

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

VARUN R (1BM24CS430)

**in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
February-May 2025**

**B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering**



This is to certify that the Lab work entitled “**ANALYSIS AND DESIGN OF ALGORITHMS**” carried out by VARUN R (1BM24CS430), who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Analysis and Design of Algorithms Lab - (23CS4PCADA) work prescribed for the said degree.

Prof. Supreeth S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Write program to obtain the Topological ordering of vertices in a given digraph. LeetCode Program related to Topological sorting	5-9
2	Implement Johnson Trotter algorithm to generate permutations.	10-12
3	Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. LeetCode Program related to sorting.	13-15
4	Sort a given set of N integer elements using Quick Sort technique and compute its time taken. LeetCode Program related to sorting.	16-19
5	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	20-22
6	Implement 0/1 Knapsack problem using dynamic programming. LeetCode Program related to Knapsack problem or Dynamic Programming.	23-25
7	Implement All Pair Shortest paths problem using Floyd's algorithm. LeetCode Program related to shortest distance calculation.	25-30
8	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.	31-36
9	Implement Fractional Knapsack using Greedy technique. LeetCode Program related to Greedy Technique algorithms.	37-40
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	41-42
11	Implement "N-Queens Problem" using Backtracking.	43-44

Course outcomes:

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

Lab program 1:

Write program to obtain the Topological ordering of vertices in a given digraph.

Code:

```
#include <stdio.h>

#define MAX 100

int adj[MAX][MAX], n, visited[MAX], stack[MAX], top = -1;

void dfs(int v) {
    visited[v] = 1;
    for (int i = 0; i < n; i++)
        if (adj[v][i] && !visited[i])
            dfs(i);
    stack[++top] = v;
}

void topologicalSort() {
    for (int i = 0; i < n; i++)
        if (!visited[i])
            dfs(i);
    while (top >= 0)
        printf("%d ", stack[top--]);
}

int main() {
    int edges, u, v;
    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &n, &edges);
```

```

for (int i = 0; i < edges; i++) {
    scanf("%d %d", &u, &v);
    adj[u][v] = 1;
}

printf("Topological Order: ");
topologicalSort();
return 0;
}

```

Result:

```

Enter number of vertices and edges: 6 7
0 1
0 2
2 1
2 3
3 4
3 5
4 5
Topological Order: 0 2 3 4 5 1

```

LEETCODE: Course Schedule

Code:

```

typedef struct Node {
    int course;
    struct Node *next;
} Node;

void addEdge(Node **graph, int src, int dest) {

```

```

Node *newNode = (Node *)malloc(sizeof(Node));
newNode->course = dest;
newNode->next = graph[src];
graph[src] = newNode;
}

bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize, int*
prerequisitesColSize) {
    Node **graph = (Node **)calloc(numCourses, sizeof(Node *));
    int *indegree = (int *)calloc(numCourses, sizeof(int));

    for (int i = 0; i < prerequisitesSize; i++) {

        int course = prerequisites[i][0];
        int pre = prerequisites[i][1];
        addEdge(graph, pre, course);
        indegree[course]++;
    }

    int *queue = (int *)malloc(numCourses * sizeof(int));
    int front = 0, rear = 0;

    for (int i = 0; i < numCourses; i++) {
        if (indegree[i] == 0) {
            queue[rear++] = i;
        }
    }

    int count = 0;
    while (front < rear) {

```

```

int current = queue[front++];
count++;

Node *temp = graph[current];
while (temp != NULL) {
    indegree[temp->course]--;
    if (indegree[temp->course] == 0) {
        queue[rear++] = temp->course;
    }
    temp = temp->next;
}
}

for (int i = 0; i < numCourses; i++) {
    Node *temp = graph[i];
    while (temp != NULL) {
        Node *next = temp->next;
        free(temp);
        temp = next;
    }
}

free(graph);
free(indegree);
free(queue);

return count == numCourses;
}

```


Result:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

numCourses =
2

prerequisites =
[[1,0]]

Output

true

Expected

true

Lab program 2 :

Implement Johnson Trotter algorithm to generate permutations.

Code:

```
#include <stdio.h>

#define SIZE 4

#define LEFT -1
#define RIGHT 1

int isMobile(int a[], int dir[], int i) {
    if (dir[i] == LEFT && i != 0 && a[i] > a[i - 1])
        return 1;
    if (dir[i] == RIGHT && i != SIZE - 1 && a[i] > a[i + 1])
        return 1;
    return 0;
}

int getMobile(int a[], int dir[]) {
    int mobile = 0;
    for (int i = 0; i < SIZE; i++)
        if (isMobile(a, dir, i))
            if (a[i] > a[mobile] || !isMobile(a, dir, mobile))
                mobile = i;
    return isMobile(a, dir, mobile) ? mobile : -1;
}

// Swap elements and their directions
void swap(int *x, int *y) {
    int temp = *x;
```

```

    *x = *y;

    *y = temp;
}

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

void johnsonTrotter() {
    int a[SIZE], dir[SIZE];
    for (int i = 0; i < SIZE; i++) {
        a[i] = i + 1;
        dir[i] = LEFT;
    }

    printPermutation(a);

    while (1) {
        int mobileIndex = getMobile(a, dir);
        if (mobileIndex == -1)
            break;

        int swapWith = mobileIndex + dir[mobileIndex];
        swap(&a[mobileIndex], &a[swapWith]);
        swap(&dir[mobileIndex], &dir[swapWith]);

        for (int i = 0; i < SIZE; i++)
            if (a[i] > a[swapWith])
                dir[i] = -dir[i];
    }
}

```

```
        printPermutation(a);
    }
}
```

```
int main() {
    johnsonTrotter();
    return 0;
}
```

Result:

```
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4
```

Lab program 3 :

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1, n2 = right - mid;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) arr[k++] = (L[i] < R[j]) ? L[i++] : R[j++];
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
```

```

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

int main() {
    int N;
    printf("Enter the size of the array: ");
    scanf("%d", &N);

    int arr[N];
    srand(time(0));
    for (int i = 0; i < N; i++) arr[i] = rand() % 10000;

    printf("\nUnsorted array:\n");
    printArray(arr, N);

    clock_t start = clock();
    mergeSort(arr, 0, N - 1);
    clock_t end = clock();

    printf("\nSorted array:\n");
    printArray(arr, N);

    printf("\nTime taken to sort %d elements: %f seconds\n", N, ((double)(end - start)) /
CLOCKS_PER_SEC);

    return 0;
}

```

Result:

Enter the size of the array: 50

Unsorted array:

4759 7097 5360 5902 1934 1104 5734 5336 2578 3865 7872 3268 3622 1306 2979 6997 1656 940 8543 3040
4018 761 4891 7206 7061 6263 5337 2447 7493 5411 3092 2252 8860 8452 8155 794 5909 241 6130
8487 458 354 8107 4081 1661 1086 7430 3317 8378 5973

Sorted array:

241 354 458 761 794 940 1086 1104 1306 1656 1661 1934 2252 2447 2578 2979 3040 3092 3268 3317 3622
3865 4018 4081 4759 4891 5336 5337 5360 5411 5734 5902 5909 5973 6130 6263 6997 7061 7097 7206
7430 7493 7872 8107 8155 8378 8452 8487 8543 8860

Lab program 4 :

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to swap two elements
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Partition function for Quick Sort
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // pivot element
    int i = (low - 1); // index of smaller element

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

// Quick Sort function
```



```

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high); // partitioning index
        quickSort(arr, low, pi - 1); // Recursively sort left subarray
        quickSort(arr, pi + 1, high); // Recursively sort right subarray
    }
}

// Main function
int main() {
    int n;

    // Input the size of the array
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];

    // Seed the random number generator
    srand(time(0));

    // Generate random numbers for the array
    printf("Generated array: ");
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000; // Generate random numbers between 0 and 999
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Measure time taken for sorting
    clock_t start = clock();

```

```

quickSort(arr, 0, n - 1);

clock_t end = clock();
double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

// Output the sorted array
printf("Sorted array: ");
for (int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
printf("\n");

// Output the time taken
printf("Time taken to sort: %f seconds\n", time_taken);

return 0;
}

```

Result:

```

Enter number of elements: 50
Generated array: 804 802 720 913 672 874 800 214 358 271 901 763 182 97 744 312 557 908 355 545 140 504 793 990 682 632
129 496 73 945 900 113 551 772 820 220 358 341 128 869 789 244 205 265 472 180 312 905 590 978
Sorted array: 73 97 113 128 129 140 180 182 205 214 220 244 265 271 312 312 341 355 358 358 472 496 504 545 551 557 590
632 672 682 720 744 763 772 789 793 800 802 804 820 869 874 900 901 905 908 913 945 978 990

```

LEETCODE: 3Sum

Code:

```

var threeSum = function (nums) {

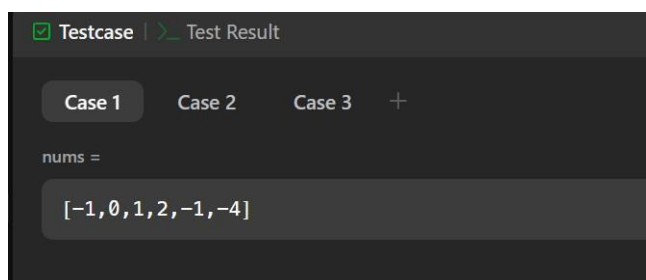
```

```

nums.sort((a, b) => a - b);
const op = [];
for (let i = 0; i < nums.length; i++) {
  if (i > 0 && nums[i] === nums[i - 1]) continue;
  const target = -nums[i];
  let left = i + 1, right = nums.length - 1;
  while (left < right) {
    const current_sum = nums[left] + nums[right];
    if (current_sum === target) {
      op.push([nums[i], nums[left], nums[right]]);
      while (left < right && nums[left] === nums[left + 1]) left++;
      while (left < right && nums[right] === nums[right - 1]) right--;
      left++;
      right--;
    } else if (current_sum < target) {
      left++;
    } else {
      right--;
    }
  }
}
return op;
};

```

Result:



Lab program 5 :

Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


// Function to swap two elements
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}


// Function to heapify a subtree rooted at index i
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1; // left child
    int right = 2 * i + 2; // right child


    // If left child is larger than root
    if (left < n && arr[left] > arr[largest])
        largest = left;


    // If right child is larger than root
    if (right < n && arr[right] > arr[largest])
        largest = right;


    // If largest is not root
    if (largest != i) {
```

```

        swap(&arr[i], &arr[largest]);

        heapify(arr, n, largest); // Recursively heapify the affected subtree
    }
}

```

// Main function to implement heap sort

```

void heapSort(int arr[], int n) {
    // Build a max heap
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    // One by one extract elements from the heap
    for (int i = n - 1; i >= 1; i--) {
        // Move the current root to the end
        swap(&arr[0], &arr[i]);

        // Call heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

```

// Function to print an array

```

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

```

```

int main() {

```

```

// Array to be sorted
int arr[] = {12, 11, 13, 5, 6, 7};
int n = sizeof(arr) / sizeof(arr[0]);

// Start measuring time
clock_t start = clock();

// Perform heap sort
heapSort(arr, n);

// Stop measuring time
clock_t end = clock();

// Print sorted array
printf("Sorted array: ");
printArray(arr, n);

// Compute and print the time taken
double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Time taken to sort: %f seconds\n", time_taken);

return 0;
}

```

Result:

```

Sorted array: 5 6 7 11 12 13
Time taken to sort: 0.000000 seconds

```

Lab program 6 :

Implement 0/1 Knapsack problem using dynamic programming.

Code:

```
#include <stdio.h>

// Function to get the maximum of two numbers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to solve 0/1 Knapsack problem using Dynamic Programming
int knapsack(int W, int wt[], int val[], int n) {
    int dp[n+1][W+1]; // dp[i][w] stores max value for i items and weight limit w

    // Build the dp table in bottom-up manner
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0; // Base case: no items or no weight capacity
            else if (wt[i-1] <= w)
                dp[i][w] = max(val[i-1] + dp[i-1][w - wt[i-1]], dp[i-1][w]);
            // Include the item or skip it, take the max
            else
                dp[i][w] = dp[i-1][w]; // Can't include the item, skip it
        }
    }

    return dp[n][W]; // Return max value for n items and total weight W
}

int main() {
```

```

int val[] = {60, 100, 120}; // Values of items
int wt[] = {10, 20, 30}; // Weights of items
int W = 50; // Maximum weight capacity
int n = sizeof(val)/sizeof(val[0]); // Number of items

int maxVal = knapsack(W, wt, val, n); // Call the knapsack function
printf("Maximum value in knapsack = %d\n", maxVal);

return 0;
}

```

Result:

```
Maximum value in knapsack = 220
```

LEETCODE: Fibonacci Number

Code:

```

int fib(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;

    int a = 0, b = 1, c;

    // Calculate Fibonacci iteratively
    for (int i = 2; i <= n; i++) {
        c = a + b; // Fibonacci relation
        a = b; // Update a to previous b
        b = c; // Update b to current Fibonacci number
    }
}

```



```
    return b; // Return the nth Fibonacci number  
}
```

Result:

☒ Testcase | > Test Result

Case 1

Case 2

Case 3

+

n =

2

Lab program 7 :

Implement All Pair Shortest paths problem using Floyd's algorithm.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#define INF 99999 // A large value representing infinity

void floydWarshall(int graph[][4], int V) {

    // dist[][] will be the output matrix that will store the shortest distance between every pair
    // of vertices

    int dist[V][V];

    // Initialize the distance matrix with the given graph values
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (graph[i][j] == 0 && i != j) {
                dist[i][j] = INF; // No path
            } else {
                dist[i][j] = graph[i][j];
            }
        }
    }

    // Floyd-Warshall algorithm: update the distance matrix
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][j] > dist[i][k] + dist[k][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
}
```

```

    }
}

// Print the shortest distance matrix
printf("The shortest distances between every pair of vertices are:\n");
for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
        if (dist[i][j] == INF) {
            printf("INF ");
        } else {
            printf("%d ", dist[i][j]);
        }
    }
    printf("\n");
}

int main() {
    // Adjacency matrix representation of the graph
    // graph[i][j] = weight of edge from vertex i to vertex j
    // 0 means no edge (in this case, we represent infinity with a large value)

    int graph[4][4] = {
        {0, 3, INF, 7},
        {8, 0, 2, INF},
        {5, INF, 0, 1},
        {2, INF, INF, 0}
    };

    int V = 4; // Number of vertices in the graph

```

```

// Call the Floyd-Warshall algorithm to find shortest paths
floydWarshall(graph, V);

return 0;
}

```

Result:

```

The shortest distances between every pair of vertices are:
0 3 5 6
5 0 2 3
3 6 0 1
2 5 7 0

```

LEETCODE: Shortest Path Visiting All Nodes

Code:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define MAXN 12
#define MAXQ (1 << MAXN) * MAXN

typedef struct {
    int node;
    int mask;
    int dist;
} State;

int shortestPathLength(int** graph, int graphSize, int* graphColSize) {
    int allVisited = (1 << graphSize) - 1;
    bool visited[MAXN][1 << MAXN] = { false };

```

```

State queue[MAXQ];
int front = 0, rear = 0;

// Initialize queue with each node as starting point
for (int i = 0; i < graphSize; i++) {
    int mask = 1 << i;
    queue[rear++] = (State){i, mask, 0};
    visited[i][mask] = true;
}

while (front < rear) {
    State curr = queue[front++];

    if (curr.mask == allVisited) {
        return curr.dist;
    }

    for (int i = 0; i < graphColSize[curr.node]; i++) {
        int neighbor = graph[curr.node][i];
        int nextMask = curr.mask | (1 << neighbor);

        if (!visited[neighbor][nextMask]) {
            visited[neighbor][nextMask] = true;
            queue[rear++] = (State){neighbor, nextMask, curr.dist + 1};
        }
    }
}

return -1; // Should never reach here
}

```

Result:

☒ Testcase | [Test Result](#)

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

graph =
[[1,2,3],[0],[0],[0]]

Output

4

Expected

4

Lab program 8 :

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

Code:

```
#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 5

int minKey(int key[], bool mstSet[]){
    int min= INT_MAX, min_index;

    for(int v=0; v<V; v++){
        if(!mstSet[v] && key[v]<min){
            min = key[v];
            min_index=v;
        }
    }
    return min_index;
}

void primMST(int graph[V][V]){
    int key[V];
    int parent[V];
    bool mstSet[V];

    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    key[0] = 0; // Start from node 0
    parent[0] = -1;
```

```

for (int count = 0; count < V - 1; count++) {
    int u = minKey(key, mstSet); // Pick min weight vertex
    mstSet[u] = true;

    for (int v = 0; v < V; v++)
        if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])
            parent[v] = u, key[v] = graph[u][v];
    }

    printMST(parent, graph);
}

void printMST(int parent[], int graph[V][V]){
    printf("Prim's MST: \n");
    for(int i=1; i<V; i++){
        printf("%d---%d==%d\n", parent[i], i, graph[i][parent[i]]);
    }
}

int main() {
    int graph[V][V] = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}
    };

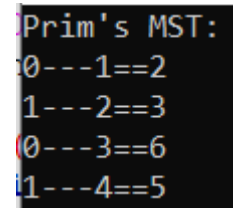
    primMST(graph);
}

```



```
    return 0;
}
```

Result:



```
Prim's MST:
0---1==2
1---2==3
0---3==6
1---4==5
```

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

Code:

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

int find(int v,int parent[10])
{
    while(parent[v]!=v)
    {
        v=parent[v];
    }
    return v;
}

void union1(int i,int j,int parent[10])
{
    if(i<j)
```

```

        parent[j]=i;
else
    parent[i]=j;
}

void kruskal(int n,int a[10][10])
{
    int count,k,min,sum,i,j,t[10][10],u,v,parent[10];
    count=0;
    k=0;
    sum=0;
    for(i=0;i<n;i++)
        parent[i]=i;
    while(count!=n-1)
    {
        min=999;
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)

                if(a[i][j]<min && a[i][j]!=0)
                {
                    min=a[i][j];
                    u=i;
                    v=j;
                }
        }

        i=find(u,parent);
        j=find(v,parent);

```

```

    if(i!=j)
    {
        union1(i,j,parent);
        t[k][0]=u;
        t[k][1]=v;
        k++;
        count++;
        sum=sum+a[u][v];
    }
    a[u][v]=a[v][u]=999;
}
if(count==n-1)
{
    printf("spanning tree\n");
    for(i=0;i<n-1;i++)
    {
        printf("%d %d\n",t[i][0],t[i][1]);
    }
    printf("cost of spanning tree=%d\n",sum);
}
else
    printf("spanning tree does not exist\n");
}

```

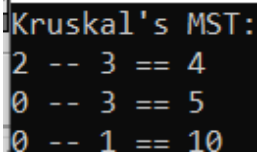
```

void main()
{
    int n,i,j,a[10][10];
    clrscr();
    printf("enter the number of nodes\n");
    scanf("%d",&n);

```

```
printf("enter the adjacency matrix\n");  
for(i=0;i<n;i++)  
    for(j=0;j<n;j++)  
        scanf("%d",&a[i][j]);  
kruskal(n,a);  
getch();  
  
}
```

Result:



```
Kruskal's MST:  
2 -- 3 == 4  
0 -- 3 == 5  
0 -- 1 == 10
```

Lab program 9 :

Implement Fractional Knapsack using Greedy technique.

Code:

```
#include<stdio.h>

int main()
{
    float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
    int n,i,j;
    printf("Enter the number of items :");
    scanf("%d",&n);
    for (i = 0; i < n; i++)
    {
        printf("Enter Weight and Profit for item[%d] :\n",i);
        scanf("%f %f", &weight[i], &profit[i]);
    }
    printf("Enter the capacity of knapsack :\n");
    scanf("%f",&capacity);

    for(i=0;i<n;i++)
        ratio[i]=profit[i]/weight[i];

    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
            if (ratio[i] < ratio[j])
            {
                temp = ratio[j];
                ratio[j] = ratio[i];
                ratio[i] = temp;

                temp = weight[j];
                weight[j] = weight[i];
```

```

        weight[i] = temp;

        temp = profit[j];
        profit[j] = profit[i];
        profit[i] = temp;
    }

    printf("Knapsack problems using Greedy Algorithm:\n");
    for (i = 0; i < n; i++)
    {
        if (weight[i] > capacity)
            break;
        else
        {
            Totalvalue = Totalvalue + profit[i];
            capacity = capacity - weight[i];
        }
    }
    if (i < n)
        Totalvalue = Totalvalue + (ratio[i]*capacity);
    printf("\nThe maximum value is :%f\n",Totalvalue);
    return 0;
}

```

Result:

```
Enter the number of items :4
Enter Weight and Profit for item[0] :
15 2
Enter Weight and Profit for item[1] :
20 1
Enter Weight and Profit for item[2] :
25 3
Enter Weight and Profit for item[3] :
10 3
Enter the capacity of knapsack :
5
Knapsack problems using Greedy Algorithm:
The maximum value is :1.500000
```

LEETCODE: Largest Odd Number in String

Code:

```
char* largestOddNumber(char* num) {
    int len = strlen(num);

    // Traverse from the end to find the rightmost odd digit
    for (int i = len - 1; i >= 0; i--) {
        if ((num[i] - '0') % 2 == 1) {
            // Temporarily terminate the string at the right place
            num[i + 1] = '\0';
            return num;
        }
    }
    return ""; // No odd digit found
}
```

Result:

☒ Testcase | >_ Test Result

Case 1

Case 2

Case 3

+

num =

"52"

Lab program 10 :

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

Code:

```
#include<stdio.h>

#include<conio.h>

void dijkstra(int n,int cost[10][10],int src)
{
    int i,j,u,dis[10],vis[10],min;
    for(i=1;i<=n;i++)
    {
        dis[i]=cost[src][i];
        vis[i]=0;
    }
    vis[src]=1;
    for(i=1;i<=n;i++)
    {
        min=999;
        for(j=1;j<=n;j++)
        {
            if(vis[j]==0 && dis[j]<min)
            {
                min=dis[j];
                u=j;
            }
        }
        vis[u]=1;
        for(j=1;j<=n;j++)
        {
            if(vis[j]==0 && dis[u]+cost[u][j]<dis[j])
```

```

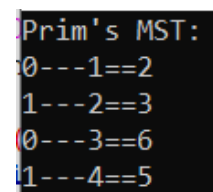
        {
            dis[j]=dis[u]+cost[u][j];
        }
    }
}

printf("shortest path\n");
for(i=1;i<=n;i++)
    printf("%d->%d=%d\n",src,i,dis[i]);
}

void main()
{
    int src,j,cost[10][10],n,i;
    clrscr();
    printf("enter the number of vertices\n");
    scanf("%d",&n);
    printf("enter the cost adjacency matrix\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            scanf("%d",&cost[i][j]);
    printf("enter the source vertex\n");
    scanf("%d",&src);
    dijkstra(n,cost,src);
    getch();
}

```

Result:



```

Prim's MST:
0---1==2
1---2==3
0---3==6
1---4==5

```

Lab program 11 :

Implement “N-Queens Problem” using Backtracking.

Code:

```
#include <stdio.h>

#define N 4

int board[N][N];
int solutionCount = 0;

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

void printBoard() {
    printf("Solution %d:\n", ++solutionCount);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%c ", board[i][j] ? 'Q' : '.');
        printf("\n");
    }
    printf("\n");
}
```

```

void solve(int row) {
    if (row == N) {
        printBoard(); // Found one solution
        return;
    }
    for (int col = 0; col < N; col++) {
        if (isSafe(row, col)) {
            board[row][col] = 1;
            solve(row + 1);    // Try next row
            board[row][col] = 0; // Backtrack
        }
    }
}

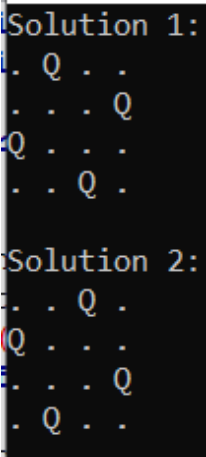
```

```

int main() {
    solve(0);
    if (solutionCount == 0)
        printf("No solution found\n");
    return 0;
}

```

Result:



```

Solution 1:
. Q . .
. . . Q
Q . . .
. . Q .

Solution 2:
. . Q .
Q . . .
. . . Q
. Q . .

```