Roll Number:____________________ Number of Pages: 02

Thapar Institute of Engineering and Technology, Patiala

Department of Computer Science and Engineering

MID SEMESTER EXAMINATION

B. E. (Second Year): Semester-II (2019/20)	Course Code: UCS301/UCS406 Course Name: Data Structures
March 07, 2019	Saturday, 10:30 Hrs – 12:30 Hrs
Time: 2 Hours, M. Marks: 25	Name of Faculty: SUG, RMT, SP, AA

Note: Attempt all questions (sub-parts) in sequence. Assume missing data, if any, suitably.

Q1. Convert the given infix expression into an equivalent postfix expression using stacks. Show contents of the stack at each intermediate step. Use the precedence table as shown in **Table 1**. **(3)**

(A $ B + C) # (K + L – M * N + O ^ P * W / U)

Table 1. Precedence of operators.

Operator	**Precedence**
#	Highest
^	↓
*, /	
+, –	
$	Lowest

Q2. Let the letters **S, E, A, T, B, L** has to be pushed on to an empty stack of characters in the order they appear from left to right. Consider that the output of **pop()** operation is appended in an initially empty output string. **(1)**

Determine the sequence in which **push()** and **pop()** operations should be called such that the contents in the output string should be **STABLE**.

Q3. Compute complexity of the following pseudo-codes giving proper justifications. **(3)**

(a)
```
for (int i = 1; i <= n; i++)
  for(int j = 1; j <= i; j++)
    for(int k = 1; k <= j; k++)
    { … }
```

(b)
```
for (int p = 1; p + n/2 <= n; p++)
  for(int q = n; q > 0; q /= 2)
    for(int r = 1; r*r <= n; r++)
      for(int s = 1; s < n; s++)
        { break; … }
```

(c)
```
m = n;
while ( n > 0)
{  for(int i = 0; i < m; i++)
   …
   n = n/3;
}
```

Q4. (a) A two-dimensional array defined as **A [-4 … 6] [-2 … 12]** requires 2 bytes of storage for each element. If the array is stored in row major order form with the address **A[4][8]** as 4142. Compute the address of **A[0][0]**. **(2)**

(b) A circular queue of positive integers is implemented using a linear array **A [1..6]**. The present contents of **A** are **{4, …, …, 7, 9, 2}** where '**…**' represents empty. After two dequeue and one enqueue operations, let the positions of **front** and **rear** pointers be **X** and **Y**, respectively. Then **4X + 5Y** will be? **(1)**

Q5. Let **S** be an empty stack and **Q** be a queue with contents as shown in **Fig. 1**. **isEmpty(Q)** or **isEmpty(S)** returns **true** if **Q** or **S** is empty, else returns **false**. **top(S)** returns the character at the *top* of **S** without removing it from **S**. **(3)**

Execute the code snippet given in **Fig. 2** and answer the following questions.

(a) Give the *contents* of **S** after each execution of the **while loop within main()**.

(b) Observe the output obtained in **Q5. (a)** and clearly state the purpose(s) of designing it.

Q:	D	A	T	A	B	A	S	E
	Front							

Fig. 1

```
function1 (char val)
{ if (isEmpty(S) || top(S) < val)
  { push(S,val);
    return;
  }
  char c = pop(S);
  function1(val);
  if (c != top(S))
    push(S,c);
}
int main()
{  while (!isEmpty(Q))
     function1(dequeue(Q));
}
```

Fig. 2

Q6. A Radix Sort algorithm utilizing Counting Sort as an intermediate stable sorting algorithm is to be applied on the contents of array **A** to arrange them in increasing order. Let **COUNT [0..9]** be the array that counting sort uses to store the frequency values during intermediate steps. **(5)**

A:	387	690	234	435	567	203	441	892

Illustrate step wise execution of counting sort, i.e. list the contents of array **COUNT**, for each of the iterations of Radix sort (starting from LSD). Also show the contents of array **A** across all the iterations of Radix sort.

Q7. Let **A[0 .. n – 1]** be an array of numbers. A number **A[i]** is called a *principal* if it is greater than the mean of all numbers from **A[i]** to **A[n – 1]**. Write an efficient pseudo-code to delete all the *principal* numbers in the array **A**. Explain the proposed logic with the help of an example. **(3)**

Q8. Write a pseudo-code to implement **Round Robin CPU Scheduling** using **singly circular linked list** assuming processes may have different arrival as well as burst times. **(4)**

Round Robin is a pre-emptive scheduling algorithm in which CPU is assigned to a process on the basis of FCFS for a fixed amount of time known as 'time quantum'. After time quantum expires, the running process is pre-empted and the processor is assigned to the next arrived process.

Note:

- ***tq** represents time quantum, **SCLL** is a singly circular linked list maintaining list of processes which are being executed currently. **LL** is a simple linked list maintaining list of processes (in FCFS order) which are freshly arrived and yet to enter **SCLL**.*
- *At the end of each **tq**, all the processes are deleted from **LL** and are inserted in the same order in **SCLL**. Similarly, when the burst time reaches zero the corresponding process is finally deleted from **SCLL**.*
- *Memory should be allocated to a process only once when it arrives freshly. Also assume that insertion is happening automatically in **LL**, so there is no need to specify it's corresponding pseudo-code.*

---**ALL THE BEST**---