Data Structure

Complexity-related InterView MCQ or Question

1.What is time complexity ?
Ans: The complexity is a way to describe the efficiency of an algorithm to determine how it's execution time grows reliably the input size . It depends on the size of input. Time complexity can be categorized into:
1. Best case O(1)
2. Average case O(log n)
3. Worst case O(N)

2.Why do we use time complexity?
Ans: We use time complexity to undustant how efficiently algorithms perform as the input size grows. It helps us compare algorithms , predict performance, and choose the best one for solving problems efficiently.

3. What is the time complexity of this following code ?

```
for(int i = 0; i < n; i++) {
        cout << i << endl;
}
```

Ans : O(N)

4. What is the time complexity of this following code?

```
for(int i =N ;i>1 ;i= i/2 ){
        cout << i << endl ;
}
```

Ans : O(LogN)

5. What is the time complexity of this following code?

```
for(int i =1 ;i<n ;i=i*2){
        cout << i << endl ;
}
```

Ans : O(LogN)

6.What is the time complexity of this following code?

```
for(int i =1 ; i*i<=n; i++){
        if(n%i==){
                cout << i << " " ;
                cout << n/i << " " ;
        }
}
```

Ans: O(sqrt(n)

7. What is the time complexity of this following code?

```
int i,j,k=0;
   for(i = n/2;i<=n;i++)   {
      for(j=2;j<=n;j*=2)  {
         k = k+n/2;
      }
   }
```

Ans: O(NLogN)

8. What is space complexity?

Ans : Space complexity is a measure of the amount of memory an algorithm uses relative to the size of the input. It helps us understand how the memory requirements of an algorithm grow as the input size increases . Space complexity is often expressed using Big-O notation

O(1) - constant space - memory usage does not change with input size.

O(n) - Linear space - memory usage grows linearly with input size.

O(n^2) Quadratic space - memory usage grows proportionally to the square of the input size.

9. What is the time complexity of a linear search algorithm?

i) O(n) ii) O(logN) iii) O(n^2) iv) O(NlogN)

Ans : i) O(n)

10. What is the time complexity of the bubble sort algorithm?

i) O(n) ii) O(logN) iii) O(n^2) iv) O(NlogN)

Ans : iii) O(n^2)

LinkedList related InterView MCQ or Question

1. What is a linked list ?

Ans : A linked list may be defined as a linear data structure which can store a collection of items, in another way the linked list can be utilized to store various objects of similar types , Each element or unit of the list is indicated as a node. Each node contains its data and the address of the next node. It is similar to a chain.

2. How many types of Linked List exist?

i) 2 ii) 3 iii)4 iv)5

Ans : iii) 4

3. What is a singly linked list?

Ans: The singly linked list includes nodes which contain a data field and next field . The next field further points to the next node in the line of nodes. It has two fields. an integer value and a pointer value (a link to the next node).

4. How to insert a new node in a singly linked list at the beginning , shown by a short code ?
Ans :

```
void insert_at_head(Node*&head,int v)
{
    Node* newNode = new Node(v);
    newNode->next= head;
    head=newNode;
    cout<<"Inserted at new head"<<endl;
}
```

5. How to insert a new node in a singly linked list at end or tail position, shown by a code ?
Ans :

```
void insert_at_tail(Node*head, int val){
    Node* newNode = new Node(val);
    int (head ==NULL){
        head = newNode;
        return;
    }
    Node * tmp = head;
    while(tmp->next !=Null){
        tmp= tmp->next;
    }
    tmp->next= newNode;
}
```

6. How to insert a new node in a singly linked list at any position, shown by a code ?

Ans:

```
void insert_at_any_position(Node*head ,int pos, int val ){
    Node * newNode = new Node(val);

    if(head == NULL){
        head= newNode;
        return;
    }
    Node* tmp = head;
    for(int i=1; i<=post-1; i++)
    {
        tmp = tmp->next;
    }
```

```
    newNode->next= tmp->next;
    tmp->next = newNode;
}
```

7. How to delete a node from the head in a singly linked list ?

Ans:

```
void Delete_node_at_head(Node*&head){
    Node* deleteNode = head;
    head = head->next;
    delete deleteNode;

}
```

8. How to delete a node at tail of a singly linked list ?

Ans:

```
void Delete_node_at_tail(Node*&head){
    if (head == NULL){
        cout<<"The list is empty!"
        return;
    }
    Node * tmp = head;
    while(tmp->next->next != NULL){
        tmp = tmp-next;
    }
    Node *deleteNode = tmp->next->next;
    delete deleteNode;
    tmp->next=NULL;

}
```

9. How to delete a node at any position in a singly linked list ?

```
Void Delete_node_at_any_position(Node *head,int pos){
    Node * tmp = head;
    if (head ==NULL){
        cout<<"The list is empty.";
        return;
    }

    Node *  tmp = head;
    for (int i=1; i<=pos-1; i++)
    {
        tmp= tmp->next;
    }
    Node * deleteNode = tmp->next;
    tmp->next= tmp->next->next;
    delete deleteNode;

}
```

10. What is a doubly linked list ?
Ans: A doubly linked list is a data structure that consists of a set of nodes, each of which contains a value and two pointers, one pointing to the previous node in the list and one pointing to the next node in the list. This allows for efficient traversal of the list in both directions, making it suitable for applications where frequent insertions and deletions are required.

11. How many fields have a doubly linked list ?
i) One ii) Two iii) Three iv) Four
Ans : iii) Three

12. How to insert a new node in a doubly linked list at the beginning , shown by a short code ?
Ans:

```
void Insert_at_head(Node*&head, Node*&tail,int val ){
```

```
    Node * newNode = new Node(val);
    if(head==NULL){
        head = newNode;
        tail = newNode;
        return;
    }
    newNode->next = head;
    head->prev= newNode;
    head= newNode;
}
```

13. How to insert a new node in a doubly linked list at end or tail position, shown by a code ?
Ans:

```
void Insert_at_tail(Node*&head , Node*&tail, int val){
    Node * newNode = new Node (val);
    if(head==NULL){
        head=  newNode;
        tail = newNode;
        return;
    }

    newNode->prev= tail;
    tail->next= newNode;
    tail=newNode;
}
```

14 . How to insert a new node in a doubly linked list at any position, shown by a code ?
Ans:

```
void Insert_at_any_position(Node *&head, Node *&tail, int pos, int val)
{
    Node *newNode = new Node(val);
    if (head == NULL)
    {
        head = newNode;
```

```
        tail = newNode;
        return;
    }
    Node *tmp = head;
    for (int i = 1; i <= pos - 1; i++)
    {
        tmp = tmp->next;
    }
    newNode->next = tmp->next;
    tmp->next->prev= newNode;
    tmp->next=newNode;
    newNode->prev= tmp;
}
```

15 . How to delete a node at head in a doubly linked list?
Ans:

```
void Delete_at_head(Node *&head, Node *&tail)
{
    Node *newNode = new Node(val);
    if (head == NULL)
    {
        cout<<"The list is empty";
        return;
    }
    Node*deleteNode  = head;
    if(head==tail){
        head=NULL;
        tail=NULL;
    }
    else {

    head= head->next;
    head->prev=NULL;
    }
    delete deleteNode;
}
```

15 . How to delete a node at tail in a doubly linked list?

```
void Delete_at_tail(Node *&head, Node *&tail)
{
    Node *newNode = new Node(val);
    if (head == NULL)
    {
        cout<<"The list is empty";
        return;
    }
    Node*deleteNode  = head;
    if(head==tail){
        head=NULL;
        tail=NULL;
    }
    else {

    tail= tail->prev;
    tail->next=NULL;
    }
    delete deleteNode;
}
```

16 . How to delete a node at any position in a doubly linked list?
Ans:

```

void Delete_at_Any_position(Node *&head, Node *&tail,int pos, int val)
{
    Node *newNode = new Node(val);
    if (head == NULL)
    {
        cout<<"The list is empty";
        return;
    }
    Node*deleteNode  = head;
    if(head==tail){
        head=NULL;
        tail=NULL;
```

```
    }
    else {
        Node *tmp = head;

        for (int i=1; i<=pos-1; i++){
            tmp= tmp->next;
        }
        deleteNode = tmp-next ;
        tmp->next = tmp->next->next; // delateNode->next
        tmp->next->prev = tmp;
        delete deleteNode;

    }
    delete deleteNode;
}
```

17. What is the time complexity for inserting a new node at the head of a singly linked list?

    A) O(n)

    B) O(1)

    C) O(log n)

    D) O(n log n)

Answer: B) O(1)

18. What happens when you try to access the next node of the last node in a singly linked list?

    A) Returns the head node

    B) Returns null (or None in Python)

    C) Causes an error

    D) Creates a new node automatically

Answer: B) Returns null (or None)

19. What is the primary disadvantage of a singly linked list over a doubly linked list?

A) Insertion at head is slow

B) Uses more memory

C) Cannot traverse backward

D) Difficult to implement

Answer: C) Cannot traverse backward

20. What is the space complexity of a linked list?

A) O(1)

B) O(n)

C) O(log n)

D) O(n^2)

Answer: B) O(n)

Stack and Queue-related interview MCQ or Question (with Answer)

1. What is stack? What approche follows to complete this operation ?
Ans : **A stack is a data structure that follows the Last In, First Out (LIFO) principle. This means that the last element added to the stack is the first one to be removed. Think of it like a stack of plates: you add new plates to the top and remove plates from the top.**

2. What is the queue? What approche follows to complete this operation ?
Ans: **A queue is a data structure that follows the First In, First Out (FIFO) principle. This means that the first element added to the queue will be the first one to be removed. Imagine a queue in a supermarket: the person who arrives first is the first to be served.**

3. What is the key operation of the stack ?
Ans :

A. **Push**: Add an element to the top of the stack.
B. **Pop**: Remove the element from the top of the stack.
C. **Top**: Retrieve the element at the top of the stack without removing it.
D. **IsEmpty**: Check if the stack is empty.

4. What is the key operation of the queue?
Ans:

Enqueue: Add an element to the end of the queue.
Dequeue: Remove the element from the front of the queue.
Front/Peek: Retrieve the element at the front of the queue without removing it.
IsEmpty: Check if the queue is empty.

5. What is the time complexity of pushing an element onto the stack?

i) O(1)

ii) O(n)

iii) O(log n)

iv) O(n log n)

Answer: i) O(1)

6. In a stack, if we try to pop an element from an empty stack, it causes:

1) Overflow

2) Underflow

3) Garbage Collection

4) None of the above

Answer: 2) Underflow

7.Which data structure is used for recursion?

i) Queue

ii) Stack

iii) Tree

iv) Graph

Answer: ii) Stack

8.What is the time complexity of enqueuing an element in a standard queue?

A) O(n)

B) O(log n)

C) O(1)

D) O(n log n)

Answer: C) O(1)

9.Which of the following is an application of a queue?

1) BFS (Breadth-First Search)

2) DFS (Depth-First Search)

3) Merge Sort

4) Quick Sort

Answer: 1) BFS (Breadth-First Search)

10.If you dequeue from an empty queue, it results in:

i) Overflow

ii) Underflow

iii) Segmentation Fault

c) Null Pointer Exception

Answer: ii) Underflow

11. How do you implement a stack that supports finding the minimum element in constant time?

```
class MinStack

{

private:

    stack<int> s;

    stack<int> minStack;



public:

    void push(int x)

    {

        s.push(x);

        if (minStack.empty() || x <= minStack.top())

            minStack.push(x);

    }

    void pop()

    {

        if (s.top() == minStack.top())

            minStack.pop();

        s.pop();

    }

    int getMin()
```

```
    {
        if (minStack.empty())
            return -1;
        return minStack.top();
    }
};
```

12. Write a function to reverse the first k elements of a queue.

Ans:

```
void reverseKElements(queue<int> &q, int k)
{
    if (q.empty() || k > q.size())
        return;
    stack<int> s;
    for (int i = 0; i < k; i++)
    {
        s.push(q.front());
        q.pop();
    }
    while (!s.empty())
```

```
    {
        q.push(s.top());
        s.pop();
    }
    for (int i = 0; i < q.size() - k; i++)
    {
        q.push(q.front());
        q.pop();
    }
}
```

13. How would you implement a stack using arrays?

Ans : A stack using an array where the top element is at the last position. The `push()` operation adds elements at the end, and `pop()` removes elements from the end.

14. What is the difference between a stack and a queue?

Ans : Stack follows the **LIFO** (Last In, First Out) principle, whereas a queue follows the **FIFO** (First In, First Out) principle.

15. What is a priority queue?

ANS: A priority queue is a special type of queue where each element has a priority assigned to it. The element with the highest priority is dequeued before elements with lower priority, regardless of their insertion order.

When is Binary Search appropriate to use?

Ans : An already-sorted list can be searched using the Binary Search algorithm in the data structure. The algorithm uses the divide-and-conquer strategy.2.

Which data structure is used to perform recursion?

Ans: Stack data structure is used in recursion due to its last in first out nature. Operating system maintains the stack in order to save the iteration variables at each function call

Can binary search be implemented recursively?
Yes, binary search can be implemented both iteratively and recursively. The recursive implementation often leads to more concise code but may have slightly higher overhead due to recursive stack space or function calls.

What is the time complexity of Binary Search?
The time complexity of binary search is O(log2n), where n is the number of elements in the array. This is because the size of the search interval is halved in each step.