💙Phenomenal ❤️

|  |  |
| --- | --- |
| Short Type | |
| 1 | What do you mean by the efficiency of an algorithm? Discuss time and space complexity?  Algorithmic efficiency is a property of an algorithm which relates to the amount of computational resources used by an algorithm.  Time complexity: It quantifies the amount of time taken by an algorithm to run as a function of the length of input, it is not the actual execution time of the machine on which the algorithm is running in.  Space complexity:- It quantifies the amount of space taken by an algorithm to run as a function of the length of the input. |
| 2 | Which asymptotic notion is used to represent for lower bound of a function? Discuss it.  Big omega(Ω) is used to represent the lower bond of running time of an algorithm.   1. It gives the best time complexity 2. Gives greatest lower bond of a function.   If f(n) and g(n) are two functions with +ve integer then  F(n) >= Cg(n)  Ω(g(n)) = f(n) |
| 3 | What is data structure? Differentiate between linear and nonlinear data structure.  *A data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.*   | S.NO | Linear Data Structure | Non-linear Data Structure | | --- | --- | --- | | 1. | In a linear data structure, data elements are arranged in a linear order where each and every element is attached to its previous and next adjacent. | In a non-linear data structure, data elements are attached in hierarchically manner. | | 2. | In linear data structure, single level is involved. | Whereas in non-linear data structure, multiple levels are involved. | | 3. | Its implementation is easy in comparison to non-linear data structure. | While its implementation is complex in comparison to linear data structure. | | 4. | In linear data structure, data elements can be traversed in a single run only. | While in non-linear data structure, data elements can’t be traversed in a single run only. | | 5. | In a linear data structure, memory is not utilized in an efficient way. | While in a non-linear data structure, memory is utilized in an efficient way. | | 6. | Its examples are: array, stack, queue, linked list, etc. | While its examples are: trees and graphs. | | 7. | Applications of linear data structures are mainly in application software development. | Applications of non-linear data structures are in Artificial Intelligence and image processing. | |
| 4 | What do you understand by stack and queue? Give real life example from each.  Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).  ***A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order****.*  Stack Data Structure |
| 5 | Suggest a data structure that can be used to reverse the content of an array? Briefly explainhow your suggestion will help in solving the problem. How your suggestion will help in solving the problem  ***Stack can be used to reverse the content of an array.***  ***Because in stack the element that is inserted last will come first.***  ***So we simply have to keep whole contents in stack and while removing it will pop out in reverse order.*** |
| 6 | Write the formulas to calculate the address of an element of a 2D array in row major order. Ifthe address of A[0][0] and A[0][1] are 1000 and 1010 respectively and each element occupies 2bytes then in which order do you think the array elements are stored and why you think so? |
| 7 | What is sparse matrix. Give an example. What are the disadvantages of representing it in normal 2D array  Matrix whose most of the elements are zero is termed as sparse matrix.  It covers memory unnecessarily so we use “Triplet Representation”  Sparse Matrix and its representations | Set 1 (Using Arrays and Linked ... |
| 8 | Discuss some areas of application of data structure?  **Arrays**  **2d arrays,** commonly known as, matrices, are used in image processing.  ***Your viewing screen is also a multidimensional array of pixels.***  ***For cpu scheduling in computer***  **Application of Strings:**   * **Spam email detection** * **Plagiarism detectin** * **Search engine** * **Spell checkers**   **Linked list:**   * **Images are linked with each other. So, an image viewer software uses a linked list to view the previous and next image using the previous and next buttons.** * **Music players** * **To keep the track of turns in a multi-player game, a circular linked list is used.** * **Used in switching between applications and programs (Alt + Tab) in the Operating system (implemented using Circular Linked List)** * **Train coaches are connected to one another in a doubly-linked list fashion.** * **It can be used to implement Stacks, Queues, Graphs, and Trees.** |
| 9 | Define queue.  ***A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order****.*  Lightbox |
| 10 | Define stack  Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out |
| 11 | Explain overflow and underflow condition in circular queue ( using array).  Overflow conditions  ((front == 0 && rear == MAX-1)||(rear-front==1))  Underlow conditin  return front==-1 |
| 12 | Write some areas of application of stack  Applications of stack:   * Stacks can be used to check for the balancing of paranthesis in an expression. * Infix to Postfix/Prefix conversion. * Redo-undo features at many places such as editors, photoshop, etc. * Forward and backward feature in web browsers. |
| 13 | Suggest a data structure that can be used to check whether an arithmetic expression has balanced parenthesis or not? Explain how your suggested data structure can help to solve the problem  **Stack**  **The idea is to put all the opening brackets in the stack. Whenever you hit a closing bracket, search if the top of the stack is the opening bracket of the same nature. If this holds then pop the stack and continue the iteration, in the end if the stack is empty, it means all brackets are well-formed . Otherwise, they are not balanced.** |
| 14 | Write some areas of application of queues  Applications of Queue: Queue is used when things don’t have to be processed immediatly, but have to be processed in First InFirst Out order like [Breadth First Search](http://en.wikipedia.org/wiki/Breadth-first_search). This property of Queue makes it also useful in following kind of scenarios:   1. When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling. 2. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc. |
| 15 | Given in the digraph G(V,E) , V={v1,v2,v3,v4,v5},  E={(v1,v2),(v2,v3),(v1,v5),(v5,v3),(v4,v3),(v4,v1)}.Draw the graph and check .whether it is cyclic or not |
| 16 | If the elements "A", "B", "C" and "D" inserted into queue and are deleted one at a time, in what, in what order will they be removed?  A B C D |
| 17 | Translate the following postfix to infix: A B \* C + A\*B+C |
| 18 | What is linked list? Name different types of linked list.  **linked lists are linear data structures. In linked lists elements are not stored at contiguous memory locations. They can be stored anywhere in the memory but for sequential access, the nodes are linked to each other using pointers**.  Singly linked list  Doubly linked list |
| 19 | Differentiate between array and linked list.   1. The size of the arrays is fixed, so we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage. On the other hand, linked lists are dynamic and the size of the linked list can be incremented or decremented during runtime. 2. Inserting a new element in an array of elements is expensive, because a room has to be created for the new elements, and to create room, existing elements have to shift.  For example, in a system, if we maintain a sorted list of IDs in an array id[].   id[] = [1000, 1010, 1050, 2000, 2040].  And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000). Deletion is also expensive with arrays unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved. On the other hand, nodes in linked lists can be inserted or deleted without any shift operation and is efficient than that of arrays. |
| 20 | Write two disadvantages of linked list over array?  Disadvantages of Linked Lists:   1. **Random access is not allowed in Linked Lists. We have to access elements sequentially starting from the first node. So, we cannot do a binary search with linked lists efficiently with its default implementation. Therefore, lookup or search operation is costly in linked lists in comparison to arrays.** 2. **Extra memory space for a pointer is required with each element of the list.** 3. **Not cache-friendly. Since array elements are present at contiguous locations, there is a locality of reference which is not there in the case of linked lists** |
| 21 | Explain Doubly Linked list with a suitable example  Similar to *Singly Linked Lists*, *Doubly Linked Lists* are also a sequential data structure with the only difference that the doubly linked lists contain two pointers instead of one to store the address of both next node and previous node respectively.   As you can see in the above image:   * Each node contains two pointers, one pointing to the next node and the other pointing to the previous node. * The prev of Head node is NULL, as there is no previous node of Head. * The next of last node is NULL, as there is no node after the last node. |
| 22 | Explain Circular Linked List with a suitable example  *A 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.*  Below is a pictorial representation of Circular Linked List: |
| 23 | What the following C code will do? Assuming that curr holds address of first node of a singly linked list , n is an integer “next” is the link  part of the node and “data” is the integer info part of the node.  n=0;  while(curr!=NULL)  { n=n+curr->data; curr=curr->next; |

|  |  |
| --- | --- |
|  | } printf(“%d”,n);  //**will print sum of all data elements of linked list** |
| 24 | Define a tree recursively. Give an example of a binary tree.  *A tree consists of a root, and zero or more subtrees T1, T2, … , A tree consists of a root, and zero or more subtrees T1, T2, … , Tk such that there is an edge from the root of the tree to the root of each subtree.Tk such that there is an edge from the root of the tree to the root of each subtree.* |
| 25 | How many structurally different binary trees are possible with 5 nodes.  5040 |
| 26 | What is graph? Give an example.  A *Graph* is a data structure that consists of the following two components:   1. A finite set of vertices also called nodes. 2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not the same as (v, u) in case of a directed graph(digraph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost. |
| 27 | What is AVL tree? Give an example.  **GM Adelson - Velsky and EM Landis**  It can be defined as height balanced binary search tree in which each node has balance factor of {-1, 0, +1}  where balance factor is **Height of left subtree- Height of right subtree**.  AVL Trees – randerson112358 – Medium |
| 28 | Represent the following general tree as a binary tree: |
| 29 | Construct an expression tree for the expression :  A \* B - (C + D) \* (P /Q) |
| 30 | What is Max-heap? Give an example.  A [max-heap](https://www.geeksforgeeks.org/binary-heap/) is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node  max-heap  #Optional  . Mapping the elements of a heap into an array is trivial: if a node is stored an index k, then its left child is stored at index 2k + 1 and its right child at index 2k + 2.  Illustration: Max Heap |
| 31 | What is path matrix of a graph? Path Matrix in Graph Theory  A path matrix in a data structure is a matrix representing a graph, where each value in mtℎ row and ntℎ column project whether there is a path from node m to node n.  The path may be direct or indirect.  It may have a single edge or multiple edges.  To understand what is path matrix in data structure, let’s consider the following graph  **Adjacency Matrix and Path Matrix**  Directed Graph  The adjacency matrix of the above graph is  **CodingHero - 2 Essential Concepts of Graph Explained for Kids Advantage** |
| 32 | Define indegree and outdegree of a node in directed graph with an example.  In a directed graph, each vertex has an indegree and an outdegree. Indegree of a Graph**Indegree of vertex V is the number of edges which are coming into the vertex V**. Notation − deg− (V). Outdegree of a Graph Outdegree of vertex V is the number of edges which are going out from the vertex V. Notation − deg+ (V). Consider the following examples.  Directed Graph The indegree and outdegree of other vertices are shown in the following table −   |  |  |  | | --- | --- | --- | | Vertex | Indegree | Outdegree | | a | 1 | 2 | | b | 2 | 0 | | c | 2 | 1 | | d | 1 | 1 | | e | 1 | 1 | | f | 1 | 1 | | g | 0 | 2 | |
| 33 | What are the three traversal strategies used in binary tree? Find the inorder traversal of the following binary tree    Three Traversal Strategies used in binary tree are:   1. **Inorder** 2. **Preorder** 3. **postorder**   F D B E A C |
| 34 | What do you mean by Internal and External Sorting?  **In internal sorting all the data to sort is stored in memory at all times while sorting is in progress. In external sorting data is stored outside memory (like on disk) and only loaded into memory in small chunks. External sorting is usually applied in cases when data can't fit into memory entirely.** |
| 35 | Compare the worst case time complexity of bubble sort, insertion sort, and selection sort  **Bubble sort**  **Worst Case: Reversely sorted / Very few elements are in proper place. [ O(N2) ] . O(N2) swaps.**  **Selection sort**  **Worst Case: Reversely sorted, and when the inner loop makes a maximum comparison. [ O(N2) ] . Also, O(N) swaps.**  **Insertion sort**  **Worst Case: Reversely sorted, and when inner loop makes maximum comparison, [ O(N2) ] . And O(N2) swaps.** |
| 36 | What is the disadvantage of selection sort?  Disadvantage:  Time complexity in all cases is O(N2), no best case scenario.  It requires n-squared number of steps for sorting n elements.  It is not scalable.  The number of swaps reduced. O(N) swaps in all cases.  In-Place sort.  @extra  Advantage:  It can also be used on list structures that make add and remove efficient, such as a linked list. Just remove the smallest element of unsorted part and end at the end of sorted part. |
| 37 | What do you understand by divide-and-conquer strategy?  **A divide and conquer strategy is used in solving huge problems through:**  **Dividing the problem into smaller sub-problems.**  **Solving the sub-problems, and**  **Merge them into one to get the output of the problem.** |
| 38 | For a list L = {8, 99, 3, 4, 6, 10}, find the output list at the end of pass 1 using bubble sorting method.  8,3,4,6,10,99 |
| 39 | Define Collision in hashing  h(6) = 6%10 = 6  The index is 6 at which the value is stored.  Collision  When hash function provides same value for two different values, then the problem occurs between the two  values, known as a collision. In the above example, the value is stored at index 6. If the key  value is 26, then the index would be:  h(26) = 26%10 = 6  Therefore, two values are stored at the same index, i.e., 6, and this leads to the collision problem. |
| 40 | What are the collision resolution methods?  When two items hash to the same slot, we must have a systematic method for**placing the second item in the hash table**. This process is called collision resolution. |
| Short Focussed Type | |
| 1 | What is algorithm? Discuss its properties  The word Algorithm means ” A set of finite rules or instructions to be followed in calculations or other problem-solving operations ” Or ” A procedure for solving a mathematical problem in a finite number of steps that frequently involves recursive operations”.  Characteristics of an Algorithm  Clear and Unambiguous: The algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.  Well-Defined Inputs: If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.  Well-Defined Outputs: The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should produce at least 1 output.  Finite-ness: The algorithm must be finite, i.e. it should terminate after a finite time.  Feasible: The algorithm must be simple, generic, and practical, such that it can be executed with the available resources. It must not contain some future technology or anything.  Language Independent: The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.  Properties of Algorithm:  It should terminate after a finite time.  It should produce at least one output.  It should take zero or more input.  It should be deterministic means giving the same output for the same input case.  Every step in the algorithm must be effective i.e. every step should do some work. |

|  |  |
| --- | --- |
| 2 | In a two dimensional array A[5][10] Deduce the address of the A[3][5] in row major order and column major order.  The base address of the array is 3000 and the size of element is  2bytes. Assume the lower bound of row and column indices to be 0. |
| 3 | Write an algorithm to find the largest and smallest element of an array   1. Input the array elements. 2. Initialize small = large = arr[0] 3. Repeat from i = 2 to n 4. if(arr[i] > large) 5. large = arr[i] 6. if(arr[i] < small) 7. small = arr[i] 8. Print small and large. |
| 4 | Write an algorithm to transpose a matrix  for (i = 0; i < 3; i++)  {  for (j = 0; j < i; j++)  {  int temp = arry[i][j];  arry[i][j] = arry[j][i];  arry[j][i] = temp;  }  } |
| 5 | Write an algorithm for binary search.   1. **Sort the array in ascending order.** 2. **Set the low index to the first element of the array and the high index to the last element.** 3. **Set the middle index to the average of the low and high indices.** 4. **If the element at the middle index is the target element, return the middle index.** 5. **If the target element is less than the element at the middle index, set the high index to the middle index – 1.** 6. **If the target element is greater than the element at the middle index, set the low index to the middle index + 1.** 7. **Repeat steps 3-6 until the element is found or it is clear that the element is not present in the array**. |
| 6 | Discuss the triplet representation of sparse matrix  **2D array is used to represent a sparse matrix in which there are three rows named as**   * **Row: Index of row, where non-zero element is located** * **Column: Index of column, where non-zero element is located** * **Value: Value of the non zero element located at index – (row,column)**   Lightbox  **//Program -- Optional**  **#include<stdio.h>**    **int main()**  **{**  **// Assume 4x5 sparse matrix**  **int sparseMatrix[4][5] =**  **{**  **{0 , 0 , 3 , 0 , 4 },**  **{0 , 0 , 5 , 7 , 0 },**  **{0 , 0 , 0 , 0 , 0 },**  **{0 , 2 , 6 , 0 , 0 }**  **};**    **int size = 0;**  **for (int i = 0; i < 4; i++)**  **for (int j = 0; j < 5; j++)**  **if (sparseMatrix[i][j] != 0)**  **size++;**    **// number of columns in compactMatrix (size) must be**  **// equal to number of non - zero elements in**  **// sparseMatrix**  **int compactMatrix[3][size];**    **// Making of new matrix**  **int k = 0;**  **for (int i = 0; i < 4; i++)**  **for (int j = 0; j < 5; j++)**  **if (sparseMatrix[i][j] != 0)**  **{**  **compactMatrix[0][k] = i;**  **compactMatrix[1][k] = j;**  **compactMatrix[2][k] = sparseMatrix[i][j];**  **k++;**  **}**    **for (int i=0; i<3; i++)**  **{**  **for (int j=0; j<size; j++)**  **printf("%d ", compactMatrix[i][j]);**    **printf("\n");**  **}**  **return 0;**  **}** |
| 7 | Convert the following Infix expression to Postfix form using a stack.  A + B – (C \* D- E + F / G ) + ( H + I\*J )   * AB+ - (CD\*E-FG/+) + HIJ\*+ * (AB+CD\*E-FG/+-) +( HIJ\*+) * AB+CD\*E-FG/+-HIJ\*++   Follow usual precedence rule and assume that the expression is legal. |
| 8 | Write an algorithm to insert an element into a circular queue implemented using array. |
| 9 | Write algorithms for Push and Pop operation of stack implemented using array |
| 10 | Discuss postfix evaluation process using stack and evaluate the following postfix expression by  125 2 \* 4 − 2 ^ 6 / +  Where ^ is exponent operator |
| 11 | Briefly discuss about priority queue and double ended queue  **Queue works on the principal of FCFS but in priority queue each element is assigned with a priority. Elements of highest priority processes easily.**  **In case of same priority (FCFS comes in action)**  Double ended Queue is a linear data structure which does not always follow FIFO principle .  Here insertion and deletion can be done from both ends. |
| 12 | Can we implement queue using stack(s)?If yes how many minimum no. of stack we need.  A queue can be implemented using two stacks. Let queue to be implemented be q and stacks used to implement q be stack1 and stack2. q can be implemented in two ways: |
| 13 | Write a code snippet for declaring the node of a doubly linked list.  and create a node dynamically.    Declaring the node of a doubly linked list  **struct node**  **{**  **int info;**  **struct node \*next;**  **struct node \*prev;**  **}**  Now for allocating memory dynamically we need to include header file #include <stdlib.h>  For allocation of memory  new = (struct node \*)malloc(sizeof(struct node)); |
| 14 | Difference between calloc() and malloc() in C.   |  |  | | --- | --- | | malloc() | calloc() | | 1. | It is a function that creates one block of memory of a fixed size. | It is a function that assigns more than one block of memory to a single variable. | | 2. | It only takes one argumemt | It takes two arguments. The calloc () function takes two arguments: the number of elements to allocate and the storage size of those elements. Typically, calloc () implementations multiply these arguments to determine how much memory to allocate. | | 3. | It is faster than calloc. | It is slower than malloc() | | 4. | It has high time efficiency | It has low time efficiency | | 5. | It is used to indicate memory allocation | It is used to indicate contiguous memory allcoation | |
| 15 | Write an algorithm to reverse a single linked list?   1. **Initialize three pointers prev as NULL, curr as head, and next as NULL.** 2. **Iterate through the linked list. In a loop, do the following:**    1. **Before changing the next of curr, store the next node**       1. **next = curr -> next**    2. **Now update the next pointer of curr to the prev**       1. **curr -> next = prev**    3. **Update prev as curr and curr as next**       1. **prev = curr**       2. **curr = next** |
| 16 | Write an algorithm to find largest data value present in a singly linked list.   * Create a variable **max** and initialize it with **INT\_MIN**. * Traverse through the list and for every node, compare its data with **max**. * If the current node’s data is greater than **max**, then store the value of the current node’s data in **max**. * In the end, **max** will contain the Maximum value of the list. |
| 17 | Write the algorithm for searching an element in a singly linked list.   * Create a variablefound and initialize it with 0. * Traverse through the list and for every node, compare its data with key. * If the current node’s data are equal to given element, then make found = 1 * If(found = 1) then key is found otherwise not |
| 18 | Write the steps of the algorithm to insert a new node after the node pointed by P in the following Linked List.  <https://media.geeksforgeeks.org/wp-content/cdn-uploads/RGIF2.gif>  void insert\_at\_end(){      new = (struct node\*)malloc(sizeof(struct node));      printf("\n enter info of new node");      scanf("%d",&new->info);      while(current->next!=0){          current = current->next;      }          current->next = new;          new->next = 0;  } |
| 19 | Briefly discuss about double ended and priority queue.   Double ended queue is a generalized version of Queue data structure that allows insert and delete at both ends.  A Deque can be implemented either using a doubly linked list or a circular array.  **Some Practical Applications of Deque**:   * Applied as both stack and queue, as it supports both operations. * Storing a web browser’s history. * Storing a software application’s list of undo operations. * Job scheduling algorithm   **priority queue** is a type of queue that arranges elements based on their priority values.  Elements with higher priority values are typically retrieved before elements with lower priority values.  In a priority queue, each element has a priority value associated with it. When you add an element to the queue, it is inserted in a position based on its priority value. For example, if you add an element with a high priority value to a priority queue, it may be inserted near the front of the queue, while an element with a low priority value may be inserted near the back. |
| 20 | Find topological ordering for the following graph:    **The first vertex in topological sorting is always a vertex with an in-degree of 0 (a vertex with no incoming edges).** |
|  |  |

|  |  |
| --- | --- |
| 21 | Construct the binary tree whose preorder and inorder traversal are given  Preorder : a b d e g h c f n  Inorder : d b g e h a c n f |
| 22 | What is adjacency matrix? Find the adjacency matrix for the following graph: |
| 23 | Write DFS algorithm for graph traversal.   * Put each vertex of the graph into one of two categories 1. Visited 2.Nonvisited * Purpose of algorithm is to mark each vertex as visited by avoiding cycles   1.start by putting any one of graph vertices on to the top of stack’  2. take the top element of the stack and add it to the visited list  3. create a list of vertex adjacent nodes and add the once which are not visited list top of the stack  4. keep repeating 2 and 3 until stack is empty |
| 24 | Define the following terms with respect to tree: height of a node, depth of a node, path, leaf node with example from each. |
| 25 | Represent the following binary tree in an array |
| 26 | Write a C function for selection sort.?  void ss(int arr[], int n)  {      int min;      for (int i = 0; i < n - 1; i++)      {          min = i;          for (int j = i + 1; j < n; j++)          {              if (arr[j] < arr[min])              {                  min = j;              }          }          if (min != i)          {              swap(&arr[i], &arr[min]);          }          /\* code \*/      }          for (int i = 0; i < n; i++)          {              printf("%d ", arr[i]);          }  }  void SelectionSort(int arr[], int size)  {      for (int i = 0; i < size; i++)      {          int min = smallest(arr, i, size);          if (min != i)          {              swap(&arr[min], &arr[i]);          }      }      for (int i = 0; i < size; i++)      {          printf("%d ", arr[i]);      }  } |
| 27 | Explain how insertion sort works taking an example and showing the result of each pass. |
| 28 | Briefly describe radix sort and Sort the following data using radix/ bucket sort: 42, 74, 11, 65, 94, 36, 87, 70, 81,1 |
| 29 | Explain the merge sort process taking an example |
| 30 | Write any two techniques to overcome hash collision  **When the two different values have the same value, then the problem occurs between the two values, known as a collision.**  **The following are the collision techniques:**  ** Open Hashing: It is also known as closed addressing.**  ** Closed Hashing: It is also known as open addressing.** |
| 31 | What are the application of hashing?  **There are many other applications of hashing, including modern day cryptography hash functions. Some of these applications are listed below:**   1. ***Message Digest*** 2. ***Password Verification*** 3. ***Data Structures(Programming Languages)*** 4. ***Compiler Operation*** 5. ***Rabin-Karp Algorithm*** 6. ***Linking File name and path together*** 7. ***Game Boards*** |
| 32 | Explain Open addressing method  **In Closed hashing, three techniques are used to resolve the collision:**   1. **Linear probing**   **Linear probing is one of the forms of open addressing. As we know that each cell in the hash table contains a key-value pair, so when the collision occurs by mapping a new key to the cell already occupied by another key, then linear probing technique searches for the closest free locations and adds a new key to that empty cell. In this case, searching is performed sequentially, starting from the position where the collision occurs till the empty cell is not found.**   1. **Quadratic probing**   **In case of linear probing, searching is performed linearly. In contrast, quadratic probing is an open addressing technique that uses quadratic polynomial for searching until a empty slot is found. It can also be defined as that it allows the insertion ki at first free location from (u+i2 )%m where i=0 to m-1.**   1. **Double Hashing technique**   **Double hashing is an open addressing technique which is used to avoid the collisions. When the collision occurs then this technique uses the secondary hash of the key. It uses one hash value as an index to move forward until the empty location is found. In double hashing, two hash functions are used. Suppose h1(k) is one of the hash functions used to calculate the locations whereas h2(k) is another hash function. It can be defined as "insert ki at first free place from (u+v\*i)%m where i=(0 to m-1)". In this case, u is the location computed using the hash function and v is equal to (h2(k)%m)** |
| Long Type | |
| 1 | 1. What is data structure? Briefly discuss the classification of data structure with examples.   Lightbox |
|  | b) What is pseudo code ? Write a pseudo code to delete an element from the array  **A pseudocode is defined as a step by step description of an algorithm.**  **It does not use any programming language in its representation instead it used the simple English language text as it is intended for human understanding rather than machine reading.**  **It is intermediate state between an idea and its implementation in a high level language.** |
| 2 | 1. Write a complete C program to implement binary search   void binary(int arr[], int size, int key)  {      int l = 0;      int h = size;      while (l <= h)      {          int mid = (l + (h)) / 2;          if (arr[mid] == key)          {              printf("%d is present at %d position ", key, mid + 1);              return;          }          else if (arr[mid] < key)          {              l = mid + 1;          }          else              h = mid - 1;      }      printf("%d is not present ", key);  } |
|  | 1. Write an algorithm to insert an element into the array at a given position   void add(int arr[], int pos,int size,int key){      int i = size;      for ( i = size; i >= pos; i--)      {          arr[i] = arr[i-1];      }      arr[pos-1] = key;      size++;      for (int i = 0; i <size; i++)      {         printf("%d ",arr[i]);      }  } |
| 3 | 1. What is sparse matrix? Discuss various types of commonly used sparse matric   parse matrices are those matrices that have the majority of their elements equal to zero. In other words, the sparse matrix can be defined as the matrix that has a greater number of zero elements than the non-zero elements.  Three types of Sparse Matrix  **types of Sparse Matrices**  **There are different variations of sparse matrices, which depend on the nature of the sparsity of the matrices. Based on these properties, sparse matrices can be**   * **Regular sparse matrices** * **Irregular sparse matrices / Non - regular sparse matrices**   Regular sparse matrices  A regular sparse matri**x** is a square matrix with a well-defined sparsity pattern, i.e., non-zero elements occur in a well-defined pattern. The various types of regular sparse matrices are:   * Lower triangular regular sparse matrices   Types of Sparse Matrices Types of Sparse Matrices   * Upper triangular regular sparse matrices   Types of Sparse Matrices Types of Sparse Matrices   * Tri-diagonal regular sparse matrices * he tridiagonal regular sparse matrix where all non-zero elements lie on one of the three diagonals, the main diagonal above and below.   Types of Sparse Matrices   * **Storing Tri-diagonal regular sparse matrices** * In a tri-diagonal regular sparse matrix, all the non-zero elements are stored in a 1-dimensional array row by row. * Types of Sparse Matrices |
|  | b) Represent the following sparse matrix in triplet format |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | | | | | | | | | **Row no** | **0** | **0** | **0** | **1** | **1** | **2** | **4** | **5** | | **5** | | **Column no** | **0** | **1** | **5** | **1** | **2** | **3** | **0** | **2** | | **5** | | **Non-zero element** | **20** | **15** | **-10** | **10** | **3** | **-6** | **90** | **30** | | **15** | |
| 4 | Write a menu driven C program to implement the following operation on stack using array  I) PUSH ii. POP iii. DIPLAY to display content of  Stack.  #include <stdio.h>  #include <stdlib.h>  #define max 5  int stack[max];  int top = -1; // 0,1,2,3,4  void push()  {      if (top == max - 1)      {          printf("\n overflow condition");      }      else      {          top++;          int x;          printf("\n what do yu wnt to enter: ");          scanf("%d", &x);          stack[top] = x;      }  }  void pop()  {      if (top == -1)      {          printf("\n underflow condition");      }      else          printf("%d is popped", stack[top]);      top--;  }  void display()  {      if (top == -1)      {          printf("\n No elements to display");      }      else      {          printf("\n  elements are : ");          for (int i = 0; i <= top; i++)          {              printf("%d ", stack[i]);          }      }  }  int main()  {      int ch;      top = -1;      while (1)      {          printf("\n Stack Operation");          printf("\n 1 Push");          printf("\n 2 Pop");          printf("\n 3 Peek");          printf("\n 4 IsEmpty");          printf("\n 5 IsFull");          printf("\n 6 Display");          printf("\n 7 Exit");          printf("\nEnter Your Choice: ");          scanf("%d", &ch);          switch (ch)          {          case 1:              push();              break;          case 2:              pop();              break;            case 3:              display();              break;          case 4:              exit(0);          default:              printf("\n Please enter between (1-4)");          }      }  } |
| 5 | Write a menu driven C program to implement the following operations on linear queue   1. INSERT ii. DELETE iii. DISPLAY to display content of queue   #define max 4  #include <stdio.h>  #include <stdlib.h>  int Queue[max];  int front = -1;  int rear = -1;  void enqueue();  {      if (rear == max - 1)      {          printf("\noverflow Condition");      }      else      {          if (front == -1)          {              front = 0;          }          rear += 1;          int x;          printf("\n what do you want to add");          scanf("%d", &x);          Queue[rear] = x;      }  }  void dequeue()  {      if (rear == -1)          printf("\nno elements to perform Dequeue ");      else      {          int y;          y = Queue[front];          if (front == rear)          {              front = -1;              rear = -1;          }          else          {              front++;          }          printf("%d is removed", y);      }  }  void traverse()  {      if (rear == -1)          printf("\nno elements to Traverse ");      else      {          int i;          printf("\n Elements in queue are: ");          for (i = front; i <= rear; i++)          {              printf("%d ", Queue[i]);          }      }  }  int main()  {      int ch;      printf("\n what do you want to perform");      while (1)      {          printf("\n Stack Operation");          printf("\n 1 Enqueue");          printf("\n 2 Dequeu");          printf("\n 3 traverse");          scanf("%d", &ch);          switch (ch)          {          case 1:              enqueue();              break;          case 2:              dequeue();              break;          case 3:              traverse();              break;          case 7:              exit(12);          default:              printf("\n Please enter between (1-7)");          }      }  } |
| 6 | 1. Write an algorithm to display circular queue. 2. START 3. If front=-1 write “no elements to display” 4. otherwise   If rear = front display queue[rear]  Else if (rear>front)  For i= rear to front  Display queue[i]  Else  For i= front to i=max-1  Display queue[i];  For I = 0 to rear  Display queue[i] |
|  | b) Write algorithm to PUSH and POP elements from stack  implemented using linked List |
| 7 | 1. Write a C program to create and display a singly linked list.   #include <stdio.h>  #include <stdlib.h>  struct node  {      int data;      struct node \*next;  };  struct node \*head = NULL;  struct node \*new;  struct node \*current;  void create()  {      char ch;      do      {          new = (struct node \*)malloc(sizeof(struct node));          printf("\n data?? ");          scanf("%d", &new->data);          new->next = NULL;          printf("\nmore??");          if (head == NULL)          {              head = new;              current = new;          }          else          {              current->next = new;              current = new;          }          ch = getch();      } while (ch == 'y');  }  void display()  {      current = head;      while (current != NULL)      {          printf("%d -> ", current->data);          current = current->next;      }  }  int main()  {      create();      display();      return 0;  } |
|  | 1. Write an algorithm to insert a node at the beginning of a doubly linked list 2. **Step 1:** IF ptr = NULL     Write OVERFLOW  Go to Step 9  [END OF IF]   1. **Step 2:** SET NEW\_NODE = ptr 2. **Step 3:** SET ptr = ptr -> NEXT 3. **Step 4:** SET NEW\_NODE -> DATA = VAL 4. **Step 5:** SET NEW\_NODE -> PREV = NULL 5. **Step 6:** SET NEW\_NODE -> NEXT = START 6. **Step 7:** SET head -> PREV = NEW\_NODE 7. **Step 8:** SET head = NEW\_NODE 8. **Step 9:** EXIT   D:\data structure and algo\LinkedList\01Ll.c Insertion in doubly linked list at beginning |
| 8 | 1. Write an algorithm to insert a node in a singly linked list at the desired position.   **to insert a given data at a specified position, the below algorithm is to be followed:**   * **Traverse the Linked list upto *position-1* nodes.** * **Once all the *position-1* nodes are traversed, allocate memory and the given data to the new node.** * **Point the next pointer of the new node to the next of current node.** * **Point the next pointer of current node to the new node.** |
|  | b) Explain with a suitable diagram to implement queue using linked list. Further write the |
| 9 | a) Write a C function to delete a node from a desired position of singly linked list |
|  | 1. Write an algorithm / C function to traverse a circular linked list with at least one node.   void display()  {      current = head;      while (current->next!= current)      {          printf("%d -> ", current->data);          current = current->next;      }  } |
| 10 | A) What is binary search tree (BST) ? For the given sequence of numbers construct a BST :  34 , 23 , 67 , 45 , 12 , 54 , 87 , 43 , 98 , 75 , 84 , 93 , 31 |
|  | 1. From the above constructed BST delete 12,67,93,34 sequentially. Show the reusltant BST after each deletion. |

|  |  |
| --- | --- |
| 11 | a) Write BFS algorithm for graph traversals. Perform BFS traversal for the following graph by considering start node as 0. |
|  | Write DFS algorithm for graph traversals. Perform DFS traversal for the above graph by considering start node as 0 |
| 12 | 1. What is AVL tree? State its properties. Construct an AVL tree by inserting data as per following order:     *MAR*. MAY, NOV, AUG, APR, *JAN*, DEC, JUL, *FEB*, JUN, OCT, SEP   * AVL trees are special kind of binary search trees. * In AVL trees, height of left subtree and right subtree of every node differs by at most one. * AVL trees are also called as **self-balancing binary search trees**. * **AVL trees can self-balance:** One of the primary concerns of computer science professionals is ensuring that their trees are balanced, and AVL trees have a higher likelihood of being balanced. An unbalanced tree means operations will take longer to complete, resulting in time-consuming lookup applications. The longer it takes to balance a tree, the longer the search will take. An AVL tree, also known as a self-balancing binary search tree, can perform three major operations: search, insert, and delete. * It is not skewed in any way. * To insert or delete a node will take low time complexity. * **It also provides faster search operations:** The most important benefit of AVL trees is that they perform faster search operations than BSTs, red-black trees, etc. This means that users can complete their tasks much faster than if they used other search operations. This is typically required for coding to ensure that projects are completed on time and in a reliable manner. * AVL tree also have the Balancing capabilities with a different type of rotation * It performs faster searches than Red-Black Trees. * Better searching time complexity than other trees, such as the binary Tree. * Height must not be greater than log(N), where N is the total number of nodes in the Tree. |
|  | b) Write C procedures/functions for the three different types of binary tree traversals |
| 13 | a) Discuss the Bubble sort technique taking the following array as example ? Explain each pass. 12,34,5,78,4,56,10,23,1 |
|  | b) Write a C program to implement Bubble sort technique to sort the elements of an integer array in descending order. |
| 14 | a) Explain how selection works. Arrange the following elements in ascending order using quicksort showing the result of every pass. |
|  | 1. Write the program to implement Insertion Sort |
| 15 | 1. What is hashing? Discuss any three hash function with example.   Hashing is one of the searching techniques that uses a constant time.  The time complexity in hashing is O(1) |
|  | b) Discuss the Linear Probing Method. Discuss the disadvantages of Linear Probing Method with an example. |

What is abstract data type?

An abstract data type is an abstraction of a data structure that provides only the interface to which the data structure must adhere. The interface does not give any specific details about something should be implemented or in what programming language.

In other words, we can say that abstract data types are the entities that are definitions of data and operations but do not have implementation details. In this case, we know the data that we are storing and the operations that can be performed on the data, but we don't know about the implementation details. The reason for not having implementation details is that every programming language has a different implementation strategy for example; a C data structure is implemented using structures while a C++ data structure is implemented using objects and classes.

**For example,** a List is an abstract data type that is implemented using a dynamic array and linked list. A queue is implemented using linked list-based queue, array-based queue, and stack-based queue. A Map is implemented using Tree map, hash map, or hash table.