SAL Engineering and Technical Institute

Computer Engineering Department

Lab Manual

Name : Ayushkumar patel

Enrollment no.:191260116029

Subject : Analysis and Design of Algorithms

Semester: 5th

1(A) Implementation of Bubble Sort

```
#include<stdio.h>
#include<conio.h>
#define MAX_SIZE 5
void bubble_sort(int[]); int main() {     int arr_sort[MAX_SIZE], i;
printf("Simple Bubble Sort Example - Array and Functions\n");
printf("\nEnter %d Elements for Sorting\n", MAX_SIZE);     for (i
= 0; i < MAX_SIZE; i++)                scanf("%d", &arr_sort[i]);
printf("\nYour Data :");       for (i = 0; i < MAX_SIZE; i++) {
printf("\t%d", arr_sort[i]);

   }
   bubble_sort(arr_sort);
getch();

} void bubble_sort(int fn_arr[])
{

   int i, j, a, t;
   for (i = 1; i < MAX_SIZE; i++) {
for (j = 0; j < MAX_SIZE - 1; j++) {
if (fn_arr[j] > fn_arr[j + 1]) {
```

```
                //Swapping Values
t = fn_arr[j];                fn_arr[j] =
fn_arr[j + 1];                fn_arr[j +
1] = t;

            }
        }         printf("\nIteration %d : ", i);
for (a = 0; a < MAX_SIZE; a++) {
printf("\t%d", fn_arr[a]);
        }
    }


    printf("\n\nSorted Data :");     for
(i = 0; i < MAX_SIZE; i++) {
printf("\t%d", fn_arr[i]);

    }
}
```

O/P

```
Simple Bubble Sort Example - Array and Functions


Enter 5 Elements for Sorting
677
45
32
1
17
```

```
Your Data   :  677   45   32   1   17
Iteration 1 :  45   32   1   17   677
Iteration 2 :  32   1   17   45   677
Iteration 3 :  1   17   32   45   677 Iteration
4 :  1   17   32   45   677

Sorted Data :  1   17   32   45   677
```

1(B Implementation of Insertion Sort

```
#include<stdio.h>
#include<conio.h>
#define MAX_SIZE 5

void insertion(int[]); int main() {     int arr_sort[MAX_SIZE],
i, j, a, t;     printf("Simple Insertion Sort Example - Array\n");
printf("\nEnter %d Elements for Sorting\n", MAX_SIZE);
for (i = 0; i < MAX_SIZE; i++)       scanf("%d", &arr_sort[i]);
printf("\nYour Data   :");     for (i = 0; i < MAX_SIZE; i++) {
printf("\t%d", arr_sort[i]);

    }
    for (i = 1; i < MAX_SIZE; i++) {
t = arr_sort[i];         j = i - 1;

   while (j >= 0 && arr_sort[j] > t) {
arr_sort[j + 1] = arr_sort[j];             j
= j - 1;

        }         arr_sort[j +
1] = t;
       printf("\nIteration %d : ", i);
for (a = 0; a < MAX_SIZE; a++) {
printf("\t%d", arr_sort[a]);

        }
    }
    printf("\n\nSorted Data :");
    for (i = 0; i < MAX_SIZE; i++) {
printf("\t%d", arr_sort[i]);
```

```
    )
  }
  getch();
}
```

O/P

```
Simple Insertion Sort Example - Array and Functions

Enter 5 Elements for Sorting
901
56
34
23
2

Your Data   :   901     56      34      23      2
Iteration 1 :   56      901     34      23      2
Iteration 2 :   34      56      901     23      2
Iteration 3 :   23      34      56      901     2
Iteration 4 :   2       23      34      56      901

Sorted Data :   2       23      34      56      901
```

1(C Implementation of Merge Sort

```
#include<stdio.h>
#include<conio.h>
#define MAX_SIZE 5
void merge_sort(int, int); void
merge_array(int, int, int, int); int
arr_sort[MAX_SIZE]; int main() {
  int i;
  printf("Simple Merge Sort Example - Functions and Array\n");
printf("\nEnter %d Elements for Sorting\n", MAX_SIZE);   for (i
= 0; i < MAX_SIZE; i++)          scanf("%d", &arr_sort[i]);
printf("\nYour Data    :");    for (i = 0; i < MAX_SIZE; i++) {
printf("\t%d", arr_sort[i]);
  }
  merge_sort(0, MAX_SIZE - 1);
printf("\n\nSorted Data :");   for (i
= 0; i < MAX_SIZE; i++) {
printf("\t%d", arr_sort[i]);
  }
getch();
} void merge_sort(int i, int j)
{
  int m;   if (i < j) {
m = (i + j) / 2;
merge_sort(i, m);
    merge_sort(m + 1, j);
    // Merging two arrays
merge_array(i, m, m + 1, j);
```

```
    )
  }
}
void merge_array(int a, int b, int c, int d) {
int t[50];   int i = a, j = c, k = 0;   while (i
<= b && j <= d) {     if (arr_sort[i] <
arr_sort[j])       t[k++] = arr_sort[i++];
else       t[k++] = arr_sort[j++];

  }
  //collect remaining elements
while (i <= b)     t[k++] =
arr_sort[i++];   while (j <= d)
t[k++] = arr_sort[j++];   for (i = a,
j = 0; i <= d; i++, j++)
arr_sort[i] = t[j];

}
```

O/P

```
Simple Merge Sort Example - Functions and Array
Enter 5 Elements for Sorting
67 57 45 32 13
Your Data   :   67      57      45      32      13

Sorted Data :   13      32      45      57      67
```

1(D Implementation of Quick Sort

```
#include<stdio.h>
```

```
#include<conio.h>
#define MAX_SIZE 5
void quick_sort(int, int); int
arr_sort[MAX_SIZE]; int
main() {
  int i;
  printf("Simple Quick Sort Example - Functions and Array\n");
printf("\nEnter %d Elements for Sorting\n", MAX_SIZE);   for (i
=  0;  i  <  MAX_SIZE;  i++)          scanf("%d",  &arr_sort[i]);
printf("\nYour Data    :");    for (i = 0; i < MAX_SIZE; i++) {
printf("\t%d", arr_sort[i]);
  }
  quick_sort(0, MAX_SIZE - 1);
printf("\n\nSorted Data :");   for (i
= 0; i < MAX_SIZE; i++) {
printf("\t%d", arr_sort[i]);
  }
getch();
} void quick_sort(int f, int l)
{
  int i, j, t, p = 0;
if (f < l) {     p =
f;     i = f; j = l;
```

```
    while (i < j) {
      while (arr_sort[i] <= arr_sort[p] && i < l)
i++;
      while (arr_sort[j] > arr_sort[p])
        j--;       if (i < j) {          t
= arr_sort[i];          arr_sort[i]
= arr_sort[j];          arr_sort[j]
= t;
      }     }     t = arr_sort[p];
arr_sort[p] = arr_sort[j];
arr_sort[j] = t;
quick_sort(f, j - 1);
quick_sort(j + 1, l);
  }
}
```

O/P

```
Simple Quick Sort Example - Functions and Array

Enter 5 Elements for Sorting
56
24
20
17
2
```

Your Data : 56 24 20 17 2

Sorted Data : 2 17 20 24 56

2. Implementation of Binary Search

```
#include<stdio.h>
#include<conio.h>
#define MAX_SIZE 5
void binary_search(int[],int); int main() {     int
arr_search[MAX_SIZE], i,element;     printf("Simple Binary Search
Example - Array and Functions\n");     printf("\nEnter %d Elements
for Searching : \n", MAX_SIZE);     for (i = 0; i < MAX_SIZE; i++)
scanf("%d", &arr_search[i]);             printf("Enter Element to
Search : ");     scanf("%d", &element);
binary_search(arr_search,element);     getch();

} void binary_search(int fn_arr[],int element)
{    int f = 0, r = MAX_SIZE,mid;      while (f
<= r) {

            mid = (f+r)/2;             if (fn_arr[mid] == element) {          printf("\nSearch
Element  : %d  : Found :  Position : %d.\n", element, mid+1);
       break;
      }
```

```
    else if (fn_arr[mid] < element)
f = mid + 1;          else
        r = mid - 1;
  }    if (f
> r)
    printf("\nSearch Element : %d  : Not Found \n", element);
}
```

O/P

```
Simple Binary Search Example - Array and Functions

Enter 5 Elements for Searching :
1001
1020
3002
4001
5000
Enter Element to Search : 3002

Search Element  : 3002  : Found :  Position : 3.
```

3. Implementation of Heap Sort Algorithm

```
#include<stdio.h>
#include<conio.h> #define
MAX_SIZE 5 void
heap_sort(); void
heap_adjust(int, int); int
arr_sort[MAX_SIZE], t, a; int
main() {
  int i;
  printf("Simple Heap Sort Example - Functions and Array\n");
printf("\nEnter %d Elements for Sorting\n", MAX_SIZE);   for
(i = 0; i < MAX_SIZE; i++)        scanf("%d", &arr_sort[i]);
printf("\nYour Data    :");    for (i = 0; i < MAX_SIZE; i++) {
printf("\t%d", arr_sort[i]);
  }
  heap_sort();   printf("\n\nSorted
Data :");   for (i = 0; i <
MAX_SIZE; i++) {
printf("\t%d", arr_sort[i]);
  }
getch();
} void heap_sort() {   for (int i =
MAX_SIZE / 2 - 1; i >= 0; i--)
heap_adjust(MAX_SIZE, i);
  for (int i = MAX_SIZE - 1; i >= 0; i--) {
//Swapping Values      t = arr_sort[0];
arr_sort[0] = arr_sort[i];
```

```
   arr_sort[i] = t;     heap_adjust(i, 0);
printf("\nHeap Sort Iteration %d : ", i);
for (a = 0; a < MAX_SIZE; a++) {
printf("\t%d", arr_sort[a]);

    }
  } } void heap_adjust(int n, int i) {   int large
= i, left = 2 * i + 1, right = 2 * i + 2;

  // adjest left child   if (left < n && arr_sort[left]
> arr_sort[large])     large = left;   // adjest right
child   if (right < n && arr_sort[right] >
arr_sort[large])     large = right;   if (large != i) {
//Swapping Values      t = arr_sort[i];     arr_sort[i]
= arr_sort[large];     arr_sort[large] = t;
heap_adjust(n, large);

  }
}
```

O/P

```
Simple Heap Sort Example - Functions and Array

Enter 5 Elements for Sorting
500
401
300
20
10

Your Data   :   500     401     300     20      10
Heap Sort Iteration 4 :         401     20      300     10      500
Heap Sort Iteration 3 :         300     20      10      401     500
Heap Sort Iteration 2 :         20      10      300     401     500
Heap Sort Iteration 1 :         10      20      300     401     500
Heap Sort Iteration 0 :         10      20      300     401     500

Sorted Data :   10      20      300     401     500
```

4 (A) Implementation Iterative Function Program

```
#include <stdio.h>
// Iterative function to find factorial of a number using for loop
unsigned long factorial(int n)
{       unsigned long fact =
1;      int i;
        for (i = 1; i <= n; i++)
        return fact;
}
// main function int
main()
{
        int n = 5;      printf("The Factorial of %d is %lu", n,
factorial(n));          return 0;               fact = fact * i;
}
```

O/P

The Factorial of 5 is 120

4 (B) Implementation Recursive Function Program

```
#include <stdio.h> long int
multiplyNumbers(int n); int
main()

{     int n;     printf("Enter a positive integer: ");
scanf("%d", &n);     printf("Factorial of %d = %ld", n,
multiplyNumbers(n));     return 0;

} long int multiplyNumbers(int
n)

{     if (n >= 1)         return
n*multiplyNumbers(n-1);     else
return 1;

}
```

O/P

Enter a positive integer: 6

Factorial of 6 = 720

5 . Implementation of a Knapsack Problem Using Dynamic Progr amming

```
#include <stdio.h> int max(int a, int b) {
return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity W int
knapsack(int W, int wt[], int val[], int n)

{    int i,
w;

  int K[n+1][W+1];
  // Build table K[][] in bottom up manner
for (i = 0; i <= n; i++)

  {
     for (w = 0; w <= W; w++)
     {
```

```
          if (i==0 || w==0)
K[i][w] = 0;            else if
(wt[i-1] <= w)
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
else
                K[i][w] = K[i-1][w];
       }
   }
 return K[n][W];
}  int main() {      int val[] =
{60, 100, 120};     int wt[] =
{10, 20, 30};
```

```
   int  W = 50;     int n = sizeof(val)/sizeof(val[0]);
printf("\nValue = %d", knapsack(W, wt, val, n));
   return 0;
}
```

O/P

```
Value = 220
```

6 . Implementation of a Chain Matrix Multiplication Using Dyna
mic Programming

```
#include <stdio.h>
#include<limits.h>
#define INFY 999999999
long int m[20][20]; int
s[20][20]; int p[20],i,j,n;
void print_optimal(int i,int j)
{ if (i == j)
printf(" A%d ",i);
else
  {       printf("( ");
print_optimal(i, s[i][j]);
print_optimal(s[i][j] + 1, j);
printf(" )");
  } }
void matmultiply(void)
{ long int q; int
k;
for(i=n;i>0;i--)
```

```
 {
for(j=i;j<=n;j++)

   {
if(i==j)
      m[i][j]=0;
else       {
       for(k=i;k<j;k++)
       {
        q=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
if(q<m[i][j])

         {
m[i][j]=q;
s[i][j]=k;

         }
        }
       }
     }
 }
}


int MatrixChainOrder(int p[], int i, int j)
{     if(i == j)
return 0;
int k;

   int min = INT_MAX;
   int count;
```

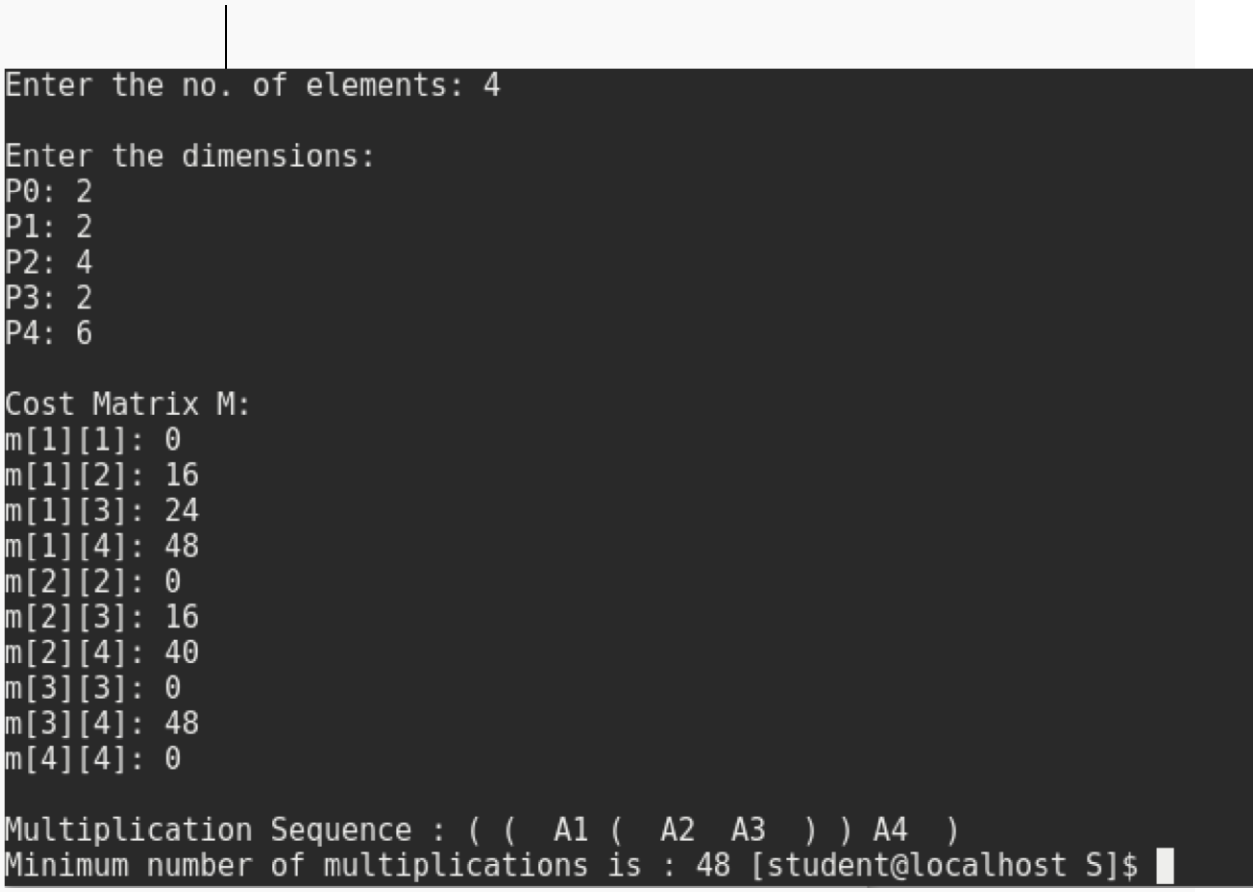
```
   for (k = i; k <j; k++)
   {
      count = MatrixChainOrder(p, i, k) +
MatrixChainOrder(p, k+1, j) +                p[i-1]*p[k]*p[j];
```

```
        if (count < min)
 min = count;

    }
   // Return minimum count
return min;

} void
main()

{ int k; printf("Enter the no. of
elements: "); scanf("%d",&n);
for(i=1;i<=n;i++)
for(j=i+1;j<=n;j++)

{  m[i][i]=0;
m[i][j]=INFY;
s[i][j]=0;

} printf("\nEnter the dimensions:
\n"); for(k=0;k<=n;k++)

{
 printf("P%d: ",k);
scanf("%d",&p[k]);

}
matmultiply();

printf("\nCost Matrix M:\n");
for(i=1;i<=n;i++)  for(j=i;j<=n;j++)
printf("m[%d][%d]: %ld\n",i,j,m[i][j]);
i=1,j=n;
```

```
printf("\nMultiplication Sequence : ");
```

```
 print_optimal(i,j);

printf("\nMinimum number of multiplications is : %d ",
                    MatrixChainOrder(p, 1, n));


}
```

O/P



7. Implementation of a Making Change Problem Using Dynamic Programming

```
    void main()
 {    int n;   clrscr();   printf("\n-------------------------
----------------------");

  printf("\n       MAKING CHANGE USING GREEDY ALGORITHM     ");
  printf("\n-------------------------------------------------");
printf("\n Enter amount you want:");
scanf("%d",&n);   make_change(n);

  getch();
 }
 void make_change(int n)
 {  int S[100],s=0,x,ind=0,i;  printf("\n----------------
AVAILABLE COINS----------------\n");  for(i=0;i<= 4;i++)
printf("%5d",C[i]);  printf("\n---------------------------------------
-------");  while(s!=n)

 {
```

```
  x=bestsol(s,n);
if(x==-1)   {}
else

  {
   S[ind++]=x;
s=s+x;

  }
 }
 printf("\n--------------MAKING CHANGE FOR %4d-------------",n);
for(i=0;i < ind;i++)

 {
printf("\n%5d",S[i]);

 }  printf("\n-----------------------------------------------
");

} int bestsol(int s,int
n)

{
 int i;  for(i=4;i>-1;i--
)
```

```
 {   if((s+C[i]) <=
n)    return C[i] ;
 }  return -
1;

}
```

O/ P

```
 MAKING CHANGE USING GREEDY ALGORITHM
------------------------------------------------------------------------------
Enter amount you want:196

--------------------------AVAILABLE COINS-------------------------
1    5   10   25  100
------------------------------------------------------------------------------ ---
------------------MAKING CHANGE FOR  196--------------------   100
   25
   25
   25
   10
   10
    1
```

8 (A) Implementation Depth First Search for Graph

```
#include<stdio.h>
#include<stdlib.h> typedef
struct node
{     struct node
*next;
   int vertex; }node;
node *G[20];
//heads of linked list
int visited[20]; int n;
void read_graph();
//create adjacency list
void insert(int,int);
//insert an edge (vi,vj) in te adjacency list
void DFS(int);  void
main()
{
   int i;
   read_graph();
//initialised visited to 0

   for(i=0;i<n;i++)
visited[i]=0;
   DFS(0);
}

void DFS(int i)
{     node *p;
printf("\n%d",i);
p=G[i];
   visited[i]=1;
   while(p!=NULL)
   {
     i=p->vertex;

if(!visited[i])
DFS(i);
       p=p->next;
   } }
void read_graph()
{
```

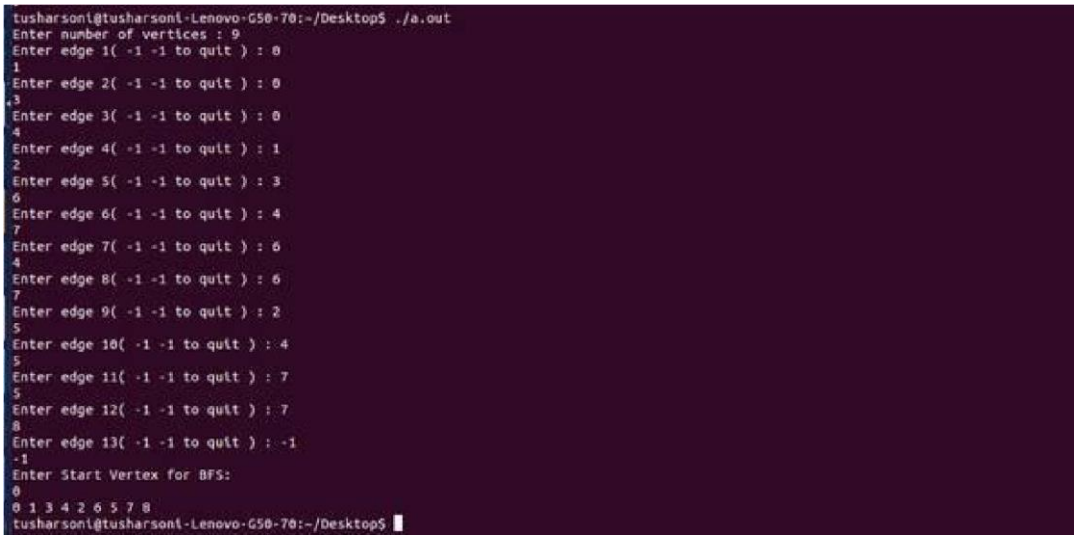
```
   int i,vi,vj,no_of_edges;
   printf("Enter number of vertices:");

   scanf("%d",&n);

   //initialise G[] with a null

   for(i=0;i<n;i++)
   {
      G[i]=NULL;
      //read edges and insert them in G[]

      printf("Enter number of edges:");
        scanf("%d",&no_of_edges);

        for(i=0;i<no_of_edges;i++)
      {
         printf("Enter an edge(u,v):");
         scanf("%d%d",&vi,&vj);
insert(vi,vj);
      }
   } }
void insert(int vi,int vj)
{     node
*p,*q;

   //acquire memory for the new node
q=(node*)malloc(sizeof(node));     q->vertex=vj;
   q->next=NULL;

   //insert the node in the linked list number vi
   if(G[vi]==NULL)
      G[vi]=q;
else
   {
      //go to end of the linked list
p=G[vi];

      while(p->next!=NULL)
p=p->next;
      p->next=q;
   }
}
```

8 (B) Implementation Breadth First Search for Graph

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];  void
create_graph(); void
BF_Traversal();
void BFS(int v);

int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex); int
delete_queue();
int isEmpty_queue();

int main() {
create_graph();
BF_Traversal();
return 0;
}

void BF_Traversal()
{     int
v;

    for(v=0; v<n; v++)
        state[v] = initial;

    printf("Enter Start Vertex for BFS: \n");
scanf("%d", &v);
    BFS(v);
}

void BFS(int v)
{
    int i;
```

```
   insert_queue(v);
state[v] = waiting;
   while(!isEmpty_queue())
   {        v =
delete_queue( );
printf("%d ",v);
       state[v] = visited;

       for(i=0; i<n; i++)
       {
          if(adj[v][i] == 1 && state[i] == initial)
          {
insert_queue(i);
state[i] = waiting;
          }
       }     }
printf("\n"); }
void insert_queue(int vertex)
{     if(rear == MAX-
1)
       printf("Queue Overflow\n");
else     {         if(front == -1)
front = 0;         rear = rear+1;
       queue[rear] = vertex ;
   } } int
isEmpty_queue()
{
   if(front == -1 || front > rear)
       return 1;     else
return 0; }  int delete_queue()
{     int delete_item;     if(front
== -1 || front > rear)
   {
       printf("Queue Underflow\n");
exit(1);
   }
   delete_item = queue[front];
   front = front+1;
return delete_item;
}
void create_graph()
{
   int count,max_edge,origin,destin;

   printf("Enter number of vertices : ");
   scanf("%d",&n);
```

```
   max_edge = n*(n-1);

   for(count=1; count<=max_edge; count++)
   {
      printf("Enter edge %d( -1 -1 to quit ) : ",count);
      scanf("%d %d",&origin,&destin);

      if((origin == -1) && (destin == -1))
break;

      if(origin>=n || destin>=n || origin<0 || destin<0)
      {
         printf("Invalid edge!\n");
         count--;
      }
else
      {
         adj[origin][destin] = 1;
      }
   }
}
```



9. Implementation Prim's Algorithm

```
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999 #define
MAX 20
int G[MAX][MAX],spanning[MAX][MAX],n;
```

```
int prims();  int main() {     int
i,j,total_cost;     printf("Enter no.
of vertices:");     scanf("%d",&n);

   printf("\nEnter the adjacency matrix:\n");

   for(i=0;i<n;i++)
for(j=0;j<n;j++)
          scanf("%d",&G[i][j]);

   total_cost=prims();
   printf("\nspanning tree matrix:\n");

   for(i=0;i<n;i++)
   {         printf("\n");
for(j=0;j<n;j++)
          printf("%d\t",spanning[i][j]);
   }

   printf("\n\nTotal cost of spanning tree=%d",total_cost);
return 0;
}

int prims() {     int
cost[MAX][MAX];
   int u,v,min_distance,distance[MAX],from[MAX];
   int visited[MAX],no_of_edges,i,min_cost,j;

   //create cost[][] matrix,spanning[][]
for(i=0;i<n;i++)
       for(j=0;j<n;j++)
       {
if(G[i][j]==0)
cost[i][j]=infinity;
else
cost[i][j]=G[i][j];
spanning[i][j]=0;
       }

   //initialise visited[],distance[] and from[]
distance[0]=0;
   visited[0]=1;

   for(i=1;i<n;i++)
   {
```

```
      distance[i]=cost[0][i];
from[i]=0;        visited[i]=0;
   }

   min_cost=0;       //cost of spanning tree
   no_of_edges=n-1;       //no. of edges to be added

   while(no_of_edges>0)
   {
      //find the vertex at minimum distance from the tree
min_distance=infinity;        for(i=1;i<n;i++)
         if(visited[i]==0&&distance[i]<min_distance)
         {
            v=i;
            min_distance=distance[i];
         }

      u=from[v];

      //insert the edge in spanning tree
spanning[u][v]=distance[v];
spanning[v][u]=distance[v];       no_of_edges--;
      visited[v]=1;

      //updated the distance[] array
for(i=1;i<n;i++)
         if(visited[i]==0&&cost[i][v]<distance[i])
         {
            distance[i]=cost[i][v];
from[i]=v;
         }

      min_cost=min_cost+cost[u][v];
   }

   return(min_cost); }
```

O/ P

```
Enter no. of vertices:6

Enter the adjacency matrix:
```

0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

spanning tree matrix:

0 3 1 0 0 0
3 0 0 0 3 0
1 0 0 0 0 4
0 0 0 0 0 2
0 3 0 0 0 0
0 0 4 2 0 0

Total cost of spanning tree=13