NAME: CHIRAG SATAPATHY

REG. No.: 19BEI0107

COURSE CODE: CSE2003

SLOT: L27 + L28

DATA STRUCTURES AND ALGORITHMS

DIGITAL ASSIGNMENT - 4

1. Menu driven C program to implement depth first search and breadth first search graph traversal algorithms.

PSEUDOCODE:

```
DFS(graph, start_node, end_node):
  frontier = new Stack()
  frontier.push(start_node)
  explored = new Set()  while frontier is not empty:
    current_node = frontier.pop()
    if current_node in explored: continue
    if current_node == end_node: return success

    for neighbor in graph.get_neigbhors(current_node):
      frontier.push(neighbor)     explored.add(current_node)

BFS(graph, start_node, end_node):
  frontier = new Queue()
  frontier.enqueue(start_node)
  explored = new Set()

  while frontier is not empty:
    current_node = frontier.dequeue()
    if current_node in explored: continue
    if current_node == end_node: return success

    for neighbor in graph.get_neigbhors(current_node):
      frontier.enqueue(neighbor)     explored.add(current_node)
```

CODE:

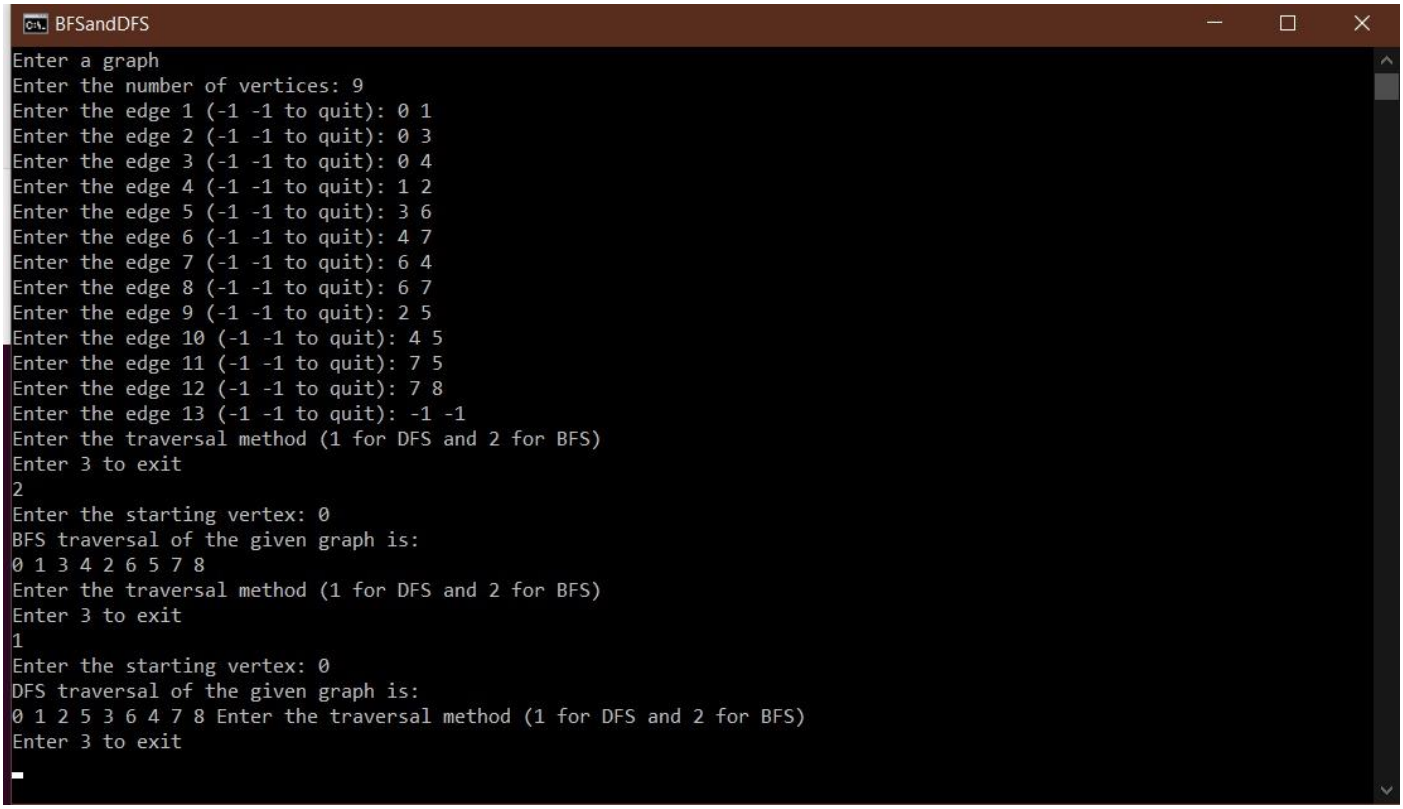
```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

int n;
int adj[MAX][MAX];
int state[MAX];
void createGraph();

void DF_Traversal();
void DFS(int v);

int queue[MAX], front = -1, rear = -1;
void enqueue(int vertex);
int dequeue();
bool isQueueEmpty();
void BF_Traversal();
void BFS(int v);

// -------------DFS--------------------

void DF_Traversal(){
   int v;
   for(int i = 0; i < n; ++i)
      state[i] = 0;  //initial state
   printf("Enter the starting vertex: ");
   scanf("%d", &v);
   printf("DFS traversal of the given graph is:\n");
   DFS(v);
}
```

```
void DFS(int v){
   state[v] = 1;    //visited
   printf("%d ", v);
   for(int i = 0; i < n; ++i){
      if(!state[i] && adj[v][i] == 1)
         DFS(i);
   }
}

// -------------DFS-------------------




// --------------BFS---------------------

void enqueue(int vertex){
   if(rear == MAX - 1)
      printf("Queue overflow!\n");
   else{
      if(front == -1)
         front++;
      queue[++rear] = vertex;
   }
}

int dequeue(){
   if(front > rear || front == -1){
      printf("Queue underflow!\n");
      exit(1);
   }
```

```
    else{
        int deleteItem = queue[front];
        front++;
        return deleteItem;
    }
}

bool isQueueEmpty(){
    if(front == -1 || front > rear)
        return true;
    return false;
}

void BF_Traversal(){
    int v;
    for(int i = 0; i < n; ++i)
        state[i] = 0;  //initial state
    printf("Enter the starting vertex: ");
    scanf("%d", &v);
    printf("BFS traversal of the given graph is:\n");
    BFS(v);
}

void BFS(int v){
    enqueue(v);
    state[v] = 1; //waiting state
    while(!isQueueEmpty()){
        v = dequeue();
        printf("%d ", v);
        state[v] = 2;     //visited
        for(int i = 0; i < n; i++){
```

```
            if(adj[v][i] == 1 && state[i] == 0){
                enqueue(i);
                state[i] = 1;
            }
        }
    }
    printf("\n");
}

// --------------BFS--------------------


// -------------Create Graph--------------

void createGraph(){
    int max, origin, dest;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    max = n * (n - 1);
    for(int i = 1; i <= max; ++i){
        printf("Enter the edge %d (-1 -1 to quit): ", i);
        scanf("%d %d", &origin, &dest);
        if(origin == -1 && dest == -1){
            break;
        }
        if(origin >= n || dest >= n || origin < 0 || dest < 0){
            printf("Invalid edge!\n");
            i--;
        }
        else{
            adj[origin][dest] = 1;
```

```
        }
    }
}

// -------------Create Graph----------

int main(){
    int choice;
    printf("Enter a graph\n");
    createGraph();
    while(1){
        printf("Enter the traversal method (1 for DFS and 2 for BFS)\n");
        printf("Enter 3 to exit\n");
        scanf("%d", &choice);
        if(choice == 1){
            DF_Traversal();
        }
        else if(choice == 2){
            BF_Traversal();
        }
        else{
            exit(1);
        }
    }

    return 0;
}
```



2. Implement Dijikstra's algorithm to find shortest path from source node to all other nodes.

PSEUDOCODE:

```
Let v1 be the origin vertex,
  and initialize W and ShortDist[u] as
   W := {v1}
   ShortDist[v1] :=0
   FOR each u in V - {v1}
      ShortDist[u] := T[v1,u]
  Now repeatedly enlarge W
   until W includes all verticies in V
   WHILE W <> V

       Find the vertex w in V - W at the minimum distance
         from v1
      MinDist := INFINITE
      FOR each v in V - W
        IF ShortDist[v] < MinDist
           MinDist = ShortDist[v]
           w := v
```

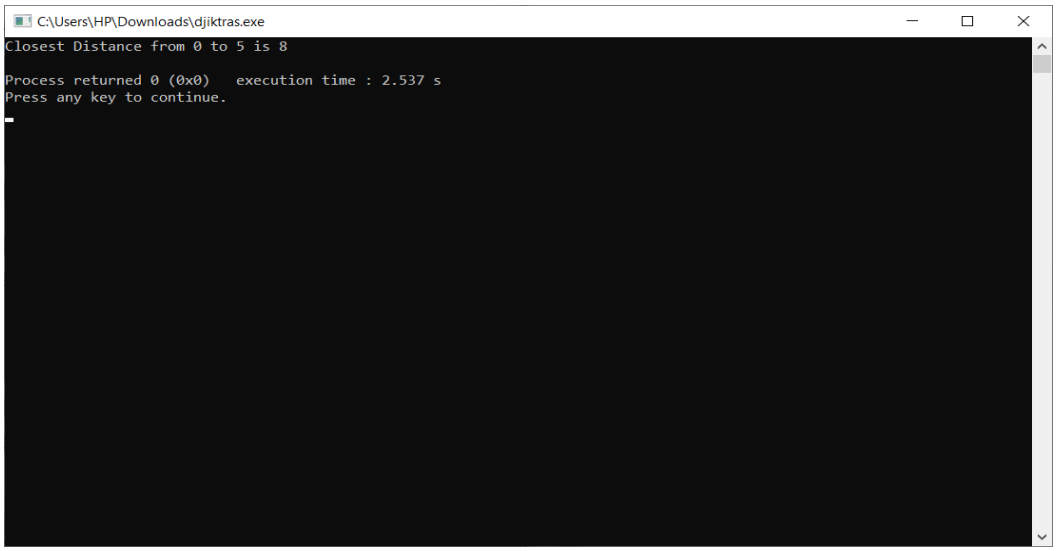
```
    END {if}
  END {for}

  Add w to W
  W := W U {w}

  Update the shortest distance to vertices in V - W
  FOR each u in V - W
    ShortDist[u] := Min(ShorDist[u],ShortDist[w] + T[w,u])
END {while}
```

CODE:

```
#include<stdio.h>

#include<stdlib.h>


#define MAX 100

#define INT_MAX 10000


int n;

int dist[MAX];

int sptSet[MAX];

int adj[MAX][MAX];



void createGraph(){

   int max, origin, dest, weight;

   printf("Enter the number of vertices: ");

   scanf("%d", &n);

   max = n * (n - 1);

   for(int i = 1; i <= max; ++i){

      printf("Enter the edge %d and it weight (-1 -1 to quit): ", i);

      scanf("%d %d %d", &origin, &dest, &weight);

      if(origin == -1 && dest == -1){

         break;

      }
```

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

void printShortestPath(){
    printf("Vertex \t\t Distance from source\n");
    for(int i = 0; i < n; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

int minDistance(){
    int min = INT_MAX, minIndex;
    for(int v = 0; v < n; v++){
        if(sptSet[v] == 0 && dist[v] < min){
            min = dist[v];
            minIndex = v;
        }
    }
    return minIndex;
}

void dijsktra(int n, int src){
    for(int v = 0; v < n; v++){
        dist[v] = INT_MAX;
        sptSet[v] = 0;
```

```
    }
    dist[src] = 0;
    for(int i = 0; i < n - 1; ++i){
        int u = minDistance();
        sptSet[u] = 1;
        for(int v = 0; v < n; v++){
            if(!sptSet[v] && adj[u][v] && dist[u] != INT_MAX && dist[u] + adj[u][v] < dist[v])
                dist[v] = dist[u] + adj[u][v];
        }
    }
    printShortestPath();
}

int main(){
    int src;
    createGraph();
    printf("Enter the source:\n");
    scanf("%d", &src);
    dijsktra(n, src);
    printShortestPath();
    return 0;
}
```



3. Menu driven C program to implement insertion, selection and bubble sort.

PSEUDOCODE:

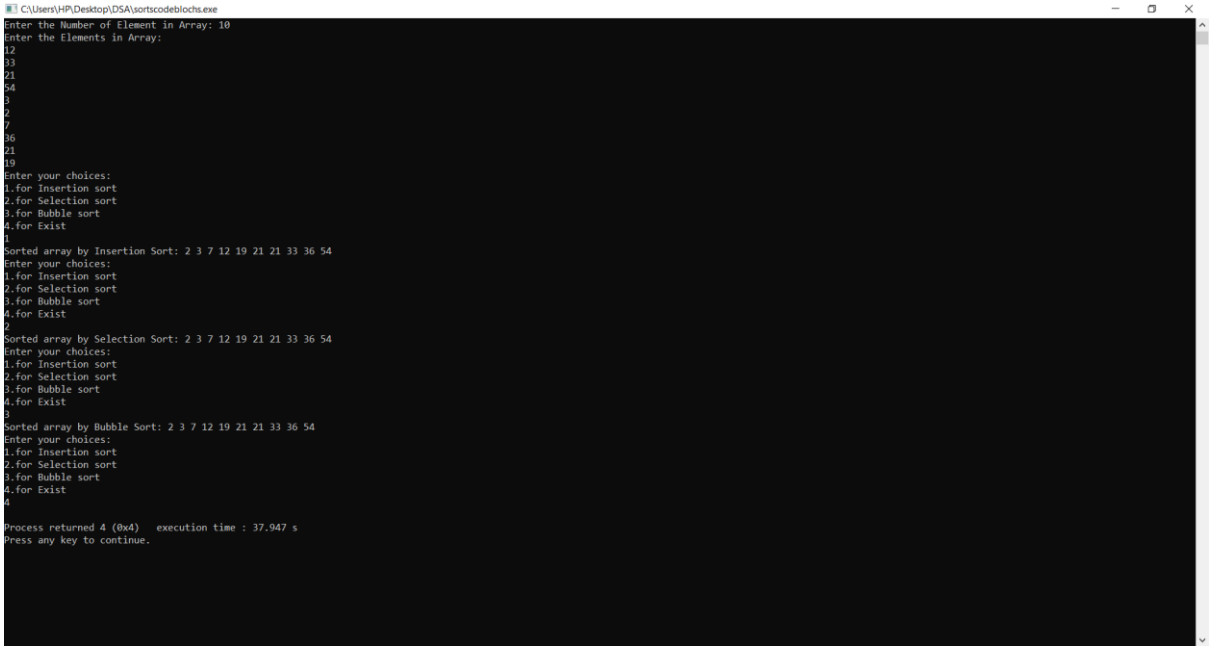
```
Selection sort()
For I = 0 to N-1 do:
       Smallsub = I
       For J = I + 1 to N-1 do:
         If A(J) < A(Smallsub)
           Smallsub = J
         End-If
       End-For
       Temp = A(I)
       A(I) = A(Smallsub)
       A(Smallsub) = Temp
     End-For
Insertion sort()
For I = 1 to N-1
       J = I
       Do while (J > 0) and (A(J) < A(J - 1)
         Temp = A(J)
         A(J) = A(J - 1)
         A(J - 1) = Temp
         J = J - 1
       End-Do
     End-For

Bubble sort()
  For I = 0 to N - 2
       For J = 0 to N - 2
         If (A(J) > A(J + 1)
           Temp = A(J)
           A(J) = A(J + 1)
           A(J + 1) = Temp
         End-If
       End-For
     End-For
```

CODE:

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

void swap(int *a, int *b){
  *a = *a + *b;
  *b = *a - *b;
  *a = *a - *b;
}
```

```
void selectionSort(int arr[], int n){
   int minIndex;
   for(int i = 0; i < n; ++i){
      minIndex = i;
      for(int j = i+1; j < n; ++j){
         if(arr[minIndex] > arr[j])
            minIndex = j;
      }
      swap(&arr[i], &arr[minIndex]);
   }
}

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] > arr[key]){
         arr[j+1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }
}

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])
            swap(&arr[j], &arr[j+1]);
      }
   }
}

void printArray(int arr[], int n){
   for(int i = 0; i < n; ++i)
      printf("%d ", arr[i]);
   printf("\n");
}

void sort(){
   int choice, n;
   int arr[50];
   printf("Enter the number of numbers you wish to enter: ");
   scanf("%d", &n);
   printf("Enter the numbers: \n");
   for(int i = 0; i < n; ++i)
      scanf("%d", &arr[i]);
   printf("Select your sorting method:\n");
   printf("1. Bubble sort\n");
   printf("2. Insertion Sort\n");
   printf("3. Bubble sort\n");
```

```
    scanf("%d", &choice);
    switch(choice){
        case 1: bubbleSort(arr, n);
                printArray(arr, n);
                break;
        case 2: insertionSort(arr, n);
                printArray(arr, n);
                break;
        case 3: selectionSort(arr, n);
                printArray(arr, n);
                break;
        default: printf("Invalid input!\n");
                break;
    }
}

int main(){
    int choice;
    while(1){
        printf("1. Sort numbers:\n");
        printf("2. Exit\n");
        scanf("%d", &choice);
        if(choice == 1)
            sort();
        else if(choice == 2)
            exit(0);
        else
            printf("Invalid input!\n");
    }


    return 0;
}
```



4. Menu driven C program to implement quick sort and merge sort using divide and conquer method.

PSEUDOCODE:

```
QuickSort(A, p, r):
        if p < r:
                q = Partition(A, p, r)
                QuickSort(A, p, q-1)
                QuickSort(A, q+1, r)
Merge(A,B,C):
        i = j = 1
        for k = 1 to n:
                if A[i] < B[j]:
                        C[k] = A[i]
                        i = i + 1
                else: (A[i] > B[j])
                        C[k] = B[j]
                        j = j + 1
```

CODE:

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

void swap(int *a, int *b){
   *a = *a + *b;
   *b = *a - *b;
   *a = *a - *b;
}

// ---------  Quick Sort ----------------

int partition(int arr[], int low, int high){
   int pivot = arr[high];
   int i = low - 1;
   for(int j = low; j < high; j++){
      if(arr[j] < pivot){
         i++;
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i+1], &arr[high]);
   return i + 1;
}
```

```
void quickSort(int arr[], int low, int high){
   if(low < high){
      int pi = partition(arr, low, high);
      quickSort(arr, low, pi - 1);
      quickSort(arr, pi + 1, high);
   }
}

// ---------- Quick Sort ----------

// ---------- Merge Sort ----------

void merge(int arr[], int l, int m, int r){
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;
   int L[n1], R[n2];
   for(i = 0; i < n1; ++i)
      L[i] = arr[l + i];
   for(j = 0; j < n2; ++j)
      R[j] = arr[m + 1 + j];

   i = 0; j = 0; k = l;
   while(i < n1 && j < n2){
      if(L[i] <= R[j]){
         arr[k] = L[i];
         i++;
      }
      else{
         arr[k] = R[j];
         j++;
      }
      k++;
   }

   while(i < n1){
      arr[k] = L[i];
      k++;
      i++;
   }
   while(j < n2){
      arr[k] = R[j];
      k++;
      j++;
   }
}

void mergeSort(int arr[], int l, int r){
   if(l < r){
      int m = l + (r-l)/2;
```

```
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}



// ---------- Merge Sort ----------



void printArray(int arr[], int n){
    printf("The sorted array is:\n");
    for(int i = 0; i < n; ++i)
        printf("%d ", arr[i]);
    printf("\n");
}

void sort(){
    int choice, n;
    int arr[50];
    printf("Enter the number of numbers you wish to enter: ");
    scanf("%d", &n);
    printf("Enter the numbers: \n");
    for(int i = 0; i < n; ++i)
        scanf("%d", &arr[i]);
    printf("Select your sorting method:\n");
    printf("1. Merge Sort\n");
    printf("2. Quick Sort\n");
    scanf("%d", &choice);
    switch(choice){
        case 1: mergeSort(arr, 0, n -1);
                printArray(arr, n);
                break;
        case 2: quickSort(arr, 0, n -1);
                printArray(arr, n);
                break;
        default: printf("Invalid input!\n");
                break;
    }
}

int main(){
    int choice;
    while(1){
        printf("1. Sort numbers:\n");
        printf("2. Exit\n");
        scanf("%d", &choice);
        if(choice == 1)
            sort();
```

```
    else if(choice == 2)
        exit(0);
    else
        printf("Invalid input!\n");
  }

  return 0;
}
```