INDEX

NAME: Ponnuri Aniruddha STD.: ____ SEC.: ____ ROLL NO.: ____ SUB.: DAA - Lab

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	24/01/23	Linear Searching	10	
2	01/02/23	Bubble sort	10	
3	01/02/23	Insertion sort		
3	08/02/23	Merge Sort	10	
4	14/02/23	Quick sort & binary search	8.5	
5	21/02/23	Strassen Matrix Multiplication Maximum Subarray Sum	8.5	
6	03/03/23	finding maximum & minimum in Array Convex Hull	9	
7	03/03/23	Huffman Coding Knapsack Problem	9	
8	03/03/23	Tree Traversal MST using Kruskal	9	
9	24/03/23	Longest Common Subsequence	9	
10	04/04/23	N Queen Problem	9.5	
11	11/04/23	Travelling Salesman Problem	9.5	
12	18/04/23	BFS and DFS implementation - with array	9.5	
13	18/04/23	Randomised Quicksort	9.5	
14	25/04/23	String Matching Algorithm	9.5	
15	25/04/23	Sorting algorithm Comparision	10	Sharp 02/05/23

Completed.

Date - 24/01/23

Experiment - 1

Aim

To perform linear search operation

Algorithm

Step 1:- Open VS code

Step 2:- Declare array of fixed size along with additional variables

Step 3:- Using for loop get the number sequence

Step 4:- Get the element to search in the number series

Step 5:- Using for loop and assignment operations, find the presence of the specified element

Step 6:- Using 'if' conditional statement print if the number exists or not.

```
If (a[i] == x)
    break;
```

Code

```
#include <stdio.h>

int search (int arr[], int n, int x) {
    for (int i=0; i<n; i++)
        if (arr[i] == x)
            return i+1;
    return -1;
}
```

```
int main() {
    int arr[50], n, x;
    printf("Enter the number of elements :-- "); scanf("%d", &n);
    printf("Enter the elements :-- \n");
    for (int i=0; i<n; i++)
        scanf("%d", &arr[i]);
    printf("Enter the element you want to search :-- "); scanf("%d", &x);
    int index = search(arr, n, x);
    if (index == -1)
        printf("element is not present in the array");
    else
        printf("Element found at position %d", index);
    return 0;
}
```

Result:-

Hence the linear search program is implemented in c successfully

01/02/23

Output

Enter the Number of elements:- 4

Enter the elements:-

5

2

3

7

Enter the element you want to search:- 3

Element found at position 3

Coding standards :- 2
(2 marks)

Program Explanation :- 2
(2 marks)

Innovation :- 1
(1 mark)

Hackerrank :- 3
(3 marks)

Output / Viva :- 2
(2 marks)

10

01/04/23

Date - 01/02/23

Experiment - 2(a)

Aim

To perform bubble sort to arrange elements of an array

Algorithm

Step 1:- Start the program

Step 2:- Initialize one variable 'n' and 1-D integer array and user input

Step 3:-

```
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
        if (array[j] > array[j+1]) {
            int temp = array[j];
            array[j] = array[j+1];
            array[j+1] = temp;
        }
    }
}
```

Step 4:- Print the sorted array

Step 5:- Stop the program

Program:-

```
#include <stdio.h>
int main() {
    int arr[50], n, i, temp, j;
    printf("Enter the number of elements :-- ");scanf("%d", &n);
    printf("Enter the element :-- \n");
    for (i=0; i<n; i++)
        scanf("%d", &arr[i]);
    printf("The original array :-- ");
    //Printing the array
    for (i=0; i<n; i++)
        printf("%d\t", arr[i]);
    printf("\n Sorting \n");
```

```
// bubble sort
for(i=0; i<n-1; i++){
    for(j=0; j<n-i-1; j++) {
        if(arr[j] > arr[j+1])
        {
            temp = array[j];
            array[j] = array[j+1];
            array[j+1] = temp;
        }
    }
    for(x=0; x<n; x++) {
        printf("%d\t", array[x]);
    printf("\n");
    }
  return 0;
}
```

Result

The code is run successfully for bubble sort.

8/2/23

Output

Enter the Number of Elements :- 7
Enter the Elements :-

32
51
27
85
23
13
57

The original array :-

32 51 27 85 23 13 57

Sorting

32 27 51 23 13 57 85

27 32 23 13 51 57 85

27 23 13 32 51 57 85

23 13 27 32 51 57 85

13 23 27 32 57 57 85

Coding standards :- 2
(2 marks)

Program Explanation :- 2
(2 marks)

Innovation :- 1
(1 marks)

Hackerrank :- 3
(3 marks)

~~Output~~ Viva :- 2
(2 marks)

10

Date :- 01/02/23

Experiment :- 2(b)

Aim

To perform insertion sort to arrange elements of an array

Algorithm

step 1 :- Start the program

step 2 :- If the element is the first element, assume already sorted

step 3 :- Pick the next element, store it seperately in a key

step 4 :- Compare the key with all elements in the sorted array

step 5 :- If the element in the sorted array is smaller than the current element, then move to the next element. Else Shift the greater to the right

step 6 :- Insert the value

step 7 :- Repeat step 3 - 6 till the array is sorted

step 8 :- Stop the program

Program

```
#include <stdio.h>
int main() {
   int arr[50], n, temp, i, j;
   printf("Enter the number of elements :-- "); scanf("%d", &n);
   printf("Enter the elements :- \n");
   for(i=0; i<n; i++)
       scanf("%d", &arr[i]);
   printf("The original array :-- ");
    // Printing the array
   for(i=0; i<n; i++)
      printf("%d\t", arr[i]);
   printf("\n Sorting \n");
```

```
// Insertion Sorting
for (i=0; i<n; i++)
    temp = arr[i]; j = i-1;
    while (j>=0 && arr[j] > temp) {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = temp;
    printf("The Sorted array:--");
    for (i=0; i<n; i++) {
        printf("%d \t", arr[i]);
    }
    return 0;
}
```

Result

Insertion Sort was successfully implemant using c program

8/2/23

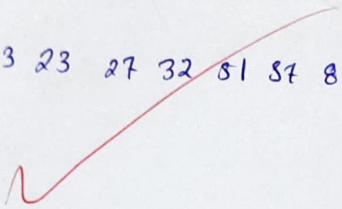
Output

Enter the number of elements :- 7
Enter the elements :
32
27
51
23
13
57
85

The original array :- 32 27 51 23 13 57 85

Sorting

The sorted array :- 13 23 27 32 51 57 85



Coding standards :- 2
(2 marks)

Program Explanation :- 2
(2 marks)

Innovation :- 1
(1 marks)

Hackerrank :- 3
(3 marks)

Output + Viva :- 2
(2 marks)

10

Date : 08/02/23

Lab-3 Merge Sort

Experiment - 3

Aim

To perform merge sort operation

Algorithm

Step 1 :- Start the program

Step 2 :- Check if the left index of array than the right index, if yes then calculate the mid point

Step 3 :- Repeat the above step, until the atomic units of the array is reached and further division is not possible

Step 4 :- After dividing the array into smallest units, start merging the elements again based on comparison of size of elements

Step 5 :- Compare the Elements for each list and then combine them into another list in a sorted manner

Step 6 :- Stop the Program

Program

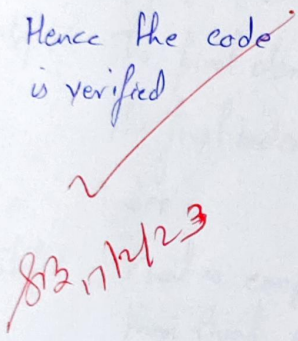
```
#include <stdio.h>

// Merge two subarrays L and M into arr
void merge(int arr[], int p, int q, int r) {
    int n1 = q - p + 1;
    int n2 = r - q;
    int L[n1], M[n2];
    for (int i = 0; i < n1; i++)
        L[i] = arr[p + i];
    for (int j = 0; j < n2; j++)
        M[j] = arr[q + 1 + j];

// Maintain current index of subarrays and main array
    int i, j, k;
    i = 0;
    j = 0;
    k = p;
```

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

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

```
int main() {
    int arr[100], size;
    printf("Enter number of elements :- "); scanf("%d", &size);
    printf("Enter the elements :- \n");
    for (int i=0; i<size ; i++)
        scanf("%d", &arr[i]);

    mergeSort(arr, 0, size-1);
    printf("Sorted array : \n");
    printArray(arr, size);
    return 0;
}
```

Result

Hence the code is executed successfully and the output is verified



output

Enter number of elements :- 6

Enter the elements :-

12
35
25
8
32
16

Sorted array :-

8 12 16 25 32 35

Coding Standards :- 2
(2 marks)

Program Explanation :- 2
(2 marks)

Innovation :- 1
(1 mark)

Hackerrank :- 3
(3 marks)

Viva :- 2
(2 marks)

2/10

Quick Sort And Binary Search

Date - 17/02/23

Experiment - 4

Aim

To implement Quick sort and binary search

Algorithm

Quick Sort

Step 1 :- Start the Program

Step 2 :- Take input from the user into an array

Step 3 :- The rightmost element is selected as pivot element, then a pointer is placed at pivot element

Step 4 :- The pivot element is compared with the elements beginning from the first index, and second pointer is also placed at first index

Step 5 :- Pivot is compared with other elements. If an element smaller than pivot element is reached, the smaller element is swapped with the greater element found

Step 6 :- The above step is repeated the the array is sorted

Step 7 :- End the Program

Binary Search

Step 1 :- Start the Program

Step 2 :- The sorted array and the element to be found is taken as input. Set two pointer low and highest at two ends of the array.

Step 3 :- Find mid using mid = $\frac{high + low}{2}$.

Step 4 :- If x == mid, then return mid, Else compare the element to be searched with m.

Step 5 :- If x > mid, compare x with the middle element of the elements on the right side of mid. This is done by low = mid + 1

Step 6:- else compare x with the middle element of the elements on the left side of mid. This is done by setting high = mid-1.

Step 7:- Repeat steps 3 to 6 until element is found.

Step 8:- End the program

Code:-

```
#include <stdio.h>
void swap (int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition (int array[], int low, int high) {
  int pivot = array[high];
  int i = (low-1); // pointer for greater element
  // traverse each element of the array & compare them with the pivot
  for (int j = low; j < high; j++) {
      if (array[j] <= pivot) {
          i++;
          swap (&array[i], &array[j]);
      }
  }
  swap (&array[i+1], &array[high]);
  return (i+1);
}
void quickSort (int array[], int low, int high) {
  if (low < high) {
    int pi = partition (array, low, high);
    quickSort (array, low, pi-1);
    quickSort (array, pi+1, high);
  }
}
```

pm 11/4/22

```
int binarySearch(int arr[], int l, int r, int x) {
    if (r >= l) {
        int mid = l + (r-1)/2
        if (arr[mid] == x)
            return mid+1;
        if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);
        return binarySearch(arr, mid+1, r, x);
    }
    return -1;
}

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

int main() {
    int arr[100], size, x;
    printf("Enter number of elements :-- "); scanf("%d", &size);
    printf("Enter the elements :-- \n");
    for(int i=0; i<size; i++)
        scanf("%d", &arr[i]);
    printf("Unsorted Array : \n"); printArray(arr, size);
    quickSort(arr, 0, size-1); printf("Sorted Array : \n");
    printArray(arr, size);
    printf("Enter the element of sorted array to Search :-- "); scanf("%d", &x);
    int result = binarySearch(arr, 0, size-1, x);
    (result == -1) ? printf("Element is not present in array")
                   : printf("Element is present at position %d", result);
    return 0;
}
```

Output

Enter number of elements :- 7

Enter the Elements :-

8
7
1
2
9
6
0

UnSorted Array:

8 7 1 2 9 6 0

Sorted Array :-

0 1 2 6 7 8 9

Enter the element of sorted array to search :- 6

Element is present at position 4

Time Complexity : $O(n^2)$ (Quick Sort)

Binary Search

Time complexity : $O(\log n)$

Result

Hence the code is executed successfully and the output is verified

Coding Standards :- 2
(2marks)

Program Explanation :- 2
(2marks)

Innovation :- 0
(1marks)

Hackerrank :- 3
(3marks)

Viva :-
(2marks)

1.5

3

Date 1/02/23

Experiment - 5(a)

Aim

To perform strassen matrix multiplication using c

Algorithm

Step 1: Start the program

Step 2: Check matrix given is square matrix or not

Step 3:- If yes the assuming two matrix given are matrix A and matrix B

Step 4:- Calculate $P = [A_{11} + A_{22}] \times [B_{11} + B_{22}]$

$Q = B_{11}(A_{12} + A_{22})$

$R = A_{12}(B_{21} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = B_{22}(A_{11} + A_{12})$

$U = B_{11} + B_{12}(A_{21} - A_{11})$

$V = B_{22}(A_{21} - A_{22})$

Step 5:- Calculate value for resultant matrix C

$C_{12} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

Step 6:- End the Program

Code:-

```
#include <stdio.h>
void insertmatrix (int a[2][2]) {
    int i, j;
    for (i=0; i<2; i++){
        for(j=0; j<2; j++)
            scanf("%d", a[i][j]);
}
void printmatrix (int a[2][2]) {
    for (int i=0; i<2; i++){
        printf("\n");
        for(int j=0; j<2; j++) {
            printf("%d \t", a[i][j]);
        printf("\n");}
```

```
Void strassmatrix (int a[2][2], int b[2][2], int c[2][2]) {
    int m1, m2, m3, m4, m5, m6, m7;
    m1 = (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
    m2 = (a[1][0] + a[1][1]) * b[0][0];
    m3 = a[0][0] * (b[0][1] - b[1][1]);
    m4 = a[1][1] * (b[1][0] - b[0][0]);
    m5 = (b[1][1]) * (a[0][0] + a[0][1]);
    m6 = (a[1][0] - a[0][0]) * (b[0][0] + b[0][1]);
    m7 = (a[0][1] - a[1][1]) * (b[1][0] + b[1][1]);

    c[0][0] = m1 + m4 - m5 + m7;
    c[0][1] = m3 + m5;
    c[1][0] = m2 + m4;
    c[1][1] = m1 - m2 + m3 + m6;

int main() {
    int a[2][2], b[2][2], c[2][2];
    printf("Enter the elements of first matrix: \n");
    insert matrix(a);
    printf("Enter the elements of second matrix: \n");
    insert matrix(b);
    printf("\n After multiplication using strassen's Algorithm \n");
    strassmatrix(a,b,c);
    printmatrix(c);
    return 0;
}
```

Result

Matrix multiplication was performed using strassen multiplication Successfully

Output

Enter the elements of first matrix

10
20
30
40

Enter the elements of second matrix

50
60
70
80

The first matrix is

10 20
30 40

The second matrix is

50 60
70 80

After multiplication using Strassen's algorithm

1800 2200
4300 5000

Time Complexity : $O(N^{\log 7})$

Date 24/02/23

Experiment 5 (b)

Aim

To find the sum of contiguous subarray with the largest sum

Algorithm

Step 1: Start the program

Step 2: Initialize array and enter the elements in the array

Step 3: Divide the array into two parts

Step 4: Find the maximum of following

- Maximum subarray sum of left subarray
- Maximum subarray sum of right subarray
- Maximum subarray sum such that subarray crosses the midpoint

Step 5: End the program

Code:-

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

void maxSubarraySum (int a[], int size)
{
    int max_so_far = INT_MIN, max_ending_here = 0, start = 0, end = 0, s = 0;
    for (int i = 0; i < size; i++) {
        max_ending_here += a[i];
        if (max_so_far < max_ending_here) {
            max_so_far = max_ending_here;
            start = s;
            end = i;
        }
        if (max_ending_here < 0) {
            max_ending_here = 0;
            s = i + 1;
        }
    }
```

```
    printf("Maximum Contiguous Sum is %d \n", max_so_far);
    printf("Starting Index %d \n", start);
    printf("Ending Index %d \n", end);

int main() {
    int arr[50], n;
    printf("Enter the number of elements in array :-- "); scanf("%d", &n);
    printf("Enter the elements in array :-- \n");
    for (int i=0; i<n; i++) {scanf("%d", &arr[i]);}
    maxSubArraySum(arr, n);
    return 0;
}
```

Result

The sum of contiguous subarray with the largest sum was successfully found.

Jos 24/2/23

Output

Enter the number of element in array :- 8

Enter the elements in array

-2
-5
6
-2
-3
1
5
-6

Maximum Contiguous Sum is 7
Starting index 2
Ending index 6

Time Complexity : $O(N^2)$

Coding Standards :- 2
(2 marks)

Program Explanation :- 2
(2 marks)

Innovation :- 0
(1 mark)

Hackerrank :- 3
(3 marks)

1.5
8.5

Viva :-
(2 marks)

Date - 03/03/23

Experiment - 6

Aim

To find maximum and minimum in a array

Algorithm

Step 1:- Start the program

Step 2:- If the subarray has only one element, the element as both the maximum and minimum value

Step 3:- If the subarray has two elements, compare them and return the maximum and minimum value

Step 4:- If the subarray has more than two elements:

a) divide the array into two halves

b) Recursively call the function on each half

c) obtain the maximum and minimum values for each half

d) Compare the maximum, minimum values found in each half to determine the overall maximum, minimum value for the entire array

~~e) Compare the maximum and~~

Step 5:- Return the overall maximum and minimum values.

Step 6:- End the program

Code

```
#include <stdio.h>

struct pair {
    int min;
    int max;
};

struct pair getMinMax (int arr[], int low, int high) {

    struct pair minmax, mml, mmr;
    int mid;
```

```
if (low == high) {
    minmax.max = arr[low];
    minmax.min = arr[low];
    return minmax;
}
if (high == low+1) {
    if (arr[low] > arr[high]) {
        minmax.max = arr[low];
        minmax.min = arr[high];
    }
    else {
        minmax.max = arr[high];
        minmax.min = arr[low];
    }
    return minmax;
}
mid = (low+high)/2
mml = getMinMax(arr, low, mid);
mmr = getMinMax(arr, mid+1, high);
if (mml.min < mmr.min)
    minmax.min = mml.min;
else
    minmax.min = mmr.min;
if (mml.max > mmr.max)
    minmax.max = mml.max;
else
    minmax.max = mml.max;
return minmax;
}
void printArray(int arr[], int n) {
    printf("The Elements in the array are:-- \n");
    for(int i=0; i<n; i++) {
        printf("%d\t", arr[i]);
    }
    printf("\n");
}
```

```
int main () {
    int arr[50], n, x;
    printf ("Enter the number of elements :-- "); scanf ("%d", &n);
    printf ("Enter the elements :-- \n");
    for (int i=0; i<n; i++)
        scanf ("%d", &arr[i]);
    struct pair minmax = getMinMax (arr, 0, n-1);
    printArray (arr, n);
    printf ("Minimum element is %d \n", minmax.min);
    printf ("Maximum element is %d \n", minmax.max);
    return 0;
}
```

Result

The minimum and maximum element of the array has been successfully found.

Output

Enter the number of element :- 8

Enter the element :-

5
6
78
57
13
9
91
-2

The Element in the array are :-

5 6 78 57 13 9 91 -2

Minimum element is -2

Maximum element is 91

Time Complexity O(nlogn) . O(n)

Date 03/03/23

Experiment - 6 (b)

Aim

To find the convex hull for given points.

Algorithm

Step 1:- Start the program

Step 2:- Define a recursive function Convex hull to compute the convex hull of the set of n points.

Step 3:- If n <= 3, join the points and return the set found as the Convex hull

Step 4:- Divide the set of n' point into two subsets, left and right by drawing a vertical line that passes through the middle of the set of points And Recursively compute the convex hull of the left and right subsets.

Step 5:- Merge the two convex hull of the left and right subsets by finding upper and lower tangent lines.

Step 6:- Return the merged Convex hull

Step 7:- Stop the program

Time Complexity

The time complexity of this algorithm is $O(n \log n)$

Code:-

```
#include <bits/stdc++.h>
    using namespace std;

pair<int, int> mid;

int quad (pair<int, int> p) {
    if (p.first >= 0 && p.second >= 0)
        return 1;
    if (p.first <= 0 && p.second >= 0)
        return 2;
    if (p.first <= 0 && p.second <= 0)
        return 3;
    return 4;
}

int orientation (pair<int, int> a, pair<int, int> b, pair<int, int> c) {
    int res = (b.second - a.second) * (c.first - b.first) - (c.second - b.second) * (b.first - a.first);
    if (res == 0)
        return 0;
    if (res > 0)
        return 1;
    return -1;
}

bool compare (pair<int, int> p1, pair<int, int> q1) {
    pair<int, int> p = make_pair (p1.first - mid.first, p1.second - mid.second);
    pair<int, int> q = make_pair (q1.first - mid.first, q1.second - mid.second);

    int one = quad (p); int two = quad (q);
    if (one != two)
        return (one < two);
    return (p.second * q.first < q.second * p.first);
}
```

```
vector<pair<int,int>> merger(vector<pair<int,int>>a, vector<pair<int,int>>b) {
int n1 = a.size(), n2 = b.size();
int ia = 0, ib = 0;
for (int i=1; i<n1; i++)
    if (a[i].first > a[ia].first)
        ia = i;
for (int i=1; i<n2; i++)
    if (b[i].first < b[ib].first)
        ib = i;
// Upper tangent
int inda = ia, indb = ib; bool done = 0;
while (!done) {
    done = 1;
    while (orientation(b[indb], a[indb], a[(inda+1)%n1]) >= 0)
        inda = (inda+1)%n1;
    while (orientation(a[inda], b[indb], b[(n2+indb-1)%n2]) <= 0) {
        indb = (n2+indb-1)%n2;
        done = 0;
    }
}
// Upper tangent
int uppera = inda, upperb = indb;
inda = ia; indb = ib;
done = 0; int g = 0;
while (!done) {
    done = 1;
    while (orientation(a[inda], b[indb], b[(indb+1)%n2]) >= 0)
        indb = (indb+1)%n2;
    while (orientation(b[indb], a[inda], a[(n1+inda-1)%n1]) <= 0) {
        inda = (n1+inda-1)%n1;
        done = 0;
    }
}
int lowera = inda, lowerb = indb; vector<pair<int,int>> ret;
```

```
int ind = upper a;
ret.push_back(a[upper a]);
while (ind != lower a) {
    ind = (ind+1) % n1;
    ret.push_back(a[ind]);
}
ind = lower b
ret.push_back(b[lower b]);
while (ind != upper b) {
    ind = (ind+1) % n2
    ret.push_back(b[ind]);
}
return ret;
}

vector<pair<int,int>> brute Hull (vector<pair<int,int>> a) {
  set<pair<int,int>> s;
  for (int i=0; i<a.size(); i++) {
    for (int j=i+1; j<a.size(); j++) {
      int x1 = a[i].first, x2 = a[j].first;
      int y1 = a[i].second, y2 = a[j].second;
      int a1 = y1-y2, b1 = x2-x1, c1 = x1*y2 - y1*x2;
      int pos=0, neg=0;
      for (int k=0; k<a.size(); k++) {
        if (a1*a[k].first + b1*a[k].second + c1 <= 0)
          neg++
        if (a1*a[k].first + b1*a[k].second + c1 >= 0)
          pos++
      }
      if (pos == a.size() || neg == a.size()) {
        s.insert(a[i]);
        s.insert(a[j]);
      }
    }
  }
```

```
vector <pair <int, int>> ret;
for (auto e : s)
    ret.push_back(e);

mid = {0, 0}
int n = ret.size();
for (int i=0 ; i<n; i++) {
    mid.first += ret[i].first ;
    mid.second += ret[i].second;
    ret[i].first *= n;
    ret[i].second *= n;
}
sort (ret.begin, ret.end(), compare);
for (int i=0; i<n; i++)
    ret[i] = make_pair (ret[i].first/n, ret[i].second/n );
return ret;
}
vector <pair <int,int>> divide ( vector <pair <int, int>>a) {
    if (a.size ()<=5)
        return brute Hull(a);
    vector <pair <int,int>> left, right;
    for ( int i = 0 ; i<a.size()/2 ; i++)
        left.push_back (a[i]);
    for ( int i = a.size()/2 ; i<a.size(); i++)
        right.push_back (a[i]);

    vector <pair<int, int>> left_hull = divide (left);
    vector <pair <int, int>> right_hull = divide (right);
    return merge (left_hull, right_hull);
}
```

```
int main () {
    vector <pair <int, int>> a; int no, x, y;
    cout << "Enter number of points :-- "; cin >> no;
    cout << " Enter the points :-- \n";
    for (int i = 0; i < no; i++) {
        cin >> x >> y;
        a.push_back (make_pair (x, y));
    }
    sort (a.begin(), a.end());
    vector <pair <int, int>> ans = divide (a);
    cout << " Convex hull \n";
    for (auto i : ans)
        cout << i.first << " " << i.second << endl;
    return 0;
}
```

Result

Successfully found Convex Hull for the given points using C++ program

Output

Enter number of point :- 10

Enter the points :-

0 0
1 -4
-1 -5
-5 -3
-3 -1
-1 -3
-2 -2
-1 -1
-2 -1
-1 1

Convex hull :-

-5 -3
-1 -5
1 -4
0 0
-1 1

Coding Standards :- 2
(2 marks)

Program Explanation :- 2
(2 marks)

Innovation :- 0.5
(1 mark)

Hackerrank :- 3
(3 marks)

Viva :- 1.5
(2 marks)

9

Date 18/03/23

Experiment- 7 (a)

Aim

To implement Huffman Coding

Algorithm

Step 1:- Start the program

Step 2:- Take a input string of characters and their frequencies

Step 3:- Sort the characters in increasing order the frequency and store in priority queue.

Step 4:- Make each unique character as a leaf node

Step 5:- Create an Empty node. Assign the minimum frequency to the left child of node and second minimum frequency to the right child of node. Set the value of the node as the sum of the above two minimum frequencies

Step 6:- Remove these two minimum frequencies from queue and add the sum into list of frequencies

Step 7:- Insert node into tree

Step 8:- Repeat steps 4 to 7 for all characters

Step 9:- For each non-leaf assign 0 to the left edge and 1 to the right edge

Step 10:- Output the Huffman for each character.

Step 11:- End the Program

Code

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

#define MAX_TREE_HT 50

struct MinHNode {
    char item;
    unsigned freq;
    struct MinHNode *left, *right;
};

struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHNode **array;
};

struct MinHNode *newNode(char item, unsigned freq) {
    struct MinHNode *temp = (struct MinHNode*) malloc (sizeof (struct MinHNode));
    temp->left = temp->right = NULL;
    temp->item = item;
    temp->freq = freq;
    return temp;
}

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

struct MinHeap *createMinH (unsigned capacity) {
    struct MinHeap *minHeap = (struct MinHeap *) malloc (sizeof (struct MinHeap));
    minHeap->size =0;
    minHeap->capacity= capacity;
    minHeap->array= (struct MinHNode **) malloc (minHeap->capacity *sizeof (struct MinH
    return minHeap;
}
```

```
Void swapMinHNode (struct MinHNode **a, struct MinHNode **b) {
    struct MinHNode *t = *a;
    *a = *b;  *b = t;
}
void minHeapify (struct MinHeap *minHeap, int idx) {
    int smallest = idx; int left = 2 * idx + 1; int right = 2 * idx + 2;
    if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]
        ->freq)
        smallest = left;
    if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array
        [smallest]->freq)
        smallest = right;
    if (smallest != idx) {
        swapMinNode (&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify (minHeap, smallest);
    }
int checkSizeOne (struct MinHeap *minHeap) {
    return (minHeap->size == 1);
}
struct MinHNode *extractMin (struct MinHeap *minHeap) {
    struct MinHNode *temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify (minHeap, 0);
    return temp;
}
Void insertMinHeap (struct MinHeap *minHeap, struct MinHNode *minHeapNode) {
    ++minHeap->size; int i = minHeap->size - 1;
    while (i && minHeapNode->freq < minHeap->array[(i-1)/2]->freq) {
        minHeap->array[i] = minHeap->array[(i-1)/2];
        i = (i-1)/2;
    }
    minHeap->array[i] = minHeapNode;
}
```

```
void buildMinHeap (struct MinHeap *minHeap) {
    int n = minHeap->size - 1; int i;
    for (i = (n-1)/2; i >= 0; --i)
        minHeapify (minHeap, i);
}
int isLeaf (struct MinHNode *root) {
    return !(root->left) && !(root->right);
}
struct MinHeap *CreateAndBuildMinHeap (char item[], int freq[], int size) {
    struct Min *minHeap = CreateMinH(size);
    for (int i=0; i < size; ++i)
        minHeap->arr[i] = newNode (item[i], freq[i]);
    minHeap->size = size;
    buildMinHeap (minHeap);
    return minHeap;
}
struct MinHNode *buildHuffmanTree (char item[], int freq[], int size) {
    struct MinHNode *left, *right, *top;
    struct MinHeap *minHeap = CreateAndBuildMinHeap (item, freq, size);
    while (!checkSizeOne (minHeap)) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = newNode ('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap (minHeap, top);
    }
    return extractMin (minHeap);
}
```

```
void printHcodes(struct MinHNode *root, int arr[], int top) {
    if (root->left) {
        arr[top] = 0;
        printHcodes(root->left, arr, top+1);
    }
    if (root->right) {
        arr[top] = 1;
        printHcodes(root->right, arr, top+1);
    }
    if (isleaf(root)) {
        printf("  %c  |", root->item);
        printArray(arr, top);
    }
}

void HuffmanCodes(char item[], int freq[], int size) {
    struct MinHNode *root = buildHuffmanTree(item, freq, size);
    int arr[MAX_TREE_HT], top = 0;
    printHcodes(root, arr, top);
}

int main() {
    char arr[50]; int freq[50], n;
    printf("Enter Number of characters :-- "); scanf("%d", &n);
    printf("Enter Characters :-- \n");
    for(int i=0; i<n; i++) { scanf("%s", &arr[i]); }
    printf("Enter their frequency :-- \n");
    for(int i=0; i<n; i++) { scanf("%d", &freq[i]); }
    printf("  Char | Huffman Code ");
    printf("\n --------------- \n");
    HuffmanCodes(arr, freq, n);
}
```

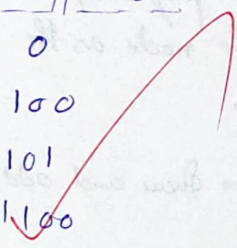
Result

Successfully found Huffman codes for the given characters using C program.

Output

```
Enter Number of characters :- 6
Enter characters :-
a
b
c
d
e
f
Enter their frequency
5
9
12
13
16
45
```

Char	Huffman Code
f	0
e	100
d	101
a	1100
b	1101
c	111



Date:- 15/03/23

Experiment - 7(b)

Aim

To implement Knapsack problem using greedy method

Algorithm

Step 1:- start the program

Step 2:- Take input containing an array of 'n' items, where each item has a weight 'w[i]' and a value v[i] and also capacity 'w' of a knapsack

Step 3:- Calculate the value per unit weight for each item 'p[i] = v[i]/w[i]'

Step 4:- Sort the items in decreasing order of their value per unit weight 'p'

Step 5:- Initialize the total value of knapsack 'v=0' and the remaining capacity of the knapsack c=w.

Step 6:- for each item i in the sorted list

a) If the weight of the item 'w[i]' is less than or equal to the remaining capacity of knapsack, add the entire item to knapsack by setting v = v + v[i] & c = c - w[i]

b) If the weight of the item 'w[i]' is greater than the remaining capacity of the knapsack 'c', add a fraction of the item to the knapsack by setting v = v + p[i]*c and c = 0

Step 7:- Return the total value 'v' of the knapsack

Step 8:- End the program

Code:-

```
#include <stdio.h>
void Knapsack (int n, float weight[], float profit[], float capacity) {
   float x[20], tp = 0; int i, j, u = capacity;
   for (i = 0; i < n; i++)
      x[i] = 0.0;
   for (i = 0; i < n; i++) {
      if (weight[i] > u)
         break;
      else {
         x[i] = 1.0;
         tp = tp + profit[i];
         u = u - weight[i];
      }
   }
   if (i < n)
      x[i] = u / weight[i];
   tp = tp + (x[i] * profit[i]);
   printf ("\n The result vector is :- ")
   for(i = 0; i < n; i++)
      printf("%.2f \t", x[i]);
   printf("\n Maximum profit is :- %.2f", tp);
}
int main() {
   float weight[20], profit[20], capacity;
   int num, i, j;
   float ratio[20], temp;
   printf ("\n Enter the no. of objects :- ");
   scanf ("%d", &num);
```

```
printf("\n Enter the wts and profits of each object :- \n");
for(i=0; i<num; i++)
    scanf("%f %f", &weight[i], &profit[i]);
printf("\n Enter the capacity of Knapsack :- ");
scanf("%f", &capacity);
for(i=0; i<num; i++) { ratio[i]= profit[i]/weight[i]; }
for(i=0; i<num; i++) {
  for(j=i+1, j<num; j++) {
      if(ratio[i]<ratio[j]) {
          temp = ratio[j];
          ratio[j]= ratio[i];
          ratio[i] = temp;
          temp = weight[i];
          weight[j] = weight[i];
          weight[i] = temp;
          temp = profit[j];
          profit[j] = profit[i];
          profit[i] = temp;
      }
  }
}
Knapsack(num, weight, profit, capacity);
return 0;
}
```

Result

Successfully solved Knapsack problem using greedy method in C.

Output

Enter the no. of object :- 7

Enter the wts and profit of each object :-

2 10
3 8
5 15
7 7
1 6
4 18
1 3

Enter the capacity of Knapsack :- 15

The result vector is :- 1.00 1.00 1.00 1.00 1.00 0.67 0.00

Maximum profit is 57.33

Coding standards :- 2
(2marks)

Program Explaination :- 2
(2marks)

Innovation :-
(1mark) 0.5

Hackerrank :- 3
(3marks)

Viva :- 1.5
(2marks)

9

Date 28/03/23

Experiment - 8 (a)

Aim

To implement tree traversal

Algorithm

Inorder Traversal

1) Traverse the left subtree by recursively calling the inorder function on the left child node
2) Visit root node
3) Traverse the right subtree by recursively calling the inorder function on the right child node.

Pre-order Traversal

1) Visit the root node
2) Traverse the left subtree by recursively calling preorder function on the left child node.
3) Traverse the right subtree by recursively calling preorder function on the right child node

Post-order Traversal

1) Traversal the left subtree by recursively calling postorder function on the left child node
2) Traversal the right subtree by recursively calling postorder function on the right child node
3) Visit the root node

Code

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

Struct node {
    int data;
    Struct node * left;
    Struct node *right;
};

Struct node* newNode (int data) {
    Struct node* node = (Struct node*) malloc (sizeof(Struct node));
    node -> data -> data;
    node -> left = NULL;
    node -> right = NULL;
    return (node);
}
void printInorder (Struct node* node) {
    if (node == NULL)
        return ;
    printInorder (node -> left);
    printf("%d", node -> data);
    printInorder (node -> right);
}
void printPreorder (Struct node* node)
    if (node == NULL)
        return;
    printPreorder (no printf("%d", node -> data);
    printPreorder (node -> left);
    printPreorder (node -> right);
}
```

```
void printPostorder(struct node* node) {
    if(node == NULL)
        return;
    printPostorder(node->left);
    printPostorder(node->right);
    printf("%d", node->data);
}
int main() {
    struct node* root = newNode(1);
    root->left = newNode(2)
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    printf("\nInorder Traversal of binary tree is \n");
    printInorder(root);
    printf("\nPreorder Traversal of binary tree is \n");
    printPreorder(root);
    printf("\nPostorder Traversal of binary tree is \n");
    printPostorder(root);
    return 0;
}
```

20/2/23

Result:
Successfully traversed binary tree using C program

Output

Inorder Traversal of binary tree is

4 2 5 1 3

Preorder Traversal of binary tree is

1 2 4 5 3

Postorder Traversal of binary tree is

4 5 2 3 1

Date : 28/03/23

Experiment - 8(b)

Aim

To implement minimum spanning tree using Kruskal's algorithm

Algorithm

1) Sort all the edges in ascending order of their weight
2) For each edge in the sorted list of edges:
   a) if including the edge in the MST doesn't create a cycle, add it to the MST
   b) otherwise discard the edge
3) Return the MST

Code:

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

struct edge{
    int src;
    int dist;
    int weight;
};
struct Graph{
    int v;
    int e;
    struct edge * edges;
};
struct subset{
    int parents;
    int rank;
};
```

```
struct Graph * Create_graph (int v, int e) {
    struct Graph* g = (struct Graph *) malloc (sizeof (struct Graph));
    malloc (sizeof (
    g->v = v;
    g-> e = e;
    g->edges = (struct edge *) malloc (sizeof (struct edge) *e))
    return g;
}
int find (struct subset subset[] , int i) {
    if (subset[i].parent != i) {
        subsets[i].parent = find (subset, subset[i].parent);
    }
    return subsets[i].parent;
}
void union_sets (struct subsets subsets[], int x, int y) {
    int x_root = find(subsets, x);
    int y_root = find (subsets, y);
    if (subsets of [x_root].rank < subset[y_root].rank) {
        subsets[x_root].parent = y_root;
    }
    else if (subset[x_root].rank > subsets[y_root].rank) {
        subsets[y_root].parent = x_root;
    }
    else {
        subset[y_root].parent = x_root
        subset[x_root].rank ++;
    }
}
int compare (const void *a , const void *b) {
    return ((struct edge *)a)->weight > ((struct edge*)b)->weight;
}
```

```
void Kruskal_mst( struct graph *g) {
    int v = g->v;
    struct edge result[v];
    int e=0; int i=0;
    qsort(g*edges, g*e, sizeof(struct edge), compare);
    struct subset* subsets = (struct subset*) malloc (sizeof(struct subset)*v);
    for(i=0; i<v; i++) {
        subset[i].parent=i;
        subset[i].rank=0;
    }
    i=0;
    while (e<v-1 && i<g*E) {
        struct edge next_edge = g->edge[i++];
        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dst);
        if (x!=y) {
            result[e++] = next_edge;
            union_set(subsets, x, y);
        }
    }
    printf("Edges in MST: \n");

    for(i=0; i<e; i++) {
        printf("%d %d %d \n", result[i].src, result[i].dst, result[i].w);
    }
}

int main() {
    int v=4, e=5;
    struct Graph* g = create_graph(v,e);
    g*edge[0].src = 0;
    g*edge[0].dst = 1;
    g* edge[0].weight = 10;
```

```
g->edge[1].src = 0
g->edge[1].dest = 2;
g->edge[1].w = 6;

g->edge[2].src = ;
g->edge[2].dest = ;
g->edge[2].w = ;
g->edge[3].src = ;
g->edge[3].dest = ;
g->edge[3].w = ;
g->edge[4].src =
g->edge[4].dest =
g->edge[4].w =
Kruskal.mst(g);
return 0;
}
```

Result

Successfully found Minimum Spanning Tree using Kruskal algorithm.

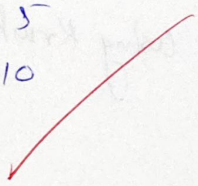
output

Edges in MST

(2, 3) : 4

(0, 3) : 5

(0, 1) : 10



Coding standards :- 2
(2marks)

Program Explanation 2
(2marks)

Innovation :- 0.5
(1marks)

Hackerrank :- 3
(3marks)

Viva :- 1.5
(2marks)

9

Date :- 28/03/23

Experiment - 9

Aim

To find longest common subsequence

Algorithm

1. Create a table of dimension n+1 * m+1 where n & m are the length of x and y strings respectively. The first row and the first column are filled with zeros
2. Fill each cell of the table using the logic
3. if the character corresponding to the current row and current column are matching then fill the current cell by adding one to the diagonal
4. else the maximum value from the previous column and previous row element for filling the current cell. point an arrow to the cell with max value
5. Repeat Step 2 until the table is filled
6. In order find the LCS, start from the last element and follow the diagonal of the array.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define max 100
char *lcs (char *x, char *y) {
    int m = strlen (x);
    int n = strlen (y);
    int L[m+1][n+1];
```

```
for (int i=0; i<=m; i++) {
    for (int j=0; j<=n; j++) {
        if (i==0 || j==0) {
            L[i][j]=0;
        } else if (X[i-1]==Y[j-1]) {
            L[i][j]=L[i-1][j-1]+1;
        } else {
            L[i][j]=(L[i-1][j]>L[i][j-1])?L[i-1][j]:L[i][j-1];
        }
    }
}
char *lcs=(char*)malloc(sizeof(char)*(L[m][n]+1));
int i=m, j=n; index=0;
while(i>0 && j>0){
    if(X[i-1]==Y[j-1]){
        lcs[index++]=X[i-1];
        i--;
        j--;
    } else if (L[i-1][j]>L[i][j-1]){
        i--;
    } else {
        j--;
    }
}
lcs[index]='\0';
int len=strlen(lcs);
for(int k=0; k<len/2; k++){
    char temp=lcs[k];
    lcs[k]=lcs[len-k-1];
    lcs[len-k-1]=temp;
}
return lcs;
}
```

```
int main () {
    char X[max], Y[max];
    printf("Enter the first string : "); scanf("%s", X);
    printf("Enter the second string : "); scanf("%s", Y);
    char * result = lcs(X,Y);
    printf("The longest common subsequence is %s\n", result);
    free(result);
    return 0;
}
```

Result.

Successfully found the longest subsequence of two strings using C program

28/2/23

Output

Enter the first string : AXACB

Enter the second string : CBDA

The longest common subsequence is : CB

Coding standards :- (2marks) 2

Program Explanation (2marks) 2

Innovation :- (1marks) 0.5

Hackerrank :- (3marks) 3

Viva :- (2marks) 1.5

9

Date 06/04/23

Experiment - 10

Aim

To find the solution for N-Queen's Problem

Algorithm

1) Start in the leftmost column
2) If all Queens are placed
   return True
3) Try all rows in the current column
   Do following for every tried row
   a) If the Queen can be placed safely in this row
      then mark this [row, column] as part of the solution and recursively check if placing Queen here leads to a solution
   b) If placing the Queen in [row, column] leads to a solution
      then return True
   c) If placing Queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
4) If all rows have tried and nothing worked, return False to trigger backtracking

Code

```
#include <stdio.h>
#include <stdbool.h>
#define N 4

void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%d", board[i][j]);
        printf("\n");
    }
}
```

```
bool isSafe(int board[N][N], int row, int col) {
    int i, j;
    for (i=0; i<col; i++)
        if (board[row][i])
            return false;
    for (i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j])
            return false;
    for (i=row, j=col; j>=0 && i<N; i++, j--)
        if (board[i][j])
            return false;
    return true;
}
bool solveNQUtil(int board[N][N], int col) {
    if (col >= N)
        return true;
    for (int i=0; i<N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQUtil(board, col+1))
                return true;
            board[i][col] = 0;
        }
    }
    return false;
}
```

```
bool solveNQ() {
    int board[N][N] = { {0,0,0,0},
                        {0,0,0,0},
                        {0,0,0,0},
                        {0,0,0,0} };

    if (solveNQUtil(board, 0) == false) {
        printf("Solution does not exist");
        return false;
    }
    printSolution(board);
    return true;
}
int main() {
    solveNQ();
    return 0;
}
```

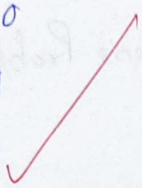
8/3/14/25

Result

Successfully Executed N-Queen's problem using C program

Output

0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0



Coding Standards :- 2
(2marks)

Program Explanation :- 2
(2marks)

Innovation :- 0.5
(1mark)

Hackerrank :- 3
(3marks)

Viva :- 2
(2marks)

9.5

Date - 11/04/23 Experiment - 11

Aim

To implement code for travelling salesman problem

Algorithm

Step 1:- Initialize cities variables

Step 2:- Set visited[0] = true and tour[0] = 0, which represents the starting city

Step 3:- Call the recursive function tsp(1,1,0) which start at city 1 has a depth of 1 and has tour cost of 0.

Step 4:- In the TSP function

→ If depth == n, which means that all cities have been visited, calculated the tour cost by adding the distance between the city

→ After returning from the recursive call, mark the city i as unvisited and remove it from the current tour

Step 5:- After TSP returns, the optimal tour is stored in best tour and its cost in min Tour cost. Return best tour and min tour cost

Code:-

```
#include <stdio.h>
int dry[10][10], completed[10][10], n, cost=0
void take input() {
    printf("Enter the Number of cities");
    scanf("%d", &n);
    printf("Enter Elements of cost matrix\n");
    for(i=0; i<n; i++) {
        printf("Enter of Row: %d", i+1);
        for (j=0; j<n; j++) {
            scanf("%d", &dry[i][j]); completed[i][j]=0; }
```

```
printf("\n\n The cost but is : ");
for (i=0; i<n; i++) {
    printf("\n")
    for(j=0; j<n; j++)
        printf("%d \t", dory[i][j]);
}
}
void mincost(int cost) {
    int i, ncity;
    completed[i] = 1;
    printf("%d -> ", city+1);
    ncity = least(city);
    if (ncity == 999) {
        ncity = 0
        printf("%d", ncity+1);
        cost += dory[city][ncity];
        return;
    }
    mincost(ncity);
}
int least(int c) {
    int i, nc=999;
    int min=999, kmin;
    for(i=0; i<n; i++) {
        if((dory[c][i]!=0) && (completed[i]==0))
            if (dory[c][i] + dory[i][c] < min) {
                min = dory[i][c] + d[c][i];
                kmin = dory[c][i];
                nc = i;
            }
    }
    if (min!=999)
        cost += kmin
        return nc;
}
```

```
int main() {
    takeInput()
    printf("This path is \n");
    minCost(0);
    printf("Minimum cost is %d \n", cost);
    return 0;
```

Result

Successfully executed code for the Travelling Salesman problem in C programming

Output

Enter the number of cities : 4

Enter the cost matrix

Enter elements of row 1:

0 4 3 1

Enter elements of row 2:

4 0 2 1

Enter elements of row 3:

1 2 0 5

Enter elements of row 4

3 1 5 0

The cost list is 0 4 3 1
4 0 2 1
1 2 0 5
3 1 5 0

The path is

1 → 3 → 2 → 4 → 1

minimum cost is 9

Coding Standards (2m) :- 2

Program Explanation (2m) :- 2

Innovation (1m) :- 1

Hackerrank (3m) :- 3

Viva (2m) :- 1.5

9.5

Date - 18/04/23

Experiment - 12

Aim

To implement codes for BFS and DFS with an array in C

Algorithm (BFS)

Step 1:- Start by selecting a starting node or vertex in graph

Step 2:- Add the starting node to Queue

Step 3:- Mark the starting node as visited

Step 4:- while Queue is not empty, repeat the steps

① Remove the first node from Queue and set it as the current node

② for each adjacent node of the current node, check if it has been visited or not. If it is not visited then add it into Queue

Step 5:- If there are no more nodes in the Queue, then the search is complete

Code:-

```
#include <stdio.h>
#include <stdlib.h>
struct Queue {
    int size
    int f, r
    int *arr;
};
int isEmpty (struct Queue *q){
    if (q->r == q->f)
        return 1;
    return 0;
}
```

```
int isFull(struct Queue *q) {
    if (q->r == q->size - 1) {
        return 1;
    }
    return 0;
}
void EnQueue(struct Queue *q, int val)
    if (isFull(q)) {
        printf("The Queue is full \n")
    else {
        q->r++;
        q->arr[q->r] = val;
    }
int dequeue(struct queue *q) {
        int a = -1;
        if (isEmpty(q)) {
            printf("The Queue is empty \n"); }
        else {
            q->f++;
            a = q->arr[q->f];
        }
        return a;
    }
int main() {
    struct Queue q;
    q.size = 400
    q.f = q.r = 0.
    q.arr = (int*) malloc (q.size * sizeof(int));
    int node, i = 1;
    int visited[7] = {0,0,0,0,0,0,0};
    int a[7][7] = {{0,1,1,1,0,0,0},
                   {1,0,1,0,0,0,0},
                   {1,1,0,1,1,0,0},
                   {1,0,1,0,1,0,0},
                   {0,0,1,1,0,1,1},
                   {0,0,0,0,1,0,0}, {0,0,0,0,1,0,0}};
```

```
printf("%d", i);
visited[i] = 1;
Enqueue (&q, i);
while (!isEmpty (&q));
  int node = dequeue (&q);
    for (int j=0; j<7; j++) {
     if (a[node][j]==1 && visited[j]==0){
          printf ("%d", j);
          visited[j]=1;
          Enque (q, i);
  }
}
return 0;
}
```

DFS

Algorithm

1) Initially all vertices are marked unvisited (false)
2) The DFS algorithm starts at a vertex of a vertex is in the graph. By starting at vertex it considers the edges from is to other vertices
   ① If the edge leads to an already visited vertex, then backtrack to current vertex
   ② If an edge leads to an unvisited vertex, then go to that vertex and start processing from that vertex
3) Follow this from process until a vertex are marked visited

Code

```
# include <stdio.h>
# include <stdlib.h>
 int vis[100];
 struct Graph{
      int v, e;
      int * adj; }
```

Output

1 0 2 3 4 5 6

```
struct Graph *adjMatrix() {
    struct Graph *G = (struct Graph*) malloc(sizeof(struct graph));
if(!G) {
    printf("Memory Error\n");
    return NULL;
}
G->V = 7;
G->E = 7;
G->Adj = (int**) malloc((G->V) * sizeof(int*));
for(int k=0; k<G->V; k++) {
    G->Adj[k] = (int*) malloc((G->V) * sizeof(int))
}
for(int u=0; u<G->V; u++){
    for(int v=0; v<G->V; v++) {
        G->Adj[u][v]=0;
    }
}
G->Adj[0][1] = G->Adj[1][0] = 1;
G->Adj[0][2] = G->Adj[2][0] = 1;
G->Adj[1][3] = G->Adj[3][0] = 1;
G->Adj[1][4] = G->Adj[4][1] = 1;
G->Adj[1][5] = G->Adj[5][1] = 1;
G->Adj[1][6] = G->Adj[6][1] = 1;
G->Adj[6][2] = G->Adj[2][6] = 1;
return G;
}
void DFStraversal(struct Graph* G) {
    for(int i=0; i<100; i++)
        vis[i]=0;
    for(int i=0; i<G->V; i++) {
        if(!vis[i]) {
            DFS(G,i);
        }
    }
}
```

```
void main () {
    struct Graph * G;
    G = adj Matrix();
    DFS
}
```

Result

Successfully implemented codes for both BFS and DFS with arrays

Output

0 1 3 4 5 6 2

Coding Standards (2m) :- 2

Program Explanation (2m) :- 2

Innovation (1m) :- 1

Hackerrank (3m) :- 3

Viva (2m) :- 1.5

9.5

Date - 18/04/23

Experiment - 13

Aim

To implement the C code for Randomized Quick Sort

Algorithm

```
Partition (arr[], lo, hi)
    pivot = arr[hi]
    i = lo
    if arr[i] <= pivot then
      Swap arr[i] with arr[j]
      i = i+1
      Swap arr[i] with arr[hi]
      return i

partition_r (arr[], lo, hi)
    r = Random Number from lo to hi
    Swap arr[r] and arr[hi]
    return partition (arr, lo, hi)

Quick Sort (arr[], lo, hi)
    if lo < hi
      p = partition_r (arr, lo, hi)
      QuickSort(arr, lo, p-1)
      QuickSort (arr, p+1, hi)
```

Code:-

```
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
int partition (int arr[], int low, int high) {
int pivot = arr[low];
int i = low-1; j = high+1;
while (1) {
    while (arr[i] < pivot);
    do {
        j--; }
    while (arr[j] > pivot);
        if (i>=j)
            return j;
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
int partition_r (int arr[], int low, int high) {
    Srand (time (0));
    int random = low + rand () % (high-low);
    int temp = arr [random];
    arr [low] = temp;
    return partition (arr, low, high);
void quick sort (int arr[], int low, int high){
    if (low < high)
        int p = partition_r (arr, low, high);
        quick sort (arr, low, p);
        quick sort (arr, pi+1, high); }
}
void printArray (int arr[], int n){
        for (int i=0; i<n; i++)
            printf ("%d ", arr[i]);
        printf ("\n");
}
```

```
int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted Array: \n");
    printArray(arr, n);
    return 0;
}
```

8/2/24

Result

Successfully implemented code for Randomized QuickSort in C programming

Output

1 5 7 8 9 10

Coding Standards :- 2
(2m)

Program Explanation :- 2
(2m)

Innovation :- 2
(1m)

Hackerrank :- 3
(3m)

Viva :- 1.5
(2m) 9.5

Date 25/04/23

Experiment - 14

Aim

To implement string Matching Algorithm in C program

Algorithm

```
match (st, pat)
    n = length of st
    m = length of pat
    for i=0 to (n-m do
        Count=0
      for j=0 to m-1 do
        if st[i+j] == pat[j] then
                count = count+1
        end if
    end for
    if count == m then
        return i+1
    end if
    end fo
    return -1
end match
```

Code :-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int match(char st[], char pat[]);
```

```
int main() {
    char st[100], pat[100];
    int status;
    printf("Enter the string:");
    st[strcspn(st,"\n")] = '\0';
    pat[strcspn(pat,"\n")] = '\0';
    status = match(st, pat);
    if (status == -1) {
        printf("No match found");
    }
    else {
        printf("match has found at %d position", status);
    }
    return 0; }
int match(char st[], char pat[]) {
    int n, m, i, j, count=0, temp=0;
    n = strlen(st);
    m = strlen(pat);
    for(i=0; i<=n; i++) {
        temp++;
        for(j=0; j<m; j++) {
            if(st[i+j] == pat[j]) {
                count++; }}
        if(count == m) {
            return temp; }
        count = 0;
    }
    return -1;
}
```

Output

Enter the String : temp.dharmendra@gmail.com

Enter the pattern to match: dhar

Match has been found on 6 position

Result

Hence Successfully executed code for implemented of string matching Algorithm

25/4

Coding Standards :- 2
(2m)

Program Explanation :- 2
(2m)

Innovation :- 1
(1m)

Hackerrank :- 3
(3m)

Viva :- 1.5
(2m) 9.5

Date - 28/04/25 Experiment - 15

Problem Statement

Sorting an array of integer in ascending order

Algorithm

Bubble Sort

Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. This algorithm a time complexity of $O(n^2)$

Step1 - Start iterating through the array from the first element

Step2 - Compare the first element with second element If the first element is greater than the second element, swap them

Step3: move to the next pair of adjacent element and repeat step 2

Step 4:- Repeat this process until the entire array is sorted.

Merge Sort

Merge sort is a divide and conquer algorithm that divides the input array into two halves calls itself for the two halves, this algorithm has a time complexity of $O(n \log n)$

Step 1:- Divide the input array into two halves
Step 2:- Call merge sort recursively for the two halves
Step 3:- Merge the two sorted halves into a single sorted array

Time Complexity Analysis

Bubble Sort has a worst-case time complexity of $O(n^2)$ and a best case time complexity of $O(n)$

Merge Sort on the other hand, has a worst case and average-case time complexity of $O(n \log n)$, making it more efficient than bubble sort for large arrays

However merge sort requires additional space for merging the two halves, making it less space efficient than Bubble Sort

25/4

In conclusion, while bubble sort is simple and easy to implement, it is less efficient for larger arrays compared to merge sort. Merge sort, on the other hand, is more efficient but requires additional space for merging the two halves, The choice of algorithm so depends on the size of the input array and the available resources.

Coding standard (2m) :- 2

Program Explanation (2m) :- 2

Innovation (1m) :- 1

Hackerrank (3m) :- 3

Viva (2m) :- 2

10