Paper code - V239 - 124 (T1)

Total No. of Questions – [03] Total No. of Printed Pages: 03

G.R. No.	

OCTOBER 2019/20 / INSEM (T1)
S. Y. B.TECH. (COMPUTER ENGG.) (SEMESTER – I)
COURSE NAME: DATA STRUCTURE AND ALGORITHMS
COURSE CODE: CSUA21184

(PATTERN 2018)
SOLUTION SCHEME

Time: [1 Hour] [Max. Marks: 20]

Q.1 a. Row major mapping address calculation: 1306 (2)
Column major mapping address calculation 1302 (2)

Sparse matrix definition: Matrix with few non zero elements (1)
Simple Transpose Algorithm (3)

```
void transpose(term a[], term b[])
/* b is set to the transpose of a */
{
   int n,i,j, currentb;
   n = a[0].value;          /* total number of elements */
   b[0].row = a[0].col; /* rows in b = columns in a */
   b[0].col = a[0].row; /* columns in b = rows in a */
   b[0].value = n;
   if (n > 0 )  { /* non zero matrix */
      currentb = 1;
      for (i = 0; i < a[0].col; i++)
      /* transpose by the columns in a */
         for (j = 1; j <= n; j++)
         /* find elements from the current column */
            if (a[j].col == i) {
            /* element is in current column, add it to b */
               b[currentb].row = a[j].col;
               b[currentb].col = a[j].row;
               b[currentb].value = a[j].value;
               currentb++;
            }
   }
}
```

Q.2 a. (8)

```
void selection_sort (int A[ ], int n) {
        // temporary variable to store the position of minimum element

        int minimum;
```

```
        // reduces the effective size of the array by one in  each
iteration.

        for(int i = 0; i < n-1 ; i++)  {

           // assuming the first element to be the minimum of the unsorted
array .
             minimum = i ;

          // gives the effective size of the unsorted  array .

             for(int j = i+1; j < n ; j++ ) {
                if(A[ j ] < A[ minimum ])  {                //finds the
minimum element
                minimum = j ;
                }
             }
          // putting minimum element on its proper position.
          swap ( A[ minimum ], A[ i ]) ;
        }
   }
```

b. Hashing is efficient searching as complexity is O(1)
Binary Search Algorithm (3)

1. Set found=false
2. Set first_location to start of list
3. Set last_location to end of list
4. Input search target
5. Repeat
6. Set pointer to middle of list.... integer(first+last)/2
7. If array(middle)=target then
8. found=true
9. Output suitable message
10. Else
11. if array(middle)>target then
12. last_location=middle-1 (4)
13. else (1)
14. first_location = middle+1
15. end if
16. End if
17. Until found = true or first>last

Complexity O(logn)

Q.3 Search node in a SLL

a. Replacing element in SLL (2)

```
void search(node *first)
{
    node *a;
    int empid;
    cout<<"\n Enter the employee ID :";
    cin>>empid;
    a=first;
    while(a!=NULL && a->empid!=empid)
    a=a->next;
```

(2)

(4)

```
    if(a==NULL)
        cout<<"\n Record not found";
    else
        cout<<a->name<<a->dpt<<a->empid<<a->sal;
}
```

OR

b. Pseudo-code of insert position in DLL

```
Void insert_DLL()
{
temp=head;
Let pos is position at which node to be inserted
i=0;
While(i<pos)
{
temp=temp->next;
i++
}
nn->next=temp->next
(temp->next)->prev=nn
temp->next=nn;
nn->prev=temp
```