Mid Semester Examination March 2019

Max. Marks: 20 Class: FYMCA Course Code: MCA23

Name of the Course: Data Structures

Duration: 1 Hr Semester: II Branch: MCA

Synoptic

1) Worst case complexity calculation of Insertion sort Worst case complexity calculation of Bubble sort

(2 mks) (2 mks)

Example: 12, 10, 8, 6, 5

Insertion sort: Number of passes and comparison in first pass = 4 Number of passes and comparison in second pass = 8 Number of passes and comparison in third pass = 12

Number of passes and comparison in forth pass = 16

Hence the complexity = $4 + 8 + 12 + 16 = O(n^2)$

Bubble sort: Number of passes and comparison in first pass = 16

Number of passes and comparison in second pass = 12 Number of passes and comparison in third pass = 8 Number of passes and comparison in forth pass = 4

Hence the complexity = $16 + 12 + 8 + 4 = O(n^2)$

Number of passes and Iterations calculation for Insertion sort Number of passes and Iterations calculation for Selection sort (2 mks)

(2 mks)

Example: 10, 2, -20, 30, 12, 34

Insertion sort:

	THISCI CION 3	UII.					
	0	1	2	3	4	5	No. Of Iterations
0 (Original Aray)	10	2	-20	30	12	34	
1	2	10					1
2	-20	2	10				2
3	-20	2	10	30			1
4	-20	2	10	12	30		2
5	-20	2	10	12	30	34	1

Selection sort:									
	0	1	2	3	4	5	No. Of Iterations		
0 (Original Aray)	10	2	-20	30	12	34	II.		
1	10	2	-20	30	12	34	5		
2	10	2	-20	12	30	34	4		
3	10	2	-20	12	30	34	3		
4	-20	2	10	12	30	34	2		
5	-20	2	10	12	30	34	1		

Comparison Table:

	No. Of passes	No. Of Iterations
Insertion Sort	5	7
Selection Sort	5	15

```
2) Applying double hashing for each key value.
   Formula is,
     h(k,i) = (h1(k) + ih2(k)) \mod n
           h1(k) = k \mod n
                                    and
                                                h2(k) = 1 + (k \mod (n-1))
    For 12,
        h(12, 0) = (h1(12) + 0 h2(12)) \mod 10
            h1(12) = 12\%10 = 2
      Hence 12 is stored at 2nd Position
    For 2,
         h(2, 0) = (h1(2) + 0 h2(2)) \mod 10
            h1(2) = 2 \% 10 = 2
           Here, 2 position is already full. So we need to calculate h(2,1)
         h(2, 1) = (h1(2) + 1 h2(2) mod 10
            h1(2) = 2
                           and h2(2) = 1 + (2 \% 9) = 3
         H(2,1) = (2+3)\%10 = 5
      Hence 2 is stored at 5th position
    Likewise 18 will be stored at 8th location and 45 will be stored at 6th location.
```

3) Solution of Construct an algorithm for reversing of singly linked list elements is

(4 mks)

(1 mk

Begin

Initialize q and s pointer with list pointer
Initialize temp pointer will NULL
Initialize r pointer with q->next
Repeat
Set temp to q
Set q to r
Set r to q->next
Set q->next to temp

Until r not equal to NULL

Set list to q

Set s->next to NULL

End

4) Pushing each symbol onto stack.

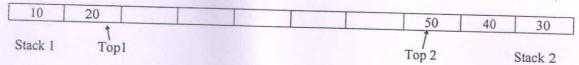
(0.25 mk each)

Symbol	(A	+	В)	*	C	1	D	1	E)
Index	0	1	2	3	4	5	6	7	8	9	10	11

Index	Symbol	Stack	Expression	
0	(((
1	A	((A	
2	+	((+	A	
3	В	((+	AB	
4)	(AB+	
5	*	(*	AB+	
6	C	(*	AB+C	
7	1	(*/	AB+C	
8	D	(*/	AB+CD	
9	1	(*/	AB+CD/	
10	E	(*/	AB+CD/E	
11)		AB+CD/E/*	

Required Postfix expression is AB+CD/E/*

Solution of construct an algorithm to implement 2 Stacks using only one array is attached here with.



Algorithm:

Step 1: Start two indexes one at left end and other at right end.

Step 2: The left index simulates the first stack amd the right index simulates second stack

Step 3: If we want to push an element into the first stack then put the element at left

(2 mks each)

index.

Step 4: Similarly, if we want to push an element into the second stack then put at right index.

Step 5: First stack grows towards right and second stack grows towards left.

5) Solution of construct enqueue and dequeue algorithms to implement queue using 2 stacks.

```
Let S1 and S2 be to stacks to be used to implement Queue.

Struct Queue

{

Struct Stack *S1; // for Enqueue

Struct Stack *s2; // for Dequeue
};

Enqueue Algorithm:

Just push on to Stack S1

void Enqueue(Struct Queue *Q, int data)

{

Push (Q -> S1, data);
```

Dequeue Algorithm:

♦ If stack S2 is not empty, then pop S2 and return that element.

♦ If stack is empty; then transfer all elements from S1 to S2 and pop the top element from S2 and return that top element.

If stack S1 is empty then throw an error.

```
int Dequeue(Struct Queue *Q)

{
    if(!IsEmptyStack(Q -> S2))
        return pop(Q -> S2);
    else
    {
        while(!IsEmptyStack(Q -> S1))
            push(Q -> S2, pop(Q -> S1));
        return pop(Q -> S2);
    }
```