DSA DIGITAL ASSESSMENT – 4

Done By: Sparsh Arya

Registration number: 17bec0656

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

PseudoCode:

```
void DF_Traversal()
        for i=0 to n:
              state[i] = 0        //initial state
        input the starting vertex  v
        print("DFS traversal of the given graph is:")
        DFS(v)


void DFS(int v)
        state[v] = 1     //visited
        print v
        for i=0 to n:
                if(!state[i] && adj[v][i] == 1)
                        DFS(i)


void enqueue(int vertex) if(rear
        == MAX - 1)
              print("Queue overflow!")
         else
```

```
        if(front == -1)
            front+=1
        queue[++rear] = vertex
int dequeue()
    if(front > rear || front == -1)
            print("Queue underflow!")
            exit(1)
    else
            int deleteItem = queue[front] front+=1
            return deleteItem
bool isQueueEmpty()
    if(front == -1 || front > rear)
            return true;
    return false;

void BF_Traversal()
    for(int i = 0; i < n; ++i)
            state[i] = 0; //initial state input
    the starting vertex v
    print("BFS traversal of the given graph is:")
    BFS(v)

Void BFS(int v)
    Enqueue(v)
    State[v] = 1            // waiting
    While( !isQueueEmpty())
```

```
        V = dequeue()
        Print v
        State[v] = 2       // visited
        For i in 0 to n
            If adj[i][v] == 1 and state[i] == 0
                Enqueue()
                State[i] = 1

Void createGraph()
    Input the number of vertexes n
    Max = n *  (n-1)
    For i = 1 to m  {
        Input origin and dest
        If origin == -1 and dest == -1
            Break
        If origin >= n or dest >= n or origin < 0 or dest < 0
            Print "Invalid edge"
        Else
            Adj[origin][dest] = 1


Int main()
    Int choice
    createGraph()
    while(1)
        input traversal method choice if
        choice == 1
```

```
            DF_Traversal() Else
        if choice == 2
            BF_Traversal()
         Else
            Exit()
Return 0
```

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;
}
```

OUTPUT:

```
Enter a graph
Enter the number of vertices: 9
Enter the edge 1 (-1 -1 to quit): 0 1
Enter the edge 2 (-1 -1 to quit): 0 3
Enter the edge 3 (-1 -1 to quit): 0 4
Enter the edge 4 (-1 -1 to quit): 1 2
Enter the edge 5 (-1 -1 to quit): 3 6
Enter the edge 6 (-1 -1 to quit): 4 7
Enter the edge 7 (-1 -1 to quit): 6 4
Enter the edge 8 (-1 -1 to quit): 6 7
Enter the edge 9 (-1 -1 to quit): 2 5
Enter the edge 10 (-1 -1 to quit): 4 5
Enter the edge 11 (-1 -1 to quit): 7 5
Enter the edge 12 (-1 -1 to quit): 7 8
Enter the edge 13 (-1 -1 to quit): -1 -1
Enter the traversal method (1 for DFS and 2 for BFS)
Enter 3 to exit
2
Enter the starting vertex: 0
BFS traversal of the given graph is:
0 1 3 4 2 6 5 7 8
Enter the traversal method (1 for DFS and 2 for BFS)
Enter 3 to exit
1
Enter the starting vertex: 0
DFS traversal of the given graph is:
0 1 2 5 3 6 4 7 8 Enter the traversal method (1 for DFS and 2 for BFS)
Enter 3 to exit
```

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

Pseudocode:

```
#define MAX 100

int n

int dist[MAX] int

sptSet[MAX]

int adj[MAX][MAX] void

createGraph(){

    int max, origin, dest, weight print("Enter

    the number of vertices:") input n

    max = n * (n - 1)

    for i=1 to max{
```

```
        print("Enter the edge %d and it weight (-1 -1 to quit): ", i) input
        origin, dest,  weight
        if(origin == -1 && dest == -1){
            break
        }
        if(origin >= n || dest >= n || origin < 0 || dest < 0){
            print("Invalid edge!\n")
            i--;
        }
        else{
            adj[origin][dest] = weight
        }
    }
}

void printShortestPath(){
    print("Vertex \t\t Distance from source\n") for i =
    0 to n; i++
        print(i,  dist[i])
}

int minDistance(){
    int min = INT_MAX, minIndex for
    v = 0 to n:{
        if(sptSet[v] == 0 && dist[v] < min){ min =
            dist[v]
            minIndex = v
```

```
        }
    }
    return minIndex
}

void dijsktra(int n, int src){ for
    v = 0 to n{
        dist[v] = INT_MAX
        sptSet[v] = 0
    }
    dist[src] = 0 for
    i = 0 to n:{
        int u = minDistance()
        sptSet[u] = 1
        for v = 0 to n{
            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")
```

```
    input src dijsktra(n,
    src)
    printShortestPath()
    return 0
}
```

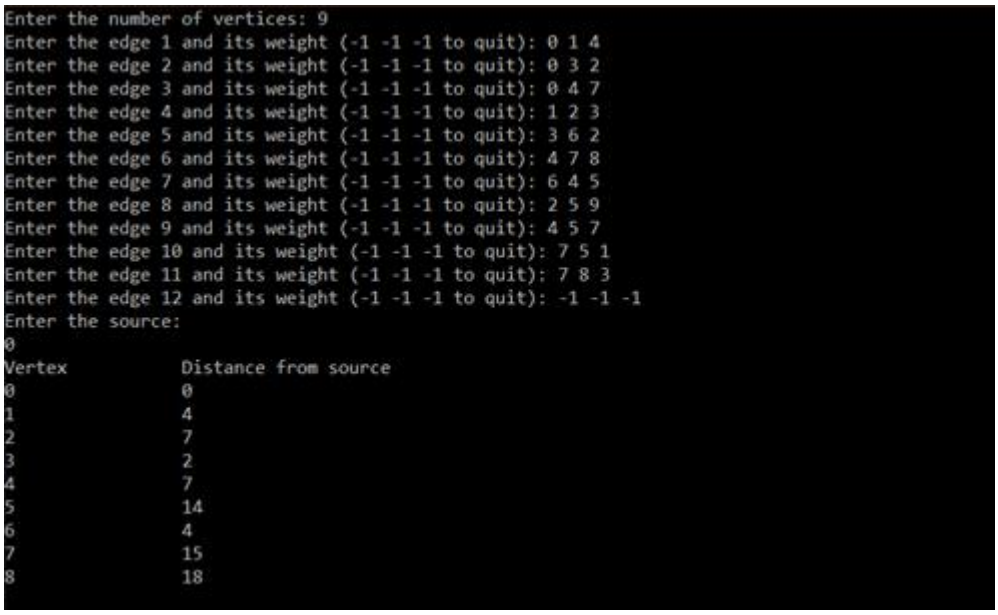
Code:

```
#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;
```

OUTPUT:



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

Pseudocode:

void **swap**(int *a, int *b){

*a = *a + *b

*b = *a - *b

*a = *a - *b

}

void **selectionSort**(int arr[], int n)

{

int minIndex

```
    for i=0 to n{
        minIndex = i for
        j = i+1 to n{
            if(arr[minIndex] > arr[j])
                minIndex = j
        }
        swap(&arr[i], &arr[minIndex])
    }
}

void insertionSort(int arr[], int n)
{
    For i = 1 to n:
        key = arr[i] 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
    i=0 to n-1:{
        for j = 0 to n-i-1:{ if(arr[j]
            > arr[j+1])
```

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

void printArray(int arr[], int n)
{
    for i = 0 to n:{
        print(arr[i])
    print("\n")
}

void sort(){
    int choice, n int
    arr[50]
    print("Enter the number of numbers you wish to enter: ") input
    number of numbers n
    printf("Enter the numbers: \n") for(int
    i = 0; i < n; ++i)
        input arr[i]
    print("Select your sorting method:\n")
    print("1. Bubble sort\n")
    print("2. Insertion Sort\n")
    print("3. Selection sort\n")
    input 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: print("Invalid input!\n") break
    }
}

int main(){ int
    choice;
    while(1){
        print("1. Sort numbers:\n")
        print("2. Exit\n")
        input choice
        if(choice == 1)
            sort()
        else if(choice == 2)
            exit(0)
        else
            print("Invalid input!\n")
    }
```

```
    return 0;
}
```

Code:

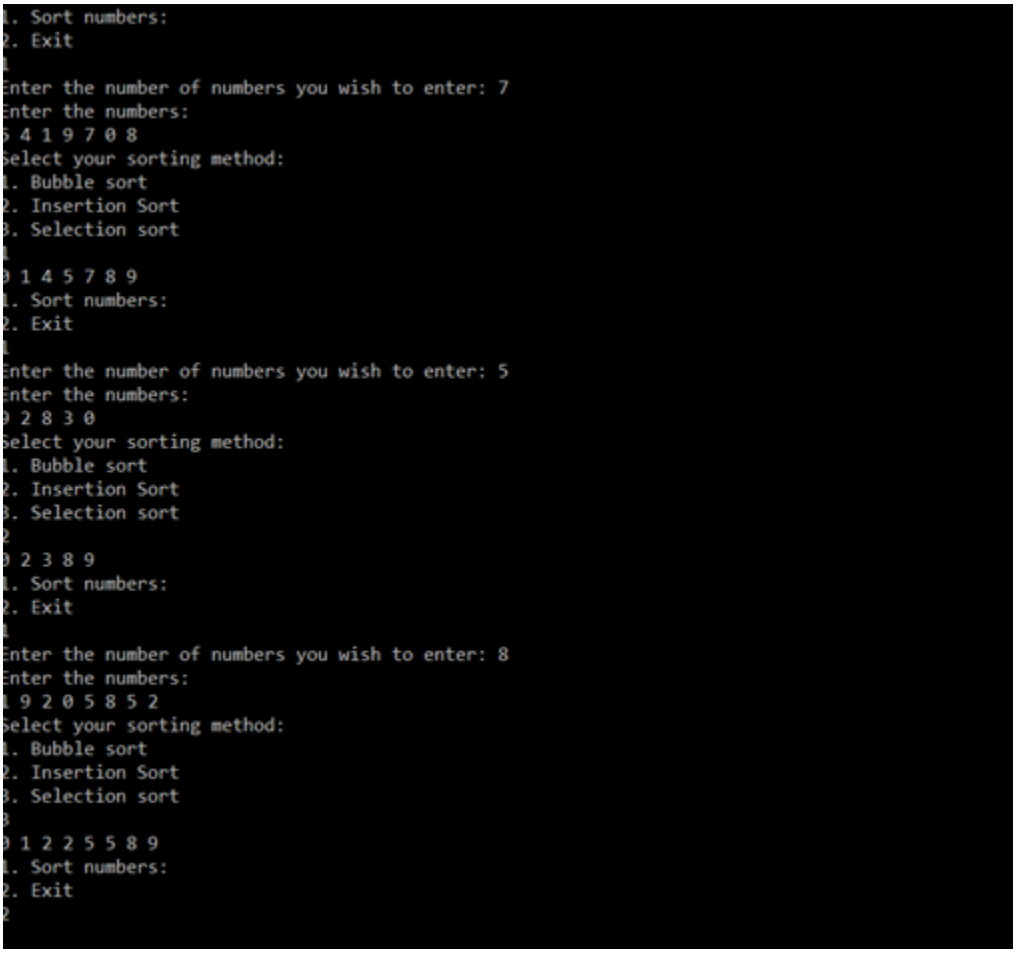
```
#include<stdio.h>
#include<stdlib.h>

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

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

void insertionSort(int arr[], int n) { int i, key, j;
```

```
    for (i = 1; i < n; i++) { key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) { arr[j + 1] =
            arr[j];
            j = j - 1;
        }
        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.
    Selection 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;
}
```

OUTPUT:



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

Pseudocode:

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 =low to high:{
        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)
    }
}
```

Merge Sort

```
void merge(int arr[], int l, int m, int r){ int
    n1 = m - l + 1
    int n2 = r - m int
    L[n1], R[n2]
```

```
for i=0 to m:
    L[i] = arr[l + i]
For j = 0 to n2:
    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)
    }
}

void printArray(int arr[], int n){
    print("The sorted array is:\n") for
    i=0 to n:
        print arr[i]
    print("\n")
}

void sort(){
    int choice, n int
    arr[50]
    printf("Enter the number of numbers you wish to enter: "); input n
    printf("Enter the numbers: \n"); for
    i=0 to n:
        input arr[i]
```

```
    print("Select your sorting method:\n")
    print("1. Merge Sort\n")
    print("2. Quick Sort\n")
    input 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: print("Invalid input!\n") break;
    }
}

int main(){ int
    choice
    while(1){
        print("1. Sort numbers:\n")
        print("2. Exit\n")
        input choice
        if(choice == 1)
            sort()
        else if(choice == 2)
            exit(0)
        else
```

```
            print("Invalid input!\n")
    }
    return 0
}
```

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; int j;
        for(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){ int i = 0;
        printf("The sorted array is:\n"); for(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"); int i = 0;
```

```
        for(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;
}
```

OUTPUT: