DSA PRACTICE QUESTIONS

Solutions By Himanshu Sharma

1. **How can we determine that an algorithm is efficient or not? List out the measures?**

   Ans.

   The efficiency of an algorithm can be computed by determining the number of resources it consumes. The primary resources that an algorithm consumes are:

   - Time: The CPU time required to execute the algorithm
   - Space: The amount of memory used by the algorithm for execution

   The lesser resources that an algorithm uses, the more efficient it is.

2. **What are asymptotic notations, explain the Big O notation and its importance.**

   Ans.

   Definition: In mathematical analysis, asymptotic analysis of algorithm is a method of defining the mathematical boundaries of its run-time performance.
   Using the asymptotic analysis, we can easily estimate about the average case, best case and worst-case scenario of an algorithm.
   In Simple words it is used to mathematically calculate the running time of any operation inside an algorithm.
   Asymptotic Algorithm analysis is to estimate the time complexity function for arbitrarily large inputs.

   Big O notation is a way to measure how well a computer algorithm scales as the amount of data involved increases.
   Big O notation is the most widely used method which describes algorithm complexity- the execution time required or space used in memory or disk by an algorithm.

   The importance of big o notation in a program is 'it is used to measure the performance of any algorithm by providing the order of growth of a function it give the upper bound A by which we can make sure that the function will never grow faster that the upper bound.'

3. **What is Data structure and explain its operations?**

   Ans.

   Data Structure is all about how you can organize/structure your data so that you can store it and use it efficiently.
   Its various operations are:

- **Insertion:** Insertion operation deals with adding an element to the data structure.
- **Deletion:** Deletion operation removes an element from the data structure.
- **Traversing:** We traverse a data structure when we visit each and every element in the structure. Traversing is required to carry out certain specific operations on the data structure.
- **Searching:** This operation is performed to search for a particular element or a key.
- **Sorting:** Sorting operation involves arranging the elements in a data structure in a particular order either ascending or descending.

4. **What is linked list and difference between array and linked list?**

Ans.

- Linked List provides an alternative to an array-based structure.
- Array has size limitation, but Linked list didn't have that limitation
- Linked List in its simplest form, is a collection of **nodes that collectively form a linear sequence.**

Arrays	Linked Lists
An array is a collection of elements of a similar data type.	Linked List is an ordered collection of elements of the same type in which each element is connected to the next using pointers.
Array elements can be accessed randomly using the array index.	Random accessing is not possible in linked lists. The elements will have to be accessed sequentially.
Data elements are stored in contiguous locations in memory.	New elements can be stored anywhere and a reference is created for the new element using pointers.
Insertion and Deletion operations are costlier since the memory locations are consecutive and fixed.	Insertion and Deletion operations are fast and easy in a linked list.
Memory is allocated during the compile time (Static memory allocation).	Memory is allocated during the run-time (Dynamic memory allocation).
Size of the array must be specified at the time of array declaration/initialization.	Size of a Linked list grows/shrinks as and when new elements are inserted/deleted.

5. **Implement the insertion and deletion operation in a singly, doubly and circular linked list?**

Ans.

Singly Linked List

```java
class NodeS {
    int data;
    NodeS next;

    public NodeS(int data) {
        this.data = data;
    }
}

public class SinglyLinkedList {
    NodeS head;

    public boolean isEmpty(){
        return head==null;
    }

    public void add(int val){
        NodeS newnode = new NodeS(val);
        if(isEmpty()){
            head = newnode;
        }
        else {
            NodeS temp = head;
            while(temp.next!=null) {
                temp = temp.next;
            }
            temp.next=newnode;
        }
    }

    public void delete(int val){
        if(isEmpty()){
            System.out.println("List is Empty.");
        }
        else{
            NodeS temp=head;
            if(temp.data==val){
                head=head.next;
            }
            else {
                while (temp.next != null && temp.next.data != val) {
                    temp = temp.next;
                }
                if (temp.next == null) {
                    System.out.println("No such value present.");
                } else {
                    temp.next = temp.next.next;
                }
            }
        }
    }

    public void display(){
        if(isEmpty()){
            System.out.println("List is Empty.");
        }
        else {
            NodeS temp=head;
            while(temp!=null){
                System.out.print(temp.data+"->");
                temp=temp.next;
            }
            System.out.println("null");
        }
    }

    public static void main(String[] args) {
        SinglyLinkedList list = new SinglyLinkedList();
        list.add(2);
        list.add(4);
        list.display(); // 2->4->null
        list.add(3);
        list.delete(4);
        list.display(); // 2->3->null
    }
}
```

Doubly Linked List

```java
class NodeD {
    int data;
    NodeD next,prev;

    public NodeD(int data) {
        this.data = data;
    }
}

public class DoublyLinkedList {
    NodeD head,tail;

    public boolean isEmpty(){
        return head==null;
    }

    public void add(int val){
        NodeD newnode = new NodeD(val);
        if(isEmpty()){
            newnode.prev=head;
            head = tail = newnode;
        }
        else {
            NodeD temp = head;
            while(temp.next!=null) {
                temp = temp.next;
            }
            newnode.prev=temp;
            temp.next=newnode;
            tail=newnode;
        }
    }

    public void delete(int val){
        if(isEmpty()){
            System.out.println("List is Empty.");
        }
        else{
            NodeD temp=head;
            if(temp.data==val){
                head=head.next;
                head.prev=null;
            }
            else {
                while (temp.next != null && temp.next.data != val) {
                    temp = temp.next;
                }
                if (temp.next == null) {
                    System.out.println("No such value present.");
                } else {
                    temp.next.next.prev=temp;
                    temp.next = temp.next.next;

                }
            }
        }
    }

    public void display(){
        if(isEmpty()){
            System.out.println("List is Empty.");
        }
        else {
            NodeD temp=head;
            while(temp!=null){
                System.out.print(temp.data+"->");
                temp=temp.next;
            }
            System.out.println("null");
        }
    }

    public void displayRev(){
        if(isEmpty()){
            System.out.println("List is Empty.");
        }
        else {
            NodeD temp=tail;
            while(temp!=null){
                System.out.print(temp.data+"->");
                temp=temp.prev;
            }
            System.out.println("null");
        }
    }

    public static void main(String[] args) {
        DoublyLinkedList list = new DoublyLinkedList();
        list.add(2);
        list.add(4);
        list.display(); // 2->4->null
        list.add(3);
        list.delete(4);
        list.display(); // 2->3->null
        list.displayRev(); // 3->2->null
    }
}
```

Circular Singly Linked List

```java
class NodeS {
    int data;
    NodeS next;

    public NodeS(int data) {
        this.data = data;
    }
}

public class CircularSinglyLinkedList {
    NodeS head;

    public boolean isEmpty(){
        return head==null;
    }

    public void add(int val){
        NodeS newnode = new NodeS(val);
        if(isEmpty()){
            head = newnode;
            newnode.next=head;
        }
        else {
            NodeS temp = head;
            while(temp.next!=head) {
                temp = temp.next;
            }
            newnode.next=head;
            temp.next=newnode;
        }
    }

    public void delete(int val){
        if(isEmpty()){
            System.out.println("List is Empty.");
        }
        else{
            NodeS temp=head;
            if(temp.data==val){
                while(temp.next!=head){
                    temp=temp.next;
                }
                temp.next=head.next;
                head=head.next;
            }
            else {
                while (temp.next != head && temp.next.data != val) {
                    temp = temp.next;
                }
                if (temp.next == head) {
                    System.out.println("No such value present.");
                } else {
                    temp.next = temp.next.next;
                }
            }
        }
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("List is Empty.");
        } else {
            NodeS temp = head;
            do {
                System.out.print(temp.data + "->");
                temp = temp.next;
            }
            while (temp != head);
            System.out.println("first-element");
        }
    }
    public static void main(String[] args) {
        CircularSinglyLinkedList list = new CircularSinglyLinkedList();
        list.add(2);
        list.add(4);
        list.display(); // 2->4->first-element
        list.add(3);
        list.display(); // 2->4->3->first-element
        list.delete(2);
        list.display(); // 4->3->first-element
    }
}
```

Circular Doubly Linked List

```java
class NodeD {
    int data;
    NodeD next,prev;

    public NodeD(int data) {
        this.data = data;
    }
}

public class CircularDoublyLinkedList {
    NodeD head,tail;

    public boolean isEmpty(){
        return head==null;
    }

    public void add(int val){
        NodeD newnode = new NodeD(val);
        if(isEmpty()){
            head=tail=newnode;
            newnode.next=newnode;
        }
        else {
            newnode.prev=tail;
            tail.next=newnode;
            head.prev=newnode;
            newnode.next=head;
            tail=newnode;
        }
    }

    public void delete(int val){
        if(isEmpty()){
            System.out.println("List is Empty.");
        }
        else{
            NodeD temp=head;
            if(temp.data==val){
                temp.next.prev=tail;
                tail.next=temp.next;
                head=temp.next;
            }
            else {
                while (temp.next != head && temp.next.data != val) {
                    temp = temp.next;
                }
                if (temp.next == head) {
                    System.out.println("No such value present.");
                } else {
                    temp.next.next.prev=temp;
                    temp.next = temp.next.next;
                }
            }
        }
    }

    public void display() {
        if (isEmpty()) {
            System.out.println("List is Empty.");
        } else {
            NodeD temp = head;
            do {
                System.out.print(temp.data + "->");
                temp = temp.next;
            }
            while (temp != head);
            System.out.println("first-element");
        }
    }

    public void displayRev(){
        if(isEmpty()){
            System.out.println("List is Empty.");
        }
        else {
            NodeD temp=tail;
            do{
                System.out.print(temp.data+"->");
                temp=temp.prev;
            }
            while(temp!=tail);
            System.out.println("last-element");
        }
    }

    public static void main(String[] args) {
        CircularDoublyLinkedList list = new CircularDoublyLinkedList();
        list.add(2);
        list.add(4);
        list.display(); // 2->4->first-element
        list.add(3);
        list.display(); // 2->4->3->first-element
        list.delete(2);
        list.display(); // 4->3->first-element
        list.add(5);
        list.display(); // 4->3->5->first-element
        list.displayRev(); // 5->3->4->last-element
    }
}
```

6. Reverse the string using stack.

Ans.

```
import java.util.Scanner;

class Node {
    char data;
    Node next;

    public Node(char data) {
        this.data = data;
    }
}

class StackByLL{
    Node top;

    public boolean isEmpty(){ return top==null; }

    public void push(char data){
        Node newnode=new Node(data);
        if(isEmpty()){
            top=newnode;
        }
        else {
            newnode.next=top;
            top=newnode;
```

```java
        }
    }

    public char pop(){
        char ans=0;
        if(!isEmpty()){
            ans=top.data;
            top=top.next;
        }
        else System.out.println("Stack Undelflow");
        return ans;
    }
}
public class RevByStack {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your string: ");
        String s=sc.nextLine();
        sc.close();
        StackByLL st=new StackByLL();
        for(int i=0;i<s.length();i++){
            st.push(s.charAt(i));
        }
        System.out.print("Reversed string: ");
        for(int i=0;i<s.length();i++){
            System.out.print(st.pop());
        }
    }
}
```

7. Check a given number is palindrome or not.

Ans.

```java
import java.util.Scanner;

class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
    }
}

```

```java
class StackLL{
    Node top;

    public boolean isEmpty(){ return top==null; }

    public void push(int data){
        Node newnode=new Node(data);
        if(isEmpty()){
            top=newnode;
        }
        else {
            newnode.next=top;
            top=newnode;
        }
    }

    public int pop(){
        int ans=0;
        if(!isEmpty()){
            ans=top.data;
            top=top.next;
        }
        else System.out.println("Stack Undelflow");
        return ans;
    }
}

public class Palindrome {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        sc.close();
        StackLL st=new StackLL();
        int temp=n;
        int ctr=0;
        while(temp>0){
            st.push(temp%10);
            temp/=10;
            ctr++;
        }
        boolean odd=true;
        if(ctr%2==0) odd=false;
        boolean isPal=true;
        for(int i=0;i<ctr;i++){
            if(odd && i==ctr/2) continue;
            if(st.pop()!=n%10) {
                isPal=false;
                break;
            }
            n/=10;
        }
        if(isPal) System.out.println("Palindrome");
        else System.out.println("Not Palindrome");
    }
}
```

8. Convert the following infix to postfix.

Ans.

Infix: ((A + B * C / D + E)

Symbol scanned	STACK	Expression P
(1)(	(	-
(2) (	((	-
(2) A	((	A
(3) +	((+	A
(4) B	((+	A B
(5) *	((+ *	A B
(6) C	((+ *	A B C
(7))	(	A B C * +
(8) /	(/	A B C * +
(9) D	(/	A B C * + D
(10) +	(+	A B C * + D /
(11) E	(+	A B C * + D / E
(12))	-	A B C * + D / E +

8.

Postfix: A B C * + D / E +

9. **What is the importance of converting infix to a postfix expression?**
Ans.
Infix expressions are readable and solvable by humans. We can easily distinguish the order of operators, and also can use the parenthesis to solve that part first during solving mathematical expressions. The computer cannot differentiate the operators and parenthesis easily, that's why postfix conversion is needed.

Postfix has a number of advantages over infix for expressing algebraic formulas.
First, any formula can be expressed without parenthesis.
Second, it is very convenient for evaluating formulas on computers with stacks.
Third, infix operators have precedence. For example, we know that a * b + c means (a * b) + c and not a * (b + c), because multiplication has been defined to have precedence over addition. Postfix eliminates this nuisance.

10. **Implement the queue using an array.**
Ans.

```
public class QueueByArray {
    int arr[];
    int front,rear;

    public QueueByArray(int size) {
        this.arr = new int[size];
        front=rear=-1;
    }

    public boolean isEmpty() { return front==-1; }
    public boolean isFull() { return rear==arr.length-1; }

    public void enqueue(int data){
        if(isFull()){
            System.out.println("Full");
        }
        else if (isEmpty()){
            front++;
            arr[++rear]=data;
        }
        else {
            arr[++rear]=data;
        }
    }

    public int dequeue(){
        int ans=0;
        if(isEmpty()){
            System.out.println("Empty");
        }
        else {
            ans=arr[front++];
```

```java
        }
        if(front==arr.length){
            front=rear=-1;
        }
        return ans;
    }

    public void display(){
        if(isEmpty()) System.out.println("Empty");
        else {
            for (int i = front; i <= rear; i++) {
                System.out.print(arr[i] + " ");
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        QueueByArray q=new QueueByArray(3);
        q.enqueue(5);
        q.enqueue(10);
        q.display(); // 5 10
        q.enqueue(15);
        q.display(); // 5 10 15
        q.enqueue(20); // Full
        q.dequeue();
        q.display(); // 10 15
        System.out.println(q.dequeue()); // 10
        q.display(); // 15
        q.dequeue();
        q.display(); // Empty
    }
}
```

11. Implement the queue using linked list.

Ans.

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

    public Node(int data) {
        this.data = data;
    }
}

public class QueueByLL {
    Node front, rear;

    public boolean isEmpty() { return front==null; }

    public void enqueue(int data){
        Node newnode=new Node(data);
        if(isEmpty()){
            front=rear=newnode;
        }
        else {
            rear.next=newnode;
            rear=newnode;
        }
    }

    public int dequeue(){
        int ans=0;
        if(isEmpty()){
            System.out.println("Empty");
        }
        else {
            ans=front.data;
            front=front.next;
        }
        return ans;
    }

    public void display(){
        if(isEmpty()) System.out.println("Empty");
        else {
            Node temp = front;
            while (temp != null) {
                System.out.print(temp.data + " ");
                temp=temp.next;
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        QueueByLL q=new QueueByLL();
        q.enqueue(5);
        q.enqueue(10);
        q.display(); // 5 10
        q.enqueue(15);
        q.display(); // 5 10 15
        q.enqueue(20);
        q.dequeue();
        q.display(); // 10 15 20
        System.out.println(q.dequeue()); // 10
        q.display(); // 15 20
        q.dequeue();
        q.display(); // 20
        q.dequeue();
        q.display(); // Empty
    }
}
```

12. What is the advantages of Circular Queue and Deque over Queue?

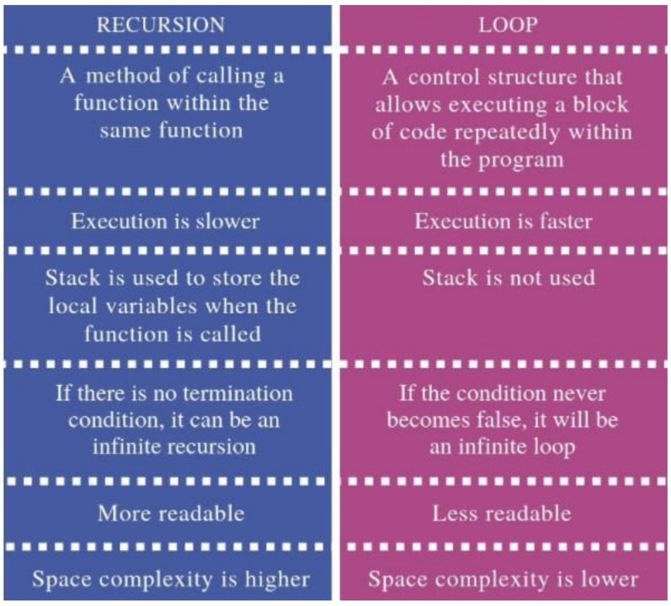
Ans.

Major advantages of using Circular Queue and Deque over Linear queue are the solution for space utilisation problem that we encounter with a linear queue. In linear queue we cannot enter more elements once the rear pointer reaches the end of the queue which gives us an overhead of wasted memory of those elements which are logically dequeued but physically still there and there's no way to use their place. Circular queue and deque allows us to use the extra space left in the queue after performing dequeue operation.

13. What is recursion? What are differences between recursion and loop?

Ans.

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.



RECURSION	LOOP
A method of calling a function within the same function	A control structure that allows executing a block of code repeatedly within the program
Execution is slower	Execution is faster
Stack is used to store the local variables when the function is called	Stack is not used
If there is no termination condition, it can be an infinite recursion	If the condition never becomes false, it will be an infinite loop
More readable	Less readable
Space complexity is higher	Space complexity is lower

14. Find a factorial of a number using recursion.

Ans.

```java
import java.util.Scanner;

public class FactByRecur {
    public int fact(int n){
        if(n==1 || n==0) return 1;
        return n*fact(n-1);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int n=sc.nextInt();
        sc.close();
        FactByRecur obj = new FactByRecur();
        System.out.println(obj.fact(n));
    }
}
```

15. Find the nth term of Fibonacci series by using recursion.

Ans.

```java
import java.util.Scanner;

public class NthFibonacci {
    public int n_Fib(int n){
        if(n<=1) return n;
        return n_Fib(n-1)+n_Fib(n-2);
    }
    public static void main(String[] args) {
     NthFibonacci obj = new NthFibonacci();
     Scanner sc = new Scanner(System.in);
     System.out.print("Enter the term: ");
     int n=sc.nextInt();
     sc.close();
     System.out.println(obj.n_Fib(n));
    }
}
```

16. Print the linked list in reverse order using recursion.

Ans.

```java
class NodeS {
    int data;
    NodeS next;

    public NodeS(int data) {
        this.data = data;
    }
}

public class SinglyLinkedList {
    NodeS head;

    public boolean isEmpty(){
        return head==null;
    }

    public void add(int val){
        NodeS newnode = new NodeS(val);
        if(isEmpty()){
            head = newnode;
        }
        else {
            NodeS temp = head;
            while(temp.next!=null) {
                temp = temp.next;
            }
            temp.next=newnode;
        }
    }
}

public class LL_Rev{
    public void display_Rev_Rec(NodeS head){
        if(head.next!=null) {
            display_Rev_Rec(head.next);
            System.out.print("<-"+head.data);
        }
        else {
            System.out.print("null<-"+head.data);
        }
    }

    public static void main(String[] args) {
        LL_Rev obj = new LL_Rev();
        SinglyLinkedList list=new SinglyLinkedList();
        list.add(10);
        list.add(20);
        list.add(30);
        obj.display_Rev_Rec(list.head); // null<-30<-20<-10
    }
}
```

17. Count the nodes of linked list by using recursion?

Ans.

```java
class NodeS {
    int data;
    NodeS next;

    public NodeS(int data) {
        this.data = data;
    }
}

public class SinglyLinkedList {
    NodeS head;

    public boolean isEmpty(){
        return head==null;
    }

    public void add(int val){
        NodeS newnode = new NodeS(val);
        if(isEmpty()){
            head = newnode;
        }
        else {
            NodeS temp = head;
            while(temp.next!=null) {
                temp = temp.next;
            }
            temp.next=newnode;
        }
    }

    public void delete(int val){
        if(isEmpty()){
            System.out.println("List is Empty.");
        }
        else{
            NodeS temp=head;
            if(temp.data==val){
                head=head.next;
            }
            else {
                while (temp.next != null && temp.next.data != val) {
                    temp = temp.next;
                }
                if (temp.next == null) {
                    System.out.println("No such value present.");
                } else {
                    temp.next = temp.next.next;
                }
            }
        }
    }
}

public class CountNodes {
    public int count_nodes(NodeS head){
        if(head==null) return 0;
        return 1+count_nodes(head.next);
    }
    public static void main(String[] args) {
        SinglyLinkedList list = new SinglyLinkedList();
        list.add(2);
        list.add(4);
        list.add(5);
        list.delete(4);
        list.add(6);
        CountNodes obj = new CountNodes();
        System.out.println(obj.count_nodes(list.head)); // 3
        }
}

```