# Data Structures Test Bank 2015

## Data Structures Linked List

**Question 1 - What does the following function do for a given Linked List with first node as head?**

```
void fun1(struct node* head)
{
 if(head == NULL)
   return;
    fun1(head->next);
 printf("%d  ", head->data);
}
```

Run on IDE

| | |
|---|---|
| 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 |

**Question 1 Explanation:**

fun1() prints the given Linked List in reverse manner. For Linked List 1->2->3->4->5, fun1() prints 5->4->3->2->1.

---

**Question 2- 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 |

**Question 3 - Consider the following function that takes reference to head of a Doubly Linked List as parameter. Assume that a node of doubly linked list has previous pointer as prev and next pointer as next.**

```
void fun(struct node **head_ref)
{
```

```
   struct node *temp = NULL;
   struct node *current = *head_ref;
    while (current !=  NULL)
   {
      temp = current->prev;
      current->prev = current->next;
      current->next = temp;
      current = current->prev;
   }

   if(temp != NULL )
      *head_ref = temp->prev;
}
```

Run on IDEAssume that reference of head of following doubly linked list is passed to above function 1 <--> 2 <--> 3 <--> 4 <--> 5 <-->6. What should be the modified linked list after the function call?

| | |
|---|---|
| A | 2 <--> 1 <--> 4 <--> 3 <--> 6 <-->5 |
| B | 5 <--> 4 <--> 3 <--> 2 <--> 1 <-->6. |
| C | 6 <--> 5 <--> 4 <--> 3 <--> 2 <--> 1. |
| D | 6 <--> 5 <--> 4 <--> 3 <--> 1 <--> 2 |

**Question 3 Explanation:**

The given function reverses the given doubly linked list.

---

**Question 4 - Which of the following sorting algorithms can be used to sort a random linked list with minimum time complexity?**

| | |
|---|---|
| A | Insertion Sort |
| B | Quick Sort |

| | |
|---|---|
| C | Heap Sort |
| D | Merge Sort |

**Question 4 Explanation:**

Both Merge sort and Insertion sort can be used for linked lists. The slow random-access performance of a linked list makes other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible. Since worst case time complexity of Merge Sort is O(nLogn) and Insertion sort is O(n^2), merge sort is preferred.

---

**Question 5- The following function reverse() is supposed to reverse a singly linked list. There is one line missing at the end of the function.**

```
/* Link list node */
struct node
{
   int data;
   struct node* next;
};
/* head_ref is a double pointer which points to head (or start) pointer
 of linked list */
static void reverse(struct node** head_ref)
{
   struct node* prev   = NULL;
   struct node* current = *head_ref;
   struct node* next;
   while (current != NULL)
   {
      next  = current->next;
      current->next = prev;
      prev = current;
      current = next;
   }
   /*ADD A STATEMENT HERE*/
}
```

# Data Structures Test Bank 2015

**What should be added in place of "/*ADD A STATEMENT HERE*/", so that the function correctly reverses a linked list.**

| | |
|---|---|
| A | *head_ref = prev; |
| B | *head_ref = current; |
| C | *head_ref = next; |
| D | *head_ref = NULL; |

**Question 5 Explanation:**

*head_ref = prev; At the end of while loop, the prev pointer points to the last node of original linked list. We need to change *head_ref so that the head pointer now starts pointing to the last node

---

**Question 6 - What is the output of following function for start pointing to first node of following linked list? 1->2->3->4->5->6**

```
void fun(struct node* start)
{
 if(start == NULL)
   return;
 printf("%d  ", start->data);
   if(start->next != NULL )
   fun(start->next->next);
 printf("%d  ", start->data);
}
```

| | |
|---|---|
| A | 1 4 6 6 4 1 |
| B | 1 3 5 1 3 5 |
| C | 1 2 3 5 |
| D | 1 3 5 5 3 1 |

**Question 6 Explanation:**

fun() prints alternate nodes of the given Linked List, first from head to end, and then from end to head. If Linked List has even number of nodes, then skips the last node

---

**Question 7-The following C function takes a simply-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank. Choose the correct alternative to replace the blank line.**

```
typedef struct node
{
 int value;
 struct node *next;
}Node;

Node *move_to_front(Node *head)
{
 Node *p, *q;
 if ((head == NULL: || (head->next == NULL))
  return head;
 q = NULL; p = head;
 while (p-> next !=NULL)
 {
  q = p;
  p = p->next;
 }
 _____
 return head;
}
```



| | |
|---|---|
| A | q = NULL; p->next = head; head = p; |
| B | q->next = NULL; head = p; p->next = head; |

C head = p; p->next = q; q->next = NULL;

D q->next = NULL; p->next = head; head = p;

---

**Question 8**

**The following C 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?**

```
struct node
{
 int value;
 struct node *next;
};
void rearrange(struct node *list)
{
 struct node *p, * q;
 int temp;
 if ((!list) || !list->next)
    return;
 p = list;
 q = list->next;
 while(q)
 {
   temp = p->value;
   p->value = q->value;
   q->value = temp;
   p = q->next;
   q = p?p->next:0;
 }
}
```

| | |
|---|---|
| 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 |

**Question 8 Explanation:**

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

---

**Question 9**

**In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is (GATE CS 2002)**

| | |
|---|---|
| A | log 2 n |
| B | n/2 |
| C | log 2 n – 1 |
| D | n |

**Question 9 Explanation:**

In the worst case, the element to be searched has to be compared with all elements of linked list.

---

**Question 10**

**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? (GATE CS 2004)**

| | |
|---|---|
| A | union only |
| B | intersection, membership |

C membership, cardinality

D union, intersection

**Question 10 Explanation:**

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. All of these methods will require more operations than intersection as we have to process intersection node plus other nodes.

---

**Question 11**

**Consider the function f defined below.**

```
struct item
{
 int data;
 struct item * next;
};

int f(struct item *p)
{
 return (
       (p == NULL) ||
       (p->next == NULL) ||
       (( P->data <= p->next->data) && f(p->next))
       );
}
```

**For a given linked list p, the function f returns 1 if and only if (GATE CS 2003)**

A the list is empty or has exactly one element

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 not all elements in the list have the same data value.

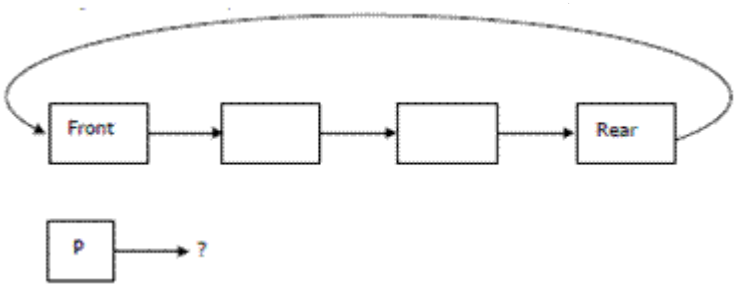
**Question 11 Explanation:**

The function f() works as follows 1) If linked list is empty return 1 2) Else If linked list has only one element return 1 3) Else if node->data is smaller than equal to node->next->data and same thing holds for rest of the list then return 1 4) Else return 0

---

**Question 12**

**A circularly linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue and deQueue can be performed in constant time? (GATE 2004)**




A rear node

B front node

C not possible with a single pointer

D node next to front

**Question 12 Explanation:**

Answer is not "(b) front node", as we can not get rear from front in O(1), but if p is rear we can implement both enQueue and deQueue in O(1) because from rear we can get front in O(1). Below are sample functions. Note that these functions are just sample are not working. Code to handle base cases is missing. 1

# Data Structures Test Bank 2015

**Question 13**

**What are the time complexities of finding 8th element from beginning and 8th element from end in a singly linked list? Let n be the number of nodes in linked list, you may assume that n > 8.**

A O(1) and O(n)

B O(1) and O(1)

C O(n) and O(1)

D O(n) and O(n)

**Question 13 Explanation:**

Finding 8th element from beginning requires 8 nodes to be traversed which takes constant time. Finding 8th from end requires the complete list to be traversed.

---

**Question 14**

**Is it possible to create a doubly linked list using only one pointer with every node.**

A Not Possible

B Yes, possible by storing XOR of addresses of previous and next nodes.

C Yes, possible by storing XOR of current node and next node

D Yes, possible by storing XOR of current node and previous node

**Question 14 Explanation:**

XOR Linked List – A Memory Efficient Doubly Linked List | Set 1

# Data Structures Test Bank 2015

**Question 15**

**Given pointer to a node X in a singly linked list. Only one pointer is given, pointer to head node is not given, can we delete the node X from given linked list?**

A Possible if X is not last node. Use following two steps (a) Copy the data of next of X to X. (b) Delete next of X.

B Possible if size of linked list is even.

C Possible if size of linked list is odd

D Possible if X is not first node. Use following two steps (a) Copy the data of next of X to X. (b) Delete next of X.

**Question 15 Explanation:**

Following are simple steps.

```
struct node *temp  = X->next;
X->data  = temp->data;
X->next  = temp->next;
free(temp);
```

---

**Question 16**

You are given pointers to first and last nodes of a singly linked list, which of the following operations are dependent on the length of the linked list?

A Delete the first element

B Insert a new element as a first element

C Delete the last element of the list

D Add a new element at the end of the list

**Question 16 Explanation:**

a) Can be done in O(1) time by deleting memory and changing the first pointer. b) Can be done in O(1) time, see push() here c) Delete the last element requires pointer to previous of last, which can only be obtained by traversing the list. d) Can be done in O(1) by changing next of last and then last.

---

**Question 17**

**Consider the following function to traverse a linked list.**

```
void traverse(struct Node *head)
{
  while (head->next != NULL)
  {
    printf("%d  ", head->data);
    head = head->next;
  }
}
```

**Which of the following is FALSE about above function?**

| | |
|---|---|
| A | The function may crash when the linked list is empty |
| B | The function doesn't print the last node when the linked list is not empty |
| C | The function is implemented incorrectly because it changes head |