**1. What do you mean by a Data structure?**

Ans) A Data structure is a programming construct used to model real life data and stores the data efficiently in the system’s memory so that we can execute instructions on it and get meaningful results. Basic data structures are arrays, lists, maps etc.

**2. What are some of the applications of DS?**

Ans) Some common Data structure and their applications are:

1. Array can be used for sorting elements, can perform matrix operation and can be used in CPU scheduling.

2. Stack is used in Expression evaluation, forward and backward feature in web browsers, syntax parsing. Used in algorithms like tower of Hanoi.

3. Queue is used when a resource is shared among multiple consumers like in cpu scheduling, disk scheduling. It is also used in Palindrome recognition.

4. Tree is used as dictionary.

5. Hash table is used for fast data lookup, database indexing, caches.

6. Graphs are used to represent networks.

**3. What are the advantages of a Linked list over an array?**

a) The Size of LinkedList is incremented at runtime. But in Array it is not possible.

b) The number of elements in the linked list are limited to the available memory space while the number of elements in the array is limited to the size of an array.

c) LinkedList stores the elements dynamically while array stores statically.

**4. Write the syntax in C to create a node in the singly linked list?**

Ans) Struct Node

{

**int** data;

Node \*next;

};

**5. What is the use of a doubly-linked list when compared to that of a singly**

**linked list?**

Ans) In singly linked list traversal is possible only in forward direction whereas the doubly linked list can be traversed both in forward and backward direction. It is more efficient in the delete and reverse operation whenever the pointer to node is given in doubly linked list.

**6. What is the difference between an Array and Stack?**

Ans) The main difference between array and stack is accessing of data and traversing of data in Array we can access data by index only or traverse the array from the index where we want but in case of stack we can only access the data in last in first out fashion and traversing is also done from last input to first input only.

**7. What are the minimum number of Queues needed to implement the priority queue?**

Ans) Minimum 2 queues are used. One for storing data and another is used for priorities.

**8. What are the different types of traversal techniques in a tree?**

Ans) 1. Preorder Traversal

2. Postorder Traversal

3. Inorder Traversal

**9. Why it is said that searching a node in a binary search tree is efficient than that of a simple binary tree?**

Ans) Binary Search Tree allows for fast retrieval of elements stored in the tree as each node key is thoroughly compared with the [root](http://www.differencebetween.net/science/nature/difference-between-root-and-stem/) node, which discards half of the tree. It is already sorted as all element in left tree is smaller than root element and all element in right tree is greater than root element.

**10. What are the applications of Graph DS?**

Ans) Graph DS is used to model scenarios where a component has relations with multiple components. Graph DS is used for modelling networks such as social media networks etc.

**11. Can we apply Binary search algorithm to a sorted Linked list?**

Ans) Yes, we can implement in in java language with the use of Array List.

**12. When can you tell that a Memory Leak will occur?**

Ans) Memory leak occurs when we create a memory in the heap and forget to delete it after using it. It may also occur when a computer program incorrectly manages memory allocations in a way that memory which is no longer needed is not released. A memory leak may also happen when an object is stored in memory but cannot be accessed by the running code.

**13. How will you check if a given Binary Tree is a Binary Search Tree or not?**

Ans) 1. If a node is a left child, then its key and the keys of the nodes in its right subtree are less than its parent’s key.

2. If a node is a right child, then its key and the keys of the nodes in its left subtree are greater than its parent’s key.

**14. Which data structure is ideal to perform recursion operation and why?**

Ans) Stack has the LIFO (Last In First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls.

**15. What are some of the most important applications of a Stack?**

Ans) Stacks are used to implement recursion, can be used to balance parenthesis and is used to implement calculations following BODMAS rule.

**16. Convert the below given expression to its equivalent Prefix And Postfix notations.**

**17. Sorting a stack using a temporary stack.**

Ans)

**package** HourOfCode;

**import** java.util.\*;

**class** Day7

{

**public** **static** Stack<Integer> reverse(Stack<Integer> Original)

{

Stack<Integer> reverse = **new** Stack<Integer>();

**while**(!Original.isEmpty())

{

**int** temp = Original.pop();

**while**(!reverse.isEmpty() && reverse.peek() > temp)

{

Original.push(reverse.pop());

}

reverse.push(temp);

}

**return** reverse;

}

**public** **static** **void** main(String args[])

{

Stack<Integer> Original = **new** Stack<Integer>();

Original.add(1);

Original.add(2);

Original.add(3);

Original.add(4);

Original.add(5);

Original.add(6);

Stack<Integer> reverse = *reverse*(Original);

System.***out***.println("Sorted numbers are:");

**while** (!reverse.empty())

{

System.***out***.print(reverse.pop()+" ");

}

}

}

**18. Program to reverse a queue.**

Ans)

package assignment;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

public class reverseQueue

{

static Queue<Integer> queue;

static void Print()

{

while (!queue.isEmpty()) {

System.out.print( queue.peek() + " ");

queue.remove();

}

}

static void revqueue()

{

Stack<Integer> stack = new Stack<>();

while (!queue.isEmpty())

{

stack.add(queue.peek());

queue.remove();

}

while (!stack.isEmpty())

{

queue.add(stack.peek());

stack.pop();

}

}

public static void main(String args[])

{

queue = new LinkedList<Integer>();

queue.add(10);

queue.add(20);

queue.add(30);

queue.add(40);

queue.add(50);

queue.add(60);

queue.add(70);

queue.add(80);

queue.add(90);

queue.add(100);

revqueue();

Print();

}

}

**19. Program to reverse first k elements of a queue**

Ans)

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

public class Reverse\_k\_element\_queue {

static Queue<Integer> queue;

static void reverseQueueFirstKElements(int k)

{

if (queue.isEmpty() == true

|| k > queue.size())

return;

if (k <= 0)

return;

Stack<Integer> stack = new Stack<Integer>();

for (int i = 0; i < k; i++) {

stack.push(queue.peek());

queue.remove();

}

while (!stack.empty()) {

queue.add(stack.peek());

stack.pop();

}

for (int i = 0; i < queue.size() - k; i++) {

queue.add(queue.peek());

queue.remove();

}

}

static void Print()

{

while (!queue.isEmpty()) {

System.out.print(queue.peek() + " ");

queue.remove();

}

}

public static void main(String args[])

{

queue = new LinkedList<Integer>();

queue.add(10);

queue.add(20);

queue.add(30);

queue.add(40);

queue.add(50);

queue.add(60);

queue.add(70);

queue.add(80);

queue.add(90);

queue.add(100);

int k = 5;

reverseQueueFirstKElements(k);

Print();

}

}

**20. Program to return the nth node from the end in a linked list**

Ans)

package assignment;

public class linkedListNth {

Node head;

class Node {

int data;

Node next;

Node(int d) {

data = d;

next = null;

}

}

void printNthFromLast(int n) {

int len = 0;

Node temp = head;

while (temp != null) {

temp = temp.next;

len++;

}

if (len < n)

return;

temp = head;

for (int i = 1; i < len - n + 1; i++)

temp = temp.next;

System.out.println(temp.data);

}

public void push(int new\_data) {

Node new\_node = new Node(new\_data);

new\_node.next = head;

head = new\_node;

}

public static void main(String[] args) {

linkedListNth lst = new linkedListNth();

lst.push(20);

lst.push(4);

lst.push(15);

lst.push(35);

lst.printNthFromLast(4);

}

}

**Q21. Reverse a linked list.**

Ans)

package assignment;

public class reverseLinkedList {

static Node head;

static class Node {

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

Node reverse(Node node)

{

Node prev = null;

Node current = node;

Node next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

node = prev;

return node;

}

void printList(Node node)

{

while (node != null) {

System.out.print(node.data + " ");

node = node.next;

}

}

public static void main(String[] args)

{

reverseLinkedList list = new reverseLinkedList();

list.head = new Node(85);

list.head.next = new Node(15);

list.head.next.next = new Node(4);

list.head.next.next.next = new Node(20);

System.out.println("Given Linked list");

list.printList(head);

head = list.reverse(head);

System.out.println("");

System.out.println("Reversed linked list ");

list.printList(head);

}

}

**Q22. Replace each element of the array by its rank in the array.**

**Ans.**

package assignment;

import java.util.Arrays;

import java.util.Map;

import java.util.TreeMap;

public class arrayRank

{

public static void transform(int[] arr)

{

Map<Integer, Integer> map = new TreeMap<>();

for (int i = 0; i < arr.length; i++) {

map.put(arr[i], i);

}

int rank = 1;

for (var val : map.values()) {

arr[val] = rank++;

}

}

public static void main(String[] args)

{

int[] A = { 10, 8, 15, 12, 6, 20, 1 };

transform(A);

System.out.println(Arrays.toString(A));

}

}

**Q23. Check if a given graph is a tree or not.**

**Ans.**

package assignment;

import java.io.\*;

import java.util.\*;

class graphTree

{

private int V;

private LinkedList<Integer> adj[];

Graph (int v)

{

V = v;

adj = new LinkedList[v];

for (int i=0; i<v; ++i)

adj[i] = new LinkedList ();

}

void addEdge(int v,int w)

{

adj[v].add(w);

adj[w].add(v);

}

Boolean isCyclicUtil(int v, Boolean visited[], int parent)

{

visited[v] = true;

Integer i;

Iterator<Integer> it = adj[v].iterator();

while (it.hasNext())

{

i = it.next();

if (!visited[i])

{

if (isCyclicUtil(i, visited, v))

return true;

}

else if (i != parent)

return true;

}

return false;

}

Boolean isTree()

{

Boolean visited[] = new Boolean[V];

for (int i = 0; i < V; i++)

visited[i] = false;

if (isCyclicUtil(0, visited, -1))

return false;

for (int u = 0; u < V; u++)

if (!visited[u])

return false;

return true;

}

public static void main(String args[])

{

graphTree g1 = new graphTree(4);

g1.addEdge(0,1);

g1.addEdge(1, 2);

g1.addEdge(2, 3);

g1.addEdge(3, 4);

if (g1.isTree())

System.out.println("Graph is Tree");

else

System.out.println("Graph is not Tree");

graphTree g2 = new graphTree(5);

g2.addEdge(0,1);

g2.addEdge(1,2);

g2.addEdge(2,3);

g2.addEdge(3,4);

g2.addEdge(4,5 );

if (g2.isTree())

System.out.println("Graph is Tree");

else

System.out.println("Graph is not Tree");

}

}

**Q24. Find out the Kth smallest element in an unsorted array**

**Ans.**

package assignment;

import java.util.Arrays;

public class kthSmallest {

public static int kthSmallest(Integer[] arr, int k)

{

Arrays.sort(arr);

return arr[k - 1];

}

public static void main(String[] args)

{

Integer arr[] = new Integer[] { 12, 3, 5, 7, 19 };

int k = 2;

System.out.print("K'th smallest element is " + kthSmallest(arr, k));

}

}

**Q25. How to find the shortest path between two vertices.**

**Ans.**

1. Input the graph.
2. Input the source and destination nodes.
3. Find the paths between the source and the destination nodes.
4. Find the number of edges in all the paths and return the path having the minimum number of edges.
5. In this way we will get the shortest path between the two vertices.
6. Shortest path between two vertices is determined by using a shortest path algorithm called Dijkstra’s Algorithm. Dijkstra’s Algorithm is also called single source shortest path algorithm.