VIDHYADEEP UNIVERSITY

Holy Flame Of Knowledge

VIDHYADEEP UNIVERSITY INSTITUTE OF B.Sc. IT & BCA

NAME :-

SUBJECT :-	**ENROLLMENT :-**
SUBMIT DATE :-	**DEPARTMENT :-**

SR NO	PROBLEMS	DATE	SIGN
1	Below is the implementation of Graph Data Structure represented using Adjacency Matrix.		
2	Below is the implementation of Graph Data Structure represented using Adjacency List.		
3	Implementation of the Linear search algorithm.		
4	Implement iterative Binary Search.		
5	Implementation of selection sort.		
6	Implementation of Bubble sort.		
7	Implementation of Insertion Sort.		
8	Program for Merge Sort.		
9	Program for Quick Sort.		

❖ PROGRAM 1: Implementation of Graph Data Structure represented using Adjacency Matrix.

```
#include<stdio.h>
#define V 4

void addEdge(int mat[V][V], int i, int j) {
   mat[i][j] = 1;
   mat[j][i] = 1; // Since the graph is undirected
}

void displayMatrix(int mat[V][V]) {
          int i,j;
   for ( i = 0; i < V; i++) {
      for ( j = 0; j < V; j++)
         printf("%d ", mat[i][j]);
      printf("\n");
   }
}

int main() {
   // Create a graph with 4 vertices and no edges
   // Note that all values are initialized as 0
   int mat[V][V] = {0};

   // Now add edges one by one
   addEdge(mat, 0, 1);
   addEdge(mat, 0, 2);
   addEdge(mat, 1, 2);
   addEdge(mat, 2, 3);

   /* Alternatively, we can also create using the below
     code if we know all edges in advance

   int mat[V][V] = { {0, 1, 0, 0}, {1, 0, 1, 0}, {0, 1, 0, 1}, {0, 0, 1, 0} };        */

   printf("Adjacency Matrix Representation\n");
   displayMatrix(mat);

   return 0;
}
```

➢ Output:

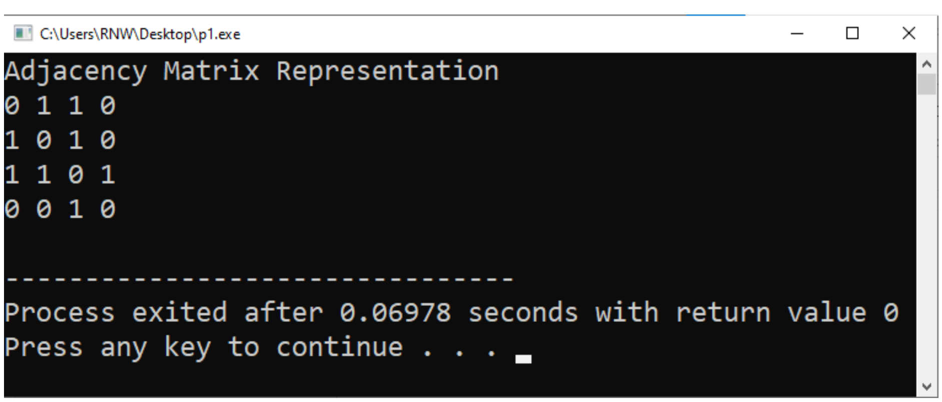
C:\Users\RNW\Desktop\p1.exe

```
Adjacency Matrix Representation
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0

--------------------------------
Process exited after 0.07382 seconds with return value 0
Press any key to continue . . .
```

❖ PROGRAM 2: Implementation of Graph Data Structure represented using Adjacency List.

```
#include<stdio.h>
#define V 4

void addEdge(int mat[V][V], int i, int j) {
   mat[i][j] = 1;
   mat[j][i] = 1; // Since the graph is undirected
}

void displayMatrix(int mat[V][V]) {
          int i,j;
   for (i = 0; i < V; i++) {
      for (j = 0; j < V; j++)
         printf("%d ", mat[i][j]);
      printf("\n");                              }
}

int main() {
   // Create a graph with 4 vertices and no edges
   // Note that all values are initialized as 0
   int mat[V][V] = {0};

   // Now add edges one by one
   addEdge(mat, 0, 1);
   addEdge(mat, 0, 2);
   addEdge(mat, 1, 2);
   addEdge(mat, 2, 3);

   /* Alternatively, we can also create using the below
     code if we know all edges in advance

   int mat[V][V] = { {0, 1, 0, 0}, {1, 0, 1, 0}, {0, 1, 0, 1}, {0, 0, 1, 0} };              */

   printf("Adjacency Matrix Representation\n");
   displayMatrix(mat);

   return 0;
}
```

➢ **Output:**



❖ PROGRAM 3: Implementation of Linear Searching Algorithm.

Algorithm:

STEP-1 : Start traversing from the start of the dataset.

STEP-2 : Compare the current element with the **key (element to be searched).**

STEP-3 : If the element is equal to the **key,** return index.

STEP-4 : Else, increment the index and repeat the step 2 and 3.

STEP-5 : If we reach the **end of the list** without finding the element equal to the key, return some value to represent that the **element is not found.**

PROGRAM:

```
// C program to implement linear search using loop
#include <stdio.h>
int linearSearch(int* arr, int n, int key) {
        int i;                    // Starting the loop and looking for the key in arr
        for (i = 0; i < n; i++) {   // If key is found, return key
    if (arr[i] == key) {
        return i;
    }
  }
  // If key is not found, return some value to indicate end
  return -1;
}
int main() {
  int arr[] = { 10, 50, 30, 70, 80, 60, 20, 90, 40 };
  int n = sizeof(arr) / sizeof(arr[0]);
  int key = 30;

  // Calling linearSearch() for arr with key = 43
  int i = linearSearch(arr, n, key);

  // printing result based on value returned by linearSearch()
  if (i == -1){
    printf("Key Not Found"); }
  else{
    printf("Key Found at Index: %d", i); }

  return 0;
}
```

➢ **Output:**

```
C:\Users\RNW\Desktop\p1.exe
Key Found at Index: 2
--------------------------------
Process exited after 0.05888 seconds with return value 0
Press any key to continue . . .
```

❖ PROGRAM 4: Implement iterative Binary Search.

Algorithm:

Step 1: Find the middle element of array. Using,

middle = initial_value + end_value / 2;

Step 2: If middle = element, return ‘element found’ and index.

Step 3: if middle > element, call the function with end_value = middle - 1.

Step 4: if middle < element, call the function with start_value = middle + 1.

Step 5: exit.

PROGRAM:

```
#include <stdio.h>

int iterativeBinarySearch(int array[], int start_index, int end_index, int element){
        while (start_index <= end_index){
          int middle = start_index + (end_index- start_index )/2;
          if (array[middle] == element)
            return middle;
          if (array[middle] < element)
            start_index = middle + 1;
          else
            end_index = middle - 1;
        }
  return -1;
}

int main(void){
       int array[] = {1, 4, 7, 9, 16, 56, 70};
       int n = 7;
       int element = 16;
       int found_index = iterativeBinarySearch(array, 0, n-1, element);
       if(found_index == -1 ) {
         printf("Element not found in the array ");
       }
       else {
         printf("Element found at index : %d",found_index);
       }
  return 0;
}
```

➢ **Output:**

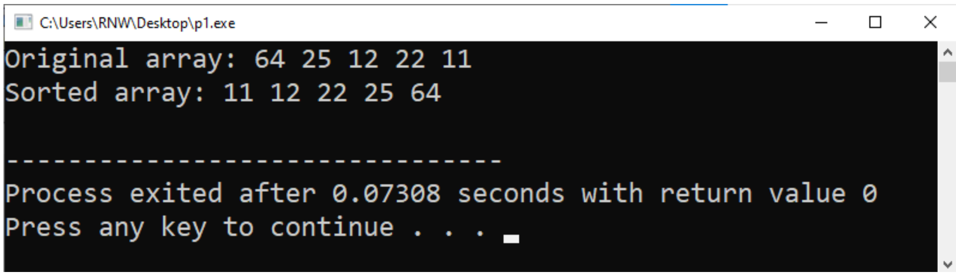
```
C:\Users\RNW\Desktop\p1.exe
Element found at index : 4
--------------------------------
Process exited after 0.05709 seconds with return value 0
Press any key to continue . . .
```

❖ **PROGRAM 5: Implementation of Selection sort.**

```
// C program for implementation of selection sort
#include <stdio.h>

void selectionSort(int arr[], int n) {
            int i,j;
    for (i = 0; i < n - 1; i++) {          // Assume the current position holds the minimum element
        int min_idx = I;
        // Iterate through the unsorted portion to find the actual minimum
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {                        // Update min_idx if a smaller element is found
                min_idx = j;
            }
        }

        // Move minimum element to its correct position
        int temp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = temp;
    }
}

void printArray(int arr[], int n) {
            int i;
    for (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: ");
    printArray(arr, n);

    selectionSort(arr, n);

    printf("Sorted array: ");
```

```
    printArray(arr, n);

    return 0;
}
```

- **Output:**



❖ PROGRAM 6: Implementation of Bubble sort.

Algorithm:

Step 1: Compare and swap the adjacent elements if they are in the wrong order starting from the first two elements.
Step 2: Do that for all elements moving from left to right. We will get the largest element at the right end.
Step 3: Start compare and swap again from the start but this time, skip the last element as its already at correct position.
Step 4: The second last element will be moved at the right end just before the last element.
Step 5: Repeat the above steps till all the elements are sorted.

PROGRAM:

```
#include <stdio.h>
void swap(int* arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
void bubbleSort(int arr[], int n) {
            int i,j;
    for (i = 0; i < n - 1; i++) {       // Last i elements are already in place, so the loop will only num n - i - 1 times
               for (j = 0; j < n - i - 1; j++) {
           if (arr[j] > arr[j + 1])
               swap(arr, j, j + 1);       }
        }
}
int main() {
    int arr[] = { 6, 10, 3, 5 };
    int n = sizeof(arr) / sizeof(arr[0]), i;
    bubbleSort(arr, n);     // Calling bubble sort on array arr
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);           }
    return 0;
}
```

➢ **Output:**

```
C:\Users\RNW\Desktop\p1.exe
3 5 6 10
--------------------------------
Process exited after 0.0692 seconds with return value 0
Press any key to continue . . .
```

❖ PROGRAM 7: Implementation of Insertion sort.

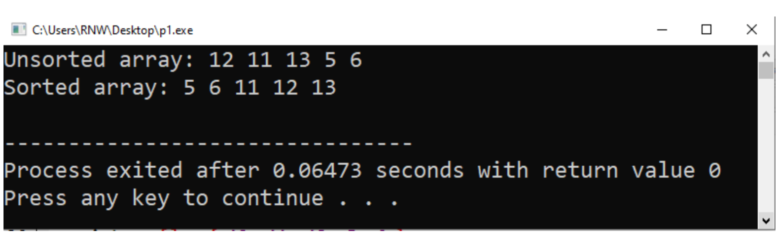
Algorithm:

Step 1: Start with the second element (index 1) as the key.
Step 2: Compare the key with the elements before it.
Step 3: If the key is smaller than the compared element, shift the compared element one position to the right.
Step 4: Insert the key where the shifting of elements stops.
Step 5: Repeat steps 2-4 for all elements in the unsorted part of the list.

PROGRAM:

```
#include <math.h>
#include <stdio.h>
void insertionSort(int arr[], int N) {
        int i;
    // Starting from the second element
    for (i = 1; i < N; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move elements of arr[0..i-1], that are greater than key, to one position to the right of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
                                        // Move the key to its correct position
        arr[j + 1] = key;
    }
}
int main() {
    int arr[] = { 12, 11, 13, 5, 6 };
    int N = sizeof(arr) / sizeof(arr[0]),i;

    printf("Unsorted array: ");
    for (i = 0; i < N; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
                                    // Calling insertion sort on array arr
    insertionSort(arr, N);
    printf("Sorted array: ");
    for (i = 0; i < N; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

➢ **Output:**



❖ **PROGRAM 8: Implementation of Merge sort.**

```
// C program for the implementation of merge sort
#include <stdio.h>
#include <stdlib.h>

// Merges two subarrays of arr[] first subarray is arr[left..mid] Second subarray is arr[mid+1..right]
void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Create temporary arrays
    int leftArr[n1], rightArr[n2];

    // Copy data to temporary arrays
    for (i = 0; i < n1; i++)
        leftArr[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        rightArr[j] = arr[mid + 1 + j];

    // Merge the temporary arrays back into arr[left..right]
    i = 0;
    j = 0;
    k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        }
        else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of leftArr[], if any
    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
```

```
        k++;
    }

    // Copy the remaining elements of rightArr[], if any
    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;                    }
    }

// The subarray to be sorted is in the index range [left-right]
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
            // Calculate the midpoint
        int mid = left + (right - left) / 2;
        // Sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        // Merge the sorted halves
        merge(arr, left, mid, right);
    }
}

int main() {
    int arr[] = { 12, 11, 13, 5, 6, 7 }, i;
    int n = sizeof(arr) / sizeof(arr[0]);

    mergeSort(arr, 0, n - 1);        // Sorting arr using mergesort

    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

- **Output:**

```
C:\Users\RNW\Desktop\p1.exe
5 6 7 11 12 13
--------------------------------
Process exited after 0.06718 seconds with return value 0
Press any key to continue . . .
```

❖ PROGRAM 9: Program for Quick sort.

```c
// C Program to sort an array using qsort() function in C.

#include <stdio.h>

#include <stdlib.h>


// If a should be placed before b, compare function should return positive value, if it should be placed after b, it should return negative value. Returns 0 otherwise

int compare(const void* a, const void* b) {

    return (*(int*)a - *(int*)b);

}


int main() {

    int arr[] = { 45, 12, 95, 33, 10 };

    int n = sizeof(arr) / sizeof(arr[0]);

    int i;


          // Sorting arr using inbuilt quicksort method

    qsort(arr, n, sizeof(int), compare);


    for (i = 0; i < n; i++)

        printf("%d ", arr[i]);


    return 0;

}
```

➢ **Output:**

```
C:\Users\RNW\Desktop\p1.exe
10 12 33 45 95
--------------------------------
Process exited after 0.06849 seconds with return value 0
Press any key to continue . . .
```