Data Structure
Chapter 3: Linked List

GATE CS PYQ

Solved By

Monalisa Pradhan

GATE 2010,Q36,2M: 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.

```
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;   }
```

Choose the correct alternative to replace the blank line.

(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;

1 | → 2 | → 3 | → 4 \

4 | → 1 | → 2 | → 3 \

- p is travelling till the end of list and assigning q to whatever p had visited & p takes next new node.
- After completion of loop. Do these.
- (i) Make q as last(q → next = NULL;)
- (ii) Set next of p as head (p → next = head;)
- (iii) Make p as head(head = p)
- Ans :(D) q→next = NULL; p→next = head; head = p;

Q2 N items are stored in a sorted doubly linked list. For a *delete* operation, a pointer is provided to the record to be deleted. For a *decrease-key* operation, a pointer is provided to the record on which the operation is to be performed.

GATE 2016 set-2,Q15,1Mark

An algorithm performs the following operations on the list in this order: $\Theta(N)$ *delete*, $O(\log N)$ *insert*, $O(\log N)$ *find*, and $\Theta(N)$ *decrease-key*. What is the time complexity of all these operations put together?

(A) $O(\log^2 N)$ (B) $O(N)$ (C) $O(N^2)$ (D) $\Theta(N^2 \log N)$

- In Doubly linked list (sorted) Delete O(1) ,Insert O(N) ,Find O(N) ,
- Decrease O(N) [O(1) for decrease O(N) for sorting]
- Now number of each operation performed is given, so finally total complexity,
- Delete = O(1) × O(N) = O(N)
- Insert = O(N) × O(log N) = O(N log N)
- Find = O(N) × O(log N) = O(N log N)
- Decrease key = O(N) × O(N) = $O(N^2)$
- So, overall time complexity is, $O(N^2)$.
- Ans: (C) $O(N^2)$

Q3:Consider the C code fragment given below. GATE 2017 set-1,Q8,1Mark

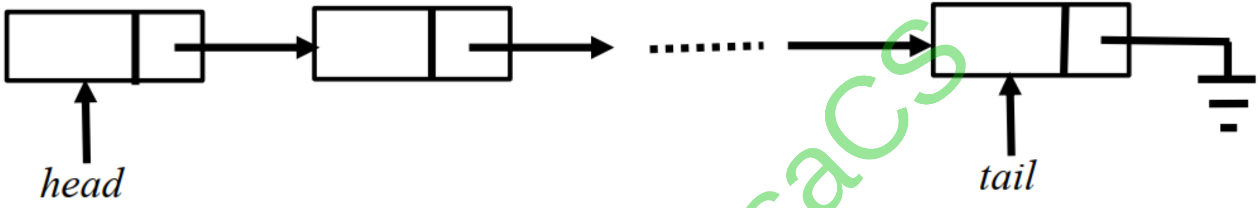
typedef struct node { int data; node next; } node;*
void join(node m, node* n) { node* p = n;*
while(p→next != NULL) { p = p→next; } p→next=m; }

Assuming that m and n point to valid NULL-terminated linked lists, invocation of join will
A)append list m to the end of list n for all inputs.
B)either cause a null pointer dereference or append list m to the end of list n.
C)cause a null pointer dereference for all inputs.
D)append list n to the end of list m for all inputs.

- m and n are valid Lists but not explicitly specified if the lists are empty or not.
- **Case 1: If lists are not NULL :** Invocation of join will append list m to the end of list n if the lists are not NULL.
- For Example: Before join operation :n =1→2→3→null , m =4→5→6→null
- After join operation : 1→2→3→4→5→6→null
- **Case 2: If lists are NULL :** If the list n is empty and itself NULL, then joining and referencing would obviously create NULL pointer issue.
- Ans : (B)either cause a null pointer dereference or append list m to the end of list n.

Q4: A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the number of nodes in the queue. Let `enqueue` be implemented by inserting a new node at the head, and `dequeue` be implemented by deletion of a node from the tail.

GATE 2018,Q11,1Mark



Which one of the following is the time complexity of the most time-efficient implementation of `enqueue` and `dequeue`, respectively, for this data structure?

(A) $\theta(1)$, $\theta(1)$ (B) $\theta(1)$, $\theta(n)$ (C) $\theta(n)$, $\theta(1)$ (D) $\theta(n)$, $\theta(n)$

- For insertion of node only head pointer is updated so $\theta(1)$ time.
- But if we have pointer to the tail of the list in order to delete it, we need the address of the 2^{nd} last node which can be obtained by traversing the list which takes θ (n) time.
- Ans: (B) θ (1), θ(n)

Q5: What is the worst case time complexity of inserting n elements into an empty linked list, if the linked list needs to be maintained in sorted order?

GATE 2020,Q16,1Mark

(A) $\Theta(n)$

(B) $\Theta(n \log n)$

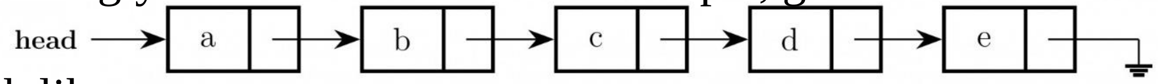
(C) $\Theta(n^2)$

(D) $\Theta(1)$

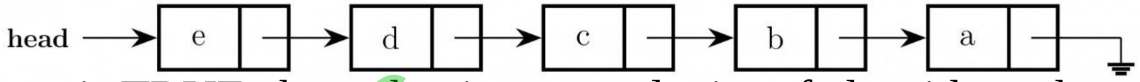
- This question is ambiguous: “needs to be maintained in sorted order”, there are two possible cases:
- 1.Needs to be maintained in *sorted order on each step (after each insertion).*
- When we are inserting *an element* in to empty linked list and to perform sorted order list of every element will take $\theta(n^2)$.
- 2.Needs to be maintained in *sorted order on final step (only after all insertion).*
- When we are inserting all elements into an empty linked list and to perform a sorted list (using merge sort) after inserting all elements will take O(n log n) time.
- Ans: (C)$\Theta(n^2)$

GATE CS 2022 | Question: 5

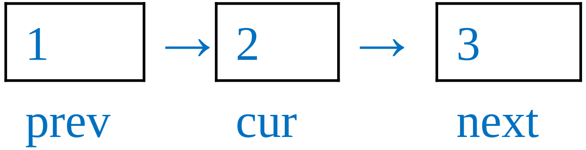
- Consider the problem of reversing a singly linked list. To take an example, given the linked list below,



- The reversed linked list should look like

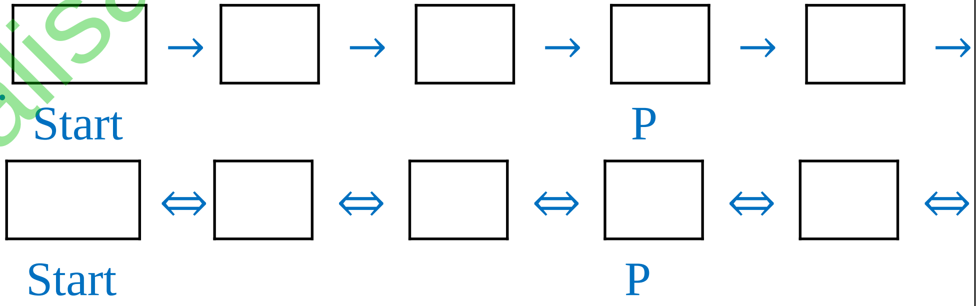


- Which one of the following statements is TRUE about the time complexity of algorithms that solve the above problem in O(1) space?
- A. The best algorithm for the problem takes θ(n) time in the worst case.
- B. The best algorithm for the problem takes θ(n log n) time in the worst case.
- C. The best algorithm for the problem takes $\theta(n^2)$ time in the worst case.
- D. It is not possible to reverse a singly linked list in O(1) space.



- While (Current != null)
- { next= current→next ;
- current →next=prev
- prev=current;
- current=next;}
- We need to traverse whole linked list so time complexity θ(n) in worst case.
- Ans : A. The best algorithm for the problem takes θ(n) time in the worst case.

- **GATE CS 2023 | Question: 3**
- Let **SLLdel** be a function that deletes a node in a singly-linked list given a pointer to the node and a pointer to the head of the list. Similarly, let **DLLdel** be another function that deletes a node in a doubly-linked list given a pointer to the node and a pointer to the head of the list. Let ***n*** denote the number of nodes in each of the linked lists. Which one of the following choices is TRUE about the worst-case time complexity of **SLLdel** and **DLLdel**?
- (A) SLLdel is $O(1)$ and DLLdel is $O(n)$
- (B) Both SLLdel and DLLdel are $O(\log n)$
- (C) Both SLLdel and DLLdel are $O(1)$
- (D) SLLdel is $O(n)$ and DLLdel is $O(1)$
- **SLLdel**
- A node temp is required to traverse the node.
- If (temp → next → data = = P → data)
- temp → next = P → next;
- free (P);
- **DLLdel**
- P → prev → next = P → next;
- P → next → prev = P → prev;
- free (P);



- Ans : (D) SLLdel is $O(n)$ and DLLdel is $O(1)$