Practical-6

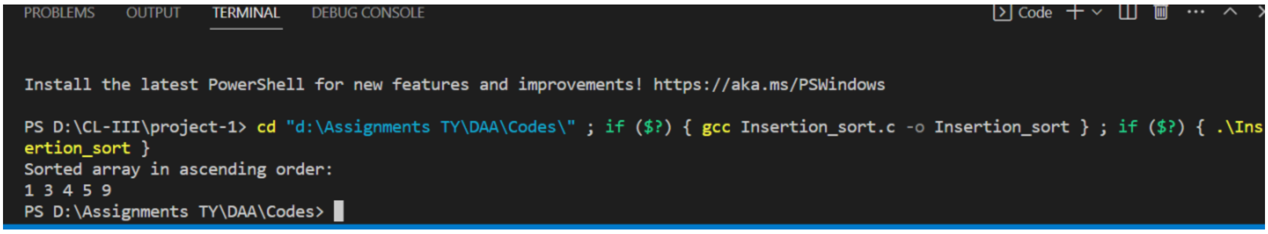
Name : shubham shivraj Suryawanshi
Reg. No: 2020BIT004

write a C/C++ program to implement Decrease and conquer algorithm
1) Insertion sort
2) DFS
3) BFS

1) Insertion Sort

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

Output:



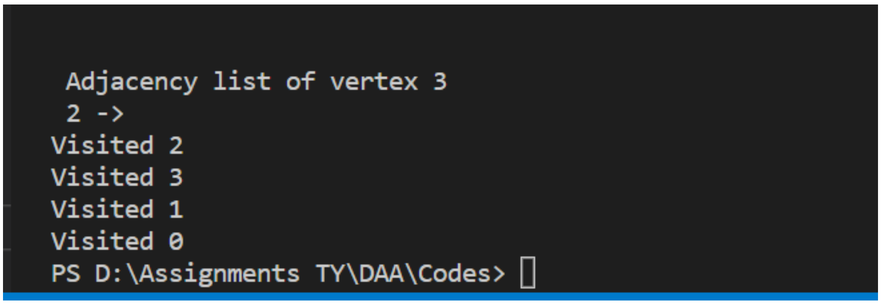
```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\CL-III\project-1> cd "d:\Assignments TY\DAA\Codes\" ; if ($?) { gcc Insertion_sort.c -o Insertion_sort } ; if ($?) { .\Ins
ertion_sort }
Sorted array in ascending order:
1 3 4 5 9
PS D:\Assignments TY\DAA\Codes>
```

2) DFS

```
#include <stdio.h>

#include <stdlib.h>

struct node {

  int vertex;

  struct node* next;

};

struct node* createNode(int v);

struct Graph {
  int numVertices;
  int* visited;
  struct node** adjLists;
};
void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;
  graph->visited[vertex] = 1;
  printf("Visited %d \n", vertex);
  while (temp != NULL) {
    int connectedVertex = temp->vertex;
    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
}
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}
void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;
```

```
 newNode = createNode(src);
 newNode->next = graph->adjLists[dest];
 graph->adjLists[dest] = newNode;
}
void printGraph(struct Graph* graph) {
 int v;
 for (v = 0; v < graph->numVertices; v++) {
  struct node* temp = graph->adjLists[v];
  printf("\n Adjacency list of vertex %d\n ", v);
  while (temp) {
   printf("%d -> ", temp->vertex);
   temp = temp->next;
  }
  printf("\n");
 }
}
int main() {
 struct Graph* graph = createGraph(4);
 addEdge(graph, 0, 1);
 addEdge(graph, 0, 2);
 addEdge(graph, 1, 2);
 addEdge(graph, 2, 3);
 printGraph(graph);

 DFS(graph, 2);

 return 0;
```

}**Output:**



```
 Adjacency list of vertex 3
 2 ->
Visited 2
Visited 3
Visited 1
Visited 0
PS D:\Assignments TY\DAA\Codes> 
```

3) BFS

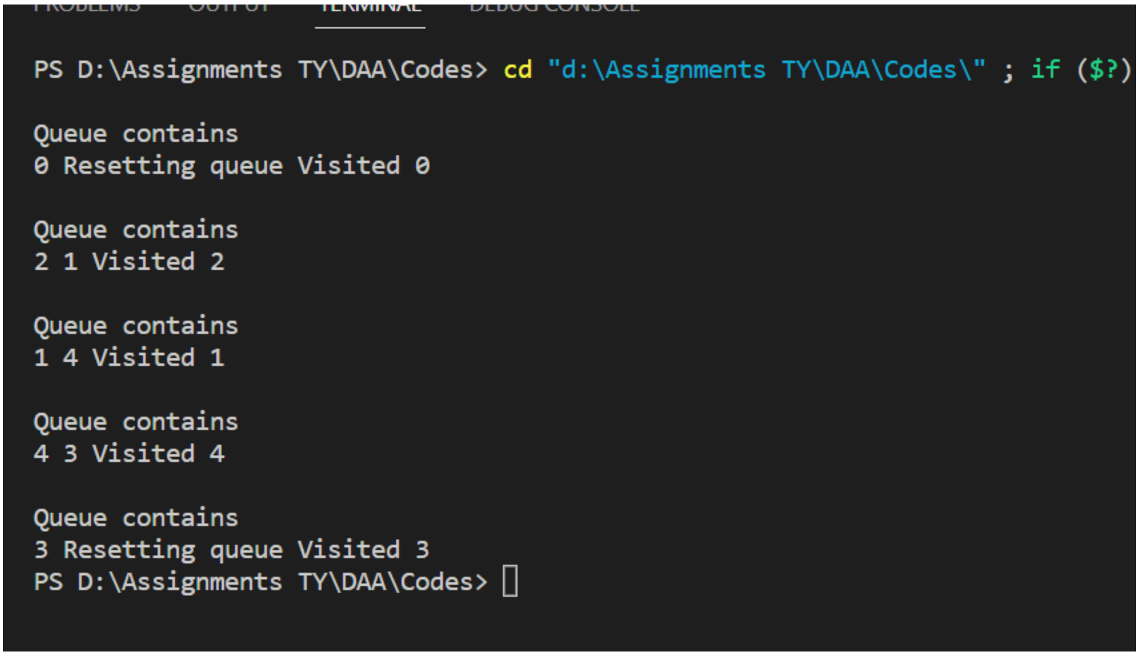
```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40

struct queue {
 int items[SIZE];
```

```
  int front;
  int rear;
};

struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);

struct node {
  int vertex;
  struct node* next;
};

struct node* createNode(int);

struct Graph {
  int numVertices;
  struct node** adjLists;
  int* visited;
};
void bfs(struct Graph* graph, int startVertex) {
  struct queue* q = createQueue();

  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);

  while (!isEmpty(q)) {
    printQueue(q);
    int currentVertex = dequeue(q);
    printf("Visited %d\n", currentVertex);

    struct node* temp = graph->adjLists[currentVertex];

    while (temp) {
      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {
        graph->visited[adjVertex] = 1;
        enqueue(q, adjVertex);
      }
      temp = temp->next;
    }
  }
}
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }

  return graph;
}
void addEdge(struct Graph* graph, int src, int dest) {
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}
struct queue* createQueue() {
  struct queue* q = malloc(sizeof(struct queue));
  q->front = -1;
  q->rear = -1;
  return q;
}
int isEmpty(struct queue* q) {
  if (q->rear == -1)
    return 1;
  else
    return 0;
}
void enqueue(struct queue* q, int value) {
  if (q->rear == SIZE - 1)
```

```
    printf("\nQueue is Full!!");
  else {
    if (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }
}
int dequeue(struct queue* q) {
  int item;
  if (isEmpty(q)) {
    printf("Queue is empty");
    item = -1;
  } else {
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
      printf("Resetting queue ");
      q->front = q->rear = -1;
    }
  }
  return item;
}
void printQueue(struct queue* q) {
  int i = q->front;

  if (isEmpty(q)) {
    printf("Queue is empty");
  } else {
    printf("\nQueue contains \n");
    for (i = q->front; i < q->rear + 1; i++) {
      printf("%d ", q->items[i]);
    }
  }
}

int main() {
  struct Graph* graph = createGraph(6);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 1, 4);
  addEdge(graph, 1, 3);
  addEdge(graph, 2, 4);
  addEdge(graph, 3, 4);
```

```
 bfs(graph, 0);

 return 0;
}
```

Output:-



```
PS D:\Assignments TY\DAA\Codes> cd "d:\Assignments TY\DAA\Codes\" ; if ($?)

Queue contains
0 Resetting queue Visited 0

Queue contains
2 1 Visited 2

Queue contains
1 4 Visited 1

Queue contains
4 3 Visited 4

Queue contains
3 Resetting queue Visited 3
PS D:\Assignments TY\DAA\Codes> 
```