- Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent Prefix and Postfix notations.

- Answer
- Prefix Notation: - * +ABC ^ - DE + FG
- Postfix Notation: AB + C * DE - FG + ^ -

- Whether Linked List is linear or Non-linear data structure?

- According to Access strategies Linked list is a linear one.
- According to Storage Linked List is a Non-linear one.

- 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);
}
```

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

Answer: B

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

Answer:E

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;
}
```

- Assume 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

Answer C

- 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

Answer: D

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.

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;

Answer: D

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

Answer: B

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

- 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

Answer: D

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

- 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

Answer: D

- 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.

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

Answer: (B)

Explanation:

The function f() works as follows

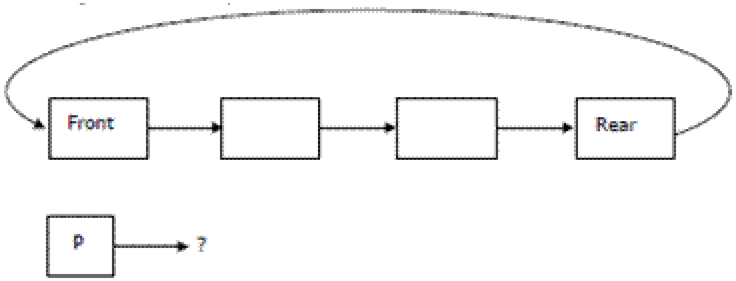
1) If linked list is empty return 1

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

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) circularLinkedList



A) rear node

B) front node

C) not possible with a single pointer

D) node next to front

Answer: A

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).

- 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.

Answer: A

Explanation:

Following are simple steps.

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

Answer: A

- Explanation:
- Following are simple steps.
- struct node *temp = X->next;
- X->data = temp->data;
- X->next = temp->next;
- free(temp);

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

Answer: C

- 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.

- 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.

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

Answer: C

- 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(log2 n)

(C) O(logn)

(D) O(1)

Answer: (D)

Explanation: A simple solution is to traverse the linked list until you find the node you want to delete. But this solution requires pointer to the head node which contradicts the problem statement.

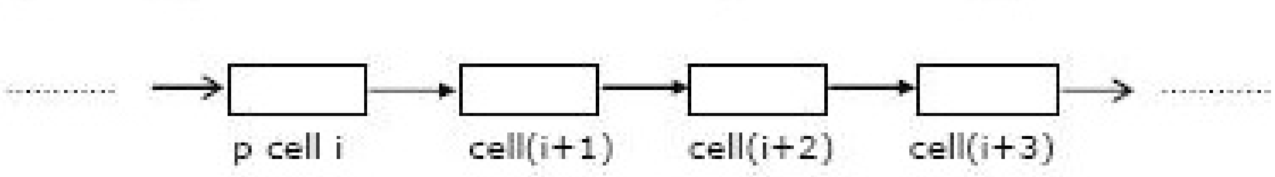
Fast solution is to copy the data from the next node to the node to be deleted and delete the next node. Something like following.

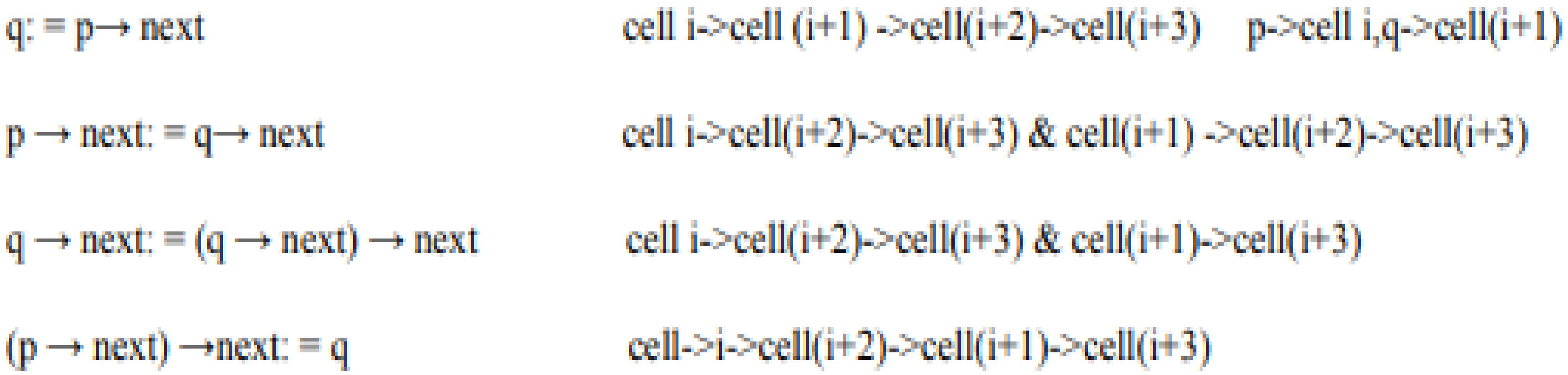
```
// Find next node using next pointer
struct node *temp  = node_ptr->next;

// Copy data of next node to this node
node_ptr->data  = temp->data;

// Unlink next node
node_ptr->next  = temp->next;

// Delete next node
free(temp);
```

Time complexity of this approach is O(1)

GATE-IT-2004

- Let p be a pointer as shown in the figure in a single linked list. What do the following assignment statements achieve ?
- q: = p → next
- p → next:= q → next
- q → next:=(q → next
- (p → next) → next:= q





q: = p→ next	cell i->cell (i+1) ->cell(i+2)->cell(i+3) p->cell i,q->cell(i+1)
p → next: = q→ next	cell i->cell(i+2)->cell(i+3) & cell(i+1) ->cell(i+2)->cell(i+3)
q → next: = (q → next) → next	cell i->cell(i+2)->cell(i+3) & cell(i+1)->cell(i+3)
(p → next) →next: = q	cell->i->cell(i+2)->cell(i+1)->cell(i+3)

Consider the following piece of 'C' code fragment that removes duplicates from an ordered list of integers.

```
Node  *remove-duplicates(Node *head, int *j)
{
    Node *t1, *t2;
    *j=0;
    t1 = head;
    if (t1! = NULL) t2 = t1 →next;
    else return head;
    *j = 1;
    if(t2 == NULL)
        return head;
    while t2 != NULL)
    {
        if (t1.val != t2.val) --------------------------→ (S1)
        {
            (*j)++; t1 -> next = t2; t1 = t2: ----------→ (S2)
        }
        t2 = t2 →next;
    }
    t1 →next = NULL;
    return head;
}
```

- Assume the list contains n elements (n≥2) in the following questions.
- a). How many times is the comparison in statement S1 made?
- b). What is the minimum and the maximum number of times statements marked S2 get executed?
- c). What is the significance of the value in the integer pointed to by j when the function completes?

Consider the following statements:

i. First-in-first out types of computations are efficiently supported by STACKS.

ii. Implementing LISTS on linked lists is more efficient than implementing LISTS on an array for almost all the basic LIST operations.

iii. Implementing QUEUES on a circular array is more efficient than implementing QUEUES on a linear array with two indices.

iv. Last-in-first-out type of computations are efficiently supported by QUEUES.

Which of the following is correct?

A) (ii) and (iii) are true

B) (i) and (ii) are true

C) (iii) and (iv) are true

D) (ii) and (iv) are true

Answer: (A)

Suppose there are two singly linked lists both of which intersect at some point and become a single linked list. The head or start pointers of both the lists are known, but the intersecting node and lengths of lists are not known.

- What is worst case time complexity of optimal algorithm to find intersecting node from two intersecting linked lists?

(A) Θ(n*m), where m, n are lengths of given lists

(B) Θ(n^2), where m>n and m, n are lengths of given lists

(C) Θ(m+n), where m, n are lengths of given lists

(D) Θ(min(n, m)), where m, n are lengths of given lists

Answer: (C)

Explanation: This takes Θ(m+n) time and O(1) space in worst case, where M and N are the total length of the linked lists.

Traverse the two linked list to find m and n.
Get back to the heads, then traverse |m − n| nodes on the longer list.
Now walk in lock step and compare the nodes until you found the common ones.

Answer: (C)

Explanation: This takes Θ(m+n) time and O(1) space in worst case, where M and N are the total length of the linked lists.

Traverse the two linked list to find m and n.

Get back to the heads, then traverse |m − n| nodes on the longer list.

Now walk in lock step and compare the nodes until you found the common ones.

Option (C) is correct.

- Consider an implementation of unsorted single linked list. Suppose it has its representation with a head and a tail pointer (i.e. pointers to the first and last nodes of the linked list). Given the representation, which of the following operation can not be implemented in O(1) time ?

A) Insertion at the front of the linked list.

B) Insertion at the end of the linked list.

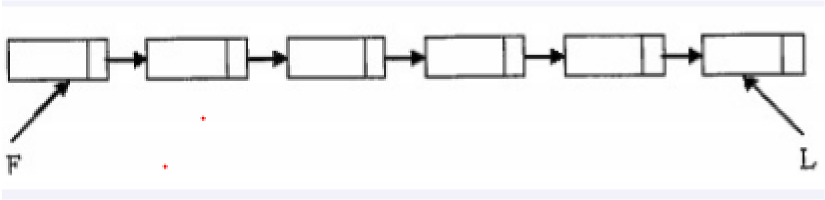
C) Deletion of the front node of the linked list.

D) Deletion of the last node of the linked list

Answer: (D)

Explanation:
Deletion of the last node of the linked list, we need address of second last node of single linked list to make NULL of its next pointer. Since we can not access its previous node in singly linked list, so need to traverse entire linked list to get second last node of linked list.
So, option (D) is correct.

Consider a single linked list where F and L are pointers to the first and last elements respectively of the linked list. The time for performing which of the given operations depends on the length of the linked list?



A) Delete the first element of the list

B) Interchange the first two elements of the list

C) Delete the last element of the list

D) Add an element at the end of the list

Answer: (C)

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,
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.

Consider the following piece of 'C' code fragment that removes duplicates from an ordered list of integers.

```
Node  *remove-duplicates(Node *head, int *j)
{
    Node *t1, *t2;
    *j=0;
    t1 = head;
    if (t1! = NULL) t2 = t1 →next;
    else return head;
    *j = 1;
    if(t2 == NULL)
        return head;
    while t2 != NULL)
    {
        if (t1.val != t2.val) --------------------------→ (S1)
        {
            (*j)++; t1 -> next = t2; t1 = t2: ----------→ (S2)
        }
        t2 = t2 →next;
    }
    t1 →next = NULL;
    return head;
}
```

Assume the list contains n elements (n≥2) in the following questions.

a). How many times is the comparison in statement S1 made?

b). What is the minimum and the maximum number of times statements marked S2 get executed?

c). What is the significance of the value in the integer pointed to by j when the function completes?

The following steps in a linked list

```
p = getnode()
 info (p) = 10
 next (p) = list
 list = p
```

result in which type of operation?

A) pop operation in stack

B) removal of a node

C) inserting a node

D) modifying an existing node

Answer: C

Explanation:

p = getnode() // getnode() allocates the space which is equal to the size of the node type structure and returns a pointer. info (p) = 10 // value of the new node is equal to 10 next (p) = list // adding new node to the list. Clearly, through these steps, insertion of a node is performed. Option (C) is correct.

The time required to search an element in a linked list of length n is

A) O (log n)

B) O (n)

C) O (1)

D) O(n2)

Which of the following operations is performed more efficiently by doubly linked list than by linear linked list?

A) Deleting a node whose location is given

B) Searching an unsorted list for a given item

C) Inserting a node after the node with a given location

D) Traversing the list to process each node

Answer: A

A doubly linked list is declared as

```
struct Node {
        int Value;
        struct Node *Fwd;
        struct Node *Bwd;
);
```

Where Fwd and Bwd represent forward and backward link to the adjacent elements of the list. Which of the following segments of code deletes the node pointed to by X from the doubly linked list, if it is assumed that X points to neither the first nor the last node of the list?

A) X->Bwd->Fwd = X->Fwd; X->Fwd->Bwd = X->Bwd ;

B) X->Bwd.Fwd = X->Fwd ; X.Fwd->Bwd = X->Bwd ;

C) X.Bwd->Fwd = X.Bwd ; X->Fwd.Bwd = X.Bwd ;

D) X->Bwd->Fwd = X->Bwd ; X->Fwd->Bwd = X->Fwd;

Answer: A

Following is C like pseudo code of a function that takes a number as an argument, and uses a stack S to do processing.

```
void fun(int n)
{
    Stack S;  // Say it creates an empty stack S
    while (n > 0)
    {
      // This line pushes the value of n%2 to stack S
      push(&S, n%2);

      n = n/2;
    }

    // Run while Stack S is not empty
    while (!isEmpty(&S))
      printf("%d ", pop(&S)); // pop an element from S and print it
}
```

What does the above function do in general?

A) Prints binary representation of n in reverse order
B) Prints binary representation of n
C) Prints the value of Logn
D) Prints the value of Logn in reverse order

Answer: B

Which one of the following is an application of Stack Data Structure?

A) Managing function calls

B) The stock span problem

C) Arithmetic expression evaluation

D) All of the above

Answer: D

Which of the following is true about linked list implementation of stack?
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 end, then in pop operation, nodes must be removed from the beginning.
C) Both of the above
D) None of the above

Answer: D

Explanation: To keep the **L**ast **I**n **F**irst **O**ut order, a stack can be implemented using linked list in two ways:

a) In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from beginning.

b) In push operation, if new nodes are inserted at the end of linked list, then in pop operation, nodes must be removed from end.

Consider the following pseudocode that uses a stack

```
declare a stack of characters
while ( there are more characters in the word to read )
{
  read a character
  push the character on the stack
}
while ( the stack is not empty )
{
  pop a character off the stack
  write the character to the screen
}
```

Run on IDE

What is output for input "test"?

A) testtest
B) tset
C) test
D) Tsettset

Answer: (B)

Explanation: Since the stack data structure follows LIFO order. When we pop() items from stack, they are popped in reverse order of their insertion (or push())

Following is an incorrect pseudocode for the algorithm which is supposed to determine whether a sequence of parentheses is balanced:

```
declare a character stack
while ( more input is available)
{
   read a character
   if ( the character is a '(' )
      push it on the stack
   else if ( the character is a ')' and the stack is not empty )
      pop a character off the stack
   else
      print "unbalanced" and exit
 }
 print "balanced"
```

Which of these unbalanced sequences does the above code think is balanced?
A) ((())
B) ())(()
C) (()()))
D) (()))()

Answer: (A)

> **Explanation:** At the end of while loop, we must check whether the stack is empty or not. For input ((()), the stack doesn’t remain empty after the loop.

The following postfix expression with single digit operands is evaluated using a stack:

8 2 3 ^ / 2 3 * + 5 1 * -

Note that ^ is the exponentiation operator. The top two elements of the stack after the first * is evaluated are:

A) 6, 1

B) 5, 7

C) 3, 2

D) 1, 5

Answer: (A)

Let S be a stack of size n >= 1. Starting with the empty stack, suppose we push the first n natural numbers in sequence, and then perform n pop operations. Assume that Push and Pop operation take X seconds each, and Y seconds elapse between the end of one such stack operation and the start of the next operation. For m >= 1, define the stack-life of m as the time elapsed from the end of Push(m) to the start of the pop operation that removes m from S. The average stack-life of an element of this stack is

A) $n(X+ Y)$

B) $3Y + 2X$

C) $n(X + Y)-X$

D) $Y + 2X$

Answer: C

A single array A[1..MAXSIZE] is used to implement two stacks. The two stacks grow from opposite ends of the array. Variables top1 and top2 (topl< top 2) point to the location of the topmost element in each of the stacks. If the space is to be used efficiently, the condition for “stack full” is (GATE CS 2004)

A) (top1 = MAXSIZE/2) and (top2 = MAXSIZE/2+1)

B) top1 + top2 = MAXSIZE

C) (top1= MAXSIZE/2) or (top2 = MAXSIZE)

D) top1= top2 -1

Answer(d)
If we are to use space efficiently then size of the any stack can be more than MAXSIZE/2.
Both stacks will grow from both ends and if any of the stack top reaches near to the other top then stacks are full. So the condition will be top1 = top2 -1 (given that top1 < top2)

Assume that the operators +, -, × are left associative and ^ is right associative. The order of precedence (from highest to lowest) is ^, x , +, -. The postfix expression corresponding to the infix expression

a + b × c - d ^ e ^ f is

A) abc × + def ^ ^ -

B) abc × + de ^ f ^ -

C) ab + c × d - e ^ f ^

D) - + a × bc ^ ^ def

Answer: (A)

Explanation: ^ is right assosciative.

To evaluate an expression without any embedded function calls:

A) One stack is enough

B) Two stacks are needed

C) As many stacks as the height of the expression tree are needed

D) A Turing machine is needed in the general case Stack

GATE-CS-2002

Answer: (A)

A function f defined on stacks of integers satisfies the following properties. f(∅) = 0 and f (push (S, i)) = max (f(S), 0) + i for all stacks S and integers i.

If a stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top, what is f(S)?

(A) 6

(B) 4

(C) 3

(D) 2

Answer: (C)

Explanation:
$f(S) = 0, \max(f(S), 0) = 0, i = 2$
$f(S)new = \max(f(S), 0) + i = 0 + 2 = 2$

$f(S) = 2, \max(f(S), 0) = 2, i = -3$
$f(S)new = \max(f(S), 0) + i = 2 - 3 = -1$

$f(S) = -1, \max(f(S), 0) = 0, i = 2$
$f(S)new = \max(f(S), 0) + i = 0 + 2 = 2$

$f(S) = 2, \max(f(S), 0) = 2, i = -1$
$f(S)new = \max(f(S), 0) + i = 2 - 1 = 1$

$f(S) = 1, \max(f(S), 0) = 1, i = 2$
$f(S)new = \max(f(S), 0) + i = 1 + 2 = 3$

Thus, option (C) is correct.

Suppose a stack is to be implemented with a linked list instead of an array. What would be the effect on the time complexity of the push and pop operations of the stack implemented using linked list (Assuming stack is implemented efficiently)?

A) O(1) for insertion and O(n) for deletion

B) O(1) for insertion and O(1) for deletion

C) O(n) for insertion and O(1) for deletion

D) O(n) for insertion and O(n) for deletion

Answer: (B)

Explanation: Stack can be implemented using link list having O(1) bounds for both insertion as well as deletion by inserting and deleting the element from the beginning of the list.

Consider n elements that are equally distributed in k stacks. In each stack, elements of it are arranged in ascending order (min is at the top in each of the stack and then increasing downwards).

Given a queue of size n in which we have to put all n elements in increasing order. What will be the time complexity of the best known algorithm?

(A) O(n logk)

(B) O(nk)

(C) O(n2)

(D) O(k2)

Answer: (A)

Explanation:
In nlogk it can be done by creating a min heap of size k and adding all the top – elements of all the stacks. After extracting the min , add the next element from the stack from which we have got our 1st minimum.
Time Complexity = O(k) (For Creating Heap of size k) + (n-k)log k (Insertions into the heap).

Which of the following permutation can be obtained in the same order using a stack assuming that input is the sequence 5, 6, 7, 8, 9 in that order?

A) 7, 8, 9, 5, 6

B) 5, 9, 6, 7, 8

C) 7, 8, 9, 6, 5

D) 9, 8, 7, 5, 6

Stack   ISRO CS 2017

Explanation:

The sequence given in option (C) is one of the only possible sequence which can be obtained.

We can obtain the sequence by performing operations in the manner: Push 5 Push 6 Push 7 Pop 7 Push 8 Pop 8 Push 9 Pop 9 Pop 6 Pop 5. hence the sequence will be 7,8,9,6,5.

The minimum number of stacks needed to implement a queue is

(A) 3

(B) 1

(C) 2

(D) 4

Answer: (C)

The seven elements A, B, C, D, E, F and G are pushed onto a stack in reverse order, i.e., starting from G. The stack is popped five times and each element is inserted into a queue.Two elements are deleted from the queue and pushed back onto the stack. Now, one element is popped from the stack. The popped item is ________.

A) A

B) B

C) F

D) G

Answer: B

Which of the following is not an inherent application of stack?

A) Implementation of recursion

B) Evaluation of a postfix expression

C) Job scheduling

D) Reverse a string

Answer: C

Stack A has the entries a, b, c (with a on top). Stack B is empty. An entry popped out of stack A can be printed immediately or pushed to stack B. An entry popped out of the stack B can be only be printed. In this arrangement, which of the following permutations of a, b, c are not possible?

A) b a c

B) b c a

C) c a b

D) a b c

Answer: (C)

Explanation: Follow these steps to print bac.
1) POP element ‘a’ from stack A, push ‘a’ to Stack B.
2) POP element ‘b’ from A, print it.
3) POP element ‘a’ from B, and print it.
4) POP element ‘c’ from A, and print it.
Now, perumtation bac has been printed.
Similarly, we can print bca and abc, but can’t print cab.

Convert the following infix expression into its equivalent post fix expression (A + B^ D) / (E – F) + G

A) ABD^ + EF – / G+

B) ABD + ^EF – / G+

C) ABD + ^EF / – G+

D) ABD^ + EF / – G+

Answer: A

Following is C like pseudo code of a function that takes a Queue as an argument, and uses a stack S to do processing.

```
void fun(Queue *Q)
{
    Stack S;  // Say it creates an empty stack S

    // Run while Q is not empty
    while (!isEmpty(Q))
    {
        // deQueue an item from Q and push the dequeued item to S
        push(&S, deQueue(Q));
    }

    // Run while Stack S is not empty
    while (!isEmpty(&S))
    {
      // Pop an item from S and enqueue the poppped item to Q
      enQueue(Q, pop(&S));
    }
}
```

What does the above function do in general?

A) Removes the last from Q
B) Keeps the Q same as it was before the call
C) Makes Q empty
C) Reverses the Q

Answer: (D)
Explanation: The function takes a queue Q as an argument. It dequeues all items of Q and pushes them to a stack S. Then pops all items of S and enqueues the items back to Q. Since stack is LIFO order, all items of queue are reversed.

Which one of the following is an application of Queue Data Structure?

A) When a resource is shared among multiple consumers.

B) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes

C) Load Balancing

D) All of the above

Answer: D

Consider the following pseudo code. Assume that IntQueue is an integer queue. What does the function fun do?

```
void fun(int n)
{
    IntQueue q = new IntQueue();
    q.enqueue(0);
    q.enqueue(1);
    for (int i = 0; i < n; i++)
    {
        int a = q.dequeue();
        int b = q.dequeue();
        q.enqueue(b);
        q.enqueue(a + b);
        ptint(a);
    }
}
```

A) Prints numbers from 0 to n-1
B) Prints numbers from n-1 to 0
C) Prints first n Fibonacci numbers
D) Prints first n Fibonacci numbers in reverse order.

Answer: (C)
Explanation: The function prints first n Fibonacci Numbers. Note that 0 and 1 are initially there in q. In every iteration of loop sum of the two queue items is enqueued and the front item is dequeued.

Suppose implementation supports an instruction REVERSE, which reverses the order of elements on the stack, in addition to the PUSH and POP instructions. Which one of the following statements is TRUE with respect to this modified stack?

A) A queue cannot be implemented using this stack.

B) A queue can be implemented where ENQUEUE takes a single instruction and DEQUEUE takes a sequence of two instructions.

C) A queue can be implemented where ENQUEUE takes a sequence of three instructions and DEQUEUE takes a single instruction.

D) A queue can be implemented where both ENQUEUE and DEQUEUE take a single instruction each.

Answer: (C)

Explanation: To DEQUEUE an item, simply POP.

To ENQUEUE an item, we can do following 3 operations
1) REVERSE
2) PUSH
3) REVERSE

A queue is implemented using an array such that ENQUEUE and DEQUEUE operations are performed efficiently. Which one of the following statements is CORRECT (n refers to the number of items in the queue)?

A) Both operations can be performed in O(1) time

B) At most one operation can be performed in O(1) time but the worst case time for the other operation will be Ω(n)

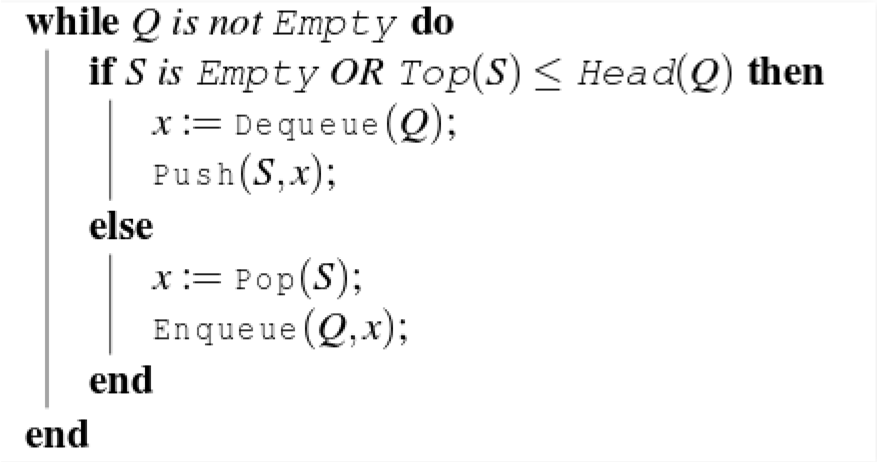
C) The worst case time complexity for both operations will be Ω(n)

D) Worst case time complexity for both operations will be Ω(log n

Answer: (A)

Explanation**:** We can use circular array to implement both in O(1) time.

Let Q denote a queue containing sixteen numbers and S be an empty stack. Head(Q) returns the element at the head of the queue Q without removing it from Q. Similarly Top(S) returns the element at the top of S without removing it from S. Consider the algorithm given below.



```
while $Q$ is not $\texttt{Empty}$ do
    if $S$ is $\texttt{Empty}$ OR $\texttt{Top}(S) \leq \texttt{Head}(Q)$ then
        $x := \texttt{Dequeue}(Q)$;
        $\texttt{Push}(S,x)$;
    else
        $x := \texttt{Pop}(S)$;
        $\texttt{Enqueue}(Q,x)$;
    end
end
```

The maximum possible number of iterations of the while loop in the algorithm is______

A) 16

B) 32

C) 256

D) 6

Answer: C

Explanation: The worst case happens when the queue is sorted in decreasing order. In worst case, loop runs n*n times.

Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are:

i. isEmpty (Q) — returns true if the queue is empty, false otherwise.

ii. delete (Q) — deletes the element at the front of the queue and returns its value.

iii. insert (Q, i) — inserts the integer i at the rear of the queue.

Consider the following function:

```
void f (queue Q) {
int i ;
if (!isEmpty(Q)) {
   i = delete(Q);
   f(Q);
   insert(Q, i);
  }
}
```

What operation is performed by the above function f ?

A) Leaves the queue Q unchanged

B) Reverses the order of the elements in the queue Q

C) Deletes the element at the front of the queue Q and inserts it at the rear keeping the other elements in the same order

D) Empties the queue Q

Answer: (B)

Explanation: As it is recursive call, and removing from front while inserting from end, that means last element will be deleted at last and will be inserted 1st in the new queue. And like that it will continue till first call executes insert(Q,i) function.

So, the queue will be in reverse.

Consider a standard Circular Queue 'q' implementation (which has the same condition for Queue Full and Queue Empty) whose size is 11 and the elements of the queue are q[0], q[1], q[2].....,q[10]. The front and rear pointers are initialized to point at q[2] . In which position will the ninth element be added?

A) q[0]

B) q[1]

C) q[9]

D) q[10]

Answer: A