1]Design develop and implement C program to **solve bubble sorting** using brute force technique

```
#include <stdio.h>
void bubbleSort(int arr[], int n) {
  for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         } } } }
void printArray(int arr[], int size) {
   for (int i = 0; i < size; i++)
      printf("%d ", arr[i]);
   printf("\n");  }
int main() {
   int arr[] = {64, 34, 25, 12, 22, 11, 90};
   int n = sizeof(arr) / sizeof(arr[0]);
printf("Original array: ");
   printArray(arr, n);
   bubbleSort(arr, n);
   printf("Sorted array: ");
   printArray(arr, n);
   return 0;  }
```

2] Design develop and implement C program to solve **selection sorting** using brute force technique

```
#include <stdio.h>
void selectionSort(int arr[], int n) {
   int i, j, minIndex, temp;
 for (i = 0; i < n - 1; i++) {
      minIndex = i;
      for (j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIndex]) {
            minIndex = j;   }}
      if (minIndex != i) {
         temp = arr[i];
         arr[i] = arr[minIndex];
         arr[minIndex] = temp;
      } }   }
void printArray(int arr[], int size) {
   for (int i = 0; i < size; i++)
      printf("%d ", arr[i]);
   printf("\n");   }
int main() {
   int arr[] = {29, 10, 14, 37, 13};
   int n = sizeof(arr) / sizeof(arr[0]);
   printf("Original array: ");
   printArray(arr, n);
  selectionSort(arr, n);
   printf("Sorted array: ");
```

```
printArray(arr, n);
   return 0; }
```

3]Design develop and implement C program to **solve string matching** using brute force technique

```
#include <stdio.h>
#include <string.h>
int main() {
   char text[100] = "hello world";
   char pattern[10] = "world";
   int i, j, found = 0;
 for (i = 0; i <= strlen(text) - strlen(pattern); i++) {
      for (j = 0; j < strlen(pattern); j++) {
         if (text[i + j] != pattern[j])
            break;   }
      if (j == strlen(pattern)) {
  printf("Pattern found at index %d\n", i);
         found = 1;
         break;
      } }
   if (!found)
      printf("Pattern not found.\n");
   return 0;  }
```

4]Design develop and implement C program to solve **binary search** using divide and conquer technique

```
#include <stdio.h>
int binarySearch(int arr[], int left, int right, int key) {
   if (left <= right) {
      int mid = (left + right) / 2;
      if (arr[mid] == key)
         return mid;
      else if (key < arr[mid])
         return binarySearch(arr, left, mid - 1, key);
      else
        return binarySearch(arr, mid + 1, right, key);
   }
   return -1;  }
int main() {
   int arr[] = {5, 10, 15, 20, 25, 30, 35};
   int n = sizeof(arr) / sizeof(arr[0]);
   int key = 25;
  int result = binarySearch(arr, 0, n - 1, key);
   if (result != -1)
      printf("Element found at index %d\n", result);
   else
      printf("Element not found\n");
   return 0;
}
```

5]Design develop and implement C program to solve **insertion sort** using divide and conquer technique

```
#include <stdio.h>
void insertionSort(int arr[], int n) {
   for (int i = 1; i < n; i++) {
      int key = arr[i];
      int j = i - 1;
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j--;   }
      arr[j + 1] = key;
   } }
void printArray(int arr[], int n) {
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");  }
int main() {
   int arr[] = {9, 5, 1, 4, 3};
   int n = sizeof(arr) / sizeof(arr[0]);
   printf("Original array: ");
   printArray(arr, n);
   insertionSort(arr, n);
   printf("Sorted array: ");
   printArray(arr, n);
   return 0;  }
```

6]Design, Develop and Implement C Programs to solve **Floyds method** using Dynamic Programming.

```
#include <stdio.h>
#define INF 999
void floydAlgo(int n, int D[10][10]) {
   int i, j, k;
   for (k = 0; k < n; k++) {
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
            if (D[i][k] + D[k][j] < D[i][j])
               D[i][j] = D[i][k] + D[k][j];
         } } }
   printf("\nAll Pairs Shortest Path Matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         if (D[i][j] == INF)
            printf("INF ");
         else
            printf("%d%d%d ", D[i][j]);
      }
      printf("\n");
   } }
int main() {
   int D[10][10], i, j, n, e, u, v, w;
```

```
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (i == j)
                D[i][j] = 0;
            else
                D[i][j] = INF;
    printf("Enter number of edges: ");
    scanf("%d", &e);
    for (i = 0; i < e; i++) {
        printf("Enter edge %d): ", i + 1);
        scanf("%d%d%d", &u, &v, &w);
        D[u][v] = w;    }
    printf("\nGiven Graph (Adjacency
Matrix):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (D[i][j] == INF)
                printf("INF ");
            else
                printf("%3d ", D[i][j]);  }
        printf("\n");    }
    floydAlgo(n, D);
    return 0;
}
```

Design, Develop and Implement C Programs to **solve Warshall's method** using Dynamic Programming.

```
#include <stdio.h>
int main() {
    int A[10][10], R[10][10];
    int i, j, k, n, e, u, v;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the number of edges: ");
    scanf("%d", &e);
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            A[i][j] = 0;
    printf("Enter the edges one by:\n");
    for (k = 0; k < e; k++) {
        printf("Edge %d : ", k + 1);
        scanf("%d%d", &u, &v);
        A[u][v] = 1;   }
    printf("\nThe digraph is\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
            printf("%d ", A[i][j]);
        printf("\n");   }
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            R[i][j] = A[i][j];
```

```
  for (k = 0; k < n; k++)
    for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
      R[i][j] = R[i][j] || (R[i][k] && R[k][j]);
  printf("\nThe transitive closure for the given digraph is\n");
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++)
      printf("%d\t", R[i][j]);
    printf("\n");    }
  return 0; }
```

PART B

1]Design, Develop and Implement C Programs to **solve Merge sort** using Divide and Conquer technique

```
#include <stdio.h>
void merge(int a[], int l, int m, int r) {
  int i = l, j = m + 1, k = 0, temp[50];
  while (i <= m && j <= r) {
    if (a[i] < a[j])
      temp[k++] = a[i++];
    else
      temp[k++] = a[j++]; }
  while (i <= m)
    temp[k++] = a[i++];
  while (j <= r)
    temp[k++] = a[j++];
  for (i = l, k = 0; i <= r; i++, k++)
    a[i] = temp[k];  }
void mergeSort(int a[], int l, int r) {
  if (l < r) {
    int m = (l + r) / 2;
    mergeSort(a, l, m);
    mergeSort(a, m + 1, r);
    merge(a, l, m, r);
  }  }
int main() {
  int a[50], n;
  printf("Enter number of elements: ");
  scanf("%d", &n);
  printf("Enter elements:\n");
  for (int i = 0; i < n; i++)
    scanf("%d", &a[i]);
  mergeSort(a, 0, n - 1);
  printf("Sorted array:\n");
  for (int i = 0; i < n; i++)
    printf("%d ", a[i]);
  return 0;  }
```

2]Design, Develop and Implement C Programs to **solve Quick sort** using Divide and Conquer technique.

```
#include <stdio.h>
int partition(int a[], int low, int high) {
   int pivot = a[low];
   int i = low + 1;
   int j = high;
   int temp;
   while (i <= j) {
 while (i <= high && a[i] <= pivot)
         i++;
 while (j >= low && a[j] > pivot)
         j--;
      if (i < j) {
         temp = a[i];
         a[i] = a[j];
         a[j] = temp;
      }  }
   temp = a[low];
   a[low] = a[j];
   a[j] = temp;
   return j; }
void quickSort(int a[], int low, int high) {
   if (low < high) {
      int pi = partition(a, low, high);
      quickSort(a, low, pi - 1);
      quickSort(a, pi + 1, high);
   } }
int main() {
   int a[50], n;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter elements:\n");
   for (int i = 0; i < n; i++)
      scanf("%d", &a[i]);
   quickSort(a, 0, n - 1);
   printf("Sorted array:\n");
   for (int i = 0; i < n; i++)
      printf("%d ", a[i]);
   return 0;  }
```

3] design, Develop and Implement C Programs to solve **Depth First Search (DFS)** using Decrease and Conquer technique.

```
#include <stdio.h>
int visited[10], a[10][10], n;
void DFS(int v) {
   visited[v] = 1;
   printf("%d ", v);
   for (int i = 0; i < n; i++) {
      if (a[v][i] && !visited[i]) {
         DFS(i);
      } } }
```

```
int main() {
   int i, j, start;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter adjacency matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         scanf("%d", &a[i][j]);
   printf("Enter starting vertex: ");
   scanf("%d", &start);
   printf("DFS traversal: ");
   DFS(start);
   return 0;  }
```

4] Design, Develop and Implement C Programs to **solve Breadth First Search (BFS)** using Decrease and Conquer technique.

```
#include <stdio.h>
int a[10][10], visited[10], queue[10];
int front = 0, rear = -1;
void BFS(int v, int n) {
   visited[v] = 1;
   queue[++rear] = v;
   while (front <= rear) {
      int current = queue[front++];
      printf("%d ", current);
      for (int i = 0; i < n; i++) {
         if (a[current][i] && !visited[i]) {
            visited[i] = 1;
            queue[++rear] = i;
         } } } }
int main() {
   int n, start;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter adjacency matrix:\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         scanf("%d", &a[i][j]);
   printf("Enter starting vertex: ");
   scanf("%d", &start);
   printf("BFS traversal: ");
   BFS(start, n);
   return 0; }
```

5] Design, Develop and Implement C Programs to **solve Topological Sorting** using Decrease and Conquer technique.

```
#include <stdio.h>
int a[10][10], visited[10], stack[10], top = -1;
int n;
void topoSort(int v) {
   visited[v] = 1;
```

```
    for (int i = 0; i < n; i++) {
        if (a[v][i] && !visited[i])
            topoSort(i);   }
    stack[++top] = v; }
int main() {
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &a[i][j]);
    for (int i = 0; i < n; i++)
        visited[i] = 0;
    for (int i = 0; i < n; i++) {
        if (!visited[i])
            topoSort(i);   }
    printf("Topological Order:\n");
    while (top >= 0)
        printf("%d ", stack[top--]);
    return 0;  }
```

6] Design, Develop and Implement C Programs to **solve Horspool's algorithm** using Space and time tradeoff technique.

```
#include <stdio.h>
#include <string.h>
#define MAX_CHAR 256
void preprocessPattern(char *pattern, int m, int shiftTable[MAX_CHAR]) {
    for (int i = 0; i < MAX_CHAR; i++) {
        shiftTable[i] = m;
    }
    for (int i = 0; i < m - 1; i++) {
        shiftTable[(int)pattern[i]] = m - i - 1;
    } }
int horspoolSearch(char *text, char *pattern) {
    int n = strlen(text); // Length of text
    int m = strlen(pattern); // Length of pattern
    int shiftTable[MAX_CHAR]; // Table for pattern shifts
    preprocessPattern(pattern, m, shiftTable);
    int i = m - 1
    while (i < n) {
        int j = m - 1;
        while (j >= 0 && text[i - (m - 1 - j)] == pattern[j]) {
            j--;  }
        if (j < 0) {
            return i - (m - 1);   }
        i += shiftTable[(int)text[i]];   }
    return -1;  }
```

```
int main() {
   char text[100], pattern[100];
   printf("Enter the text: ");
   fgets(text, sizeof(text), stdin);
   text[strcspn(text, "\n")] = 0;
   printf("Enter the pattern: ");
   fgets(pattern, sizeof(pattern), stdin);
   pattern[strcspn(pattern, "\n")] = 0;
   int result = horspoolSearch(text, pattern);
   if (result != -1) {
      printf("Pattern found at index %d\n",
result);
   } else {
      printf("Pattern not found\n");   }
   return 0; }
```

7] Design, Develop and Implement C Programs to **solve Heap Sorting** using Transform and Conquer technique.

```
#include <stdio.h>
void swap(int* a, int* b) {
   int temp = *a;
   *a = *b;
   *b = temp; }
void heapify(int arr[], int n, int i) {
   int largest = I;
   int left = 2 * i + 1;
   int right = 2 * i + 2;
   if (left < n && arr[left] > arr[largest]) {
      largest = left;  }
   if (right < n && arr[right] > arr[largest]) {
      largest = right;  }
   if (largest != i) {
      swap(&arr[i], &arr[largest]);
      heapify(arr, n, largest);    } }
void heapSort(int arr[], int n) {
   for (int i = n / 2 - 1; i >= 0; i--) {
      heapify(arr, n, i);  }
   for (int i = n - 1; i >= 1; i--) {
      heapify(arr, i, 0);   } }
void printArray(int arr[], int size) {
   for (int i = 0; i < size; i++) {
      printf("%d ", arr[i]);  }
   printf("\n"); }
int main() {
   int arr[] = {12, 11, 13, 5, 6, 7};
   int n = sizeof(arr) / sizeof(arr[0]);
   printf("Original array: \n");
   printArray(arr, n);
   heapSort(arr, n);
   printf("Sorted array: \n");
   printArray(arr, n);
   return 0; }
```