GATE

CS & DA

Data Structure through Python

DPP: 1

Linked List

Q1 What does the following function do for a given Linked List with first node as head ?

```
def fun1(head):
    if head is None:
        return
    fun1(head.next)
    print(head.data, end=" ")
```

(A) Prints all nodes of linked lists
(B) Prints all nodes of linked list in reverse order
(C) Prints alternate nodes of Linked List
(D) Prints alternate nodes in reverse order

Q2 The following function takes a single-linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution?

```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
def rearrange(head):
    if head is None or head.next is None:
        return
    p = head
    q = head.next
     while q is not None:
        # Swap values of p and q
        temp = p.value
        p.value = q.value
        q.value = temp
       # Move p and q to next pair of nodes
        p = q.next
        if p is not None:
            q = p.next
        else:
            q = None
# Helper function to print the linked list
def print_list(head):
    current = head
    while current is not None:
        print(current.value, end=" ")
        current = current.next
    print()
# Example usage:
if __name__ == "__main__":
    # Creating a linked list: 1 -> 2 -> 3 -> 4 -> 5
    head = Node(1)
    head.next = Node(2)
    head.next.next = Node(3)
    head.next.next.next = Node(4)
    head.next.next.next.next = Node(5)
    print("Original list:")
    print_list(head)
    rearrange(head)
    print("Rearranged list:")
    print_list(head)
```

(A) 1,2,3,4,5,6,7 (B) 2,1,4,3,6,5,7
(C) 1,3,2,5,4,7,6 (D) 2,3,4,5,6,7,1

Q3 In a doubly linked list, the number of pointers affected for an insertion operation will be

(A) 0 (B) 5
(C) 1 (D) None of these

Q4 Suppose each set is represented as a linked list with elements in arbitrary order. Which of the operations among union, intersection, membership, cardinality will be the slowest ?

(A) union
(B) membership
(C) cardinality
(D) union, intersection

Q5 Consider the function f defined below.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def f(p):
    if p is None or p.next is None:
        return True
    return p.data <= p.next.data and f(p.next)
```

For a given linked list p, the function f returns 1 if and only if

(A) not all elements in the list have the same data value

(B) the elements in the list are sorted in non-decreasing order of data value

(C) the elements in the list are sorted in non-increasing order of data value

(D) None of them

Q6 Which of the following points is/are true about Linked List data structure when it is compared with array ?

(A) Arrays have better cache locality that can make them better in terms of performance.

(B) It is easy to insert and delete elements in Linked List

(C) Random access is not allowed in a typical implementation of Linked Lists

(D) The size of array has to be pre-decided, linked lists can change their size any time.

(E) All of the above

Q7 In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is

(A) log n

(B) n/2

(C) log n – 1

(D) n

Q8 In a circular linked list organization, insertion of a record involves modification of

(A) One pointer.

(B) Two pointers.

(C) Multiple pointers

(D) No pointer.

Q9 What does the following function do for a given Linked List with first node as head ?

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def fun1(head):
    if head.next is None:
        return
    print(head.next.data, end=' ')
    fun1(head.next)
```

(A) Prints all nodes of linked lists

(B) Prints all nodes of linked list except first node

(C) Prints alternate nodes of Linked List

(D) Prints all nodes of linked list except last node

Q10 What does the following function print for a given Linked List with input 1,2,3,4,5,6 ?

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
def fun1(head):
    if head.next is None:
        return
    print(head.data, end=' ')
    fun1(head.next)
    print(head.data, end=' ')
```

(A) 1,2,3,4,5,6,6,5,4,3,2,1

(B) 1,2,3,4,4,3,2,1

(C) 1,2,3,4,5,5,4,3,2,1

(D) 1,2,3,4,5,1,2,3,4,5

GATE

Answer Key

Q1	(B)	Q6	(E)
Q2	(B)	Q7	(D)
Q3	(D)	Q8	(B)
Q4	(D)	Q9	(B)
Q5	(B)	Q10	(C)

Hints & Solutions

Q1 Text Solution:

This function fun1 recursively prints the elements of a linked list in reverse order.

It first checks if the head is NULL, and if it is, it returns without doing anything. Otherwise, it calls the fun1 function recursively with the next node in the linked list (head–>next). This continues until the last node is reached, which is the node whose next pointer is NULL.

At this point, the function starts returning, and as it returns it prints the value of each node in the linked list starting from the last node, working its way backwards to the first node. This is achieved through the printf statement which prints the value of head–>data.

Q2 Text Solution:

The function rearrange() exchanges data of every node with its next node. It starts exchanging data from the first node itse lf.

For eg. 3,5,7,9,11

answer:- 5,3,9,7,11

Q3 Text Solution:

Explanation - Actually, It depends on the position at which the insertion is done. However, there can be at most 3 cases :-

(1) Insertion at Begin - 3 pointers affected

(2) Insertion at Middle - 4 pointers affected

(3) Insertion at End 3 pointers affected

Q4 Text Solution:

Cardinality and membership are definitely not the slowest one. For cardinality, just count the number of nodes in a list. Fo r membership, just traverse the list and look for a match

For getting intersection of L1 and L2, search for each element of L1 in L2 and print the elements we find in L2.

There can be many ways for getting union of L1 and L2. One of them is as follows

(a) Print all the nodes of L1 and print only those which are not present in L2.

(b) Print nodes of L2.

Q5 Text Solution:

The function checks if the current element is less than or equal to the next element, and recursively applies the same check to the next element. If the end of the list is reached (i.e., p->next is NULL), or the next element is less than the current element, the function returns 1. Otherwise, it returns 0. Therefore, the function returns 1 only if the linked list is sorted in non-decreasing order.

Q6 Text Solution:

The following points are true when comparing Linked List data structure with an array:

Insertion and deletion: Linked lists allow for efficient insertion and deletion operations at any point in the list, as they inv olve simply adjusting pointers, while in an array, these operations can be expensive as all the elements after the insertion or deletion point need to be shifted.

Memory allocation: Linked lists use dynamic memory allocation, so they can grow or shrink as needed, while arrays have a fixed size and need to be allocated a contiguous block of memory upfront.

Access time: Arrays provide constant-time access to any element in the array (assuming the index is known), while accessi ng an element in a linked list takes

linear time proportional to the number of elements in the list, as the list needs to be tr aversed from the beginning to find the desired element.

Random access: Arrays support random access, which means that we can directly access any element in the array with its index, while linked lists do not support random access and we need to traverse the list to find a specific element.