Practical No 10

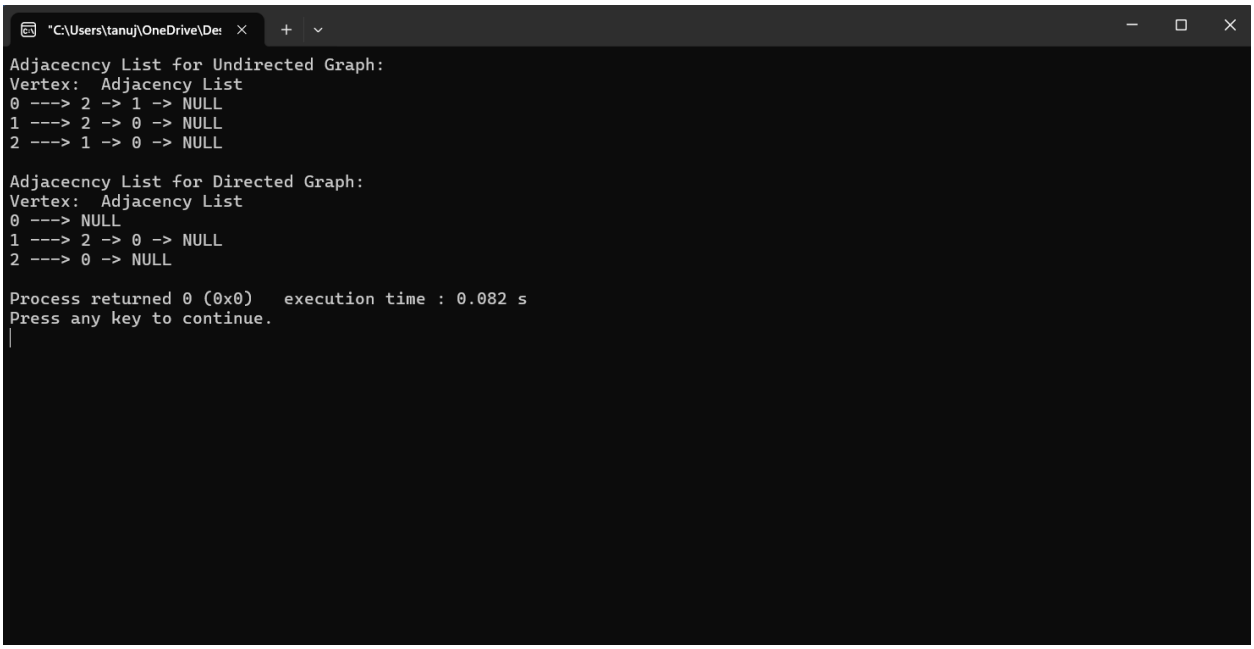
1.write a program to create n vertices using adjacency list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int vertex;
   struct Node* next;
};
struct Graph {
   int numVertices;
   struct Node** adjLists;
   int isDirected;
};
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, int isDirected) {
   struct Graph* graph = malloc(sizeof(struct Graph));
   graph->numVertices = vertices;
   graph->isDirected = isDirected;
   graph->adjLists = malloc(vertices * sizeof(struct Node*));
   for (int i = 0; i < vertices; i++) {
      graph->adjLists[i] = NULL;
   }
```

```
    return graph;
}
void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    if (!graph->isDirected) {
        newNode = createNode(src);
        newNode->next = graph->adjLists[dest];
        graph->adjLists[dest] = newNode;
    }
}
void printGraph(struct Graph* graph) {
    printf("Vertex:  Adjacency List\n");
    for (int v = 0; v < graph->numVertices; v++) {
        struct Node* temp = graph->adjLists[v];
        printf("%d --->", v);
        while (temp) {
            printf(" %d ->", temp->vertex);
            temp = temp->next;
        }
        printf(" NULL\n");
    }
}

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

```
    printf("Adjacecncy List for Undirected Graph:\n");
    printGraph(undirectedGraph);
    struct Graph* directedGraph = createGraph(3, 1);
    addEdge(directedGraph, 1, 0);
    addEdge(directedGraph, 1, 2);
    addEdge(directedGraph, 2, 0);
    printf("\nAdjacecncy List for Directed Graph:\n");
    printGraph(directedGraph);
    return 0;
}
```



Practical No 11

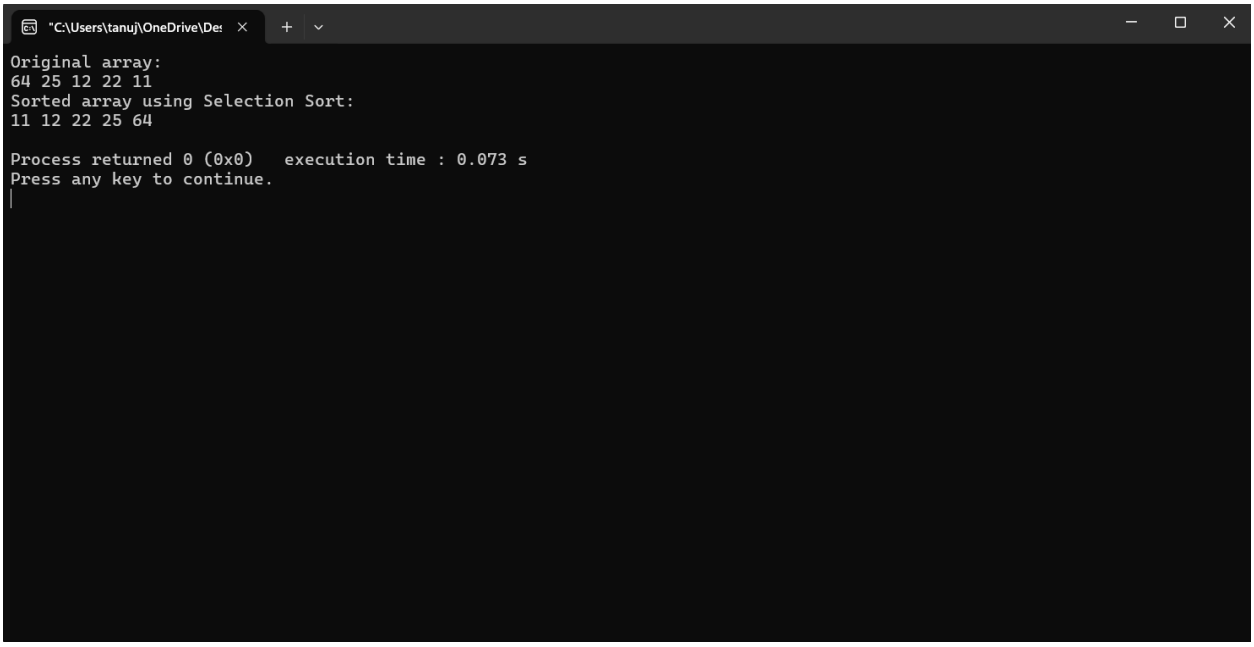
1.wite a program for selection sort ,bubble sort ,insertion sort.

```
#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;
         }
      }
      // Swap the found minimum element with the first element
      temp = arr[minIndex];
      arr[minIndex] = arr[i];
      arr[i] = temp;
   }
}
void printArray(int arr[], int n) {
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}

int main() {
```

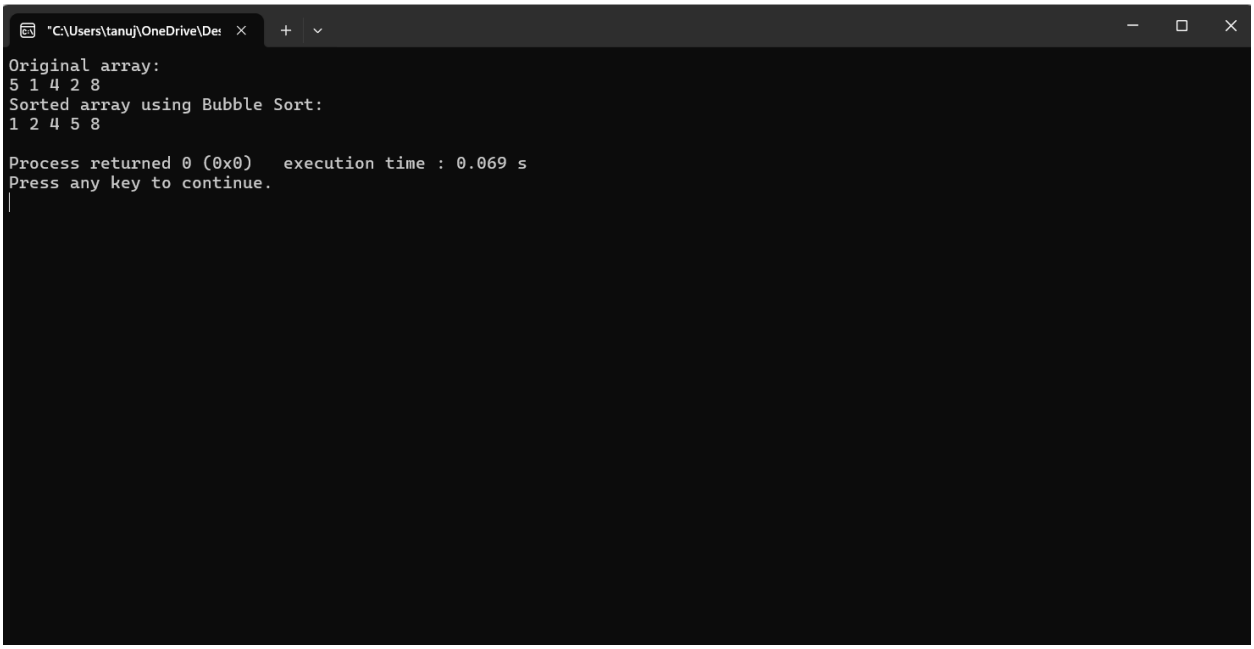
```
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: \n");
    printArray(arr, n);
    selectionSort(arr, n);
    printf("Sorted array using Selection Sort: \n");
    printArray(arr, n);

    return 0;
}
```




```
#include <stdio.h>
void bubbleSort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
```

```
            // Swap arr[j] and arr[j+1]
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
      }
   }
}
void printArray(int arr[], int n) {
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}
int main() {
   int arr[] = {5, 1, 4, 2, 8};
   int n = sizeof(arr) / sizeof(arr[0]);
   printf("Original array: \n");
   printArray(arr, n);
   bubbleSort(arr, n);
   printf("Sorted array using Bubble Sort: \n");
   printArray(arr, n);
   return 0;
}
```

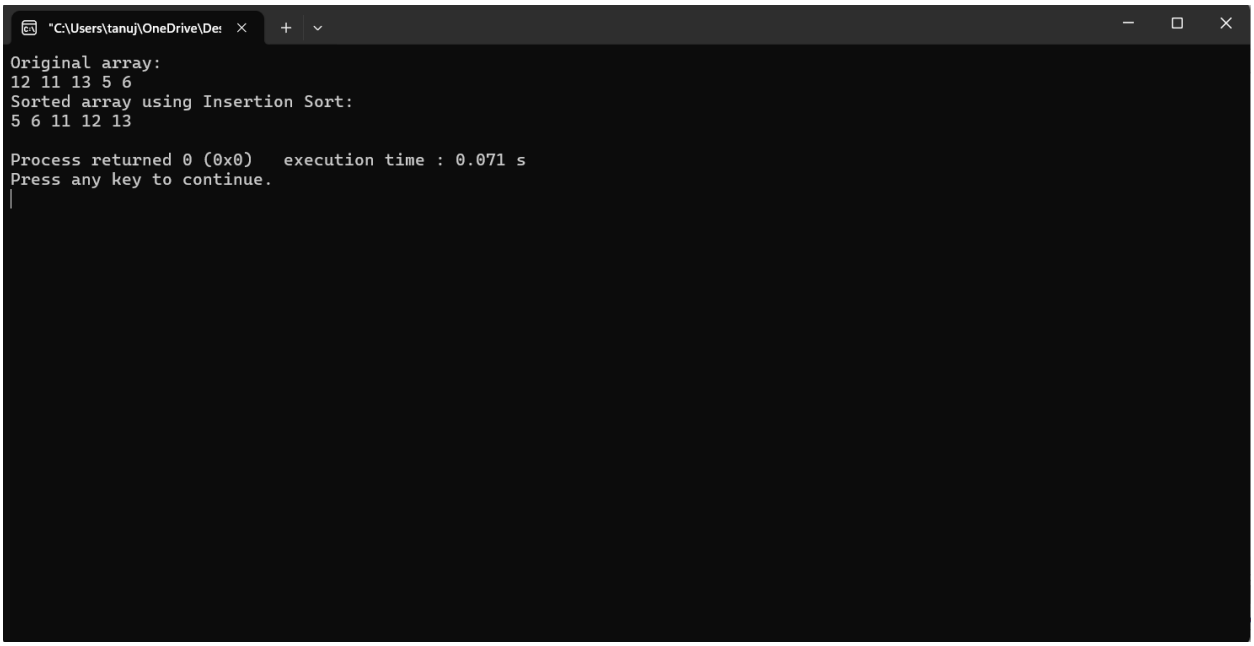



```
#include <stdio.h>

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

```
    printf("\n");
}
int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: \n");
    printArray(arr, n);
    insertionSort(arr, n);
    printf("Sorted array using Insertion Sort: \n");
    printArray(arr, n);
    return 0;
}
```



Practical No 12

1.write program for search an element using hashing technique

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10
int hashTable[TABLE_SIZE];
void initHashTable() {
   for (int i = 0; i < TABLE_SIZE; i++) {
      hashTable[i] = -1;
   }
}
int hashFunction(int key) {
   return key % TABLE_SIZE;
}
void insert(int key) {
   int index = hashFunction(key);
   while (hashTable[index] != -1) {
      index = (index + 1) % TABLE_SIZE;
   }
   hashTable[index] = key;
}
int search(int key) {
   int index = hashFunction(key);
   int startIndex = index;
   while (hashTable[index] != -1) {
      if (hashTable[index] == key)
         return index;
```

```
        index = (index + 1) % TABLE_SIZE;
        if (index == startIndex)
            break;
    }
    return -1;
}
void display() {
    printf("Hash Table:\n");
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (hashTable[i] != -1)
            printf("Index %d: %d\n", i, hashTable[i]);
        else
            printf("Index %d: ~\n", i);
    }
}
int main() {
    initHashTable();
    insert(23);
    insert(43);
    insert(13);
    insert(27);
    display();
    int key;
    printf("Enter the element to search: ");
    scanf("%d", &key);
    int result = search(key);
    if (result != -1)
```

```
        printf("Element %d found at index %d\n", key, result);
    else
        printf("Element %d not found in the hash table\n", key);
    return 0;
}
```