4 Linked List

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

(a) $\log_2 n$ (b) $\frac{n}{2}$

(c) $\log_2 n - 1$ (d) n

[2002 : 1 M]

4.2 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

(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

[2003 : 2 M]

4.3 Let P be a singly linked list. Let Q be the pointer to an intermediate node x in the list. What is the worst-case time complexity of the best-known algorithm to delete the node x from the list?

(a) O(n) (b) $O(\log^2 n)$
(c) O(log n) (d) O(1)

[2004 : 1 M]

4.4 The following C function takes a singly-linked list of integers as a parameter and rearranges the elements of the list. The list is represented as pointer to a structure. 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

[2005 : 2 M]

4.5 The following C function takes a singly-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

[2008 : 2 M]

4.6 The following C function takes a singly-linked list as input argument. It modified 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.

```
typed of struct node
{
  int value;
  struct node *next;
} Node;
Node *mode_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;

[2010 : 2 M]

4.7 Consider the C code fragment given below:

```
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.

[2017 (Set-1) : 1 M]

4.8 Consider the following ANSI C program:

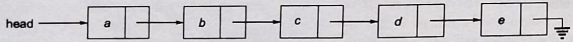
```
#include <stdio.h>
#include <stdlib.h>
struct Node{
        int value;
        struct Node *next;};
int main( ) {
    struct Node *boxE, *head, *boxN; int index = 0;
    boxE = head = (struct Node *)
    malloc(sizeof(struct Node));
    head → value = index;
    for (index = 1; index <= 3; index++) {
        boxN = (struct Node *) malloc(sizeof(struct Node));
        boxE → next = boxN;
        boxN → value = index;
        boxE = boxN; }
    for (index = 0; index <= 3; index++) {
        printf("Value at index %d is %d\n", index, head → value);
        head = head → next;
        printf("Value at index %d is %d\n", index+1, head → value); } }
```

Which one of the statements below is correct about the program?

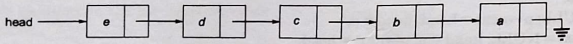
(a) Upon execution, the program creates a linked-list of five nodes.
(b) It has a missing return which will be reported as an error by the compiler.
(c) It dereferences an uninitialized pointer that may result in a run-time error.
(d) Upon execution, the program goes into an infinite loop.

[2021 (Set-2) : 2 M]

4.9 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 $\theta(n)$ time in the worst case.
(b) The best algorithm for the problem takes $\theta(n \log n)$ 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.

[2022 : 1 M]

4.10 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?

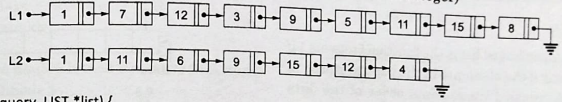
(a) SLLdel is O(1) and DLLdel is $O(n)$
(b) Both SLLdel and DLLdel are $O(\log(n))$
(c) Both SLLdel and DLLdel are O(1)
(d) SLLdel is $O(n)$ and DLLdel is O(1)

[2023 : 1 M]

4.11 Let **LIST** be a datatype for an implementation of linked list defined as follows:

```
typedef struct list {
    int data;
    struct list *next;
}  LIST;
```

Suppose a program has created two linked lists, L1 and L2, whose contents are given in the figure below (code for creating L1 and L2 is not provided here). L1 contains 9 nodes, and L2 contains 7 nodes.
Consider the following C program segment that modifies the list L1. The number of nodes that will be there in L1 after the execution of the code segment is _____. (Answer in integer)




```
int find (int query, LIST *list) {
    while (list != NULL) {
    if(list -> data == query) return 1;
    list = list -> next;
    }
        return 0;
}
int main( ) {
... ... ...
ptr1=L1; ptr2=L2;
while (ptr1 -> next != NULL) {
    query = ptr1 -> next -> data;
    if (find (query, L2))
        ptr1 -> next = ptr1 -> next -> next;
    else ptr1 = ptr1 - >next;
}
    ... ... ...
    return 0;
}
```

[2025 (Set-1) : 2 M]

4.12 Consider an unordered list of N distinct integers. What is the minimum number of element comparisons required to find an integer in the list that is NOT the largest in the list?

(a) 1
(b) $N-1$
(c) N
(d) $2N-1$

[2025 (Set-2) : 1 M]

4.13 A meld operation on two instances of a data structure combines them into one single instance of the same data structure. Consider the following data structures:
P: Unsorted doubly linked list with pointers to the head node and tail node of the list.
Q: Min-heap implemented using an array.
R: Binary Search Tree.
Which ONE of the following options gives the worst-case time complexities for meld operation on instances of size n of these data structures?

(a) P: $\Theta(1)$, Q: $\Theta(n)$, R: $\Theta(n)$
(b) P: $\Theta(1)$, Q: $\Theta(n \log n)$, R: $\Theta(n)$
(c) P: $\Theta(n)$, Q: $\Theta(n \log n)$, R: $\Theta(n^2)$
(d) P: $\Theta(1)$, Q: $\Theta(n)$, R: $\Theta(n \log n)$

[2025 (Set-2) : 2 M]

■■■■

Answers Linked List

4.1	(d)	4.2	(b)	4.3	(a)	4.4	(b)	4.5	(b)	4.6	(d)	4.7	(b)	4.8	(c)	4.9	(a)
4.10	(d)	4.11	(5)	4.12	(a)	4.13	(a)										

Explanations Linked List

4.1 (d)

Linear search number of comparisons: O(n).

4.2 (b)

For a given linked list p, the function f returns 1 if and only if the elements in the list are stored in non-decreasing (increasing) order of the data value.

f returns 1 if p == NULL or p → Next == NULL or p → data <= p → next → data,
satisfies in linked list for every node p.

4.3 (a)

To delete node x, we should findout the previous node to the x. To find the previous nodes to the x will take O(n).
Therefore it takes O(n).

4.4 (b)

2 1 4 3 6 5 7: as p and q are swapping each other where q is p → next all the time.

4.5 (b)

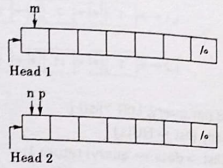
The given C code is nonrecursive which interchange the two consecutive elements of the list. So the input sequence 1, 2, 3, 4, 5, 6, 7 produces the output 2, 1, 4, 3, 6 , 5, 7

4.6 (d)

This is the program by moving last element to front of list and returns modified list.
When while (P → next ! = NULL) execute then p pass to q and now p point to next. When "while" condition is false then we get q → next = NULL; p → next = head; head = p; means next to q made null, and points p → next is head and then make p as the head.

4.7 (b)

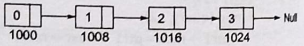
Consider two linked lists:



After the code execution, list 'm' will be appended to the end of list 'n'. But in some cases it can cause a null pointer dereference also.

4.8 (c)

As we can see in the loop, i runs from 1 to 3. So, four nodes will be created because one node is already created with value 0.



When index = 3, then head value = 3
Head = Head → Next **(Now head will point to NULL)**
Head → Value [which is in print. So it will generate Run time Error].
So, option (c) is correct.

4.10 (d)

In order to delete a node in a singly linked list we need to traverse the nodes sequentially, this will take O(n).

```
struct node {
    int val;
    struct node * next;
};
```



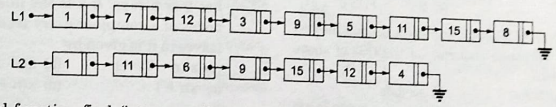
```
SLLdel (struct node * delnode, struct node *
  head) {
  struct node * P;
  struct node * t = head;
  while (t ! = delnode)
    {
      P = t;
      t = t → next;
    }
  P → next = delnode → next;
  free (delnode);
  }
```

Deletion in double linked list takes O(1) time.

```
struct DLLnode {
    int val;
    struct DLLnode * next;
    struct DLLnode * prev;
};
DLLdel (struct DLLnode * delnode, struct
  DLLnode * head) {
  struct DLLnode * t = delnode → prev;
  t → next = delnode → next;
  delnode → next → prev = t;
  free (delnode);
}
```

Note: Corners cases in the above codes are not handled.

4.11 (5)



In the find function find (int query, list * list) check if query exists in L2:

- If query is present in L2, it returns 1 (indicates that the node should be deleted) otherwise. It returns 0 (indicating that the node should be kept).

Let's go node by node and apply the logic:

Current node (ptr1)	Next node (ptr1 -> next)	Exists in L2?	Action Taken	Updated L1
1	7	No	Keep	1 → 7 → 12 → 3 → 9 → 5 → 11 → 15 → 8
7	12	Yes	Delete	1 → 7 → 3 → 9 → 5 → 11 → 15 → 8
7	3	No	Keep	1 → 7 → 3 → 9 → 5 → 11 → 15 → 8
3	9	Yes	Delete	1 → 7 → 3 → 5 → 11 → 15 → 8
3	5	No	Keep	1 → 7 → 3 → 5 → 11 → 15 → 8
5	11	Yes	Delete	1 → 7 → 3 → 5 → 15 → 8
5	15	Yes	Delete	1 → 7 → 3 → 5 → 8
5	8	No	Keep	1 → 7 → 3 → 5 → 8

4.12 (a)

Compare any two elements

```
if (a[1] < a[z])
    return(a[1]);
else
    return(a[z]);
```

4.13 (a)

Merging of two unsorted DLL : θ(1)
Merging of two min heaps : θ(n)
Using bottom up
Merging of two Binary Search Trees : θ(n)
[Inorder traversal of both lists : θ(n) time
Merge two sorted lists : θ(n) time
Build BST over sorted list : θ(n) time]

■■■■