1. Which operation is used to traverse a linked list and process each node one by one?

a) Insertion

b) Deletion

**c) Traversal**

d) Searching

2. #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

Node\* insertAtEnd(Node\* head, int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = nullptr;

if (head == nullptr) {

return newNode;

}

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

return head;

}

void display(Node\* head) {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

}

int main() {

Node\* head = nullptr;

head = insertAtEnd(head, 1);

head = insertAtEnd(head, 2);

head = insertAtEnd(head, 3);

head = insertAtEnd(head, 4);

display(head);

return 0;

}

What will be the output when the above code is executed?

**a) 1 2 3 4**

b) 4 3 2 1

c) 1 4 2 3

d) 1 2 4 3

3. Which operation is used to add an element at the beginning of a linked list?

**a) Insertion at the head**

b) Insertion at the tail

c) Deletion from the head

d) Deletion from the tail

4. 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) 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

5. The time complexity for inserting an element at the end of a linked list with n nodes is:

a) O(1)

b) O(log n)

c) O(n)

**d) O(1) (if tail is known, otherwise O(n))**

6. 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**

d) None of the above

7. The process of removing a node from a linked list is known as:

a) Deletion

b) Insertion

**c) Deletion**

d) Searching

8. Which of the following code snippets correctly appends a new node to the end of a singly linked list?

a)

void append(Node\* head, int data) {

Node\* newNode = new Node;

newNode->data = data;

head = newNode;

}

b)

void append(Node\* head, int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = head;

}

**c)**

**void append(Node\* head, int data) {**

**Node\* newNode = new Node;**

**newNode->data = data;**

**newNode->next = nullptr;**

**Node\* temp = head;**

**while (temp->next) {**

**temp = temp->next;**

**}**

**temp->next = newNode;**

**}**

d)

void append(Node\* head, int data) {

Node\* newNode = new Node;

newNode->data = data;

head->next = newNode;

}

9. Which type of linked list has a loop/cycle in the list?

a) Singly linked list

b) Doubly linked list

**c) Circular linked list**

d) Cyclic linked list

10. #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

bool detectLoop(Node\* head) {

Node\* slow = head;

Node\* fast = head;

while (slow != NULL && fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

return true;

}

}

return false;

}

int main() {

Node\* list = NULL;

for (int i = 1; i <= 5; i++) {

Node\* newNode = new Node;

newNode->data = i;

newNode->next = list;

list = newNode;

}

// Create a loop by connecting the last node to the second node

list->next->next->next->next->next = list->next;

if (detectLoop(list)) {

cout << "Loop detected";

} else {

cout << "No loop detected";

}

return 0;

}

What will be the output when the above code is executed?

**a) Loop detected**

b) No loop detected

c) 1 2 3 4 5

d) Compilation Error

11. In a doubly linked list, what is the time complexity to delete a node with a given value?

a) O(1)

b) O(log n)

**c) O(n)**

d) O(n log n)

12.

Node\* find(Node\* head) {

if (head == NULL) return NULL;

Node\* slow = head;

Node\* fast = head->next;

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

What the above code snippet do?

a) Find first element of list

**b) Find middle element of list**

c) Find last element of list

d) Find second element of list

13. What is the space complexity of a linked list with n nodes?

a) O(n)

b) O(1)

**c) O(n)**

d) O(log n)

14. #include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

Node\* mergeLists(Node\* list1, Node\* list2) {

if (list1 == NULL) return list2;

if (list2 == NULL) return list1;

Node\* merged = NULL;

if (list1->data <= list2->data) {

merged = list1;

merged->next = mergeLists(list1->next, list2);

} else {

merged = list2;

merged->next = mergeLists(list1, list2->next);

}

return merged;

}

void printList(Node\* head) {

Node\* current = head;

while (current != NULL) {

cout << current->data << " ";

current = current->next;

}

}

int main() {

Node\* list1 = NULL;

Node\* list2 = NULL;

for (int i = 2; i <= 8; i += 2) {

Node\* newNode = new Node;

newNode->data = i;

newNode->next = list1;

list1 = newNode;

}

for (int i = 1; i <= 9; i += 2) {

Node\* newNode = new Node;

newNode->data = i;

newNode->next = list2;

list2 = newNode;

}

Node\* merged = mergeLists(list1, list2);

printList(merged);

return 0;

}

What will be the output when the above code is executed?

a) 1 2 3 4 5 6 7 8 9

b) 2 4 6 8 1 3 5 7 9

c) 1 2 8 7 6 5 4 3 9

**d) 8 6 4 2 9 7 5 3 1**

15. Which operation is used to insert a new node after a specific node in a linked list?

a) Insertion at the head

**b) Insertion after a given node**

c) Insertion at the tail

d) Insertion before a given node

16. Node\* operation(Node\* head) {

if (head == NULL || head->next == NULL) return head;

Node\* slow = head;

Node\* fast = head->next;

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

}

Node\* secondHalf = slow->next;

slow->next = NULL;

return secondHalf;

}

What the above function do?

**a) Split the linked list in half**

b) Remove the alternate elements

c) Reverse the linked list

d) Remove alternate elements

17. In a doubly linked list, what is the time complexity to insert a new node after a given node?

a) O(1)

b) O(log n)

**c) O(1)**

d) O(n)

18. Node\* operation(Node\* list1, Node\* list2) {

Node\* dummy = new Node;

Node\* tail = dummy;

while (list1 != NULL && list2 != NULL) {

if (list1->data < list2->data) {

tail->next = list1;

list1 = list1->next;

} else if (list1->data > list2->data) {

tail->next = list2;

list2 = list2->next;

} else {

tail->next = list1;

list1 = list1->next;

tail->next->next = list2;

list2 = list2->next;

tail = tail->next;

}

tail = tail->next;

}

if (list1 != NULL) {

tail->next = list1;

} else {

tail->next = list2;

}

return dummy->next;

}

What will be the above function do?

a) Split the linked list in half

**b) Merge two sorted list**

c) Reverse the linked list

d) Sort the list

19. The process of freeing the memory occupied by a node after it has been deleted from a linked list is known as:

a) Memory deallocation

b) Memory release

**c) Memory deallocation**

d) Memory garbage collection

20. Node\* pairwiseSwap(Node\* head) {

if (head == NULL || head->next == NULL) return head;

Node\* first = head;

Node\* second = head->next;

while (first != NULL && second != NULL) {

int temp = first->data;

first->data = second->data;

second->data = temp;

first = second->next;

if (first != NULL) {

second = first->next;

}

}

return head;

}

What will be the above function do?

a) Swap the linked list elements

b) Merge two sorted list

c) Reverse the linked list

**d) Pairwise swapping of elements in linked list**