, even though it may not be in NP itself. If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable. These problems are called NP-complete.

### 14.Define Topological sorting. Give algorithm.

a topological sort or topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering.

```
topological_sort(N, adj[N][N])
    T = []
    visited = []
    in_degree = []
    for i = 0 to N
        in_degree[i] = visited[i] = 0

    for i = 0 to N
        for j = 0 to N
            if adj[i][j] is TRUE
                in_degree[j] = in_degree[j] + 1

    for i = 0 to N
        if in_degree[i] is 0
            enqueue(Queue, i)
            visited[i] = TRUE

    while Queue is not Empty
        vertex = get_front(Queue)
        dequeue(Queue)
        T.append(vertex)
        for j = 0 to N
            if adj[vertex][j] is TRUE and visited[j] is FALSE
                in_degree[j] = in_degree[j] - 1
                if in_degree[j] is 0
                    enqueue(Queue, j)
                    visited[j] = TRUE
    return T
```

## 15.List out the applications of DFS and BFS

| **BFS** | **DFS** |
|---|---|
| It uses a queue to keep track of the next location to visit. | It uses a stack to keep track of the next location to visit. |
| BFS traverses according to tree level. | DFS traverses according to tree depth. |
| It is implemented using FIFO list. | It is implemented using LIFO list. |

### 1.What is the worst case running time for Dijikstra's Algorithm?

Each pop operation takes **O(log V)** time assuming the heap implementation of priority queues. So the total time required to execute the main loop itself is O(V log V).

PART 5.

### 1.Define Program.

A computer is a tool that provides information and entertainment by means of a computer program written in a programming language. A computer program in its human-readable form is called source code.

### 2.Define Algorithm.

Algorithm is a step-by-step procedure, which **defines a set of instructions to be executed in a certain order to get the desired output**. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

### 3.Define Problem Definition Phase

As your intuition would suggest, a **problem is a task to be performed**. It is best thought of in terms of inputs and matching outputs. A problem definition should not include any constraints on how the problem is to be solved.

### 4.What are the problem solving strategies?

- Developing an approach to understanding the problem.
- Thinking of a correct basic solution.
- Designing step-by-step pseudocode solution.
- Analyzing the efficiency of a solution.
- Optimizing the solution further.
- Transforming pseudocode into a correct code.

### 5.Define Top Down Design.

Top-down and bottom-up are both strategies of information processing and knowledge ordering, used in a variety of fields including software, humanistic and scientific theories, and management and organization. In practice, they can be seen as a style of thinking, teaching, or leadership.

## 6.What is the basic idea behind Divide & Conquer Strategy?

The divide-and-conquer paradigm is often used to find an optimal solution of a problem. Its basic idea is **to decompose a given problem into two or more similar, but simpler, subproblems, to solve them in turn, and to compose their solutions to solve the given problem**.

## 7.Define Program Verification.

Program verification aims **to use formal proofs to demonstrate that programs behave according to formal specifications**. Proving programs correct or even partially correct has been a hot topic for decades. ... The basic idea is to use symbolic execution and some Hoare-style proof rules to generate verification conditions.

## 9. Define Symbolic Execution

symbolic execution is a means of analyzing a program to determine what inputs cause each part of a program to execute. An interpreter follows the program, assuming symbolic values for inputs rather than obtaining actual inputs as normal execution of the program would.

## 11.Define the qualities of good algorithm.

Input: a good algorithm **must be able to accept a set of defined input**. Output: a good algorithm should be able to produce results as output, preferably solutions. Finiteness: the algorithm should have a stop after a certain number of instructions. Generality: the algorithm must apply to a set of defined inputs.

## 12. Define Computational Complexity.

Computational complexity theory focuses on classifying computational problems according to their resource usage, and relating these classes to each other. A computational problem is a task solved by a computer. A computation problem is solvable by mechanical application of mathematical steps, such as an algorithm.

## 13.What is O –notation?

The Big O notation is **used to express the upper bound of the runtime of an algorithm** and thus measure the worst-case time complexity of an algorithm. It analyses and calculates the time and amount of memory required for the execution of an algorithm for an input value.

## 14.What is Recursion? Explain with an example.

Recursion is **a process in which the function calls itself indirectly or directly in order to solve the problem**. The function that performs the process of recursion is called a recursive function. There are certain problems that can be solved pretty easily with the help of a recursive algorithm.

For example, we can define the operation "**find your way home**" as: If you are at home, stop moving. Take one step toward home.

## 15.List out the performance measures of an algorithm.

There are many ways in which the resources used by an algorithm can be measured: the two most common measures are **speed and memory usage**; other measures could include transmission speed, temporary disk usage, long-term disk usage, power consumption, total cost of ownership, response time to external stimuli, etc.

## 16.Define Algorithm & Notion of algorithm

An algorithm is a finite sequence of well-defined instructions, typically used to solve a class of specific problems or to perform a computation.

An algorithm (pronounced AL-go-rith-um) is **a procedure or formula for solving a problem, based on conducting a sequence of specified actions**. A computer program can be viewed as an elaborate algorithm. ... Algorithms are widely used throughout all areas of IT (information technology).