VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

Analysis and Design of Algorithms

Submitted by

PRANAV HEGDE (1BM22CS202)

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 HEGDE (1BM22CS202),** 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.

Name of the Lab-In charge:	Dr. Jyothi S Nayak
Designation	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

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.

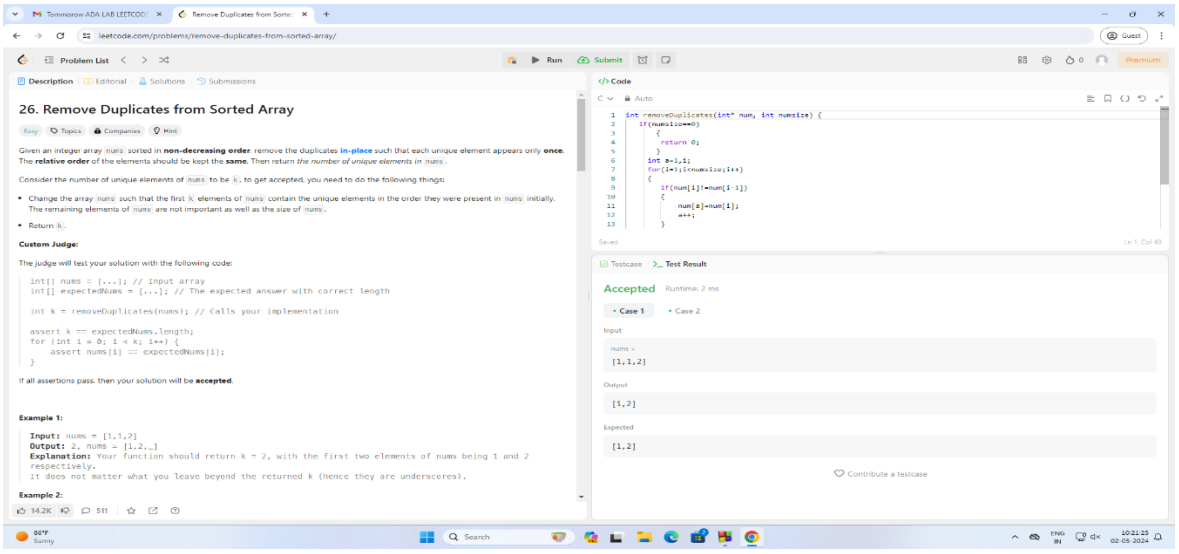
Lab Program - 1

Leetcode (Remove Duplicates from Sorted Array)

Code:

```
int removeDuplicates(int nums[], int numsSize) {
    if(numsSize==0)
        return 0;
    int a=1,i;
    for (i=1;i<numsSize;i++)
    {
        if(nums[i]!=nums[i-1])
        {
            nums[a]=nums[i];
            a++;
        }
    }
    return a;
}
```

Output:



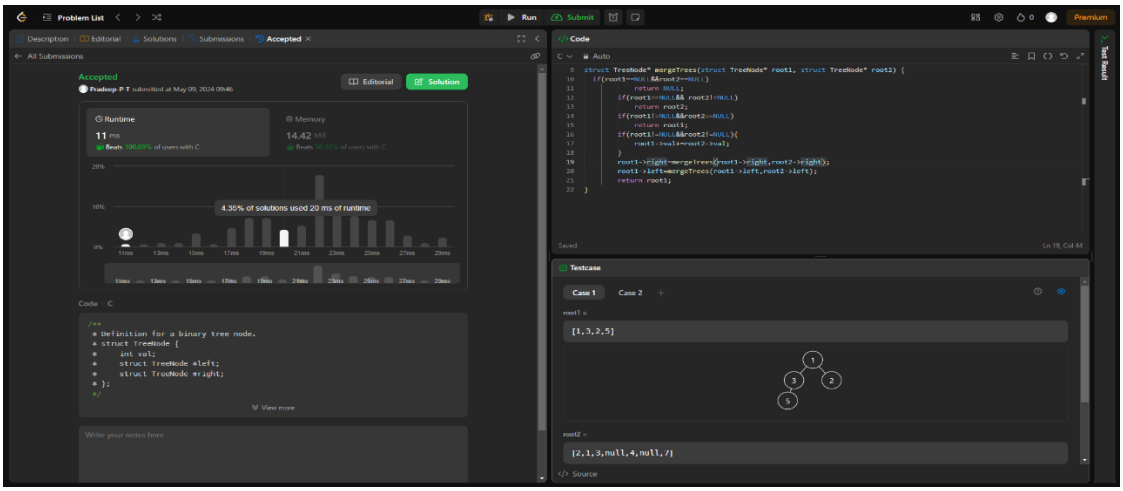
Lab Program - 2

Leetcode (Merge Two Binary Trees)

Code:

```
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
  if(root1==NULL&&root2==NULL)
        return NULL;
      if(root1==NULL&& root2!=NULL)
        return root2;
      if(root1!=NULL&&root2==NULL)
        return root1;
      if(root1!=NULL&&root2!=NULL){
        root1->val+=root2->val;
      }
      root1->right=mergeTrees(root1->right,root2->right);
      root1->left=mergeTrees(root1->left,root2->left);
      return root1;
}
```

Output:



Lab Program - 3

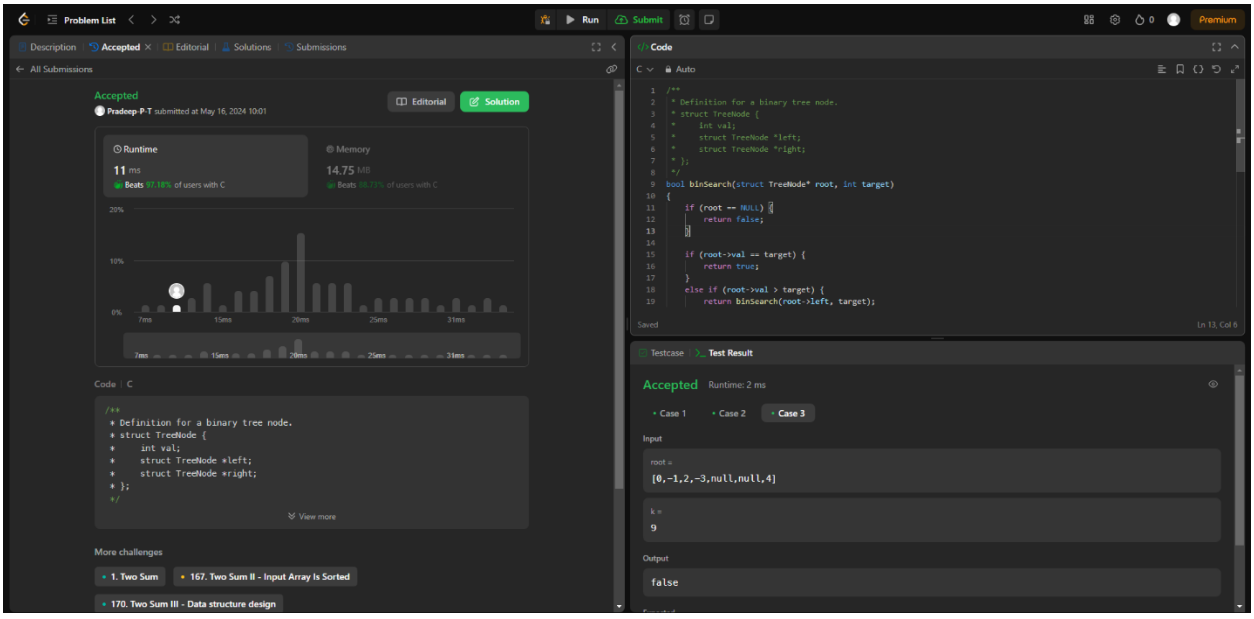
Leetcode (Two Sum IV – Input is a BST)

Code:

```
bool binSearch(struct TreeNode* root, int target)
{
    if (root == NULL) {
        return false;
    }

    if (root->val == target) {
        return true;
    }
    else if (root->val > target) {
        return binSearch(root->left, target);
    }
    else {
        return binSearch(root->right, target);
    }
}

bool dfs(struct TreeNode *root, struct TreeNode *curr, int target)
{
    if (curr==NULL) {
        return false;
    }
```

```
    if (target - curr->val != curr->val) {
        if (binSearch(root, target - curr->val) == true) {
            return true;
        }
    }

    return (dfs(root, curr->left, target) || dfs(root, curr->right, target));
}

bool findTarget(struct TreeNode* root, int k){
    return dfs(root, root, k);
}
```

Output:



Lab Program - 4

Topological Sorting (Source Removal and DFS)

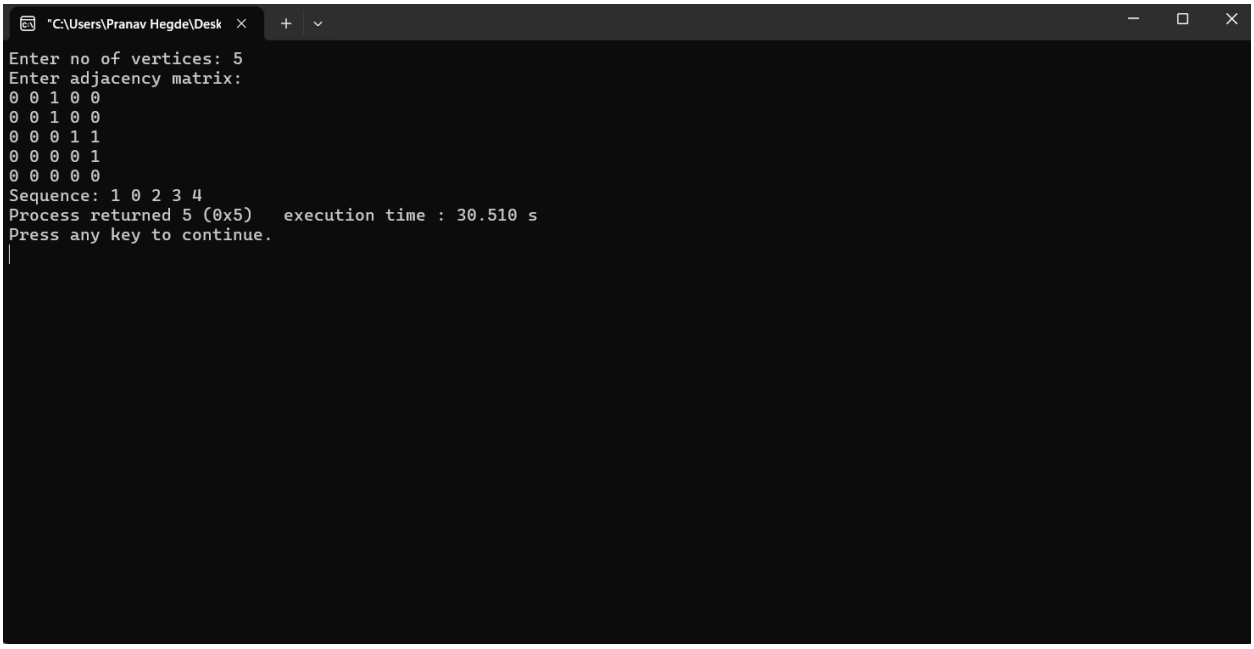
Code: Source Removal

```
#include<stdio.h>


void main()
{
 int n, a[30][30],i,j,sum,in[30],s[30],t[30],k=0;
 printf("Enter no of vertices: ");
 scanf("%d",&n);
 printf("Enter adjacency matrix:\n");
 for(i=0;i<n;i++)
 {
    for(j=0;j<n;j++)
    {
       scanf("%d",&a[i][j]);
    }
 }
 for(j=0;j<n;j++)
 {
    sum=0;
    for(i=0;i<n;i++)
    {
       sum+=a[i][j];
    }
    in[j]=sum;
 }
```

```
int top=-1;
for(i=0;i<n;i++)
{
   if(in[i]==0)
   {
      top++;
      s[top]=i;
   }

}
while(top!=-1)
{
  int u=s[top];
  top--;
  t[k++]=u;
  for(int i=0;i<n;i++)
  {
     if(a[u][i]==1)
     {
        in[i]--;
        if(in[i]==0)
        {
           top++;
           s[top]=i;
        }
     }
  }
}
```

```
    printf("Sequence: ");
    for(i=0;i<n;i++)
    {
        printf("%d ",t[i]);
    }
}
```

Output:



DFS

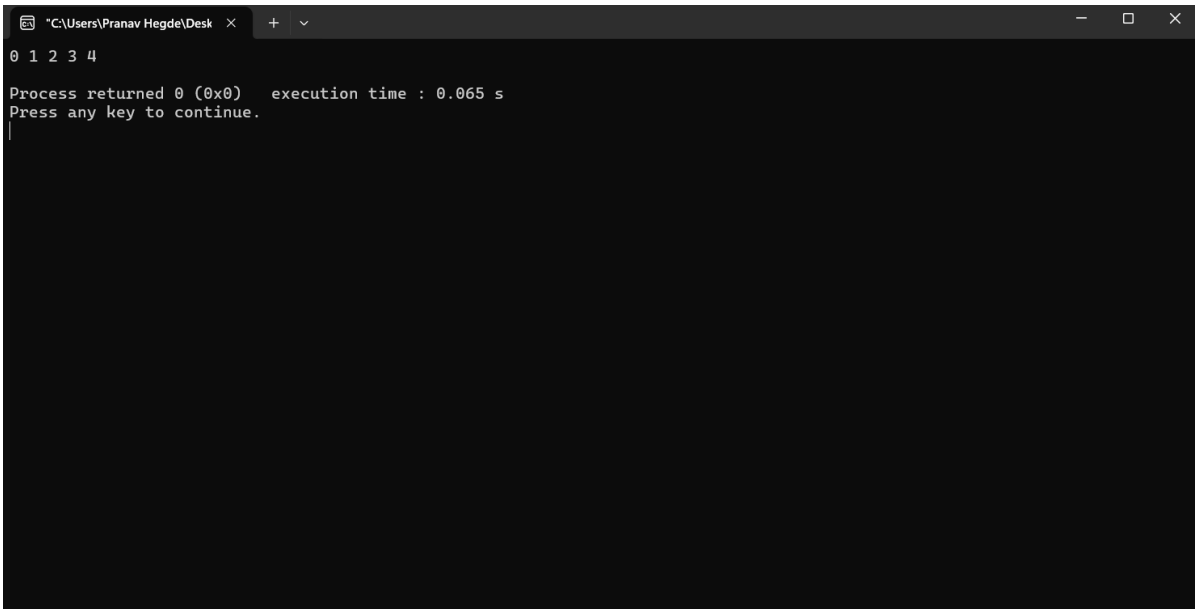
Code:

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

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

```
            DFS(v, n, a, s, j, res);
        }
    }
    (*j)++;
    res[*j] = u;
}

void topological_order(int n, int a[n][n]) {
    int s[n];
    for (int i = 0; i < n; i++) {
        s[i] = 0;
    }
    int j = -1;
    int res[n];

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

int main() {
    int adjacency_matrix[5][5] = {
```

```
        {0, 1, 0, 0, 0},
        {0, 0, 1, 0, 0},
        {0, 0, 0, 1, 1},
        {0, 0, 1, 0, 1},
        {0, 1, 1, 0, 0}
    };
    int num_vertices = 5;
    topological_order(num_vertices, adjacency_matrix);
    return 0;
}
```

Output:



Lab Program - 5

Johnson Trotter

Code:

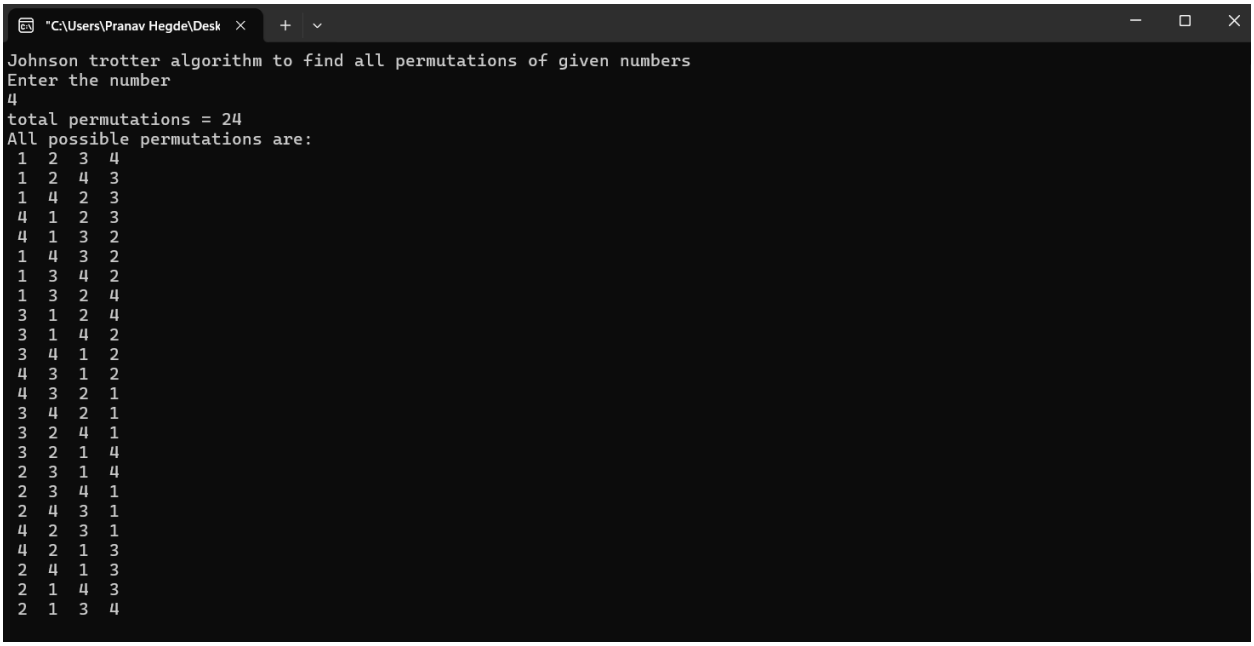
```
#include <stdio.h>
#include <stdlib.h>
int flag = 0;
int swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}
int search(int arr[], int num, int mobile) {
  int g;
  for (g = 0; g < num; g++) {
    if (arr[g] == mobile)
      return g + 1;
    else {
      flag++;
    }
  }
  return -1;
}

int find_Moblie(int arr[], int d[], int num) {
  int mobile = 0;
  int mobile_p = 0;
  int i;
```

```
  for (i = 0; i < num; i++) {
    if ((d[arr[i] - 1] == 0) && i != 0) {
      if (arr[i] > arr[i - 1] && arr[i] > mobile_p) {
        mobile = arr[i];
        mobile_p = mobile;
      } else {
        flag++;
      }
    } else if ((d[arr[i] - 1] == 1) & i != num - 1) {
      if (arr[i] > arr[i + 1] && arr[i] > mobile_p) {
        mobile = arr[i];
        mobile_p = mobile;
      } else {
        flag++;
      }
    } else {
      flag++;
    }
  }
  if ((mobile_p == 0) && (mobile == 0))
    return 0;
  else
    return mobile;
}

void permutations(int arr[], int d[], int num) {
  int i;
  int mobile = find_Moblie(arr, d, num);
```

```
  int pos = search(arr, num, mobile);
  if (d[arr[pos - 1] - 1] == 0)
    swap(&arr[pos - 1], &arr[pos - 2]);
  else
    swap(&arr[pos - 1], &arr[pos]);
  for (int i = 0; i < num; i++) {
    if (arr[i] > mobile) {
      if (d[arr[i] - 1] == 0)
        d[arr[i] - 1] = 1;
      else
        d[arr[i] - 1] = 0;
    }
  }
  for (i = 0; i < num; i++) {
    printf(" %d ", arr[i]);
  }
}

int factorial(int k) {
  int f = 1;
  int i = 0;
  for (i = 1; i < k + 1; i++) {
    f = f * i;
  }
  return f;
}
int main() {
  int num = 0;
```

```
  int i;
  int j;
  int z = 0;
  printf(
      "Johnson trotter algorithm to find all permutations of given numbers \n");
  printf("Enter the number\n");
  scanf("%d", &num);
  int arr[num], d[num];
  z = factorial(num);
  printf("total permutations = %d", z);
  printf("\nAll possible permutations are: \n");
  for (i = 0; i < num; i++) {
    d[i] = 0;
    arr[i] = i + 1;
    printf(" %d ", arr[i]);
  }
  printf("\n");
  for (j = 1; j < z; j++) {
    permutations(arr, d, num);
    printf("\n");
  }
  return 0;
}
```

Output:

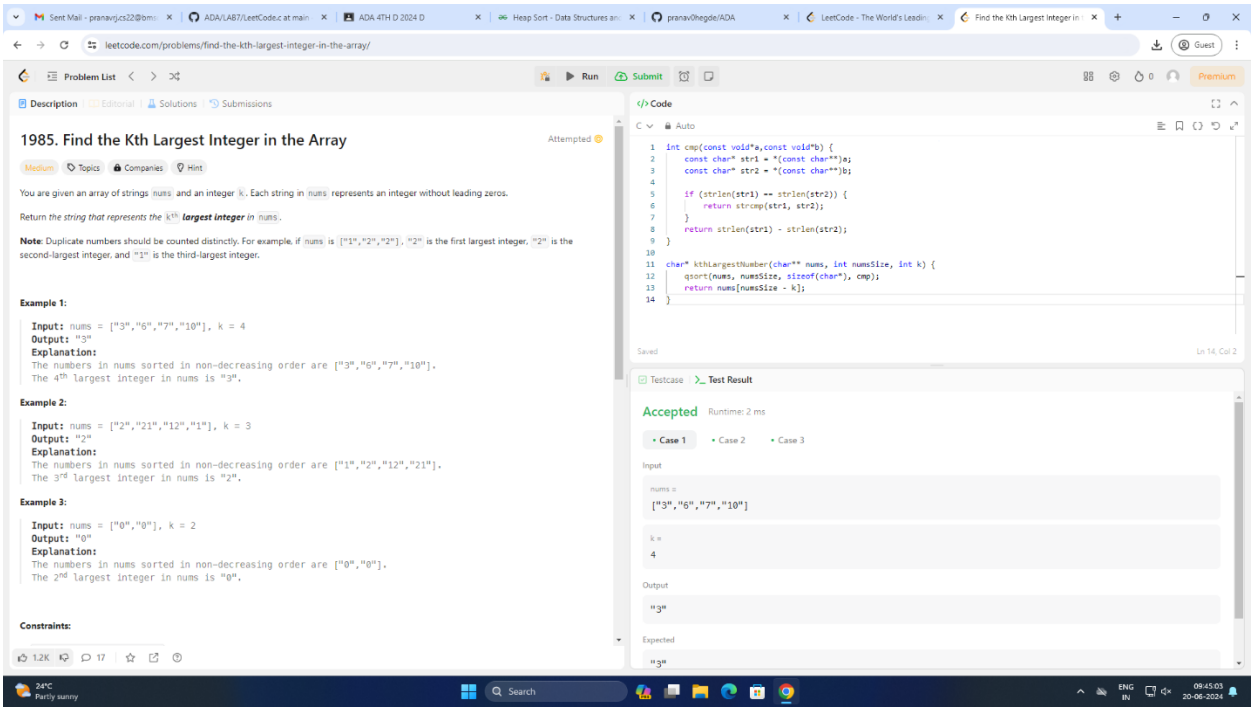


Leetcode (Kth largest integer in the array)

Code:

```
int cmp(const void*a,const void*b) {
    const char* str1 = *(const char**)a;
    const char* str2 = *(const char**)b;

    if (strlen(str1) == strlen(str2)) {
        return strcmp(str1, str2);
    }
    return strlen(str1) - strlen(str2);
}

char* kthLargestNumber(char** nums, int numsSize, int k) {
    qsort(nums, numsSize, sizeof(char*), cmp);
    return nums[numsSize - k];
}
```

Output:


leetcode.com/problems/find-the-kth-largest-integer-in-the-array/
Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
1985. Find the Kth Largest Integer in the Array
Attempted
Medium
Topics
Companies
Hint
You are given an array of strings nums and an integer k. Each string in nums represents an integer without leading zeros.
Return the string that represents the kth largest integer in nums.
Note: Duplicate numbers should be counted distinctly. For example, if nums is ["1","2","2"], "2" is the first largest integer, "2" is the second-largest integer, and "1" is the third-largest integer.
Example 1:
Input: nums = ["3","6","7","10"], k = 4
Output: "3"
Explanation:
The numbers in nums sorted in non-decreasing order are ["3","6","7","10"].
The 4th largest integer in nums is "3".
Example 2:
Input: nums = ["2","21","12","1"], k = 3
Output: "2"
Explanation:
The numbers in nums sorted in non-decreasing order are ["1","2","12","21"].
The 3rd largest integer in nums is "2".
Example 3:
Input: nums = ["0","0"], k = 2
Output: "0"
Explanation:
The numbers in nums sorted in non-decreasing order are ["0","0"].
The 2nd largest integer in nums is "0".
Constraints:
1.2K
17
Code
Auto
int cmp(const void*a,const void*b) {
const char* str1 = *(const char**)a;
const char* str2 = *(const char**)b;
if (strlen(str1) == strlen(str2)) {
return strcmp(str1, str2);
}
return strlen(str1) - strlen(str2);
}
char* kthLargestNumber(char** nums, int numsSize, int k) {
qsort(nums, numsSize, sizeof(char*), cmp);
return nums[numsSize - k];
}
Saved
Ln 14, Col 2
Testcase
Test Result
Accepted
Runtime: 2 ms
Case 1
Case 2
Case 3
Input
nums =
["3","6","7","10"]
k =
4
Output
"3"
Expected
"3"
24°C
Partly sunny
Search
ENG
IN
09:45:03
20-06-2024

Lab Program - 6

MergeSort

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++)
                   {
```

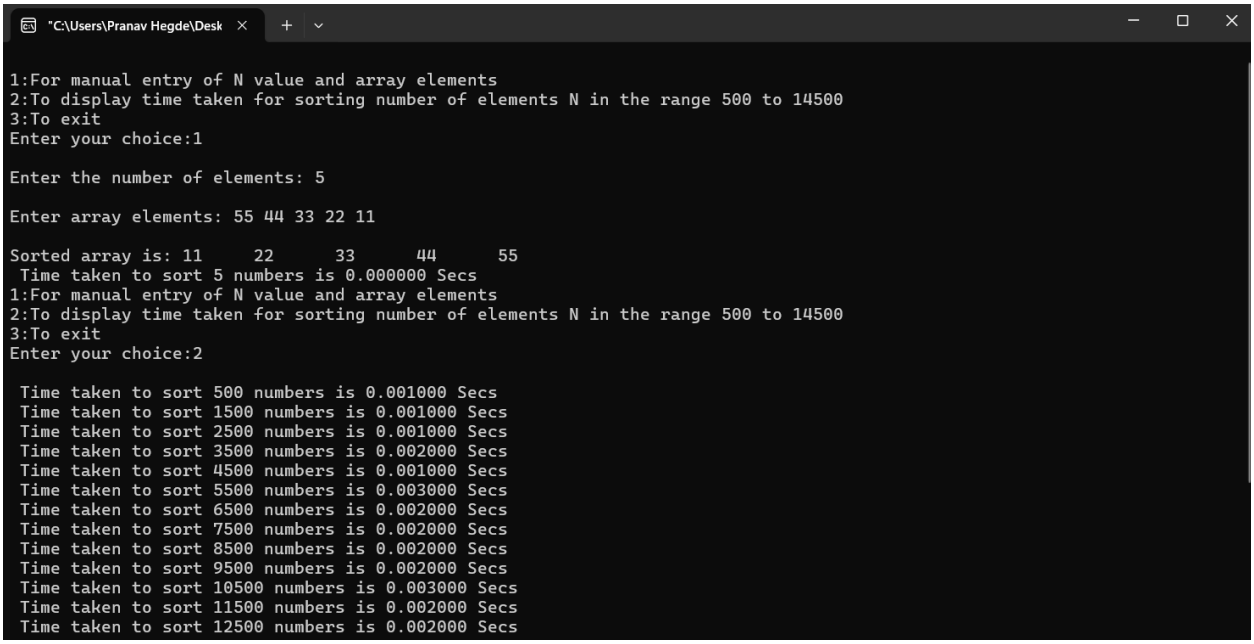
```
                scanf("%d",&a[i]);
               }
              start=clock();
              split(a,0,n-1);
              end=clock();
              printf("\nSorted array is: ");
              for(i=0;i<n;i++)
              printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n, (((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;
```

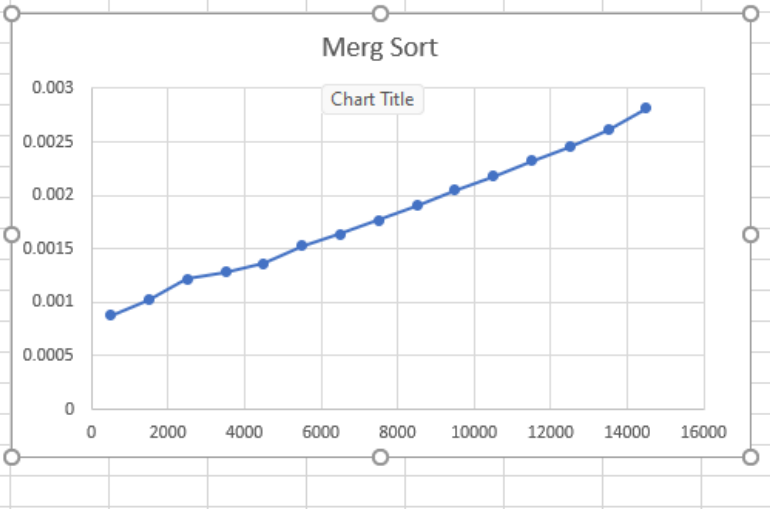
```
 }
 else
 {
  c[k]=a[j];
  ++k;
  ++j;
 }
}
if(i>mid)
{
 while(j<=high)
 {
  c[k]=a[j];
  ++k;
  ++j;
 }
}
if(j>high)
{
 while(i<=mid)
 {
  c[k]=a[i];
  ++k;
  ++i;
 }
}
for(i=low;i<=high;i++)
{
```

```
  a[i]=c[i];
 }
}
```

Output:



Values	Time Taken(Secs)
500	0.000881
1500	0.001025
2500	0.001219
3500	0.00128
4500	0.001366
5500	0.001529
6500	0.001637
7500	0.001768
8500	0.001907
9500	0.00205
10500	0.002179
11500	0.002326
12500	0.002451
13500	0.002615
14500	0.002814



SelectionSort

Code:

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
void selsort(int n,int a[]);

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]);
```

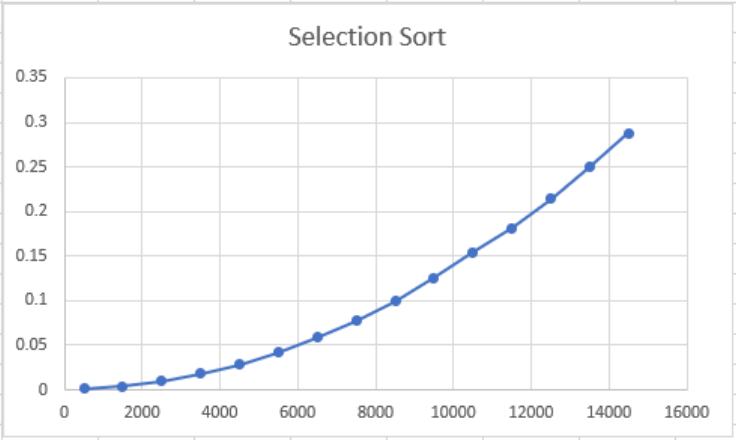
```
                }
                start=clock();
                selsort(n,a);
                end=clock();
                printf("\nSorted array is: ");
                for(i=0;i<n;i++)
                printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
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();
            selsort(n,a);
        //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 selsort(int n,int a[])
{
    int i,j,t,small,pos;
    for(i=0;i<n-1;i++)
     {
      pos=i;
      small=a[i];
      for(j=i+1;j<n;j++)
      {
            if(a[j]<small)
            {
              small=a[j];
              pos=j;
            }
      }
      t=a[i];
      a[i]=a[pos];
      a[pos]=t;
   }
}
```

Output:

```
"C:\Users\Pranav Hegde\Desk
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: 22 33 11 55 44

Sorted array is: 11     22      33      44      55
 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:2

 Time taken to sort 500 numbers is 0.001000 Secs
 Time taken to sort 1500 numbers is 0.004000 Secs
 Time taken to sort 2500 numbers is 0.008000 Secs
 Time taken to sort 3500 numbers is 0.015000 Secs
 Time taken to sort 4500 numbers is 0.017000 Secs
 Time taken to sort 5500 numbers is 0.023000 Secs
 Time taken to sort 6500 numbers is 0.037000 Secs
 Time taken to sort 7500 numbers is 0.049000 Secs
 Time taken to sort 8500 numbers is 0.057000 Secs
 Time taken to sort 9500 numbers is 0.056000 Secs
 Time taken to sort 10500 numbers is 0.044000 Secs
 Time taken to sort 11500 numbers is 0.050000 Secs
 Time taken to sort 12500 numbers is 0.092000 Secs
 Time taken to sort 13500 numbers is 0.097000 Secs
```

Values	Time Taken(Secs)
500	0.00156
1500	0.004306
2500	0.009696
3500	0.017867
4500	0.028555
5500	0.042328
6500	0.058556
7500	0.077831
8500	0.099727
9500	0.125303
10500	0.154173
11500	0.181203
12500	0.214116
13500	0.249417
14500	0.288084



Lab Program - 7

QuickSort

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);
```

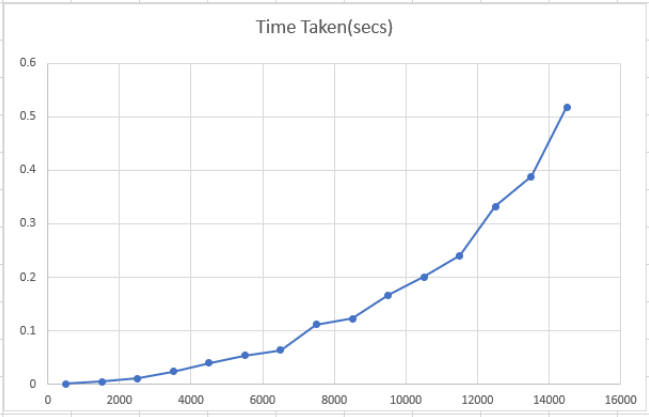
```
        quicksort(arr, low, pi - 1);
        quicksort(arr, pi + 1, high);
    }
}

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();
      }
}
```

Output:

```
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
```

Values	Time Taken(secs)
500	0.001255
1500	0.00495
2500	0.011442
3500	0.023983
4500	0.039485
5500	0.054163
6500	0.064276
7500	0.111262
8500	0.123074
9500	0.167025
10500	0.200293
11500	0.240502
12500	0.332143
13500	0.387682
14500	0.518015



Lab Program - 8

HeapSort

Code:

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling with gcc*/
void selsort(int n,int a[]);

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();
heapsort(a,n);
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();
     heapsort(a,n);
        //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);
```

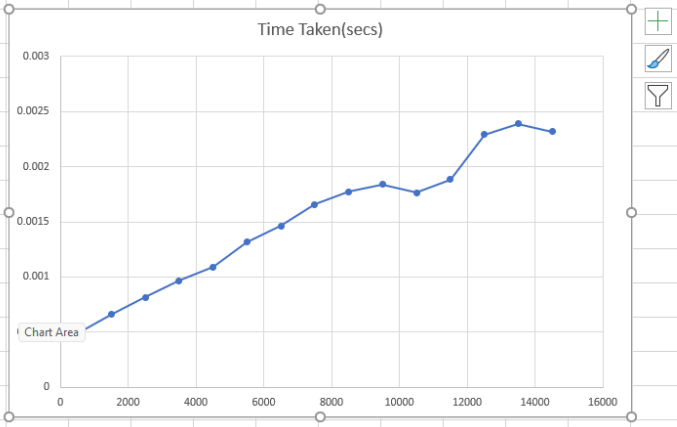
```
  }
  getchar();
  }
}


void heapify(int arr[], int n, int i) {
  int temp, maximum, left_index, right_index;
  maximum = i;
  right_index = 2 * i + 2;
  left_index = 2 * i + 1;

  if (left_index < n && arr[left_index] > arr[maximum])
    maximum = left_index;
  if (right_index < n && arr[right_index] > arr[maximum])
    maximum = right_index;

  if (maximum != i) {
    temp = arr[i];
    arr[i] = arr[maximum];
    arr[maximum] = temp;
    heapify(arr, n, maximum);
  }
}

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

```
        heapify(arr, n, i);
    }
    for (i = n - 1; i > 0; i--) {
        temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}
```

Output:

```
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:
```

Values	Time Taken(secs)
500	0.000488
1500	0.000661
2500	0.000815
3500	0.000965
4500	0.001091
5500	0.001315
6500	0.001464
7500	0.001657
8500	0.001773
9500	0.00184
10500	0.001765
11500	0.001886
12500	0.002294
13500	0.002389
14500	0.002318



Lab Program - 9

Knapsack

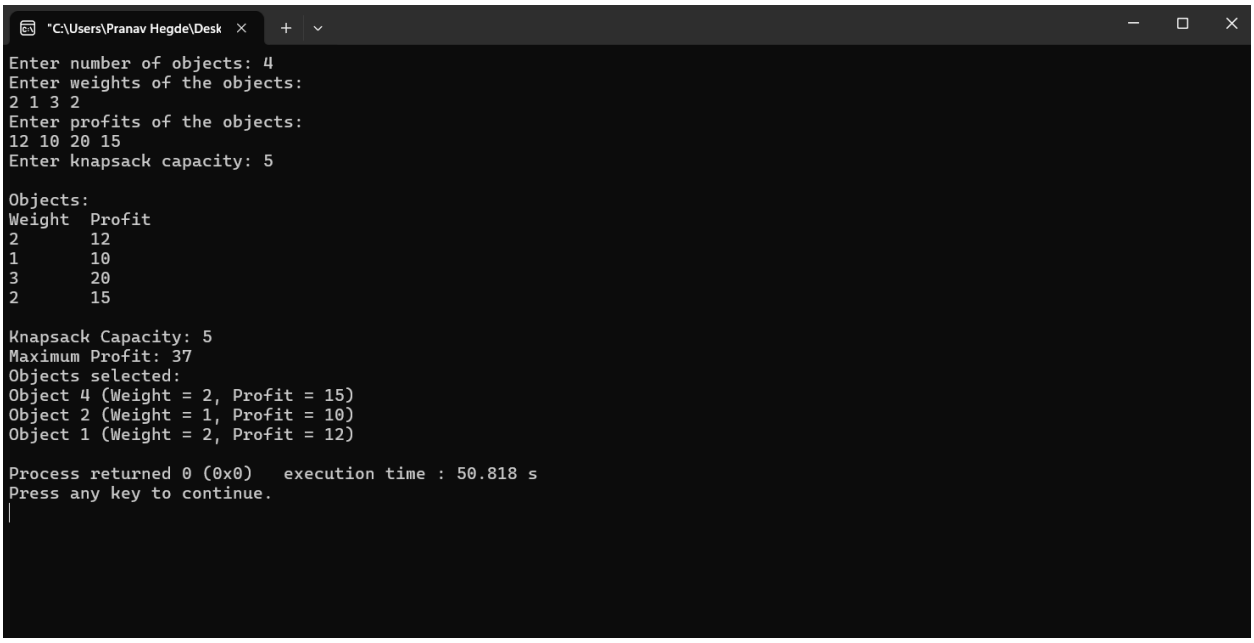
Code:

```
#include <stdio.h>

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

void knapsack(int n, int capacity, int weights[], int profits[]) {
   int dp[n+1][capacity+1]; // DP table
   int i, w;


   for (i = 0; i <= n; i++) {
      for (w = 0; w <= capacity; w++) {
         if (i == 0 || w == 0)
            dp[i][w] = 0;
         else if (weights[i-1] <= w)
            dp[i][w] = max(profits[i-1] + dp[i-1][w - weights[i-1]], dp[i-1][w]);
         else
            dp[i][w] = dp[i-1][w];
      }
   }


   int max_profit = dp[n][capacity];
   printf("Maximum Profit: %d\n", max_profit);
```

```
    printf("Objects selected:\n");
    w = capacity;
    for (i = n; i > 0 && max_profit > 0; i--) {
        if (max_profit == dp[i-1][w])
            continue;
        else {

            printf("Object %d (Weight = %d, Profit = %d)\n", i, weights[i-1], profits[i-1]);
            max_profit -= profits[i-1];
            w -= weights[i-1];
        }
    }
}

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

    int weights[n];
    int profits[n];

    printf("Enter weights of the objects:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
    }
```

```
    printf("Enter profits of the objects:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &profits[i]);
    }

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

    printf("\nObjects:\n");
    printf("Weight\tProfit\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\n", weights[i], profits[i]);
    }

    printf("\nKnapsack Capacity: %d\n", capacity);

    knapsack(n, capacity, weights, profits);

    return 0;
}
```

Output:


"C:\Users\Pranav Hegde\Desk
Enter number of objects: 4
Enter weights of the objects:
2 1 3 2
Enter profits of the objects:
12 10 20 15
Enter knapsack capacity: 5
Objects:
Weight Profit
2 12
1 10
3 20
2 15
Knapsack Capacity: 5
Maximum Profit: 37
Objects selected:
Object 4 (Weight = 2, Profit = 15)
Object 2 (Weight = 1, Profit = 10)
Object 1 (Weight = 2, Profit = 12)
Process returned 0 (0x0) execution time : 50.818 s
Press any key to continue.

Lab Program - 10

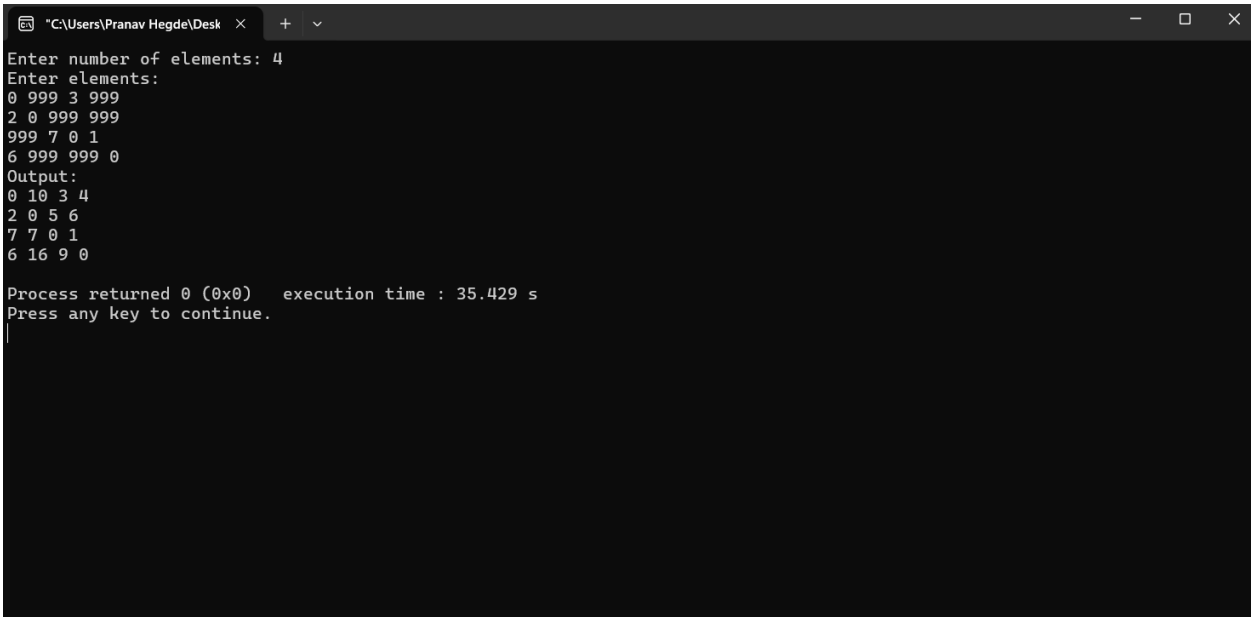
Floyd's Algorithm

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;
}
```

Output:



Lab Program - 11

Prim's Algorithm

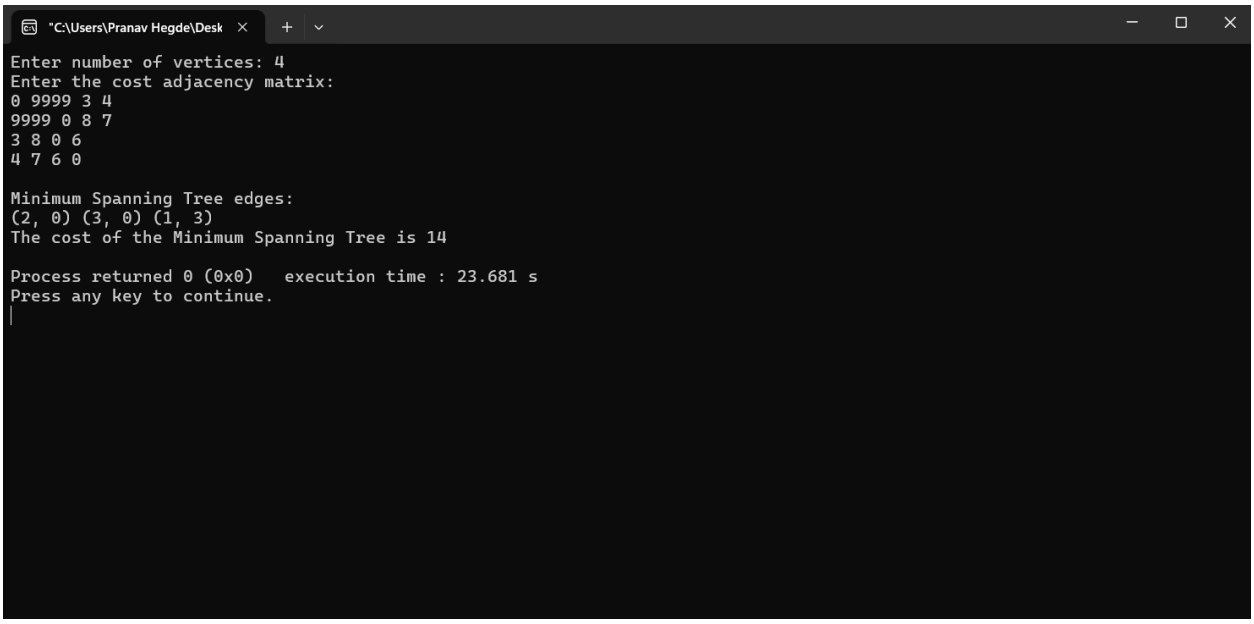
Code:

```
#include <stdio.h>
#define INF 9999

void prim(int n, int cost[n][n]) {
   int s[n];
   int d[n];
   int p[n];

   int i, j, min, source, sum = 0, k = 0;



   min = INF;
   source = 0;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         if (cost[i][j] != 0 && cost[i][j] < min) {
            min = cost[i][j];
            source = i;
         }
      }
   }



   for (i = 0; i < n; i++) {
      s[i] = 0;
```

```
    d[i] = cost[source][i];
    p[i] = source;
}


s[source] = 1;


for (i = 1; i < n; i++) {

    min = INF;
    int u = -1;
    for (j = 0; j < n; j++) {
        if (s[j] == 0 && d[j] < min) {
            min = d[j];
            u = j;
        }
    }

    printf("(%d, %d) ", u, p[u]);
    sum += cost[u][p[u]];
    s[u] = 1; // Add u to the MST

    for (j = 0; j < n; j++) {
        if (s[j] == 0 && cost[u][j] < d[j]) {
            d[j] = cost[u][j];
            p[j] = u;
        }
```

```
        }
    }

    if (sum >= INF) {
        printf("\nSpanning tree does not exist\n");
    } else {
        printf("\nThe cost of the Minimum Spanning Tree is %d\n", sum);
    }
}

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

    int cost[n][n];
    printf("Enter the cost adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }

    printf("\nMinimum Spanning Tree edges:\n");
    prim(n, cost);

    return 0;
}
```

Output:


"C:\Users\Pranav Hegde\Desk
Enter number of vertices: 4
Enter the cost adjacency matrix:
0 9999 3 4
9999 0 8 7
3 8 0 6
4 7 6 0
Minimum Spanning Tree edges:
(2, 0) (3, 0) (1, 3)
The cost of the Minimum Spanning Tree is 14
Process returned 0 (0x0) execution time : 23.681 s
Press any key to continue.

Kruskal's Algorithm

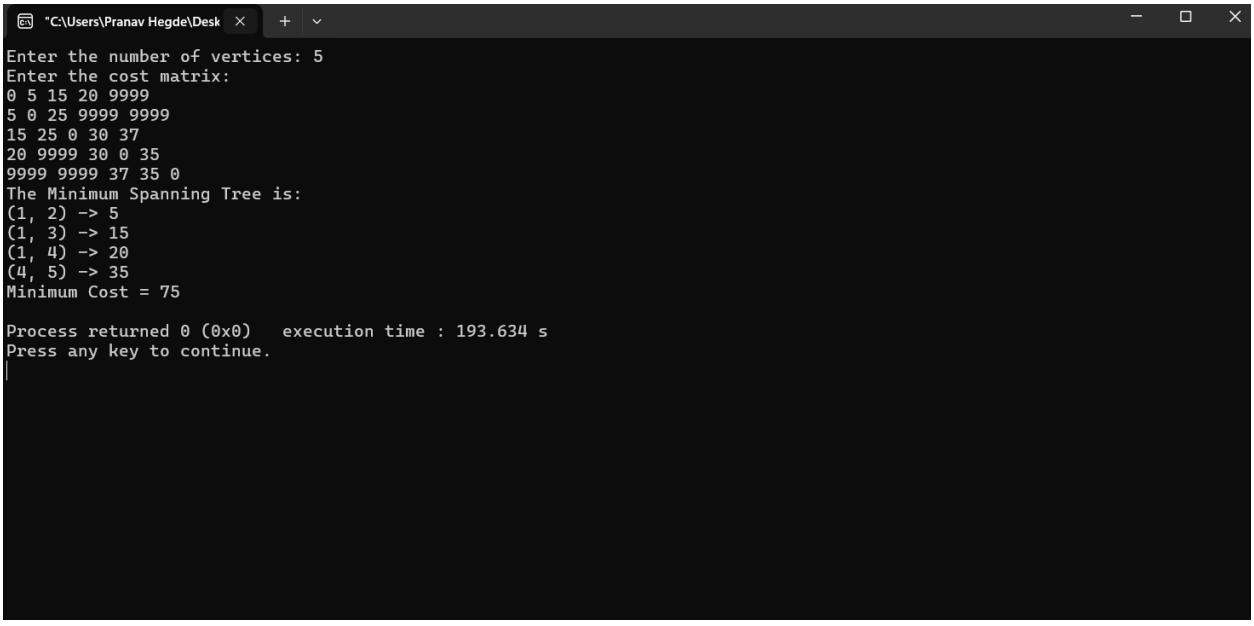
Code:

```
#include <stdio.h>
#define MAX 30
typedef struct edge {
   int u, v, cost;
} Edge;
Edge edges[MAX];
int parent[MAX];
int find(int i) {
   while (parent[i])
      i = parent[i];
   return i;
}
int uni(int i, int j) {
   if (i != j) {
      parent[j] = i;
      return 1;
   }
   return 0;
}
void kruskals(int c[MAX][MAX], int n) {
   int i, j, u, v, a, b, min, ne = 0, mincost = 0;
   for (i = 1; i <= n; i++)
      parent[i] = 0;
   while (ne < n - 1)
 {
```

```
        min = 9999;
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
                if (c[i][j] < min) {
                    min = c[i][j];
                    u = a = i;
                    v = b = j;
                }
            }
        }
        u = find(u);
        v = find(v);
        if (uni(u, v)) {
            printf("(%d, %d) -> %d\n", a, b, min);
            mincost += min;
            ne++;
        }
        c[a][b] = c[b][a] = 9999;
    }
    printf("Minimum Cost = %d\n", mincost);
}
int main() {
    int c[MAX][MAX], n, i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &n)
    printf("Enter the cost matrix:\n");
    for (i = 1; i <= n; i++) {
```

```
        for (j = 1; j <= n; j++) {
            scanf("%d", &c[i][j]);
            if (c[i][j] == 0)
                c[i][j] = 9999;
        }
    }
    printf("The Minimum Spanning Tree is:\n");
    kruskals(c, n);
    return 0;
}
```

Output:



Lab Program - 12

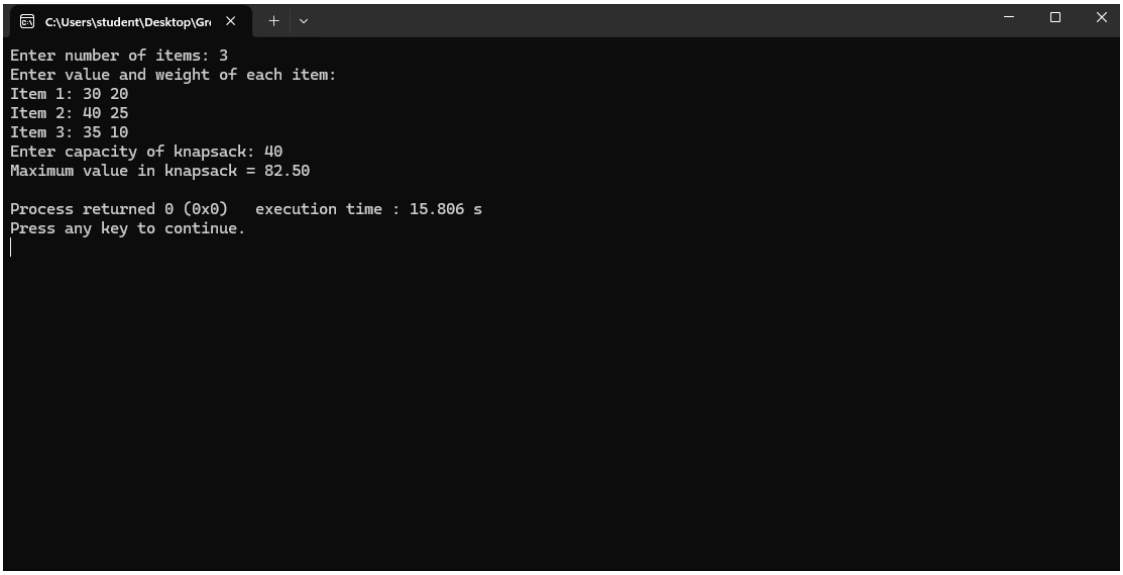
Fractional Knapsack using Greedy technique

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;
}
```

Output:



Lab Program - 13

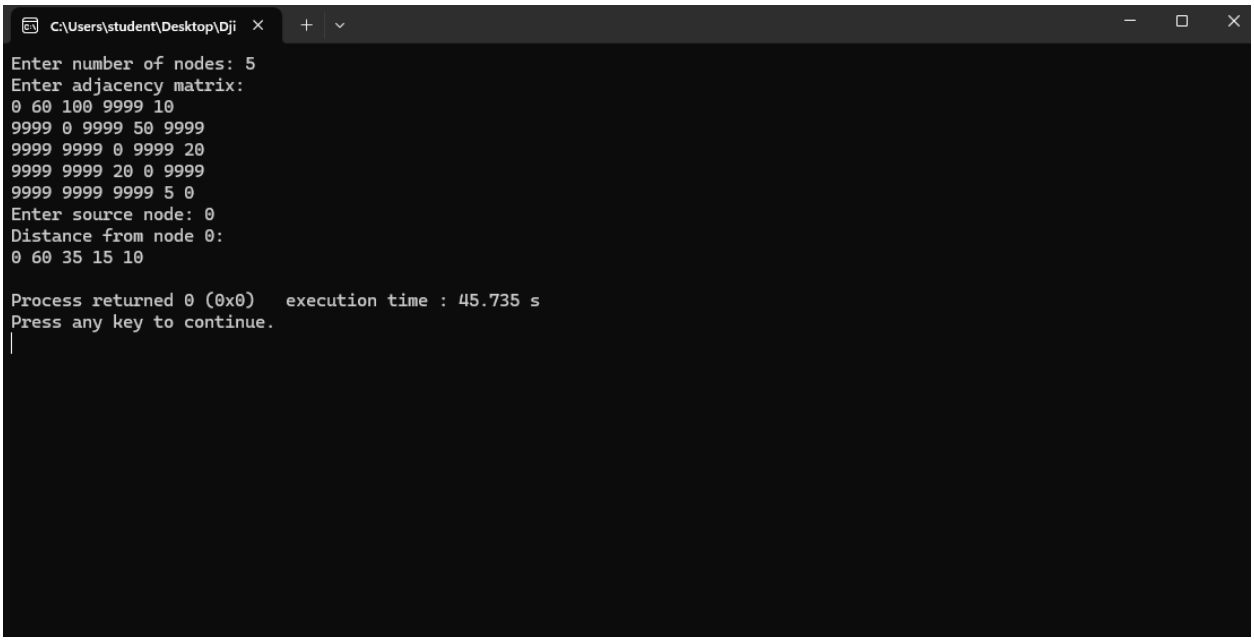
Dijkstras Algorithm

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;
}
```

Output:


C:\Users\student\Desktop\Dji
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 : 45.735 s
Press any key to continue.

Lab Program - 14

NQueens Problem using Backtracking

Code:

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

void printSolution(int board[N][N])
{
	for (int i = 0; i < N; i++) {
		for (int j = 0; j < N; j++) {
			if(board[i][j])
				printf("Q ");
			else
				printf(". ");
		}
		printf("\n");
	}
}

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

```
            if (board[i][j])
                return false;
        for (i = row, j = col; j >= 0 && i < N; i++, j--)
            if (board[i][j])
                return false;

        return true;
}

bool solveNQUtil(int board[N][N], int col)
{

        if (col >= N)
            return true;

        for (int i = 0; i < N; i++) {
            if (isSafe(board, i, col)) {
                board[i][col] = 1;
                if (solveNQUtil(board, col + 1))
                    return true;
                board[i][col] = 0;
            }
        }
        return false;
}

bool solveNQ()
```

```
{
        int board[N][N] = { { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 } };

        if (solveNQUtil(board, 0) == false) {
                printf("Solution does not exist");
                return false;
        }

        printSolution(board);
        return true;
}

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

Output:

```
C:\Users\Hp\Desktop\IV SEM\LABScd "c:\Users\Hp\Desktop\IV SEM\LABS\ADA\" && gcc NQueens.c -o NQueens && "c:\Users\Hp\Desktop\IV SEM\LABS\ADA\"NQueens
. . Q .
Q . . .
. . . Q
. Q . .
```