VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

HITHA HARISH (1BM23CS115)

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 **HITHA HARISH (1BM23CS115)**,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.

RAMYA K M	**Dr. Kavitha Sooda**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

GITHUB LINK : https://github.com/HithaHarish-csbmsce/ADA

ADA 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 1.1:

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

Program full details

Code

```
#include <stdio.h>

#include <stdbool.h>

#define MAX 100


int graph[MAX][MAX];

bool  visited[MAX];  int

stack[MAX]; int top = -

1;

int n;


void push(int v) {

   stack[++top] = v;

}

void dfs(int node) {

   visited[node] = true; for

   (int i = 0; i < n; i++) {

      if (graph[node][i] == 1 && !visited[i]) {

         dfs(i);

      } }

   push(

   node)

   ;

}
```

```
void topologicalSort() {

    for (int i = 0; i < n; i++) {
        visited[i] = false;
    }

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i);
        } }
    printf("Topological Order: "); while
    (top != -1) {
        printf("%d ", stack[top--]);
    } printf("\n");
}

int main() {

    printf("Enter number of vertices: "); scanf("%d",
    &n);
    printf("Enter the adjacency matrix (0 or 1):\n");
    for (int i = 0; i < n; i++) { for (int j = 0; j < n;
    j++) {
            scanf("%d", &graph[i][j]);
        } }
    topologicalSort();
    return 0;
}
```

Screenshot of Output

```
Enter number of vertices: 5
Enter the adjacency matrix (0 or 1):
0 1 0 0 0
0 0 1 1 0
0 0 0 1 0
0 0 0 0 0
0 1 0 0 0
Topological Order: 4 0 1 2 3
```

5. Write a algorithm to obtain a topological ordering of vertices in a given digraph.

Algorithm: DFS Method

Step 1: Select any arbitrary Vertex.

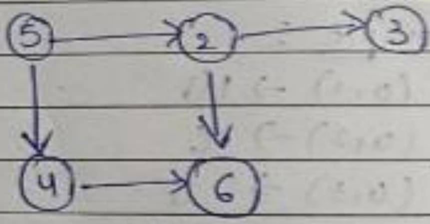
Step 2: When a vertex is visited for the first time, push onto the stack.

Step 3: When a vertex becomes a dead end it is removed from the stack.

Step 4: Repeat step 2 and 3 for all the vertices in the graph.

Step 5: Reverse the order of deleted items to get the topological sequence.

Example:



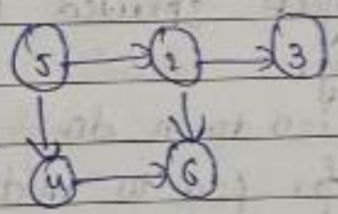
Step	Stack	Adjacent Vertex	Node Visited	Stack [illegible]
initial	5	-	5	-
1	5	2	5, 2	-
2	5, 2	3	5, 2, 3	-
3	5, 2, 3	-	5, 2, 3	3
4	5, 2	6	5, 2, 6, 3	-
5	5, 2, 6	-	5, 2, 3, 6	6
6	5, 2	-	5, 2, 3, 6	2
7	5	4	5, 2, 3, 4, 6	-
8	5, 4	-	5, 2, 3, 4, 6	4
9	5	-	5, 2, 3, 4, 6	5

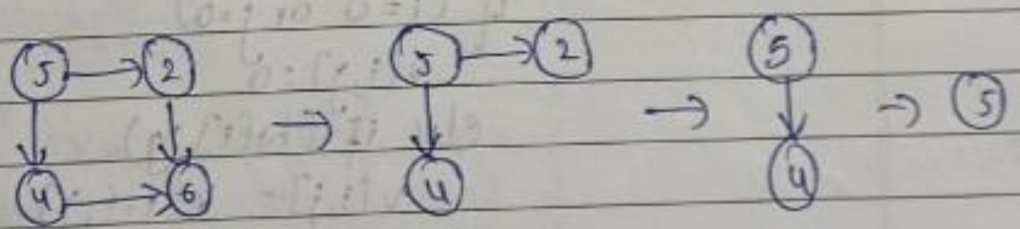
Topological Order :- 5 → 4 → 2 → 6 → 3

Algorithm → Source Method.

```
function TopologicalOrder(G)
  for i=1 to n
     indegree [i] = 0
     for j=1 to n
        indegree[i] = indegree[i] + A[j][i]
  for i=1 to n
     Choose j with indegree (j]= 0
       enumerate j
       indegree [j] = -1
       for k=1 to n
          if A[j][k] ==1
             indegree(k)= indegree[k]-1.
```

Example :-





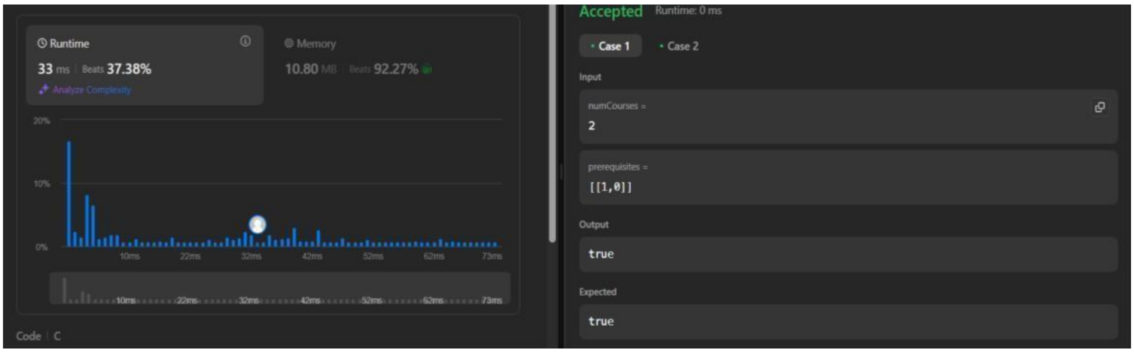
Topological order :- [3, 6, 2, 4, 5]

↓

[5, 4, 2, 6, 3]

Lab program 1.2:

```
class Solution { public:     bool canFinish(int numCourses,
vector<vector<int>>& prerequisites) {        vector<vector<int>>
graph(numCourses);        vector<int> indegree(numCourses, 0);        for
(const auto& pre : prerequisites) {
graph[pre[1]].push_back(pre[0]);           indegree[pre[0]]++;
        }
        queue<int> q;
        for (int i = 0; i < numCourses; ++i) {
if (indegree[i] == 0) q.push(i);
        }        int count = 0;        while
(!q.empty()) {              int curr =
q.front(); q.pop();            count++;
            for (int next : graph[curr]) {
indegree[next]--;
                if (indegree[next] == 0) q.push(next);
            }
        }
        return count == numCourses;
    }
};
```



Lab program 2:

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 right, int mid) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int L[n1], R[n2];

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

    i = 0; j
    = 0; k
    = left;


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

```
    while(i < n1) {
        arr[k] = L[i];
        i++; k++;
    }

    while(j < n2) {
        arr[k] = R[j];
        j++; k++;
    }
}

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, right, mid);
    }
}

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

int main() {
    int n;
    clock_t start, end;
```

```
printf("Enter the number of elements in the array: "); scanf("%d",
&n);

int arr[n];

srand(time(NULL));

for(int i = 0; i < n; i++) { arr[i]
    = rand() % 1000;
}

printf("Original Array: "); print(arr,
n);

start = clock();

mergeSort(arr, 0, n - 1);

end = clock();

printf("Sorted Array: "); print(arr,
n);

printf("Time taken: %f seconds\n",1000* (double)(end - start) / CLOCKS_PER_SEC);
```

```
    return 0;

}
```

Screenshot of Output

```
921 921 921 921 921 921 922 922 922 922 922 922 922 922 922 922 923 923 923 923 923 923 923 924 924 924 924 924 924 924
924 924 924 925 925 925 925 925 925 925 925 925 926 926 926 926 927 927 927 927 927 927 928 928 928 928 928 928 928 928
928 928 929 929 929 929 929 929 929 929 930 930 930 930 930 930 930 930 930 931 931 931 931 931 931 931 931 931 931 931
931 931 932 932 932 932 932 932 932 932 932 933 933 933 933 933 933 933 933 933 933 934 934 934 934 934 934 934 934 934
934 935 935 935 935 935 935 935 935 936 936 936 936 936 936 936 936 936 936 936 936 936 936 937 937 937 937 937 937 937
937 937 937 937 938 938 938 938 938 938 938 938 939 939 939 939 939 939 939 939 939 939 940 940 940 940 940 940 940 940
941 941 941 941 941 941 941 941 941 941 942 942 942 942 942 942 942 942 942 942 943 943 943 943 943 944 944 944 944 944
944 944 944 944 944 944 944 945 945 945 945 945 945 945 945 946 946 946 946 947 947 947 947 947 948 948 948 948 948 948
948 948 948 949 949 949 949 949 949 949 950 950 950 950 950 950 950 950 951 951 951 951 951 951 951 951 951 951 951 952
952 952 952 952 952 952 952 952 952 953 953 953 953 953 953 953 953 953 953 953 954 954 954 954 954 954 954 954 955 955
955 955 955 955 955 955 955 956 956 956 956 956 956 956 956 956 956 956 956 956 956 957 957 957 957 957 957 957 957 957
957 957 957 958 958 958 958 958 958 958 958 959 959 959 959 959 959 959 959 959 959 960 960 960 960 960 960 960 961 961
961 961 961 961 961 961 961 961 961 962 962 962 962 962 962 962 963 963 963 963 963 963 963 963 963 963 963 963 963 964
964 964 964 964 964 964 964 964 964 964 965 965 965 965 965 965 965 965 965 965 966 966 966 966 966 966 966 966 966 966
967 967 967 967 967 967 967 967 967 968 968 968 968 968 968 968 968 968 969 969 969 969 969 969 969 969 969 969 970 970
970 970 970 970 970 970 970 971 971 971 971 971 971 971 971 971 971 972 972 972 972 972 972 972 972 972 972 972 972 973
973 973 973 973 973 973 973 974 974 974 974 974 974 974 974 974 974 975 975 975 975 975 975 975 975 975 975 975 975 975
976 976 976 976 976 976 976 977 977 977 978 978 978 978 978 978 978 978 978 978 979 979 979 979 979 979 979 979 979 979
979 979 979 979 980 980 980 980 980 980 980 980 980 980 980 980 981 981 981 981 981 981 981 981 981 981 981 982 982 982
982 982 982 983 983 983 983 983 983 983 983 983 983 984 984 984 985 985 985 985 985 985 986 986 986 986 986 986 986 986
986 987 987 987 987 987 987 987 987 988 988 988 988 988 988 988 988 988 989 989 989 989 989 989 989 989 989 989 990 990
990 990 990 990 990 991 991 991 991 991 991 991 991 991 991 991 991 992 992 992 992 992 992 992 992 992 992 993 993 993
993 994 994 994 994 994 994 994 994 994 994 994 994 994 994 995 995 995 995 995 995 995 995 995 995 996 996 996 996 996
996 997 997 997 997 997 997 997 997 998 998 998 998 998 998 998 998 998 998 998 998 999 999 999 999 999 999 999 999 999
999 999
Time taken: 1.000000 seconds
```

DATE: 4/04/2025

1. Merge Sort of N integers :-

```
#include <stdio.h>
void merge (int arr[], int low, int mid, int high){
    int i=low, j=mid+1, k=0;
    int temp [high-low+1];
    while (i<=mid && j<=high)
    {
        if (arr[i] < arr[j])
            temp [k++] = arr[i++];
        else
            temp [k++] = arr[j++];
    }
    while (i<=mid){
        temp [k++] = arr[i++];
    }
    while ( j<=high) {
        temp [k++] = arr[j++];
    }
    for (i=low, k=0; i<=high ; i++ ; k++ ) {
        arr[i] = temp[k];
    }
}
Void mergesort (int arr[], int low, int high){
    if (low <high) {
        int mid = (low+high)/2;
        mergesort (arr, low, mid);
        mergesort (arr, mid+1, high);
        merge (arr ,low, mid, high);
    }
}
```

```
void printArray (int arr[], int size) {
    for (int i=0; i<size ; i++) {
        printf ("%d ", arr[i]);
    }
    printf ("\n");
}
int arr[] = { 38, 27, 43, 3, 9, 82, 10 };
int size = sizeof (arr) / sizeof [arr[0]];
printf ("Original array: \n");
printArray ( arr, size);
mergesort (arr, 0, size-1);
printf (" Sorted array: \n");
printArray (arr, size);
return 0;
}
```

Output

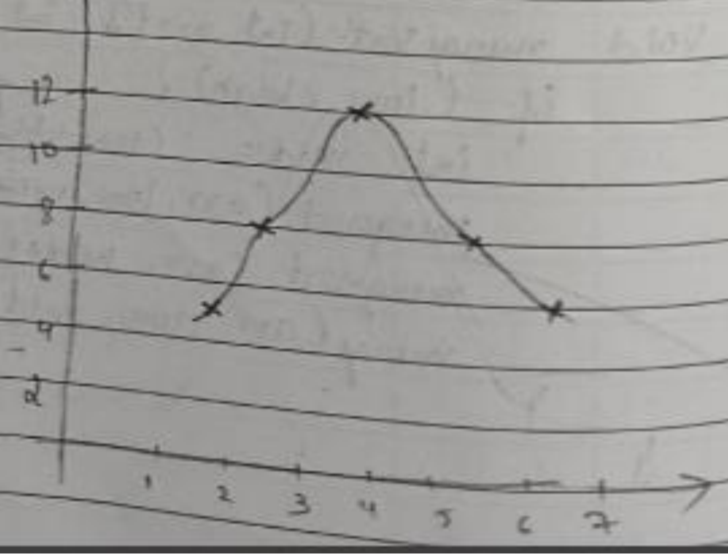
Original array :

38 27 43 3 9 82 10

Sorted array:

3 9 10 27 38 43 82

Graph:



4/4/25

Lab program 3:

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>



int partition(int arr[], int low, int high) { int
   pivot = arr[high];
   int i = low - 1;



   for (int j = low; j <= high - 1; j++) { if
      (arr[j] < pivot) {
          i++;

          int temp = arr[i];
          arr[i] = arr[j];
          arr[j] = temp;
      }
   }



   int temp = arr[i + 1];
   arr[i + 1] = arr[high];
   arr[high] = temp;

   return (i + 1);
}
```

```
void quickSort(int arr[], int low, int high) {
    if (low < high) {

        int pi = partition(arr, low, high);



        quickSort(arr, low, pi - 1); quickSort(arr,
        pi + 1, high);
    }
}


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


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


    int arr[n];
```

```
srand(time(NULL));

for (int i = 0; i < n; i++) {
    arr[i] = rand() % 1001;
}

printf("Original Array: "); print(arr,
n);

start = clock();

quickSort(arr, 0, n - 1);
end = clock();

printf("Sorted Array: "); print(arr,
n);

 printf("Time taken: %f seconds\n",1000* (double)(end - start) / CLOCKS_PER_SEC);

return 0;
```

}

Screenshot of Output

```
 992 992 992 992 992 992 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993
 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993
 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993
 993 993 993 993 993 993 993 993 993 993 993 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994
 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994
 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994
 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994
 994 994 994 994 994 994 994 994 994 994 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995
 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995
 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995
 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 996 996 996 996 996 996
 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996
 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996
 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 997 997
 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997
 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997
 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 998 998 998 998 998
 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998
 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998
 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998
 998 998 998 998 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
 999 999 999 999 999 999 999 999 999 999 999 999 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
Time taken: 10.000000 seconds
```

2. Quick Sort

ALGORITHM

Input : An array of integers nums, and the starting index start and ending index 'end' of the subarray to be sorted.

Output : The subarray nums [start ---- end] sorted in ascending order.

1. Function quick sort (nums, start, end)
2. If start is less than end:
   // Base case : If subarray has more than 1 element.
3. low = start
4. high = end
5. mid = start + (end - start)/2.
6. pivot = nums [mid]
7. while low is <= high.
8. while nums [low] < Pivot
   low++
9. While nums [high] > Pivot
   high --
10. If low <= high
    swap nums[low] and nums [high]
    low ++
    high --
   call quick sort (nums, start, high)
   call quick sort (nums, low, end)
End funct

Main Program

1. Declare an integers array 'arr' with initial values

2. Calculate the length of the array n.

3. Calculate quick sort (arr, 0, n-1) to sort the entire array

4. For each element ~~in~~ the sorted array 'arr':

5. Print element.

<u>Tracing</u>

~~12 4 5 6 7 3 1 9 14 15~~

12 4 5 6 7 3 1 15 14 9 ∞
p i j

(12 > 4 , 12 < ∞)

12 4 5 6 7 3 1 15 14 9 ∞
p i j

(12 > 5 ✓, 12 < 9 ✗

(12

12 4 5 6 7 3 1 15 14 9 ∞
p i j

(12 > 6)

12 4 5 6 7 3 1 15 14 9
i j

12 4 5 6 7 3 1 15 14 9
i j

DATE :

12 4 5 6 7 3 1 15 14 9
p i j

12 4 5 6 7 3 1 15 14 9
p i j

12 4 5 6 7 3 1 15 14 9
p i j

12 4 5 6 7 3 1 9 14 15
p i j

12 4 5 6 7 3 1 9 14 15
p i j

12 4 5 6 7 3 1 9 14 15
p j i

9 4 5 6 7 | 3 1 12 14 15

9 4 5 6 7
p i j

9 4 5 6 7
p i j

9 4 5 6 7
p i j

9 4 5 6 7
j

9 4 5 6 7
j i

7 4 5 6 9 ∞

6 4 5 7 9 ∞

5 4 6 7 9 ∞

4 5 6 7 9

3 1 12 14 15
p i j

3 1 12 14 15
p i j

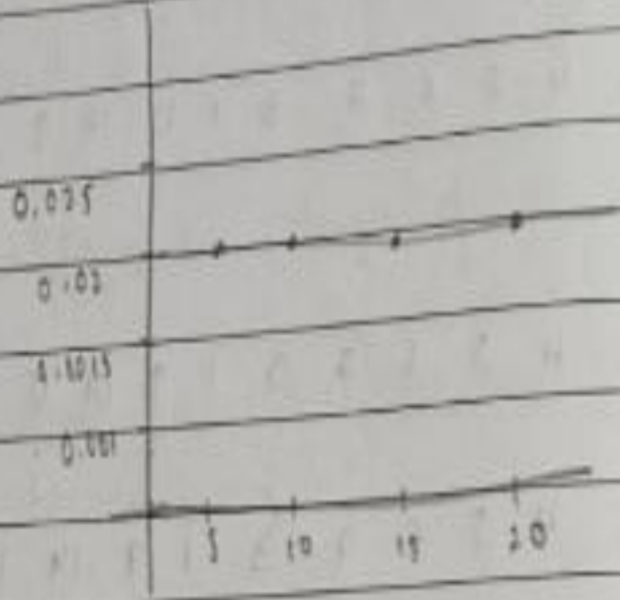
3 1 12 14 15
p i j

3 1 12 14 15
p i
j

3 1 12 14 15
i j i

1 3 12 14 15

Sorted array : 1 3 4 5 6 7 9 12 14 15



N	Time taken.
5	0.000002
10	0.000002
15	0.000002
20	0.00002

Lab program 4:

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

Code

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

int
cost[10][10],vt[10],et[10][10],vis[10],j,n;
int sum=0; int x=1; int e=0;
void prims();


void main()
{
   int i;


   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]);
      } vis[i]=0; } prims();
   printf("edges of spanning tree\n");
   for(i=1;i<=e;i++)
   { printf("%d,%d\t",et[i][0],et[i][1]);
   }
   printf("weight=%d\n",sum); getch();
}


void prims()
{ int s,min,m,k,u,v;
  vt[x]=1;
```

```
  vis[x]=1;
  for(s=1;s<n;s++)
   { j=x;
      min=999;
      while(j>0
      )
      { k=vt[j];
            for(m=2;m<=n;m++)
            {
             if(vis[m]==0)
             { if(cost[k][m]<min)
j--;
   } vt[++x]=v;
   et[s][0]=u;
   et[s][1]=v;
   e++;
   vis[v]=1;
   sum=sum+min;
 }
}
```

Screenshot of Output

```
enter the number of vertices
5
enter the cost adjacency matrix
999 2 999 6 999
2 999 3 8 5
999 3 999 999 7
6 8 999 999 9
999 5 7 9 999
edges of spanning tree
1,2     2,3     2,5     1,4     weight=16
```

3. ~~Steps~~ Prim's Algorithm.

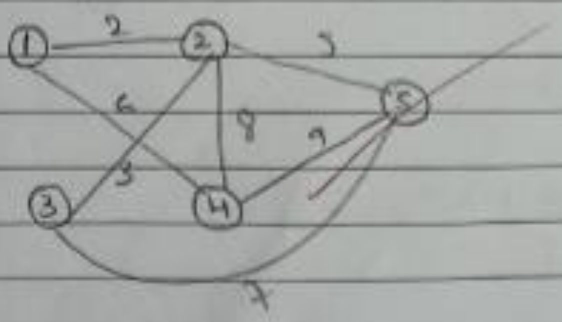
Steps to Prim's Algorithm.

1. Start from any node (usually node 0).
2. Mark it as visited.
3. Find the smallest edge connecting a visited node to an unvisited node.
4. Add that edge to the MST
5. Repeat untill all nodes are Included in the MST.

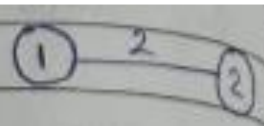
Tracing

Adjacency Matrix.

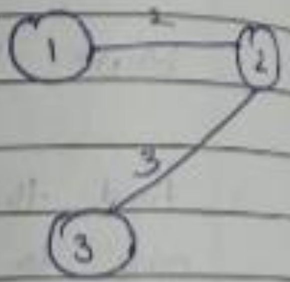
	1	2	3	4	5
1	0	2	0	6	0
2	2	0	3	8	5
3	0	3	0	0	7
4	6	8	0	0	9
5	0	5	7	9	0



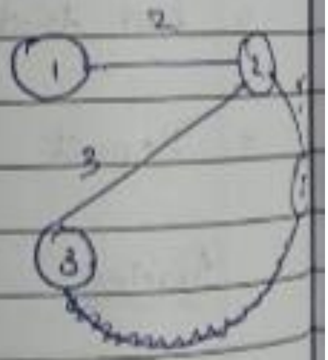
1(–,–)	2(1,2)
	3(–,–)
	4(1,6)
	5(–,–)



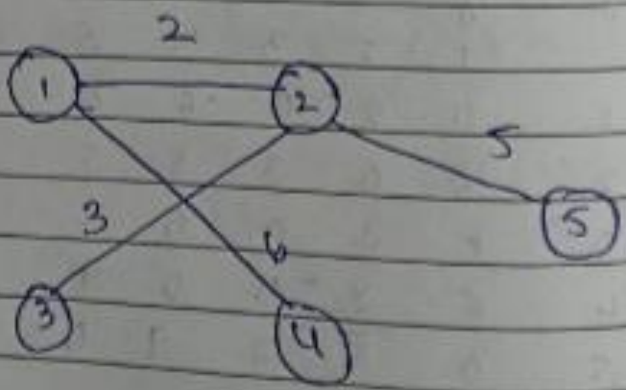
2(1,2)	3(2,3)
	4(1,6)
	5(2,5)



3(2,3)	4(1,6)
	5(2,5)



5(2,5)	4(1,6)



Lab program 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();
}
```

Screenshot of Output

```
enter the number of nodes
8
enter the adjacency matrix
0 2 0 6 0 0 0 0
2 0 3 8 5 0 0 0
0 3 0 0 7 0 0 0
0 8 0 0 9 0 0 0
0 5 7 9 0 4 0 0
0 0 0 0 4 0 2 3
0 0 0 0 0 2 0 6
0 0 0 0 0 3 6 0
spanning tree
0 1
5 6
1 2
5 7
4 5
1 4
0 3
cost of spanning tree=25
```

4) Kruskal's Algorithm:

Steps of Kruskal's Algorithm.

Step : 1 :- Sort all edges in non-decreasing order of their weights.

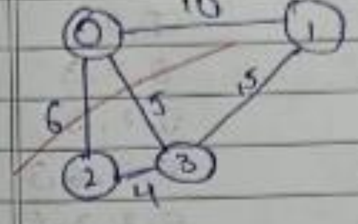
Step : 2 :- Initialize MST as an empty set.

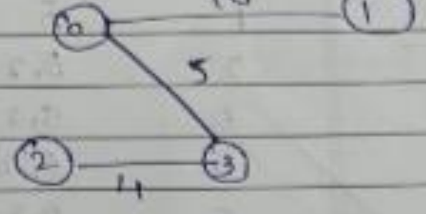
Step : 3 :- For each edge in the sorted list:
- If including the edge doesn't form a cycle, add it to the MST
- Use Disjoint set Union (DSU) or Union-Find to detect cycles.

Step 4 :- Repeat unit MST has (v-1) edges where v is the number of vertices.

Example :

(0,1) → 10	(2,3) → 4 ✓
(0,2) → 6	(0,3) → 5 ✓
(0,3) → 5	(0,2) → 6 ×
(1,3) → 15	(0,1) → 10 ✓
(2,3) → 4	(1,3) → 15 ×





10 + 5 + 4 = 19

Lab program 6:

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

Code

```
#include <stdio.h>
#define INF 999


void dijkstra(int n, int cost[10][10], int src) {
   int i, j, u, dis[10], vis[10], min;


   // Initialize distances and visited flags for
   (i = 1; i <= n; i++) {
      dis[i] = cost[src][i];
      vis[i] = 0;
   }


   vis[src] = 1;


   for (i = 1; i < n; i++) {
      min = INF;
      u = -1;


      // Find the unvisited vertex with the smallest distance
      for (j = 1; j <= n; j++) {
         if (vis[j] == 0 && dis[j] < min) {
            min = dis[j];
            u = j;
         }
      }
```

```
        if (u == -1) break; // All reachable vertices visited


        vis[u] = 1;
        // Update distances to neighboring vertices
        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 paths from vertex %d:\n", src); for
    (i = 1; i <= n; i++) {
        if (dis[i] == INF)
            printf("%d -> %d = INF\n", src, i);
        else
            printf("%d -> %d = %d\n", src, i, dis[i]); }
}


int main() {
    int src, j, cost[10][10], n, i;


    printf("Enter the number of vertices: "); scanf("%d",
    &n);


    printf("Enter the cost adjacency matrix (use 999 for no connection):\n"); for
    (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &cost[i][j]);
        }
```

```
    }

    printf("Enter the source vertex: "); scanf("%d",
    &src);

    dijkstra(n, cost, src);

    return 0;
}
```

Screenshot of Output

```
Enter the number of vertices: 4
Enter the cost adjacency matrix (use 999 for no connection):
0 1 4 999
1 0 2 6
4 2 0 3
999 6 3 0
Enter the source vertex: 1
Shortest paths from vertex 1:
1 -> 1 = 0
1 -> 2 = 1
1 -> 3 = 3
1 -> 4 = 6
```

8) Write a Dijkstra's Algorithm.

// Dijkstra's algorithm for single - source shortest paths.

// Input: A weighted connected graph $G=(V,E)$ with non-negative weights and its vertex s.

// output: The length d_v of a shortest path from s to v, and its penultimate vertex p_v for every vertex v in V.

Initialize (Q)

for every vertex v in V.

$d_v \leftarrow \infty$; $p_v \leftarrow$ null

Insert (Q, v, d_v)

$d_s \leftarrow 0$; Decrease (Q, s, d_s)

$V_T \leftarrow \phi$

for $i \leftarrow 0$ to $|V|-1$ do

$u^* \leftarrow$ DeleteMin(Q)

$V_T \leftarrow V_T \cup \{u^*\}$

for every vertex u in $V - V_T$ that is adjacent to u^* do

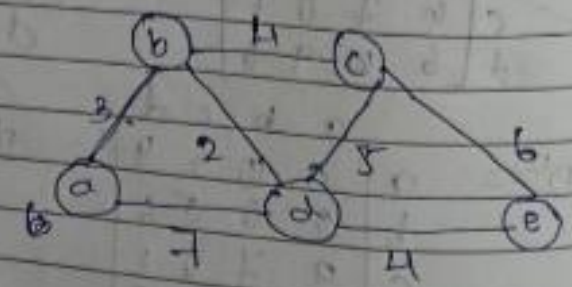
if $d_{u^*} + w(u^*, u) < d_u$

$d_u \leftarrow d_{u^*} + w(u^*, u)$;

$p_u \leftarrow u^*$;

Decrease (Q, u, d_u).

Example:



Tree vertices	Remaining vertices.
a(-, -)	b(a, 3) c(-, ∞) d(a, 7) e(-, ∞)
b(a, 3)	c(b, 7) d(b, 5) e(-, ∞)
d(b, 5)	c(b, 7) e(d, 9)
c(b, 7)	e(d, 9)

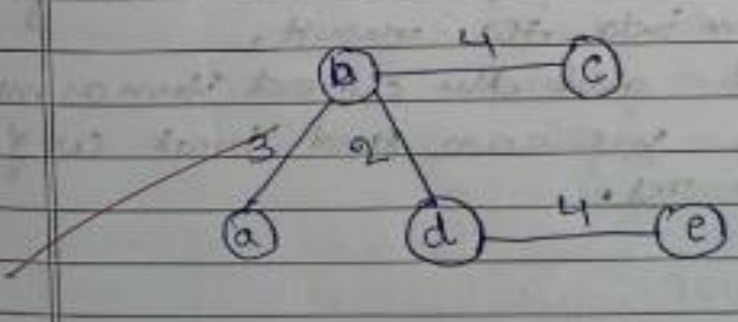
$\therefore$ e(d, 9)

$\Rightarrow$ $a \to b = 3$

$a \to b \to d = 5$

$a \to b \to c = 7$

$a \to b \to d \to e = 9$



Lab program 7:

Implement Johnson Trotter algorithm to generate permutations.

Code

```
#include <stdio.h>


#define LEFT_TO_RIGHT 1

#define RIGHT_TO_LEFT 0
int searchArr(int a[], int n, int mobile) {
   for (int i = 0; i < n; i++)

      if (a[i] == mobile)

         return i + 1;

   return -1;

}


int getMobile(int a[], int dir[], int n) {

   int mobile_prev = 0, mobile = 0;


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

      if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {

         if (a[i] > a[i - 1] && a[i] > mobile_prev) {

            mobile = a[i];

            mobile_prev = mobile;

         } }

      if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {

         if (a[i] > a[i + 1] && a[i] > mobile_prev) {

            mobile = a[i];

            mobile_prev = mobile;

         }

      }

   }
```

```
    return mobile;
}

void printOnePerm(int a[], int dir[], int n) {
    int mobile = getMobile(a, dir, n); int
    pos = searchArr(a, n, mobile);

    if (mobile == 0) return;

    if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT) {
        int temp = a[pos - 1]; a[pos
        - 1] = a[pos - 2]; a[pos - 2]
        = temp;
    } else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT) {
        int temp = a[pos]; a[pos] = a[pos - 1]; a[pos - 1] =
        temp;
    }

    for (int i = 0; i < n; i++) {
        if (a[i] > mobile) {
            dir[a[i] - 1] = !dir[a[i] - 1]; // toggle direction
        }
    }

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

int fact(int n) {
```

```
    int res = 1; for (int i = 1;
    i <= n; i++) res = res * i;

    return res;
}

void printPermutation(int n) {
    int a[n], dir[n];

    for (int i = 0; i < n; i++) {
    a[i] = i + 1; printf("%d",
    a[i]); }
    printf("\n");

    for (int i = 0; i < n; i++)
        dir[i] = RIGHT_TO_LEFT;

    for (int i = 1; i < fact(n); i++)
        printOnePerm(a, dir, n);
}

int main() { int n = 4;
    printPermutation(n);
    return 0;
}
```

Screenshot of Output

12) Implement Johnson Trotter algorithm to generate permutations.

Algorithm.

1) Initialize
  - create an array prem = [1, 2, ..., n]
  - create an array dir of size n, where each elements direction is initially set to left (←).
2. Print the initial permutation.
3. Repeat until no mobile element exists.
  a. Find the largest mobile element exists: k in perm:
    - An element perm[i] is mobile if:
      - Its direction is left and i>0 and perm[i] > perm [i-1]
      - OR its direction is Right and i<n-1 and perm[i] > perm [i+1]
    - Among all mobile elements, select the one with the largest Value.
  b. Swap the largest mobile element k with the adjacent element in the direction it's moving
  c. Reverse the direction of all elements greater than k.
  d. Print the current permutation

DATE :

<u>Example</u>

$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}\ \overleftarrow{4}$	-1	$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}\ \overleftarrow{4}$
$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}\ \overleftarrow{3}$	-2	$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}\ \overleftarrow{3}$
$\overleftarrow{1}\ \overleftarrow{4}\ \overleftarrow{2}\ \overleftarrow{3}$	-3	$\overleftarrow{1}\ \overleftarrow{4}\ \overleftarrow{2}\ \overleftarrow{3}$
$\overleftarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}$	-4	$\overleftarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}$
$\overrightarrow{4}\ \overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}$	-5	$\overrightarrow{4}\ \overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}$
$\overrightarrow{1}\ \overrightarrow{4}\ \overleftarrow{3}\ \overleftarrow{2}$	-6	$\overleftarrow{1}\ \overrightarrow{4}\ \overleftarrow{3}\ \overleftarrow{2}$
$\overrightarrow{1}\ \overleftarrow{3}\ \overrightarrow{4}\ \overleftarrow{2}$	-7	$\overleftarrow{1}\ \overleftarrow{3}\ \overrightarrow{4}\ \overleftarrow{2}$
$\overrightarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}$	-8	$\overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}$
$\overleftarrow{3}\ \overrightarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}$	-9	$\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}$
$\overrightarrow{3}\ \overleftarrow{2}\ \overrightarrow{1}\ \overrightarrow{4}$	-10	$\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{4}\ \overleftarrow{2}$
$\overleftarrow{2}\ \overrightarrow{3}\ \overrightarrow{1}\ \overleftarrow{4}$	-11	$\overleftarrow{3}\ \overleftarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}$
$\overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{4}\ \overrightarrow{1}$	-12	$\overleftarrow{4}\ \overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}$
$\overleftarrow{2}\ \overleftarrow{4}\ \overrightarrow{3}\ \overrightarrow{1}$	-13	$\overrightarrow{4}\ \overrightarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}$
$\overleftarrow{4}\ \overleftarrow{2}\ \overrightarrow{3}\ \overrightarrow{1}$	-14	$\overrightarrow{3}\ \overrightarrow{4}\ \overleftarrow{2}\ \overleftarrow{1}$
	15	$\overrightarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}$
	16	$\overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{4}$
	17	$\overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{1}\ \overleftarrow{4}$
	18	$\overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{4}\ \overleftarrow{1}$
	19	$\overleftarrow{2}\ \overleftarrow{4}\ \overrightarrow{3}\ \overleftarrow{1}$
	20	$\overleftarrow{4}\ \overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{1}$
$\overrightarrow{4}\ \overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{3}$	21	$\overrightarrow{4}\ \overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}$
$\overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}\ \overrightarrow{3}$	22	$\overleftarrow{3}\ \overrightarrow{4}\ \overleftarrow{2}\ \overleftarrow{1}$
$\overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{4}\ \overrightarrow{3}$	23	$\overleftarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}$
$\overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{3}\ \overrightarrow{4}$	24	$\overleftarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{4}$

//.

23/5/25

Lab program 8.1:

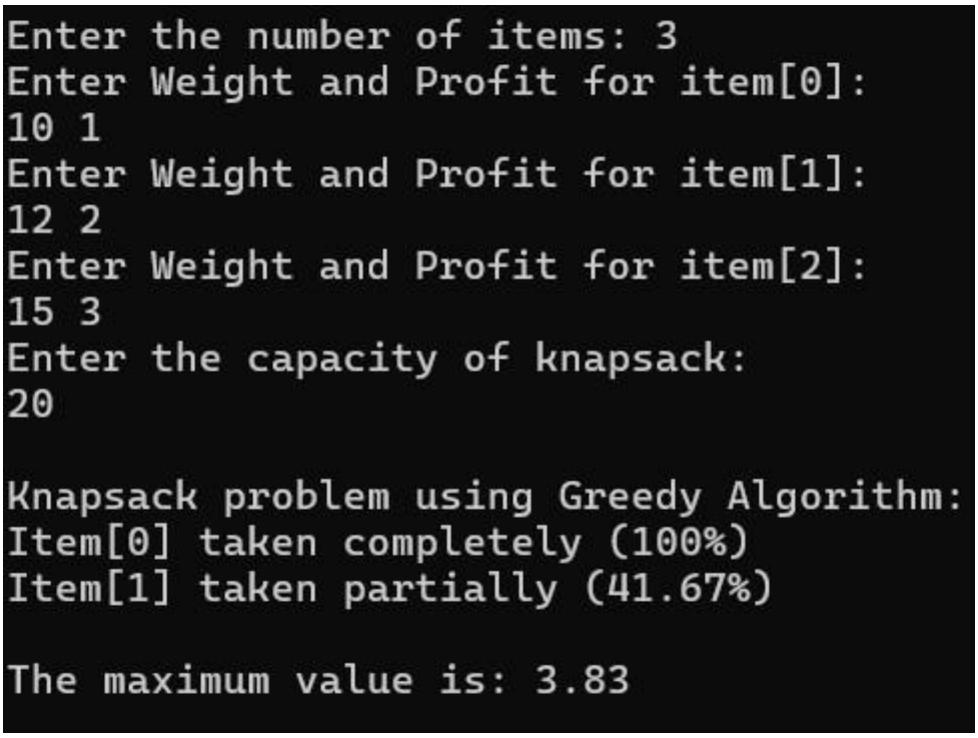
Implement Fractional Knapsack using Greedy technique.
Code

```
#include <stdio.h>


int main() {
   float weight[50], profit[50], ratio[50]; float
   Totalvalue = 0.0, 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);


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


   // Sort items by descending ratio for
   (i = 0; i < n; i++) {
      for (j = i + 1; j < n; j++) { if
         (ratio[i] < ratio[j]) {
```

```
            // Swap ratio
            temp = ratio[i];
            ratio[i] = ratio[j];
            ratio[j] = temp;

            // Swap weight temp
            = weight[i];
            weight[i] =
            weight[j]; weight[j]
            = temp;

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

printf("\nKnapsack problem using Greedy Algorithm:\n");
for (i = 0; i < n; i++) { if (weight[i] <= capacity) { // Take
full item
        printf("Item[%d] taken completely (100%%)\n", i);
        Totalvalue += profit[i]; capacity -= weight[i];
    } else {
        // Take fraction of item float
        fraction = capacity / weight[i];
        Totalvalue += profit[i] * fraction;
        printf("Item[%d] taken partially
```

```
            (%.2f%%)\n", i, fraction * 100);
            break; // Knapsack is now full
        }
    }

    printf("\nThe maximum value is: %.2f\n", Totalvalue); return
    0;
}
```

Screenshot of Output



```
Enter the number of items: 3
Enter Weight and Profit for item[0]:
10 1
Enter Weight and Profit for item[1]:
12 2
Enter Weight and Profit for item[2]:
15 3
Enter the capacity of knapsack:
20

Knapsack problem using Greedy Algorithm:
Item[0] taken completely (100%)
Item[1] taken partially (41.67%)

The maximum value is: 3.83
```

DATE :

9) Implement Fractional Knapsack using Greedy technique.

Given two Arrays, val[] and wt[], representing the values and weights of items, and an integer. Capacity representing the maximum weight a knapsack can hold, the task is to determine the maximum total value that can be achieved by putting items in the knapsack.

Steps by step's approach:

1. Calculate the ratio (profit / weight) for each item
2. Sort all the items is decreasing order of the ratio:
3. Initialize res = 0. current capacity = given capacity.
4. Do the following for every items i in the sorted order:
   - If the weight of the current item is less than or equal to the remaining capacity then add the value of that item into the result,
   - Else add the current item as much as we can and break out of the loop
5. Return res.

17/5/25

DATE :

Example:

For the given set of items and the Knapsack capacity of 10 Kg, find the subset of the items to be added in the knapsack such that the profit is maximum.

Items	1	2	3	4	5
Weight (in kg)	3	3	2	5	1
Profits	10	15	10	12	8
P_i/w_i	3.3	5	5	4	8
P_i/w_i	8	5	5	4	3.3

Item	Value	Weight	P_i/W_i
1	60	10	$60/10 = 6.0$
2	100	20	$100/20 = 5.0$
3	120	30	$120/30 = 4.0$

item 1 :- Capacity $= 50 - 10 = 40$

Total value $= 60$

item 2 :- Capacity $= 40 - 20 = 20$

Total value :- $60 + 100 = 160$

item 3 :- 2/3 of item 3

$20/30 = 2/3$

$120 \times \frac{2}{3} = 80$

$160 + 80 = 240$

∴ Maximum value in the Knapsack :- 240.00

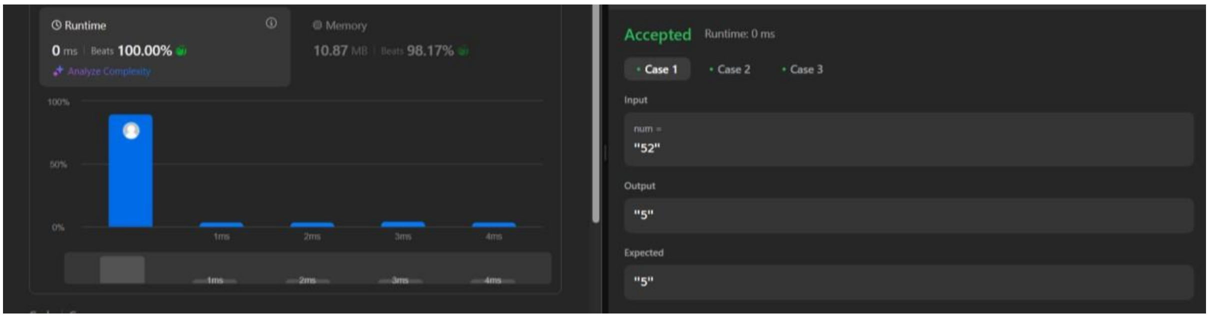
Lab program 8.2:

LeetCode Program related to Greedy Technique algorithms

Code

```
char* largestOddNumber(char* num) {
int len = strlen(num);
for (int i = len - 1; i >= 0; i--) { if ((num[i] - '0') % 2 == 1) {
num[i + 1] = '\0'; // Truncate string at that position return
num; // Return the longest odd-suffix (greedy)
      }
   }


   return ""; // No odd digit found
}
```

Screenshot of Output



Lab program 9.1:

Implement 0/1 Knapsack problem using dynamic programming.

Code

```
#include <stdio.h>


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

```
// Function to solve the 0/1 Knapsack problem
int knapsack(int weight[], int profit[], int n, int capacity) {

    int i, w;

    int K[n + 1][capacity + 1];


    // Build the DP table K[][] bottom up

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

    w <= capacity; w++) { if (i == 0 || w

    == 0)

                K[i][w] = 0; else if

            (weight[i - 1] <= w)

                K[i][w] = max(profit[i - 1] + K[i - 1][w - weight[i - 1]], K[i - 1][w]);

            else

                K[i][w] = K[i - 1][w];

        }

    }


    // Optional: Print the items included

    printf("\nItems included:\n"); w =

    capacity;

    for (i = n; i > 0 && w > 0; i--) { if (K[i][w] != K[i - 1][w]) { printf("Item %d

        (Weight: %d, Profit: %d)\n", i, weight[i - 1], profit[i - 1]); w -= weight[i - 1];

        }

    }


    return K[n][capacity];

}
int main() {
```

```
    int     n,     capacity;     int
    weight[50], profit[50]; int
    i;

    printf("Enter number of items: "); scanf("%d",
    &n);

    printf("Enter weight and profit for each item:\n"); for
    (i = 0; i < n; i++) {
        printf("Item[%d] - Weight Profit: ", i + 1); scanf("%d
        %d", &weight[i], &profit[i]);
    }

    printf("Enter the capacity of knapsack: "); scanf("%d",
    &capacity);

    int maxProfit = knapsack(weight, profit, n, capacity);

    printf("\nMaximum profit: %d\n", maxProfit); return
    0;
}
```

Screenshot of Output

```
Enter number of items: 4
Enter weight and profit for each item:
Item[1] - Weight Profit: 2 12
Item[2] - Weight Profit: 3 15
Item[3] - Weight Profit: 1 25
Item[4] - Weight Profit: 2 10
Enter the capacity of knapsack: 4

Items included:
Item 3 (Weight: 1, Profit: 25)
Item 2 (Weight: 3, Profit: 15)

Maximum profit: 40
```

6) Implement 0/1 Knapsack problem using dynamic programing.

Algorithm :

Aim: To find the Optimal solution for the knapsack problem using Dynamic Programming

Input : n → Number of objects to be selected
m → Capacity of knapsack
w → weights of all the Objects
p → profits of all the Objects.

Output : v → Optimal solution for the number of objects selected with specified remaining capacity.

```
for i=0 to n do
  for j=0 to m do
    if (i=0 or j=0)
      v[i,j]=0
    else if (w[i]>j)
      v[i,j] = v[i-1,j]
    else
      v[i,j] = max[v[i-1,j], v[i-1][j-w[i]]
                                   + p[j]]
    end if
  end for
end for.
```

Optimal Solution in Knapsack

```
for i=0 to n-1 do
        x[i]=0
end for
i=n;  j=m;
while (i!=0 and j!=0)
        if (v[i][j] != v[i-1][j])
                x[i] = 1
                j = j - wi
        end if
        i=i-1
end while
for i=0 to n do
        if x[i]=1
                write 'object selected is';
        end if
end for.
```

Example :

Item	Weight	Profit
1	2	12 $
2	1	10 $
3	3	20 $
4	2	15 $

0.2-2 +12

	wi		0	1	2	3	4	5
	0	0	0	0	0	0	0	0
12	2	1	0	0	12	12	12	12
10	1	2	0	10	12	22	22	22
20	3	3	0	10	12	22	30	32
15	2	4	0	10	15	25	30	37

$(x_1, x_2, x_3, x_4) = (0,0,0,0)$

Step 1 : $V[4,5]$ $V[3,5]$

$37 != 32$

$x[4] = 1$ $(x_1, x_2, x_3, x_4) = (0,0,0,1)$

$j = 5 - 2 = 3$

$i = 3$

step 2 : $V[3,3] != V[2,3]$

$22 != 22$ ×

$i = 3 - 1 = 2$ $(x_1, x_2, x_3, x_4) = (0,0,0,1)$

step : 3 : $V[2,3] != V[1,3]$

$22 != 12$ $i = 2 - 1 = 1$

$x[2] = 1$

$j = 3 - 1 = 2$ $(x_1, x_2, x_3, x_4) = (0,1,0,1)$

step 4 : $V[1,2] != V[0,2]$

$12 != 0$ T

$x[1] = 1$

$j = 2 - 2 = 0$

$i = 1 - 1 = 0$

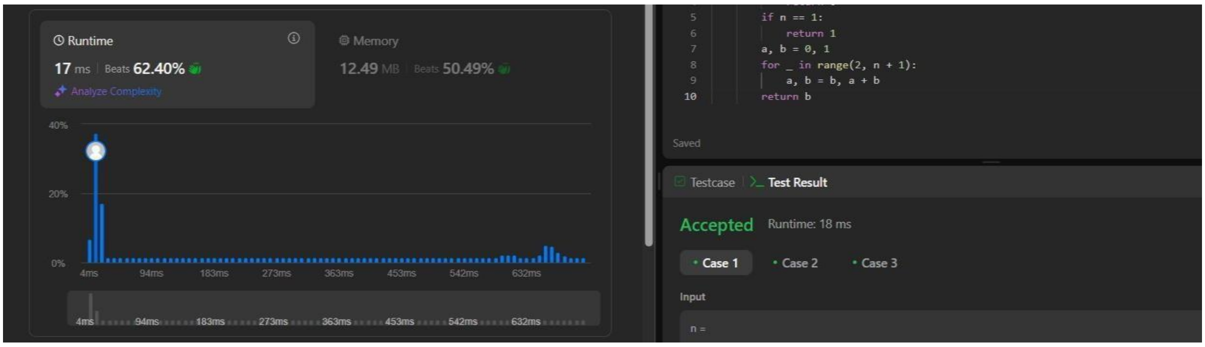
Solution $\langle x_1, x_2, x_3, x_4 \rangle = \langle 1,1,0,1 \rangle$.

Lab program 9.2:

Code

```
class Solution(object):

    def fib(self, n):

        if n == 0:

            return 0

        if n == 1:

            return 1

        a, b = 0, 1 for _ in

        range(2, n + 1): a, b =

        b, a + b

        return b
```

Screenshot of Output



Lab program 10:

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

Code

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


void heapify(int arr[], int n, int i) {
   int largest = i; int left = 2 * i + 1;
   int right = 2 * i + 2;


   if (left < n && arr[left] > arr[largest]) largest
      = left;
```

```
    if (right < n && arr[right] > arr[largest]) largest
        = right;


    if (largest != i) { int
        temp = arr[i]; arr[i]
        = arr[largest];
        arr[largest] = temp;


        heapify(arr, n, largest);
    }
}

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

    for (int i = n - 1; i >= 0; i--) {
        int temp = arr[0]; arr[0] =
        arr[i]; arr[i] = temp;


        heapify(arr, i, 0);
    } }
int main() {
    int arr[1000], n; clock_t
    start, end;
    double time_taken;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d integer elements:\n", n);
    for (int i = 0; i < n; i++)
    scanf("%d", &arr[i]); start =
    clock(); heapSort(arr, n);

    end = clock();
    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("\nSorted array is:\n"); for
    (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    printf("\n\nTime taken by Heap Sort: %f seconds\n", time_taken);

    return 0;
```

}

Screenshot of Output

```
Enter number of elements: 7
Enter 7 integer elements:
50
25
30
75
100
45
80

Sorted array is:
25 30 45 50 75 80 100

Time taken by Heap Sort: 0.000000 seconds
```

11) Sort a given set of N integer element using heap sort technique and compute its time taken. Given Algorithm.

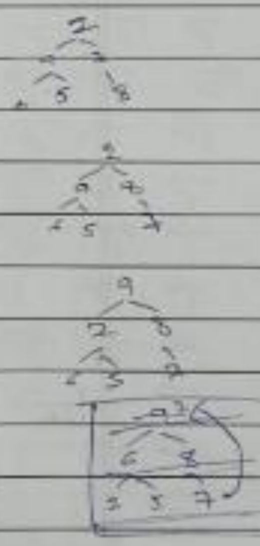
Algorithm

Function: heapSort (arr,n)

1. Build a Max Heap from the input data
2. Repeat
   - Swap the root (maximum element) with the last element
   - Reduce the heap size by 1.
   - Heapify the root to restore heap property.
3. The array is now sorted in ascending order.

Stage 1 (heap construction)

2 9 7 6 5 8
2 9 8 6 5 7
2 9 8 6 5 7
9 2 8 6 5 7
9 6 8 2 5 7



Stage 2 (maximum deletions)

9 6 8 2 5 7
7 6 8 2 5 | 9
8 6 7 2 5
5 6 7 2 | 8
7 6 5 2
2 6 5 | 7
6 2 5
5 2 | 6
5 2
2 | 5
2

Lab program 11.1:

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

Code

```
#include <stdio.h>


#define INF 99999 // Use a large number to represent infinity

#define MAX 100


void floydWarshall(int graph[MAX][MAX], int n) {

  int dist[MAX][MAX];

  int i, j, k;


  // Initialize the solution matrix same as input graph for

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

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

        dist[i][j] = graph[i][j];


  // Floyd-Warshall algorithm for

  (k = 0; k < n; k++) {

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

        for (j = 0; j < n; j++) {

           if (dist[i][k] + dist[k][j] < dist[i][j])

              dist[i][j] = dist[i][k] + dist[k][j];

        }

     }

  }


  // Print the final shortest distance matrix
  printf("\nAll-Pairs Shortest Paths (Floyd-Warshall):\n"); for

  (i = 0; i < n; i++) {
```

```
        for (j = 0; j < n; j++) {
            if (dist[i][j] == INF)
                printf("INF ");
            else
                printf("%3d ", dist[i][j]);
        } printf("\n");
    }
}

int main() {
    int graph[MAX][MAX], n;

    printf("Enter number of vertices: "); scanf("%d",
    &n);

    printf("Enter the adjacency matrix (use 99999 for no direct path):\n"); for
    (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    floydWarshall(graph, n);

    return 0;
}
```

Screenshot of Output

```
Enter number of vertices: 4
Enter the adjacency matrix (use 99999 for no direct path):
0 4 3 9
99 0 1 99
99 990 99999
5 2 6 0
2 99 99999 99999

All-Pairs Shortest Paths (Floyd-Warshall):
  0   4   3   8
  8   0   1   6
  7  11   5   5
  2   6   0   2
```

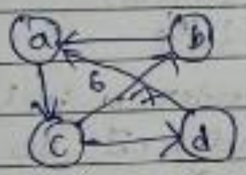
7) Implement All Pair Shortest paths problem using Floy'd algorithm.

// Implements Floyd's algorithm for the all-pairs Shortest - paths problem.
// Input : The Weight matrix W of a graph with no negative - length cycle.
// Output: The distance matrix of the shortest paths' lengths.

D ← W // Is not necessary if W can be overwritten

for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i,j] ← min[D[i,j], D[i,k] + D[k,j]]

return D.



$$D^{(0)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & \infty & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \infty & 0 \end{array}$$

$$D^{(1)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{array}$$

$$D^{(2)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & 9 & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{array}$$

$$D^{(3)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 10 & 3 & 4 \\ b & 2 & 0 & 5 & 6 \\ c & 9 & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \end{array}$$

$$D^{(4)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 10 & 3 & 4 \\ b & 2 & 0 & 5 & 6 \\ c & 7 & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \end{array}$$

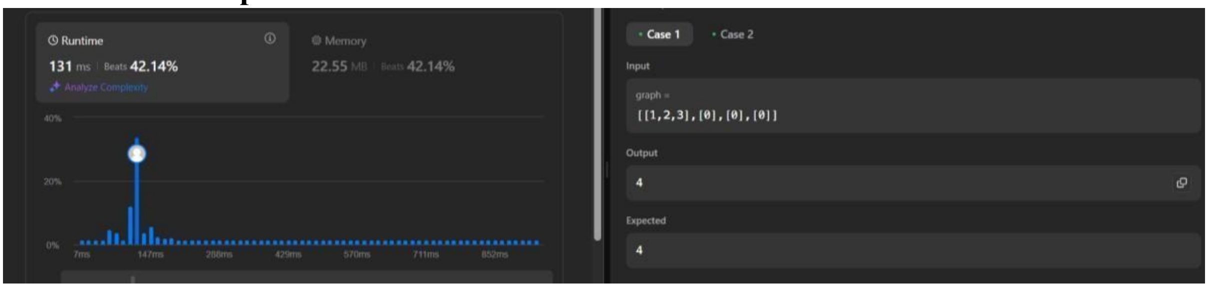
Lab program 11.2:

LeetCode Program related to shortest distance calculation

Code

```
class Solution:
    def shortestPathLength(self, graph: List[List[int]]) -> int:
        n=len(graph)
        queue=deque([(i,1<<i) for i in range(n)])
        seen=set(queue) ans=0 while queue:
            for _ in
                range(len(queue)):
                u,m=queue.popleft() if
                m==(1<<n)-1:
                    return ans
                for v in graph[u]:
                    if (v,m|1<<v) not in seen:
                        queue.append((v,m|1<<v)) seen.add((v,m|1<<v))
            ans+=1
```

Screenshot of Output



Lab program 12:

Implement “N-Queens Problem” using Backtracking.

Code

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

#define MAX 20

int board[MAX];
int found = 0;

// Function to print one solution void
printSolution(int n) { printf("One solution for
%d-Queens:\n", n); for (int i = 1; i <= n; i++)
{ for (int j = 1; j <= n; j++) { if (board[i] == j)
printf("Q "); else
            printf(". ");
  } printf("\n"); }
  found = 1;
}

// Check if placing queen at (k, i) is safe
int isSafe(int k, int i) {
  for (int j = 1; j < k; j++) {
      if (board[j] == i || fabs(board[j] - i) == fabs(j - k))
         return 0;
  } return 1;
}

// Recursive backtracking to find one solution
void nQueens(int k, int n) {
  for (int i = 1; i <= n && !found; i++) {
      if (isSafe(k, i)) {
```

```
            board[k] = i;
            if (k == n)
                printSolution(n);
            else
                nQueens(k + 1, n);
        }
    } }
int main() {
    int n;
    printf("Enter number of queens (N): "); scanf("%d",
    &n);

    if (n < 1 || n > MAX) {
        printf("Please enter N between 1 and %d.\n", MAX); return
        1;
    }

    nQueens(1, n);

    if (!found)
        printf("No solution exists for N = %d\n", n);

    return 0;
}
```

Screenshot of Output

```
Enter number of queens (N): 8
One solution for 8-Queens:
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
```

10) Implement "N-Queens Problem" Using Backtracking.

Algorithm

1. Start with an empty board of size N x N
2. Define a recursive function solve (row) that tries to place a queen in the given row
3. Base Case
   - If row == N all queens are placed successfully
   - print or save the current board configuration
4. Recursive Case:
   For each column col in the current row:
   - Check if placing a queen at position (row, col) is safe.
     - No other queen in the same column.
     - No other queen in the major diagonal (top-left to bottom-right)
     - No other queen in the minor diagonal (top-right to bottom-left).
   - If it safe:
     - Place the queen at (row, col).
     - Call (solve (row+1) recursively to place the queen in the next row.
     - Back track: remove the queen from (row, col) (undo the placement).
5. Repeat Until all rows are processed.

Example:-

Consider 4- Queen Problem
- Each of the four queens has to be placed in its own row, all we need to

do it to assign a column for each queen on the board.