VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

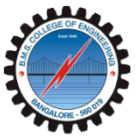
LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Pranav Anantha Rao (1BM2CS201)

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
April-2024 to August-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated to Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **Pranav Anantha Rao (1BM22CS201),** 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 academic semester April-2024 to August-2024. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (23CS4PCADA)** work prescribed for the said degree.

Dr. Nandhini Vineeth	Dr. Jyothi S Nayak
Associate Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

	• Leet Code.	
11	• 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.	36
12	Implement Fractional Knapsack using Greedy technique.	41
13	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	43
14	Implement "N-Queens Problem" using Backtracking.	46

Course Outcome

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.

Program -1

LEET CODE 26 - Remove Duplicates from Sorted Array

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.

Return k.

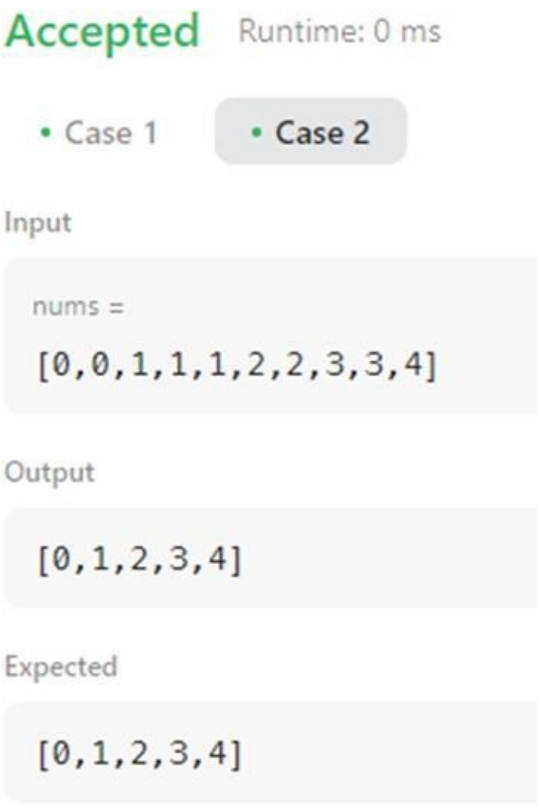
CODE:

```
int removeDuplicates(int* nums, int numsSize){
    int a[100000];
    int n = 0;
    for(int i = 0; i < numsSize; i++){
    int x = nums[i];
    int flag = 0;
    for(int j = 0; j < n; j++){
    if(a[j] == x){
        flag = 1;
        break;
    }
```

```
        }
        if(flag ==
        0){ a[n] = x;
        n++;
        }
        }
        for(int i = 0; i < n; i++){
        nums[i] = a[i];
        }
        return n;
}
```

OUTPUT:


Accepted
Runtime: 0 ms
• Case 1
• Case 2
Input
nums =
[1,1,2]
Output
[1,2]
Expected
[1,2]


Accepted
Runtime: 0 ms
• Case 1
• Case 2
Input
nums =
[0,0,1,1,1,2,2,3,3,4]
Output
[0,1,2,3,4]
Expected
[0,1,2,3,4]

Program -2

LEETCODE 617 – Merge Two Binary Trees

You are given two binary trees root1 and root2.

Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.

Return the merged tree.

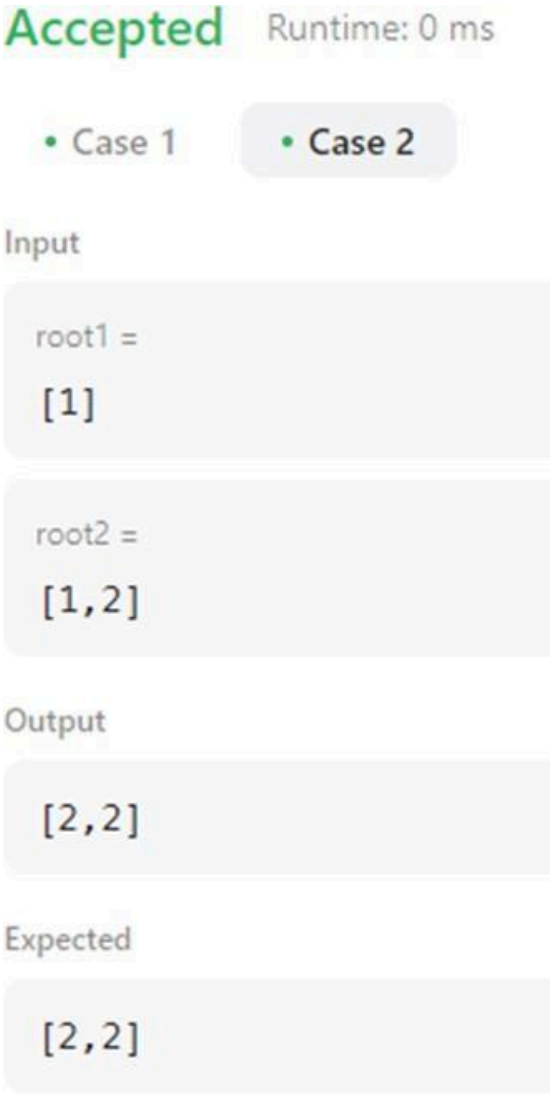
Note: The merging process must start from the root nodes of both trees.

CODE:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
    if(!root1) return root2;
    if(!root2) return root1;
    root1->val +=
    root2->val;
    root1->left = mergeTrees(root1->left, root2->left);
    root1->right = mergeTrees(root1->right,
    root2->right); return root1;
}
```

OUTPUT:


Accepted
Runtime: 0 ms
• Case 1
• Case 2
Input
root1 =
[1,3,2,5]
root2 =
[2,1,3,null,4,null,7]
Output
[3,4,5,5,4,null,7]
Expected
[3,4,5,5,4,null,7]


Accepted
Runtime: 0 ms
• Case 1
• Case 2
Input
root1 =
[1]
root2 =
[1,2]
Output
[2,2]
Expected
[2,2]

Program -3

LEETCODE 653 – Two Sum IV - Input is a BST

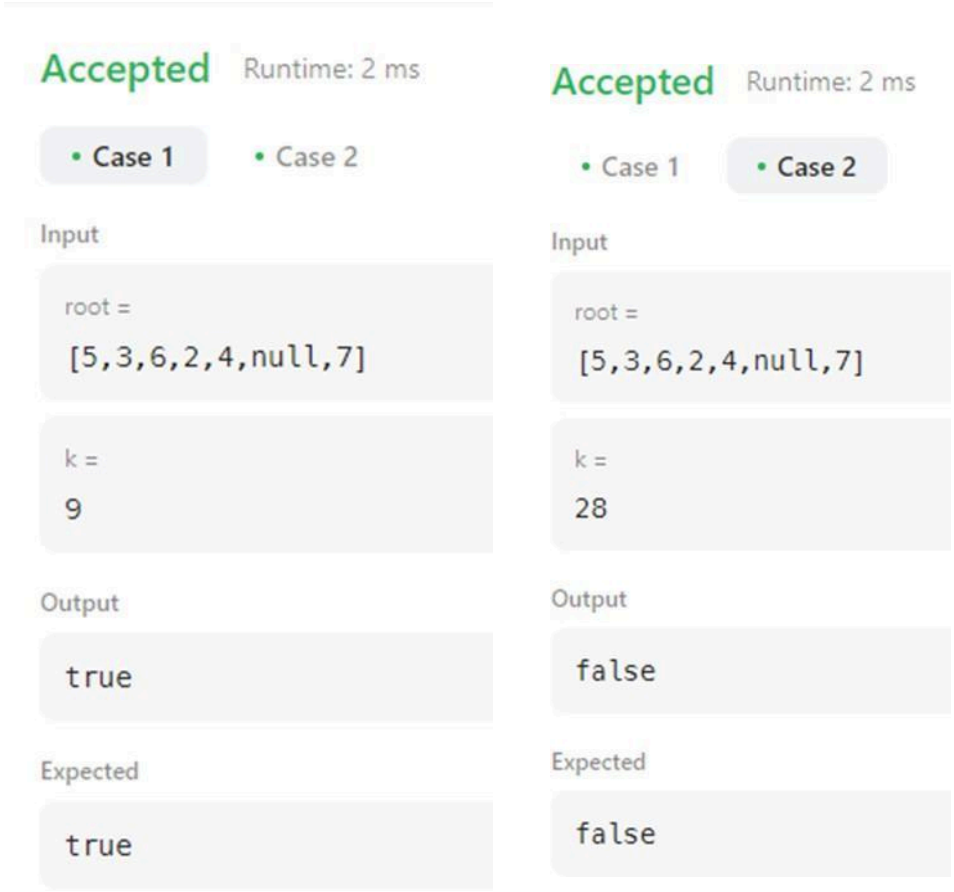
Given the root of a binary search tree and an integer k, return true if there exist two elements in the BST such that their sum is equal to k, or false otherwise.

CODE:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
int v[10001];
int n = 0;
void bst(struct TreeNode*
    root){ if(root == NULL)
    return; bst(root->left);
    v[n] = (root->val);
    n++;
    bst(root->right);
}
bool findTarget(struct TreeNode* root, int k) {
    n = 0;
    bst(root);
    int l = 0;
    int r = n - 1;
    while(l <
    r){
        if(v[l] + v[r] == k){
```

```
            return true;
        }
        else if(v[l] + v[r] < k){ l++;
        }
        else{
            r--;
        }
    }
    return false;
}
```

OUTPUT:



Program -4

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

Code:

Source Removal:

```
#include<stdio.h>


int main(){
    int n;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    int a[n][n];
    printf("Enter adjacency matrix: ");
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            scanf("%d", &a[i][j]);
        }
    }
    int indegree[n];
    for(int j = 0; j < n; j++){
        int sum = 0;
        for(int i = 0; i < n; i++){
            sum += a[i][j];
        }
        indegree[j] = sum;
    }
    int s[n];
    int top = -1;
    for(int i = 0; i < n; i++){
```

```
        if(indegree[i] == 0){
            top++;
            s[top] = i;
        }
    }
    int soln[n];
    int si = 0;
    while(top != -1){
        int u = s[top];
        top--;
        soln[si++] = u;
        for(int i = 0; i < n; i++){
            if(a[u][i] == 1){
                indegree[i]--;
                if(indegree[i] == 0){
                    top++;
                    s[top] = i;
                }
            }
        }
    }
    printf("Soln: \n");
    for(int i = 0; i < n;
    i++){
        printf("%d ", soln[i]);
    }
    return 0;
}
```

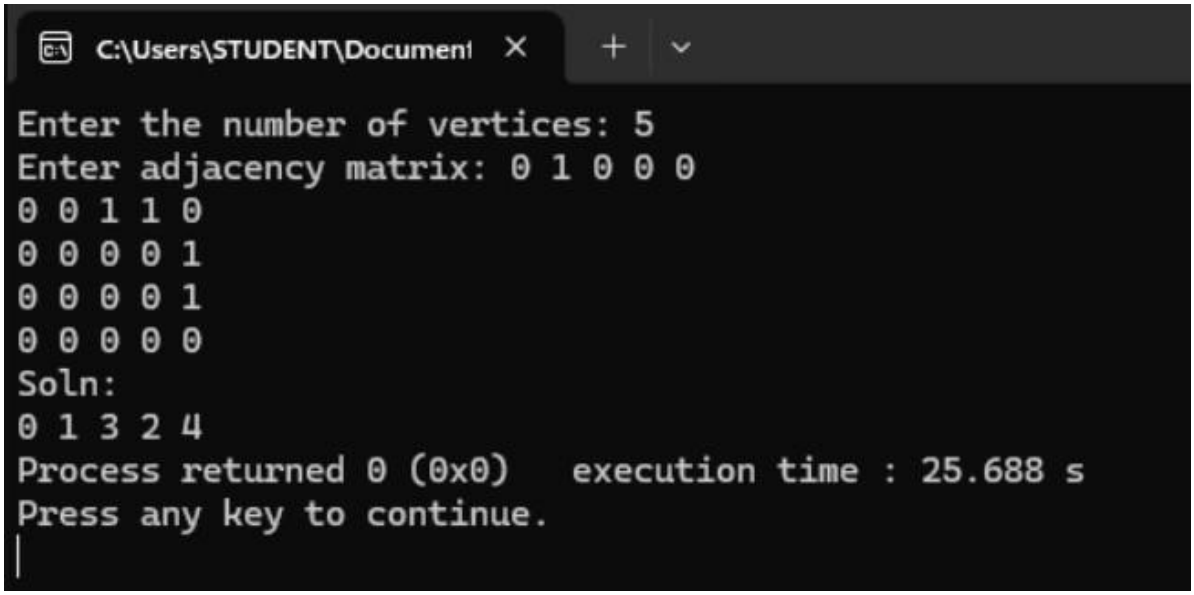
DFS:

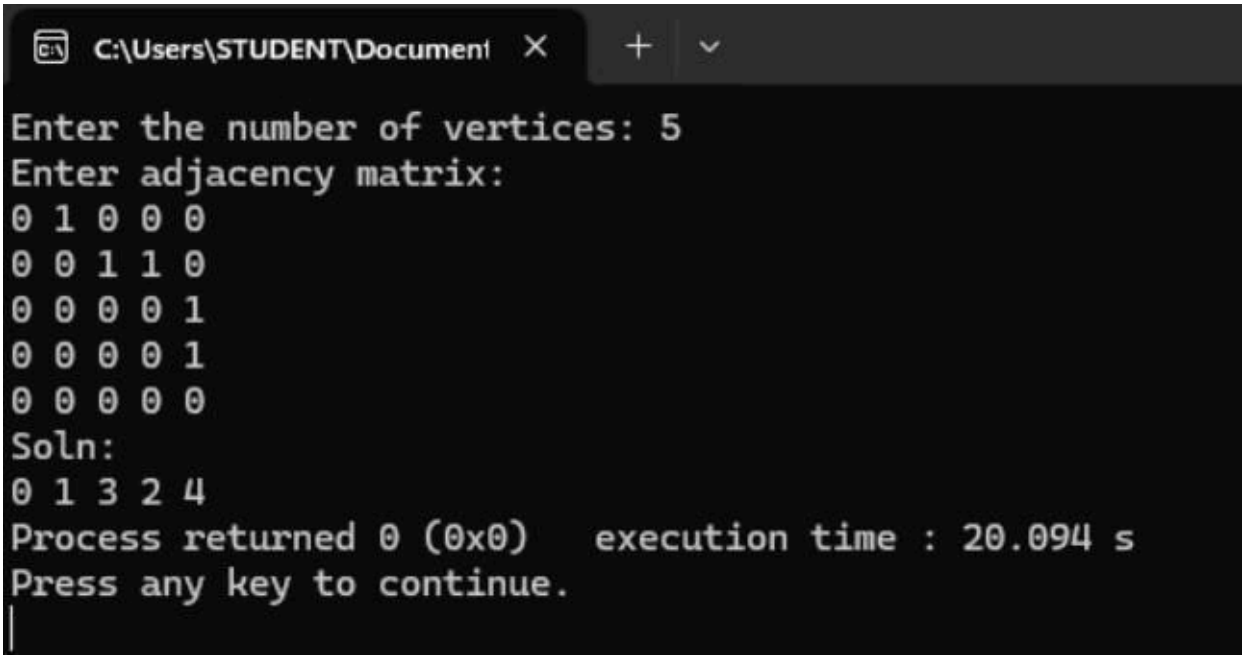
```
#include<stdio.h>
```

```
int s[20];
int j;
int res[20];

void dfs(int u, int n, int a[n][n]){
    s[u] = 1;
    for(int v = 0; v < n; v++){
        if(a[u][v] == 1 && s[v] == 0){
            dfs(v, n, a);
        }
    }
    res[j++] = u;
}

int main(){
    int n;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    int a[n][n];
    printf("Enter adjacency matrix: ");
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            scanf("%d", &a[i][j]);
        }
    }
    for(int i = 0; i < n; i++){
        s[i] = 0;
    }
```

```
    j = 0;
    for(int i = 0; i < n; i++){
        if(s[i] == 0){
            dfs(i, n, a);
        }
    }
    printf("Soln: \n");
    for(int i = n - 1; i >= 0; i--){
        printf("%d ", res[i]);
    }

    return 0;
}
```

Result:




C:\Users\STUDENT\Document
Enter the number of vertices: 5
Enter adjacency matrix:
0 1 0 0 0
0 0 1 1 0
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0
Soln:
0 1 3 2 4
Process returned 0 (0x0)   execution time : 20.094 s
Press any key to continue.

Program -5

Question: Implement Johnson Trotter algorithm to generate permutations

Code:

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

struct  ele{
   int
   num;
   int dir;
};

void main(){
   int n;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   struct ele e[n];
   printf("Enter sorted elements: ");
   for(int i = 0; i < n; i++){
      scanf("%d", &e[i].num);
      e[i].dir = 1;
   }
   while(1){
      for(int i = 0; i < n; i++){
         printf("%d ",
         e[i].num);
      }
```

```
printf("\n");
int curri =
-1;

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

   if(curri == -1 || e[i].num > e[curri].num){
      if(e[i].dir == 1){
         if(i < n - 1 && e[i+1].num < e[i].num){
            curri = i;
         }
      }
      else{
         if(i > 0 && e[i].num > e[i-1].num){
            curri = i;
         }
      }
   }
}
if(curri == -1) break;
struct ele temp = e[curri];
if(e[curri].dir == 1){
   e[curri] = e[curri + 1];
   e[curri + 1] = temp;
}
else{
   e[curri] = e[curri - 1];
   e[curri - 1] = temp;
}
for(int i = 0; i < n; i++){
```

```
        if(e[i].num > temp.num){
           e[i].dir = !e[i].dir;
        }
        }
  }
```

Result:

```
Johnson trotter algorithm to find all permutations of given numbers
Enter the number
4
total permutations = 24
All possible permutations are:
 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
```

Program -6

Question: 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<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with
gcc*/ void split(int[],int,int);
void combine(int[],int,int,int);
void main()
{
  int a[15000],n, i,j,ch, temp;
  clock_t start,end;

  while(1)
  {
printf("\n1:For manual entry of N value and array elements");
printf("\n2:To display time taken for sorting number of elements N in the range 500 to
14500"); printf("\n3:To exit");
    printf("\nEnter your choice:");
    scanf("%d", &ch);
    switch(ch)
    {
     case 1: printf("\nEnter the number of elements: ");
                scanf("%d",&n);
```

```
                printf("\nEnter array elements: ");
                for(i=0;i<n;i++)
                 {
                  scanf("%d",&a[i]);
                 }
                start=clock();
                split(a,0,n-1)
                ;
                end=clock();
                printf("\nSorted array is: ");
                for(i=0;i<n;i++)
                printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f
Secs",n, (((double)(end-start))/CLOCKS_PER_SEC));
                break;
    case 2:
            n=500;
            while(n<=14500) {
            for(i=0;i<n;i++)
                  {
                   //a[i]=random(1000)
                    ; a[i]=n-i;
                  }
            start=clock();
            split(a,0,n-1)
            ;
        //Dummy loop to create delay
           for(j=0;j<500000;j++){
           temp=38/600;}
              end=clock();
```

```
printf("\n Time taken to sort %d numbers is %f
Secs",n, (((double)(end-start))/CLOCKS_PER_SEC));
n=n+1000;
                        }
                break;
   case 3: exit(0);
   }
   getchar();
    }
}
```

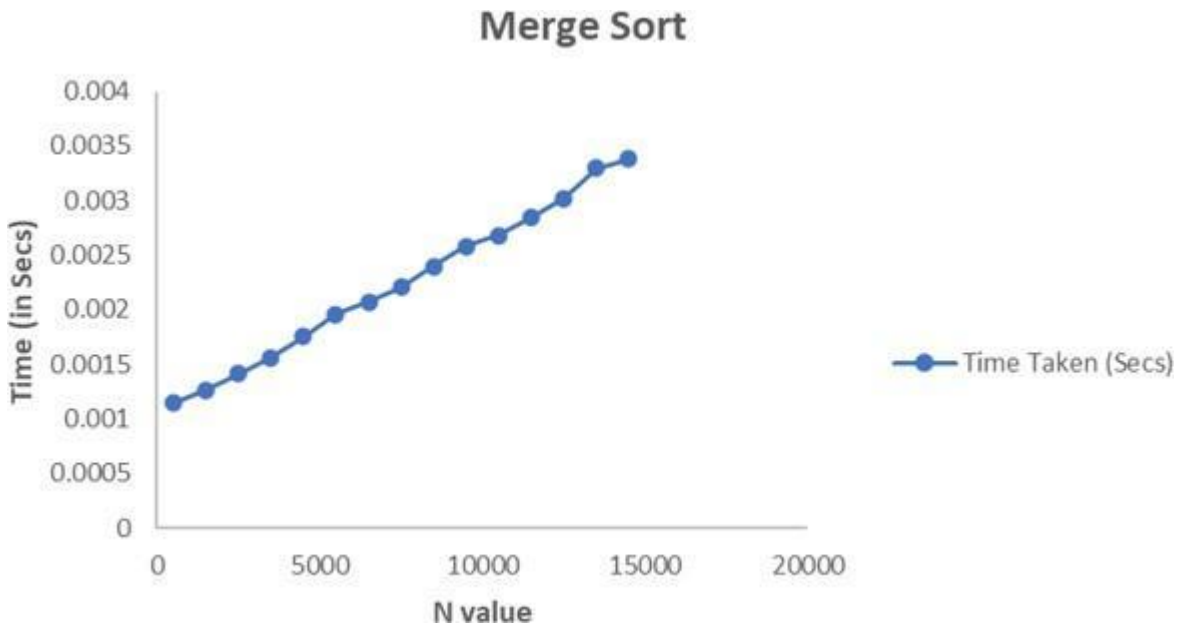
```
void split(int a[],int low,int high)
{
 int mid;
 if(low<high)
 {
  mid=(low+high)/2;
  split(a,low,mid);
  split(a,mid+1,high);
  combine(a,low,mid,high);
 }
}

void combine(int a[],int low,int mid,int high)
{
 int c[15000],i,j,k;
 i=k=low;
```

```
j=mid+1;
while(i<=mid&&j<=high)
{
 if(a[i]<a[j])
 {
  c[k]=a[i];
  ++k;
  ++i;
 }
 else
 {
  c[k]=a[j];
  ++k;
  ++j;
 }
}
if(i>mid)
{
 while(j<=high)
 {
  c[k]=a[j];
  ++k;
  ++j;
 }
}
if(j>high)
{
 while(i<=mid)
```

```
 {
  c[k]=a[i];
  ++k;
  ++i;
 }
}
 for(i=low;i<=high;i++)
 {
  a[i]=c[i];
 }
}
```

Result:

```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1

Enter the number of elements: 5

Enter array elements: 5 4 3 2 1

Sorted array is: 1      2       3       4       5
 Time taken to sort 5 numbers is 0.000000 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:3

Process returned 0 (0x0)   execution time : 10.407 s
Press any key to continue.
```


Merge Sort
0.004
0.0035
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
Time (in Secs)
0
5000
10000
15000
20000
N value
Time Taken (Secs)

Program -7

Question: Sort a given set of N integer elements using Quick Sort technique and compute

its time taken.

Code:

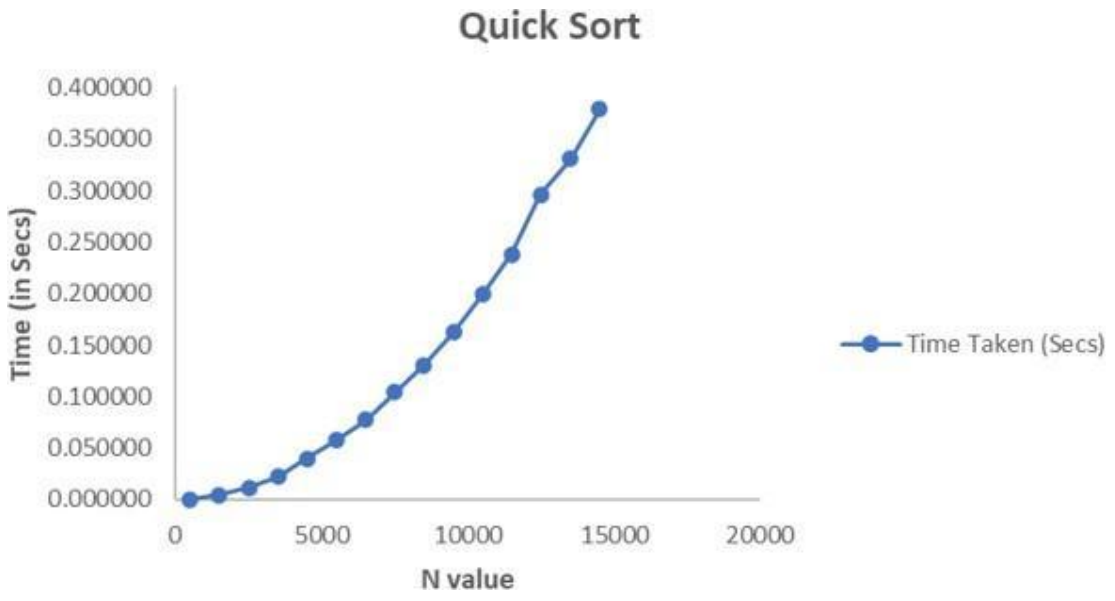
```
#include <stdio.h>
#include <time.h>

void swap(int* a, int* b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

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

int main() {
      int a[15000],n, i,j,ch, temp;
      clock_t start,end;

       while(1)
       {
         printf("\n1:For manual entry of N value and array elements");
         printf("\n2:To display time taken for sorting number of elements N in the range 500
to 14500");
         printf("\n3:To exit");
         printf("\nEnter your choice:");
         scanf("%d", &ch);
         switch(ch)
         {
           case 1: printf("\nEnter the number of elements: ");
                scanf("%d",&n);
                printf("\nEnter array elements: ");
                for(i=0;i<n;i++)
                 {
                  scanf("%d",&a[i]);
                 }
                start=clock();
```

```
            quicksort(a,0,n-1);
            end=clock();
            printf("\nSorted array is: ");
            for(i=0;i<n;i++)
            printf("%d\t",a[i]);
            printf("\n Time taken to sort %d numbers is %f
Secs",n, (((double)(end-start))/CLOCKS_PER_SEC));
            break;
         case 2:
           n=500;
            while(n<=14500) {
              for(i=0;i<n;i++)
               {
                  a[i]=n-i;
               }
              start=clock();
              quicksort(a,0,n-1)
              ;
               for(j=0;j<500000;j++){ temp=38/600;}
              end=clock();
               printf("\n Time taken to sort %d numbers is %f
Secs",n, (((double)(end-start))/CLOCKS_PER_SEC));
                   n=n+1000;
           }
           break;
        case 3: exit(0);
     }
     getchar();
      }
}
```

Result:

```
Enter the number of elements: 5

Enter array elements: 4 1 2 3 5

Sorted array is: 1      2       3       4       5
 Time taken to sort 5 numbers is 0.000001 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2

 Time taken to sort 500 numbers is 0.001701 Secs
 Time taken to sort 1500 numbers is 0.007335 Secs
 Time taken to sort 2500 numbers is 0.016030 Secs
 Time taken to sort 3500 numbers is 0.030166 Secs
 Time taken to sort 4500 numbers is 0.050065 Secs
 Time taken to sort 5500 numbers is 0.072405 Secs
 Time taken to sort 6500 numbers is 0.102483 Secs
 Time taken to sort 7500 numbers is 0.140362 Secs
 Time taken to sort 8500 numbers is 0.175819 Secs
 Time taken to sort 9500 numbers is 0.214041 Secs
 Time taken to sort 10500 numbers is 0.265348 Secs
 Time taken to sort 11500 numbers is 0.315092 Secs
 Time taken to sort 12500 numbers is 0.384209 Secs
 Time taken to sort 13500 numbers is 0.438603 Secs
 Time taken to sort 14500 numbers is 0.503937 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
```



Program -8

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

Code:

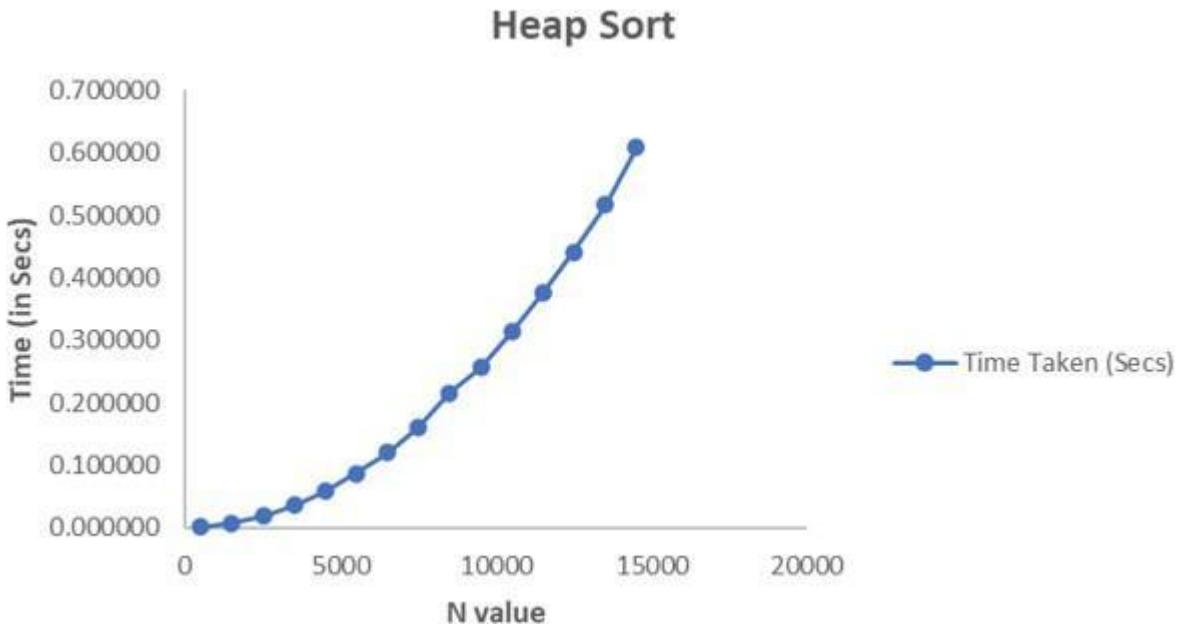
```
#include  <stdio.h>
#include
<stdlib.h>
#include <time.h>

int a[15000];

void heapify(int n){
   for(int p = (n - 1) / 2; p >= 0; p--){
      int item = a[p];
      int c = 2 * p + 1;
      while(c <= n - 1){
         if(c + 1 <= n - 1 && a[c] < a[c + 1]){
            c++;
         }
         if(item >= a[c]){
            break;
         }
         a[p] = a[c];
         p = c;
         c = 2 * p + 1;
      }
      a[p] = item;
   }
}
```

```
void sort(int n){
    heapify(n);
    for(int i = n - 1; i > 0; i--){
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;
        heapify(i);
    }
}

int main(){
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements: \n");
    for(int i = 0; i < n; i++){
        scanf("%d", &a[i]);
    }
    clock_t start, end;
    start = clock();
    sort(n);
    end = clock();
    printf("Sorted: \n");
    for(int i = 0; i < n;
    i++){
        printf("%d ", a[i]);
    }
    printf("\nTime taken to sort %d numbers is %f", n, (double)(end -
start)/CLOCKS_PER_SEC);
```

```
    n = 500;
    while(n <= 14500){
        for(int i = 0; i < n;
        i++){
            a[i] = n - i;
        }
        start = clock();
        sort(n);
        for(int j = 0; j < 500000; j++){
            int temp = 38/600;
        }
        end = clock();
        printf("\nTime taken to sort %d numbers is %f", n, (double)(end -
start)/CLOCKS_PER_SEC);
        n += 1000;
    }
    return 0;
}
```

Result:

```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1

Enter the number of elements: 5

Enter array elements: 9 5 1 6 3

Sorted array is: 1       3       5       6       9
 Time taken to sort 5 numbers is 0.000002 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2

 Time taken to sort 500 numbers is 0.002176 Secs
 Time taken to sort 1500 numbers is 0.010551 Secs
 Time taken to sort 2500 numbers is 0.028930 Secs
 Time taken to sort 3500 numbers is 0.057662 Secs
 Time taken to sort 4500 numbers is 0.091070 Secs
 Time taken to sort 5500 numbers is 0.126118 Secs
 Time taken to sort 6500 numbers is 0.173874 Secs
 Time taken to sort 7500 numbers is 0.235562 Secs
 Time taken to sort 8500 numbers is 0.286235 Secs
 Time taken to sort 9500 numbers is 0.375800 Secs
 Time taken to sort 10500 numbers is 0.443151 Secs
 Time taken to sort 11500 numbers is 0.539377 Secs
 Time taken to sort 12500 numbers is 0.651570 Secs
 Time taken to sort 13500 numbers is 0.864491 Secs
 Time taken to sort 14500 numbers is 0.861665 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:
```



Program -9

Question: Implement 0/1 Knapsack problem using dynamic programming.

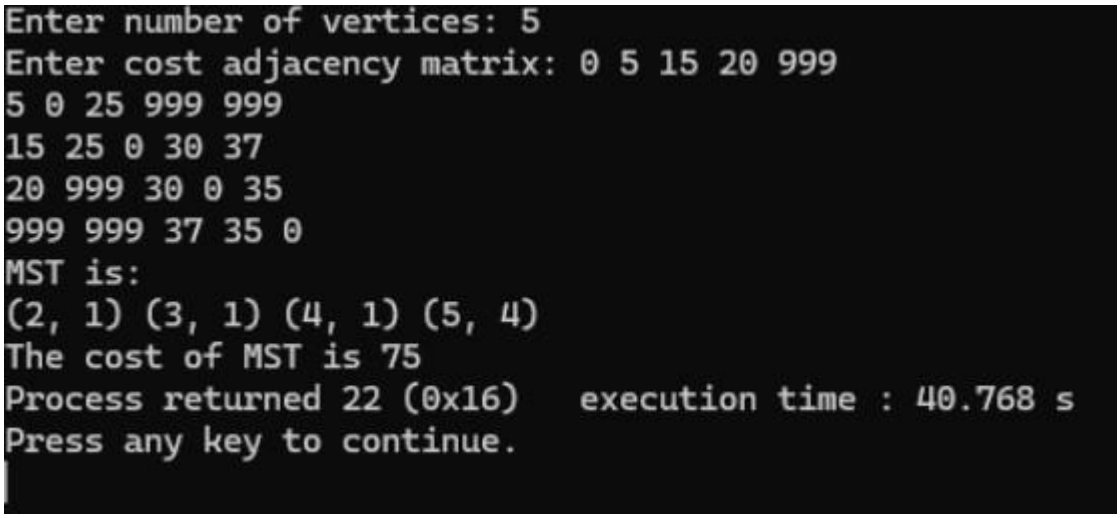
Code:

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

int max(int a, int b){
   if(a > b){
      return a;
   }
   return b;
}

void main(){
   int n;
   printf("Enter number of weights: ");
   scanf("%d", &n);
   int w[n], c[n];
   printf("Enter weights: ");
   for(int i = 0; i < n; i++){
      scanf("%d", &w[i]);
   }
   printf("Enter coins in weights: ");
   for(int i = 0; i < n; i++){
      scanf("%d", &c[i]);
   }
   int bag;
```

```
printf("Enter the capacity of knapsack: ");
scanf("%d", &bag);
int dp[n + 1][bag + 1];
for(int i = 0; i < n + 1;
i++){
    for(int j = 0; j < bag + 1; j++){
        dp[i][j] = 0;
    }
}
for(int i = 1; i < n + 1; i++){
    for(int j = 1; j < bag + 1; j++){
        if(j - w[i - 1] >= 0){
            dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i - 1]] + c[i - 1]);
        }
        else{
            dp[i][j] = dp[i-1][j];
        }
    }
}
for(int i = 0; i < n + 1; i++){
    if(i == 0){
        printf("-\t-\t%d\t", i);
    }
    else{
        printf("%d\t%d\t%d\t", w[i-1], c[i-1], i);
    }
    for(int j = 0; j < bag + 1; j++){
        printf("%d\t", dp[i][j]);
    }
```

```
        printf("\n");
    }
    printf("Maximum possible: %d", dp[n][bag]);
}
```

Result:



Program -10

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

Code:

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

int cost[1000][1000];

void floyd(int n){
   int d[n][n];
   for(int i = 0; i < n; i++){
      for(int j = 0; j < n;
      j++){
         d[i][j] = cost[i][j];
      }
   }
   for(int k = 0; k < n; k++){
      for(int i = 0; i < n;
      i++){
         for(int j = 0; j < n; j++){
            if(d[i][j] > d[i][k] + d[k][j]){
               d[i][j] = d[i][k] + d[k][j];
            }
         }
      }
   }
   printf("Output: \n");
   for(int i = 0; i < n;
```

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

int main(){
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements: \n");
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            scanf("%d",
            &cost[i][j]);
        }
    }
    floyd(n);
    return 0;
}
```

Result:

```
Enter number of elements: 4
Enter elements:
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0
Output:
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0

Process returned 0 (0x0)   execution time : 30.142 s
Press any key to continue.
```

Program -11

Question: 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.

Code:

Prim's:

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

void main(){
   printf("Enter number of vertices: ");
   int n;
   scanf("%d", &n);
   int cost[n][n];
   printf("Enter cost adjacency matrix: ");
   for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
         scanf("%d",
         &cost[i][j]);
      }
   }
   int min = INT_MAX, source = 0;
   for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
         if(cost[i][j] != 0 && cost[i][j] < min){
            min = cost[i][j];
            source = i;
         }
```

```
    }
}
int s[n], d[n], p[n];
for(int i = 0; i < n;
i++){
    s[i] = 0;
    d[i] = cost[source][i];
    p[i] = source;
}
s[source] = 1;
int sum = 0, k = 0, t[n][2];
for(int i = 0; i < n; i++){
    min = INT_MAX;
    int u = -1;
    for(int j = 0; j < n; j++){
        if(s[j] == 0 && d[j] <= min){
            min = d[j];
            u = j;
        }
    }
    if (u == -1) break;

    t[k][0] = u;
    t[k][1] = p[u];
    k++;
    sum += cost[u][p[u]];
    s[u] = 1;
    for(int j = 0; j < n; j++){
        if(s[j] == 0 && cost[u][j] < d[j]){
```

```
            d[j] = cost[u][j];
            p[j] = u;
        }
    }
}
if(sum >= INT_MAX){
    printf("Spanning tree does not
    exist");
}
else{
    printf("MST is:\n");
    for(int i = 0; i < k;
    i++){
        printf("(%d, %d) ", t[i][0] + 1, t[i][1] + 1);
    }
    printf("\nThe cost of MST is %d", sum);
}
}
```

Kruskal's:

```
#include<stdio.h>
void main(){
    int n;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    int c[n][n];
    printf("Enter cost adjacency matrix: \n");
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            scanf("%d", &c[i][j]);
        }
```

```
}
int ne = 0, mincost = 0;
int parent[n];
for(int i = 0; i < n; i++){
   parent[i] = -1;
}
int u = -1, v = -1, a, b;
while(ne != n - 1){
   int min = 9999;
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++)
      {
         if (c[i][j] < min && c[i][j] > 0) {
            int root_u = i;
            int root_v = j;
            while (parent[root_u] != -1) {
               root_u = parent[root_u];
            }
            while (parent[root_v] != -1) {
               root_v = parent[root_v];
            }
            if (root_u != root_v) {
               min = c[i][j];
               u = i;
               v = j;
            }
         }
      }
   }
```

```
        if (u != -1 && v != -1) {
            printf("(%d, %d, %d) ", u + 1, v + 1, min);
            parent[v] = u;
            mincost += min;
            ne++;
        }
        c[a][b] = 9999;
        c[b][a] = 9999;
    }
    printf("Mincost: %d", mincost);
}
```

Result:

```
Enter number of weights: 4
Enter weights: 2 1 3 2
Enter coins in weights: 12 10 20 15
Enter the capacity of knapsack: 5
-       -       0       0       0       0       0       0       0
2       12      1       0       0       12      12      12      12
1       10      2       0       10      12      22      22      22
3       20      3       0       10      12      22      30      32
2       15      4       0       10      15      25      30      37
Maximum possible: 37
Process returned 20 (0x14)   execution time : 8.422 s
Press any key to continue.
```

```
Enter the number of nodes: 5
Enter cost adjacency matrix:
0 5 15 20 999
5 0 25 999 999
15 25 0 30 37
20 999 30 0 35
999 999 37 35 0
(1, 2, 5) (1, 3, 15) (1, 4, 20) (4, 5, 35) Mincost: 75
Process returned 11 (0xB)   execution time : 32.205 s
Press any key to continue.
```

Program -12

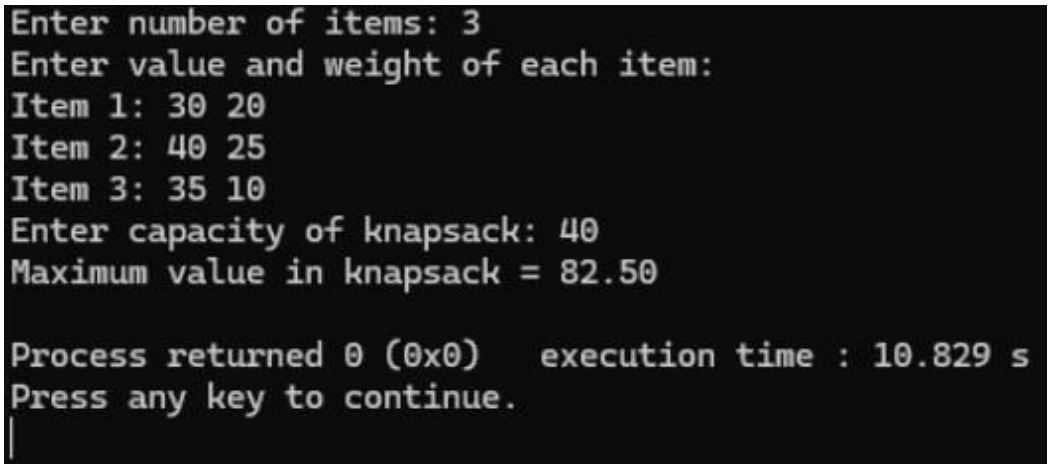
Question: Implement fractional Knapsack problem using Greedy technique

Code:

```
#include  <stdio.h>
#include
<stdlib.h>      struct
Item {
    int   value;
    int
    weight;
};
int compare(const void *a, const void *b) {
    double ratio1 = (double)(((struct Item *)a)->value) / (((struct Item *)a)->weight);
    double ratio2 = (double)(((struct Item *)b)->value) / (((struct Item *)b)->weight);
    return (ratio2 > ratio1) - (ratio2 < ratio1);
}
int main() {
    int n;
    printf("Enter number of items: ");
    scanf("%d", &n);
    struct Item items[n];
    printf("Enter value and weight of each item:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d: ", i + 1);
        scanf("%d %d", &items[i].value, &items[i].weight);
    }
    int W;
    printf("Enter capacity of knapsack: ");
```

```
    scanf("%d", &W);

    qsort(items, n, sizeof(items[0]),
    compare); int currentWeight = 0;
    double finalValue = 0.0;
    for (int i = 0; i < n; i++)
    {
        if (currentWeight + items[i].weight <= W)
            { currentWeight += items[i].weight;
            finalValue += items[i].value;
        } else {
            int remainingWeight = W - currentWeight;
            finalValue += items[i].value * ((double)remainingWeight /
            items[i].weight); break;
        }
    }
    printf("Maximum value in knapsack = %.2f\n",
    finalValue); return 0;
}
```

Result:



Program -13

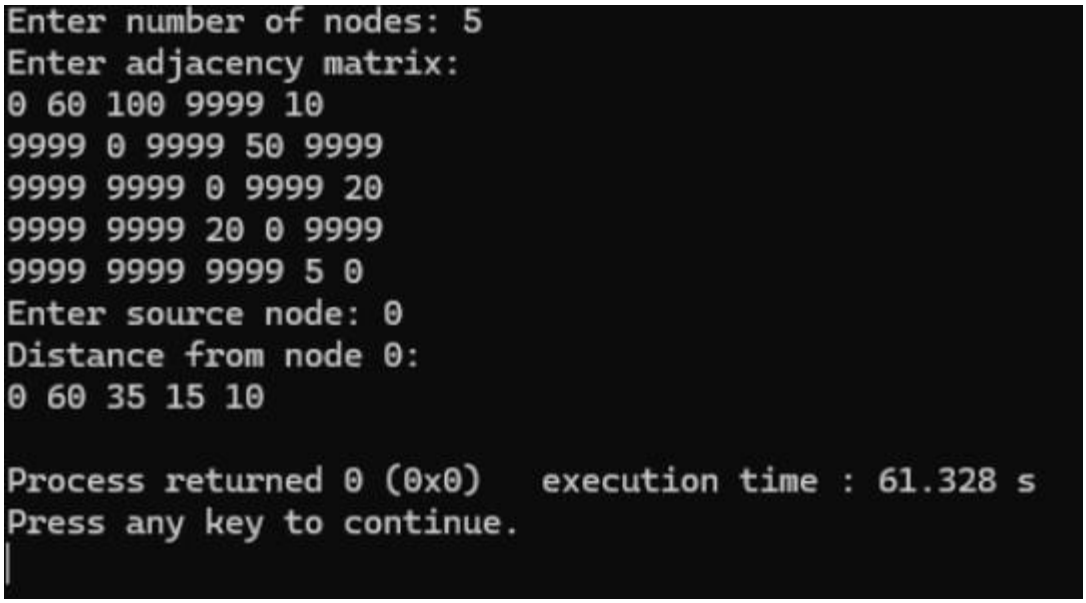
Question: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

Code:

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

int main() {
    printf("Enter number of nodes: ");
    int n;
    scanf("%d", &n);
    int g[n][n];
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &g[i][j]);
        }
    }
    int s;
    printf("Enter source node: ");
    scanf("%d", &s);
    int d[n];
    int v[n];
    for (int i = 0; i < n; i++) {
        d[i] = INT_MAX;
        v[i] = 0;
    }
    d[s] = 0;
```

```
    for (int count = 0; count < n - 1; count++) {
        int u = -1;
        for (int i = 0; i < n; i++) {
            if (!v[i] && (u == -1 || d[i] < d[u])) {
                u = i;
            }
        }
        if (d[u] == INT_MAX) break;
        v[u] = 1;
        for (int i = 0; i < n; i++) {
            if (g[u][i] && !v[i] && d[u] != INT_MAX && d[u] + g[u][i] < d[i]) {
                d[i] = d[u] + g[u][i];
            }
        }
    }
    printf("Distance from node %d:\n", s);
    for (int i = 0; i < n; i++) {
        if (d[i] == INT_MAX) {
            printf("INF ");
        } else {
            printf("%d ", d[i]);
        }
    }
    printf("\n");
    return 0;
}
```

Result:


Enter number of nodes: 5
Enter adjacency matrix:
0 60 100 9999 10
9999 0 9999 50 9999
9999 9999 0 9999 20
9999 9999 20 0 9999
9999 9999 9999 5 0
Enter source node: 0
Distance from node 0:
0 60 35 15 10
Process returned 0 (0x0) execution time : 61.328 s
Press any key to continue.

Program -14

Question: Implement “N-Queens Problem” using Backtracking.

Code:

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

#define MAX 100

int place(int x[], int k) {
    for (int i = 1; i < k; i++) {
        if (x[i] == x[k] || abs(x[i] - x[k]) == abs(i - k)) {
            return 0;
        }
    }
    return 1;
}

void nqueens(int n) {
    int x[MAX];
    int k = 1;
    x[k] = 0;

    while (k > 0) {
        x[k] = x[k] +
        1;

        while (x[k] <= n && !place(x, k)) {
            x[k] = x[k] + 1;
        }
```

```
        if (x[k] <= n) {
            if (k == n) {
                for (int i = 1; i <= n; i++) {
                    printf("%d ", x[i]);
                }
                printf("\n");
            } else {
                k++;
                x[k] =
                0;
            }
        } else
            {
            k--;
        }
    }
}

int main() {
    int n;
    printf("Enter the number of queens: ");
    scanf("%d", &n);

    if (n < 1 || n > MAX) {
        printf("Invalid number of queens. Please enter a value between 1 and %d.\n", MAX);
        return 1;
    }

    printf("The solutions are:\n");
    nqueens(n);
```

```
    return 0;
}
```

Result:

```
Enter the number of queens: 5
The solutions are:
1 3 5 2 4
1 4 2 5 3
2 4 1 3 5
2 5 3 1 4
3 1 4 2 5
3 5 2 4 1
4 1 3 5 2
4 2 5 3 1
5 2 4 1 3
5 3 1 4 2
```