Which data structure is used for implementing recursion?

a) Stack

b) Queue

c) List

d) Array

The data structure required to check whether an expression contains a balanced parenthesis is?

a) Queue

b) Stack

c) Tree

d) Array

Which of the following is not the application of stack?

a) Data Transfer between two asynchronous process

b) Compiler Syntax Analyzer

c) Tracking of local variables at run time

d) A parentheses balancing program

Which data structure is needed to convert infix notation to postfix notation?

a) Tree

b) Branch

c) Stack

d) Queue

What is the value of the postfix expression 6 3 2 4 + – \*?

a) 74

b) -18

c) 22

d) 40

What data structure would you mostly likely see in non recursive implementation of a recursive algorithm?  
a) Stack  
b) Linked List  
c) Tree  
d) Queue

Which of the following statement(s) about stack data structure is/are NOT correct?  
a) Top of the Stack always contain the new node  
b) Stack is the FIFO data structure  
c) Null link is present in the last node at the bottom of the stack  
d) Linked List are used for implementing Stacks

1 The prefix form of A-B/ (C \* D ^ E) is?  
a) -A/B\*C^DE  
b) -A/BC\*^DE  
c) -ABCD\*^DE  
d) -/\*^ACBDE

1 Which of the following points is/are not true about Linked List data structure when it is compared with an array?  
a) Random access is not allowed in a typical implementation of Linked Lists  
b) Access of elements in linked list takes less time than compared to arrays  
c) Arrays have better cache locality that can make them better in terms of performance  
d) It is easy to insert and delete elements in Linked List

1 Which data structure is based on the Last In First Out (LIFO) principle?  
a) Tree  
b) Linked List  
c) Stack  
d) Queue

1 Which of the following application makes use of a circular linked list?  
a) Recursive function calls  
b) Undo operation in a text editor  
c) Implement Hash Tables  
d) Allocating CPU to resources

1 Which of the following tree data structures is not a balanced binary tree?  
a) Splay tree  
b) B-tree  
c) AVL tree  
d) Red-black tree

1 Which of the following is not the type of queue?  
a) Priority queue  
b) Circular queue  
c) Single ended queue  
d) Ordinary queue

1 Which of the following data structures can be used for parentheses matching?  
a) n-ary tree  
b) queue  
c) priority queue  
d) stack

What is the need for a circular queue?

a) easier computations

b) implement LIFO principle in queues

c) effective usage of memory

d) to delete elements based on priority

2 What will be the output of the following program?

main()

{

str=[];

int len =str.size;

int i;

for(i=0;i<len;i++)

push(str[i]); *// pushes an element into stack*

for(i=0;i<len;i++)

pop(); *//pops an element from the stack*

}

2 What is an AVL tree?  
a) a tree which is unbalanced and is a height balanced tree  
b) a tree which is balanced and is a height balanced tree  
c) a tree with atmost 3 children  
d) a tree with three children

2 Which is the most appropriate data structure for reversing a word?  
a) stack  
b) queue  
c) graph  
d) tree

What is the functionality of the following piece of code?

display()

{

**if**(size == 0)

cout << ("underflow");

**else**

{

Node current = first;

**while**(current != **null**)

{

cout << (current.getEle());

current = current.getNext();

}

}

}

a) display the list  
b) reverse the list  
c) reverse the list excluding top-of-the-stack-element  
d) display the list excluding top-of-the-stack-element

What is a dequeue?  
a) A queue implemented with both singly and doubly linked lists  
b) A queue with insert/delete defined for front side of the queue  
c) A queue with insert/delete defined for both front and rear ends of the queue  
d) A queue implemented with a doubly linked list

A data structure in which elements can be inserted or deleted at/from both ends but not in the middle is?  
a) Priority queue  
b) Dequeue  
c) Circular queue  
d) Queue

Process of inserting an element in stack is called \_\_\_\_\_\_\_\_\_\_\_\_  
a) Create  
b) Push  
c) Evaluation  
d) Pop

Process of removing an element from stack is called \_\_\_\_\_\_\_\_\_\_  
a) Create  
b) Push  
c) Evaluation  
d) Pop

In a stack, if a user tries to remove an element from an empty stack it is called \_\_\_\_\_\_\_\_\_  
a) Underflow  
b) Empty collection  
c) Overflow  
d) Garbage Collection

Pushing an element into stack already having five elements and stack size of 5, then stack becomes \_\_\_\_\_\_\_\_\_\_\_  
a) Overflow  
b) Crash  
c) Underflow  
d) User flow

Entries in a stack are “ordered”. What is the meaning of this statement?  
a) A collection of stacks is sortable  
b) Stack entries may be compared with the ‘<‘ operation  
c) The entries are stored in a linked list  
d) There is a Sequential entry that is one by one

Which of the following is not the application of stack?  
a) A parentheses balancing program  
b) Tracking of local variables at run time  
c) Compiler Syntax Analyzer  
d) Data Transfer between two asynchronous process

Consider the usual algorithm for determining whether a sequence of parentheses is balanced. The maximum number of parentheses that appear on the stack AT ANY ONE TIME when the algorithm analyzes: (()(())(()))?  
a) 1  
b) 2  
c) 3  
d) 4 or more

Consider the usual algorithm for determining whether a sequence of parentheses is balanced. Suppose that you run the algorithm on a sequence that contains 2 left parentheses and 3 right parentheses (in some order). The maximum number of parentheses that appear on the stack AT ANY ONE TIME during the computation?  
a) 1  
b) 2  
c) 3  
d) 4 or more

What is the value of the postfix expression 6 3 2 4 + – \*?  
a) 1  
b) 40  
c) 74  
d) -18

Here is an infix expression: 4 + 3\*(6\*3-12). Suppose that we are using the usual stack algorithm to convert the expression from infix to postfix notation. The maximum number of symbols that will appear on the stack AT ONE TIME during the conversion of this expression?  
a) 1  
b) 2  
c) 3  
d) 4

What is the best case time complexity of deleting a node in a Singly Linked list?  
a) O (n)  
b) O (n2)  
c) O (nlogn)  
d) O (1)

Which of the following statements are not correct with respect to Singly Linked List(SLL) and Doubly Linked List(DLL)?  
a) Complexity of Insertion and Deletion at known position is O(n) in SLL and O(1) in DLL  
b) SLL uses lesser memory per node than DLL  
c) DLL has more searching power than SLL  
d) Number of node fields in SLL is more than DLL

What is the functionality of the following piece of code?

display()

{

**if**(size == 0)

cout << ("underflow");

**else**

{

Node current = first;

**while**(current != **null**)

{

cout << (current.getEle());

current = current.getNext();

}

}

}

a) reverse the list  
b) display the list  
c) display the list excluding top-of-the-stack-element  
d) reverse the list excluding top-of-the-stack-element

What does ‘stack overflow’ refer to?  
a) accessing item from an undefined stack  
b) adding items to a full stack  
c) removing items from an empty stack  
d) index out of bounds exception

Consider these functions:  
push() : push an element into the stack  
pop() : pop the top-of-the-stack element  
top() : returns the item stored in top-of-the-stack-node  
What will be the output after performing these sequence of operations

push(20);

push(4);

top();

pop();

pop();

push(5);

top();

a) 20  
b) 4  
c) stack underflow  
d) 5

Which of the following data structures can be used for parentheses matching?  
a) n-ary tree  
b) queue  
c) priority queue  
d) stack

Which of the following properties is associated with a queue?  
a) First In Last Out  
b) First In First Out and Last in Last Out  
c) Last In First Out  
d) Last In Last Out Only

In a circular queue, how do you increment the rear end of the queue?  
a) rear++  
b) (rear+1) % CAPACITY  
c) (rear % CAPACITY)+1  
d) rear–

What is the term for inserting into a full queue known as?  
a) overflow  
b) underflow  
c) null pointer exception  
d) program won’t be compiled

What is the time complexity of enqueue operation?  
a) O(logn)  
b) O(nlogn)  
c) O(n)  
d) O(1)

What is the need for a circular queue?  
a) effective usage of memory  
b) easier computations  
c) to delete elements based on priority  
d) implement LIFO principle in queues

Which of the following represents a dequeue operation? (count is the number of elements in the queue)  
a)

dequeue()

{

**if**(count == 0)

{

cout << ("Queue underflow");

**return** 0;

}

**else**

{

ele = q[front];

q[front] = **null**;

front = (front+1)%CAPACITY;

count--;

**return** ele;

}

}

b)

dequeue()

{

**if**(count == 0)

{

cout << ("Queue underflow");

**return** 0;

}

**else**

{

ele = q[front];

front = (front+1)%CAPACITY;

q[front] = **null**;

count--;

**return** ele;

}

}

c)

dequeue()

{

**if**(count == 0)

{

cout << ("Queue underflow");

**return** 0;

}

**else**

{

front = (front+1)%CAPACITY;

ele = q[front];

q[front] = **null**;

count--;

**return** ele;

}

}

d)

dequeue()

{

**if**(count == 0)

{

cout << ("Queue underflow");

**return** 0;

}

**else**

{

ele = q[front];

q[front] = **null**;

front = (front+1)%CAPACITY;

**return** ele;

count--;

}

}

Which of the following best describes the growth of a linear queue at runtime? (Q is the original queue, size() returns the number of elements in the queue)  
a)

**private**expand()

{

**int** length = size();

**int**[] newQ = **new** **int**[length<<1];

**for**(**int** i=front; i<=rear; i++)

{

newQ[i-front] = Q[i%CAPACITY];

}

Q = newQ;

front = 0;

rear = size()-1;

}

b)

**private**expand()

{

**int** length = size();

**int**[] newQ = **new** **int**[length<<1];

**for**(**int** i=front; i<=rear; i++)

{

newQ[i-front] = Q[i%CAPACITY];

}

Q = newQ;

}

c)

**Private** expand()

{

**int** length = size();

**int**[] newQ = **new** **int**[length<<1];

**for**(**int** i=front; i<=rear; i++)

{

newQ[i-front] = Q[i];

}

Q = newQ;

front = 0;

rear = size()-1;

}

d)

**private**expand()

{

**int** length = size();

**int**[] newQ = **new** **int**[length\*2];

**for**(**int** i=front; i<=rear; i++)

{

newQ[i-front] = Q[i%CAPACITY];

}

Q = newQ;

}

What is the space complexity of a linear queue having n elements?  
a) O(n)  
b) O(nlogn)  
c) O(logn)  
d) O(1)

In linked list implementation of queue, if only front pointer is maintained, which of the following operation take worst case linear time?  
a) Insertion  
b) Deletion  
c) To empty a queue  
d) Both Insertion and To empty a queue

In linked list implementation of a queue, where does a new element be inserted?  
a) At the head of link list  
b) At the centre position in the link list  
c) At the tail of the link list  
d) At any position in the linked list

In linked list implementation of a queue, front and rear pointers are tracked. Which of these pointers will change during an insertion into a NONEMPTY queue?  
a) Only front pointer  
b) Only rear pointer  
c) Both front and rear pointer  
d) No pointer will be changed

In linked list implementation of a queue, front and rear pointers are tracked. Which of these pointers will change during an insertion into EMPTY queue?  
a) Only front pointer  
b) Only rear pointer  
c) Both front and rear pointer  
d) No pointer will be changed

In linked list implementation of a queue, from where is the item deleted?  
a) At the head of link list  
b) At the centre position in the link list  
c) At the tail of the link list  
d) Node before the tail

In linked list implementation of a queue, the important condition for a queue to be empty is?  
a) FRONT is null  
b) REAR is null  
c) LINK is empty  
d) FRONT==REAR-1

The essential condition which is checked before insertion in a linked queue is?  
a) Underflow  
b) Overflow  
c) Front value  
d) Rear value

The essential condition which is checked before deletion in a linked queue is?  
a) Underflow  
b) Overflow  
c) Front value  
d) Rear value

Which of the following is true about linked list implementation of queue?  
a) In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from end  
b) In push operation, if new nodes are inserted at the beginning, then in pop operation, nodes must be removed from the beginning  
c) In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from end  
d) In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from beginning

With what data structure can a priority queue be implemented?  
a) Array  
b) List  
c) Heap  
d) Tree

Which of the following is not an application of priority queue?  
a) Huffman codes  
b) Interrupt handling in operating system  
c) Undo operation in text editors  
d) Bayesian spam filter

What is the time complexity to insert a node based on key in a priority queue?  
a) O(nlogn)  
b) O(logn)  
c) O(n)  
d) O(n2)

What is not a disadvantage of priority scheduling in operating systems?  
a) A low priority process might have to wait indefinitely for the CPU  
b) If the system crashes, the low priority systems may be lost permanently  
c) Interrupt handling  
d) Indefinite blocking

Which of the following is not an advantage of a priority queue?  
a) Easy to implement  
b) Processes with different priority can be efficiently handled  
c) Applications with differing requirements  
d) Easy to delete elements in any case

What is a dequeue?  
a) A queue with insert/delete defined for both front and rear ends of the queue  
b) A queue implemented with a doubly linked list  
c) A queue implemented with both singly and doubly linked lists  
d) A queue with insert/delete defined for front side of the queue

What are the applications of dequeue?  
a) A-Steal job scheduling algorithm  
b) Can be used as both stack and queue  
c) To find the maximum of all sub arrays of size k  
d) All of the mentioned

What is the time complexity of deleting from the rear end of the dequeue implemented with a singly linked list?

a) O(nlogn)

b) O(logn)

c) O(n)

d) O(n2)

After performing these set of operations on a queue, what does the final list look like?

InsertFront(10);

InsertFront(20);

InsertRear(30);

DeleteFront();

InsertRear(40);

InsertRear(10);

DeleteRear();

InsertRear(15);

display();

a) 10 30 10 15  
b) 20 30 40 15  
c) 20 30 40 10  
d) 10 30 40 15