VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT on

Analysis and Design of Algorithms

Submitted by

NAVNEETH K S (1BM22CS183)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING BENGALURU-560019 April-2024 to August-2024

(Autonomous Institution under VTU)

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 **NAVNEETH K S (1BM22CS174),** who is Bonafede 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.

Prof. Vikranth B M
Associate Professor
Department of CSE
BMSCE.

Dr. Jyothi S Nayak
Professor and Head of
Department of CSE
BMSCE.

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.

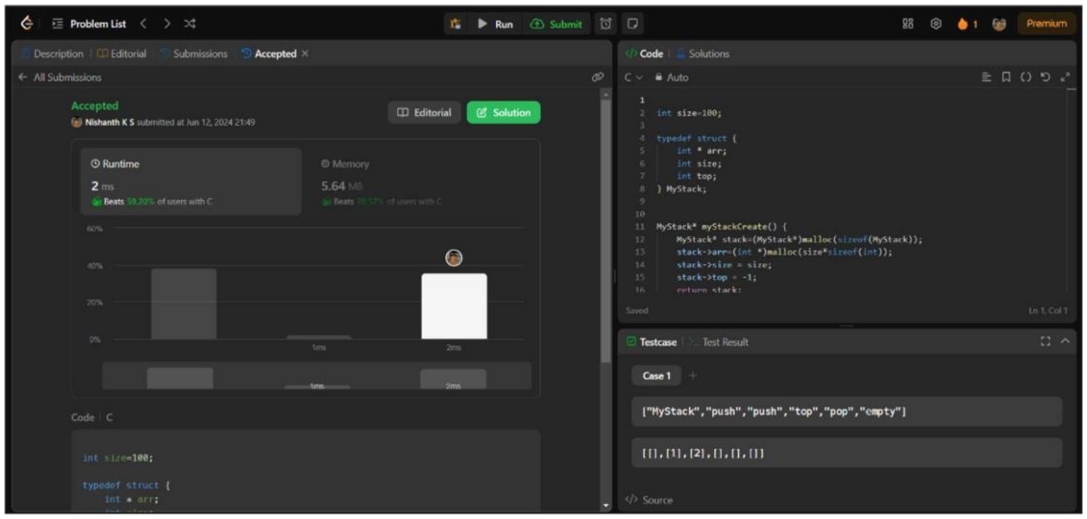
1

Leetcode (Implement stack using queues)

Code:

```
#include <stdbool.h>


int size=100;


typedef struct {
    int * arr;
    int size;
    int top;
} MyStack;



MyStack* myStackCreate() {
    MyStack* stack=(MyStack*)malloc(sizeof(MyStack));
    stack->arr=(int *)malloc(size*sizeof(int));
    stack->size = size;
    stack->top = -1;
    return stack;
}


void myStackPush(MyStack* obj, int x) {
    if(obj->top < obj->size-1){
        obj->top++;
        obj->arr[obj->top]=x;
    }
}

int myStackPop(MyStack* obj) {
    if(obj->top==-1){
        return -1;
    }
    return obj->arr[obj->top--];
}

int myStackTop(MyStack* obj) {
    if(obj->top==-1){
        return -1;
    }
    return obj->arr[obj->top];
}

bool myStackEmpty(MyStack* obj) {
    return obj->top==-1;
}

void myStackFree(MyStack* obj) {
    free(obj->arr);
    free(obj);
}

/**
```

Output:


Accepted
Nishanth K S submitted at Jun 12, 2024 21:49

Runtime 2 ms
Beats 59.20% of users with C

Memory 5.64 MB

Code | C

```
int size=100;

typedef struct {
    int * arr;
```

```
int size=100;

typedef struct {
    int * arr;
    int size;
    int top;
} MyStack;


MyStack* myStackCreate() {
    MyStack* stack=(MyStack*)malloc(sizeof(MyStack));
    stack->arr=(int *)malloc(size*sizeof(int));
    stack->size = size;
    stack->top = -1;
    return stack;
```

Testcase

Case 1

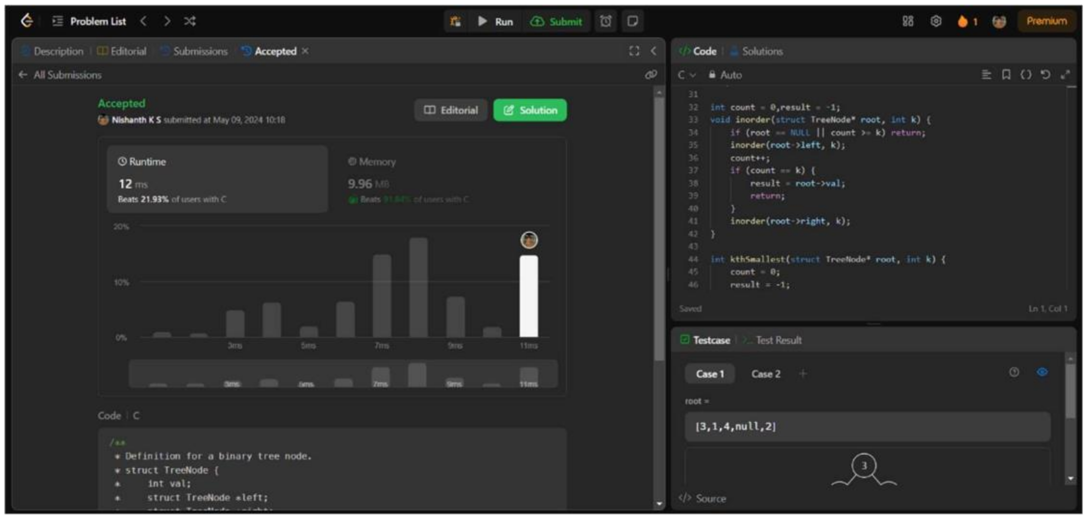
```
["MyStack","push","push","top","pop","empty"]
[[],[1],[2],[],[],[]]
```

2

Leetcode (kth smallest element in a bst)

Code:

```
#include <stdlib.h>


struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};


int count = 0, result = -1;


void inorder(struct TreeNode* root, int k) {
    if (root == NULL || count >= k) return;
    inorder(root->left, k);
    count++;
    if (count == k) {
        result = root->val;
        return;
    }
    inorder(root->right, k);
}


int kthSmallest(struct TreeNode* root, int k) {
    count = 0;
    result = -1;
    inorder(root, k);
    return result;
}
```
Output:



Accepted

Nishanth K S submitted at May 09, 2024 10:18

Runtime 12 ms Beats 21.93% of users with C

Memory 9.96 MB

Code | C

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
```

```
int count = 0,result = -1;
void inorder(struct TreeNode* root, int k) {
    if (root == NULL || count >= k) return;
    inorder(root->left, k);
    count++;
    if (count == k) {
        result = root->val;
        return;
    }
    inorder(root->right, k);
}

int kthSmallest(struct TreeNode* root, int k) {
    count = 0;
    result = -1;
```

Saved Ln 1, Col 1

root =

```
[3,1,4,null,2]
```

3

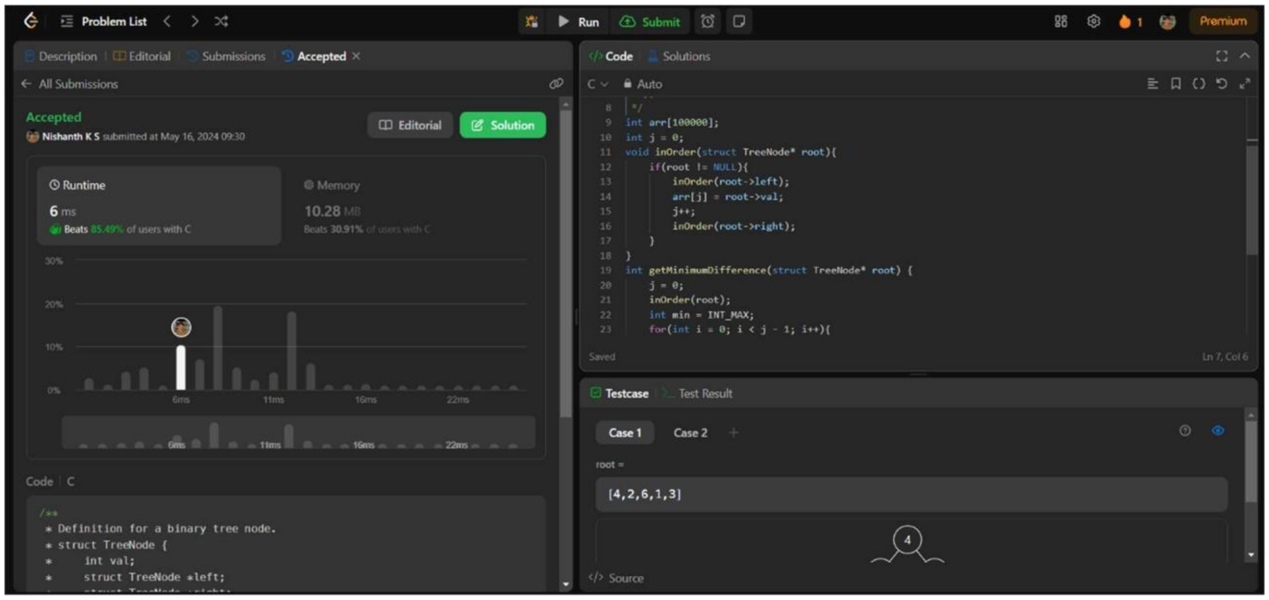
Leetcode (Minimum absolute difference in bs)

Code:

```
#include <limits.h>

#include <stddef.h> // Add this line to include the necessary header file for 'NULL'
```

```
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

int arr[100000];
int j = 0;

void inOrder(struct TreeNode* root){
    if(root != NULL){
        inOrder(root->left);
        arr[j] = root->val;
        j++;
        inOrder(root->right);
    }
}

int getMinimumDifference(struct TreeNode* root) {
    j = 0;
    inOrder(root);
    int min = INT_MAX;
    for(int i = 0; i < j - 1; i++){
        int diff = arr[i+1] - arr[i];
        if(diff<min && diff > 0){
            min = diff;
        }
    }
    return min;
}
```

Output:



Accepted
Nishanth K S submitted at May 16, 2024 09:30

Runtime
6 ms
Beats 85.49% of users with C

Memory
10.28 MB
Beats 30.91% of users with C

Code | C

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
```

```
 */
int arr[100000];
int j = 0;
void inOrder(struct TreeNode* root){
    if(root != NULL){
        inOrder(root->left);
        arr[j] = root->val;
        j++;
        inOrder(root->right);
    }
}
int getMinimumDifference(struct TreeNode* root) {
    j = 0;
    inOrder(root);
    int min = INT_MAX;
    for(int i = 0; i < j - 1; i++){
```

Saved
Ln 7, Col 6

root =

```
[4,2,6,1,3]
```

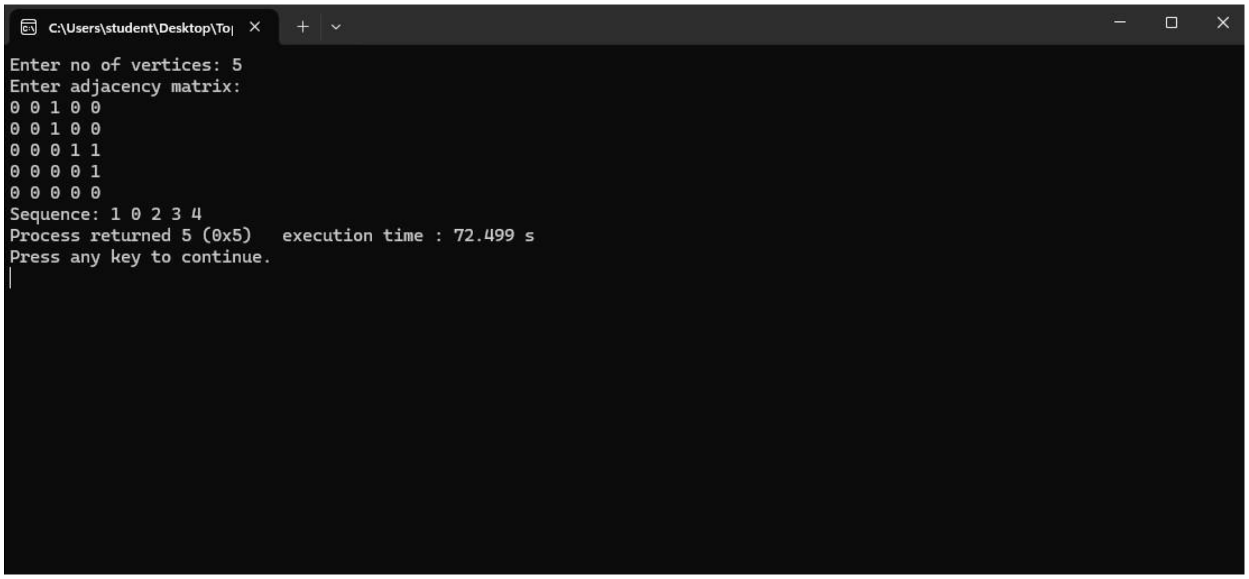
Lab Program - 4 Topological Sorting (Source Removal and DFS)

Code:

Source Removal

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

```
#define MAX 100 // Define the maximum number of vertices

// Function to find the topological order using the Source Removal Technique
void topological_order(int n, int a[MAX][MAX]) {
    int in_degree[MAX] = {0}; // Array to store in-degrees of each vertex
    int res[MAX]; // Array to store the topological order
    int j = 0; // Index for the result array

    // Step 1: Calculate in-degrees of all vertices
    for (int u = 0; u < n; u++) {
        for (int v = 0; v < n; v++) {
            if (a[u][v] == 1) {
                in_degree[v]++;
            }
        }
    }

    // Step 2: Initialize the queue for vertices with in-degree 0
    int queue[MAX], front = 0, rear = 0;

    for (int i = 0; i < n; i++) {
        if (in_degree[i] == 0) {
            queue[rear++] = i; // Enqueue vertices with in-degree 0
        }
    }

    // Step 3: Process the queue
    while (front < rear) {
        int u = queue[front++]; // Dequeue a vertex

        res[j++] = u; // Add it to the result

        // Step 4: Decrease the in-degree of adjacent vertices
        for (int v = 0; v < n; v++) {
            if (a[u][v] == 1) {
                in_degree[v]--; // Decrease in-degree
                if (in_degree[v] == 0) {
                    queue[rear++] = v; // Enqueue if in-degree becomes 0
                }
            }
        }
    }

    // Step 5: Print the topological order
    printf("\nTopological order:\n");
    for (int i = 0; i < j; i++) {
        printf("-->%d", res[i]);
    }
    printf("\n");
}

int main() {
    int a[MAX][MAX] = {
        {0, 1, 0, 0, 0},
        {0, 0, 1, 0, 0},
        {0, 0, 0, 0, 0},
        {0, 0, 1, 0, 1},
        {0, 0, 0, 0, 0}
    };

    printf("\nTopological order using Source Removal Technique:\n");
    topological_order(5, a); // Call the topological order function


    return 0;
}
```
Output:



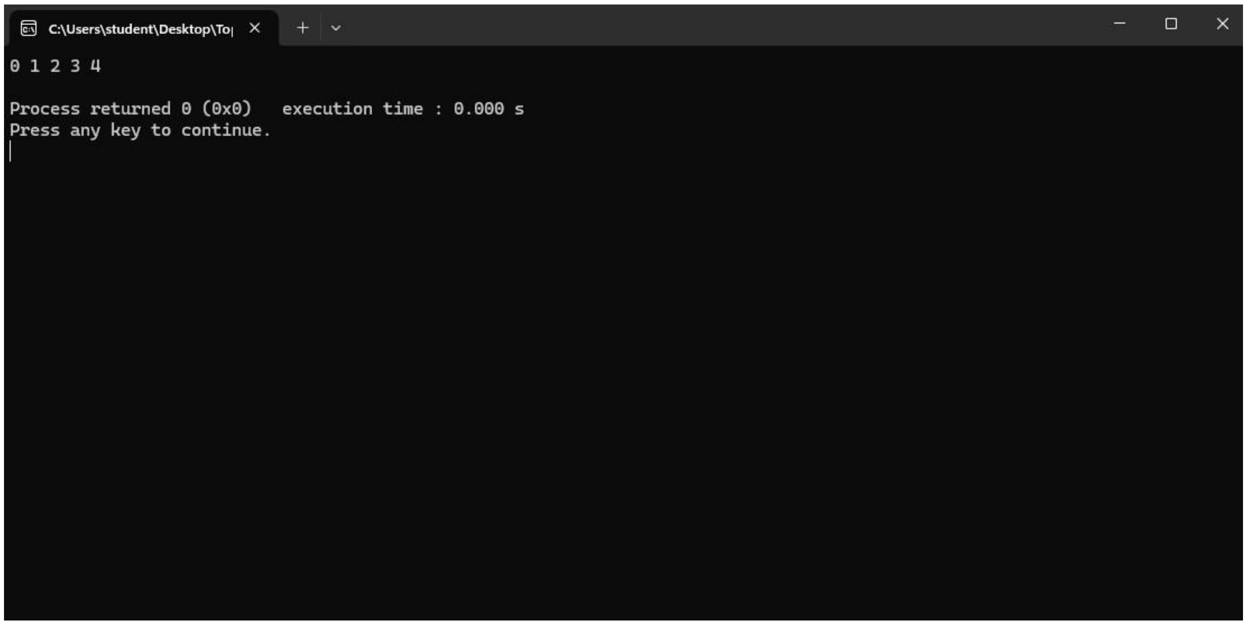
```
Enter no of vertices: 5
Enter adjacency matrix:
0 0 1 0 0
0 0 1 0 0
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0
Sequence: 1 0 2 3 4
Process returned 5 (0x5)   execution time : 72.499 s
Press any key to continue.
```

DFS

Code:

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


int s[100], j, res[100]; /* GLOBAL VARIABLES */ //s-> visited, res-> 0


void dfs(int u, int n, int a[5][5]) {
    int v;
    s[u] = 1;
    for (v = 0; v < n; v++) {  /* Change n-1 to n to include all vertices */
        if (a[u][v] == 1 && s[v] == 0) {
            dfs(v, n, a);
        }
    }
    res[j++] = u; /* Increment j after assignment */
}

void topological_order(int n, int a[5][5]) {
    // make all vertex not visited
    for (int i = 0; i < n; i++) {
        s[i] = 0;
    }
    j = 0;
    for (int u = 0; u < n; u++) {
        if (s[u] == 0) {
            dfs(u, n, a);
        }
    }
}

int main() {
    int a[5][5] = {
        {0, 1, 0, 0, 0},
        {0, 0, 1, 0, 0},
        {0, 0, 0, 0, 0},
        {0, 0, 1, 0, 1},
        {0, 0, 0, 0, 0}
    };

    printf("\nTopological order:\n");
    topological_order(5, a);

    for (int i = j - 1; i >= 0; i--) {
        printf("-->%d", res[i]);
    }

    printf("\n");
    return 0;
}
```

Output:



```
0 1 2 3 4

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

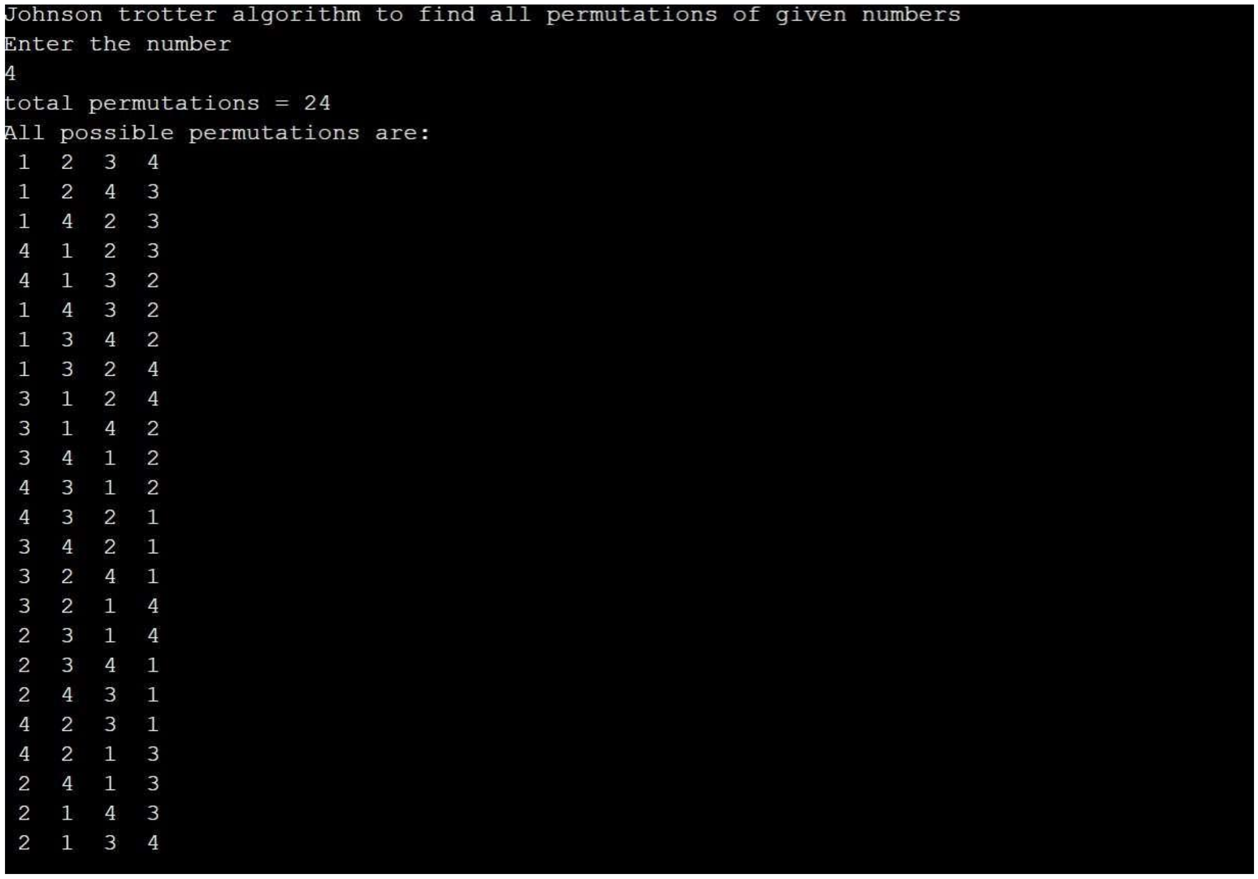
Lab Program - 5 Johnson Trotter

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:


```
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
```

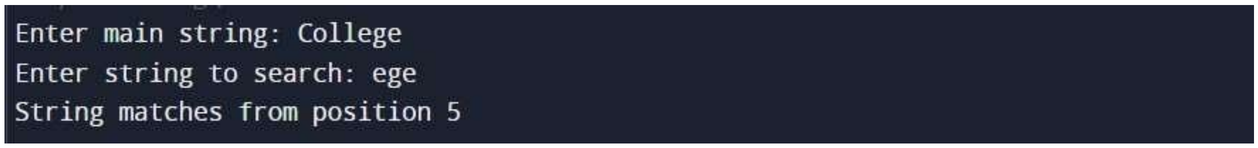
Substring Matching

Code:

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


void  main(){   char  a[1000];
char   b[1000];   printf("Enter
main string: ");     gets(a);
    printf("Enter string to search: ");
gets(b);
```

```
    for(int i = 0; i < strlen(a) - strlen(b) + 1; i++){
for(int j = 0; j < strlen(b); j++){              if(a[i+j]
!= b[j]){                  break;
             }             if(j ==
strlen(b) - 1){
                printf("String matches from position %d", i + 1);
return;
             }
         }
    }
    printf("String doesn't match");
}
```

Output:



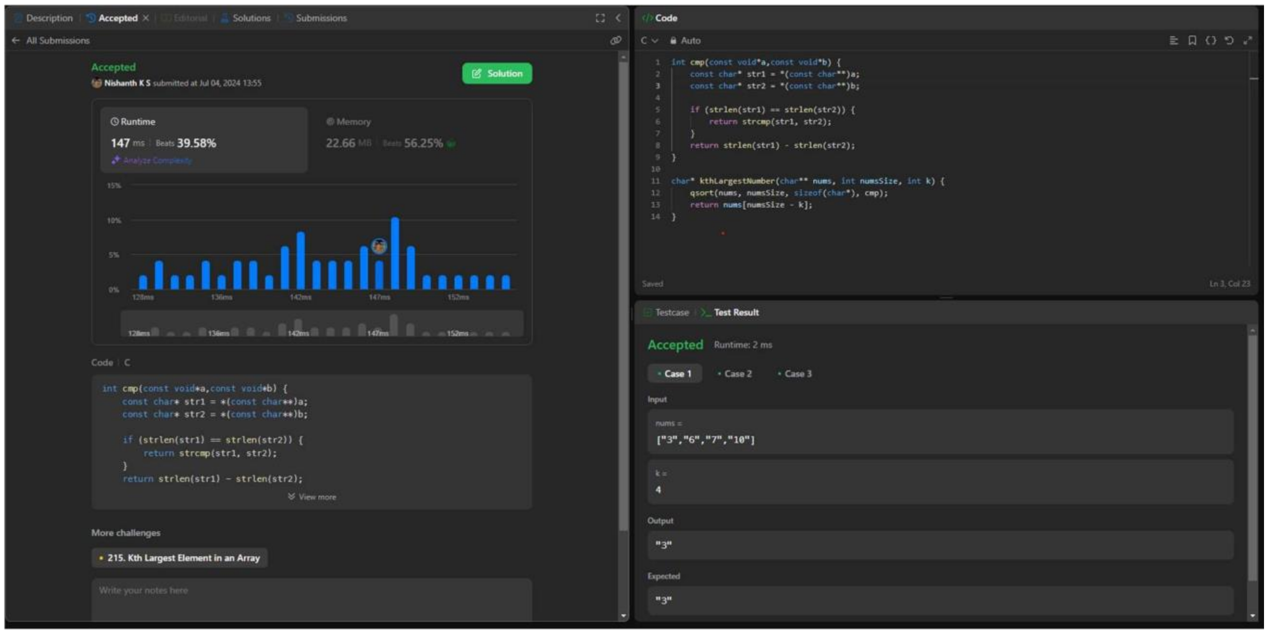
```
Enter main string: College
Enter string to search: ege
String matches from position 5
```

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:



Accepted
Nishanth K S submitted at Jul 04, 2024 13:55

Runtime
147 ms | Beats 39.58%

Memory
22.66 MB | Beats 56.25%

Code | C

```
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);
```

View more

More challenges
215. Kth Largest Element in an Array

Write your notes here

Code
C 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 3, Col 23

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"

Lab Program - 6 MergeSort

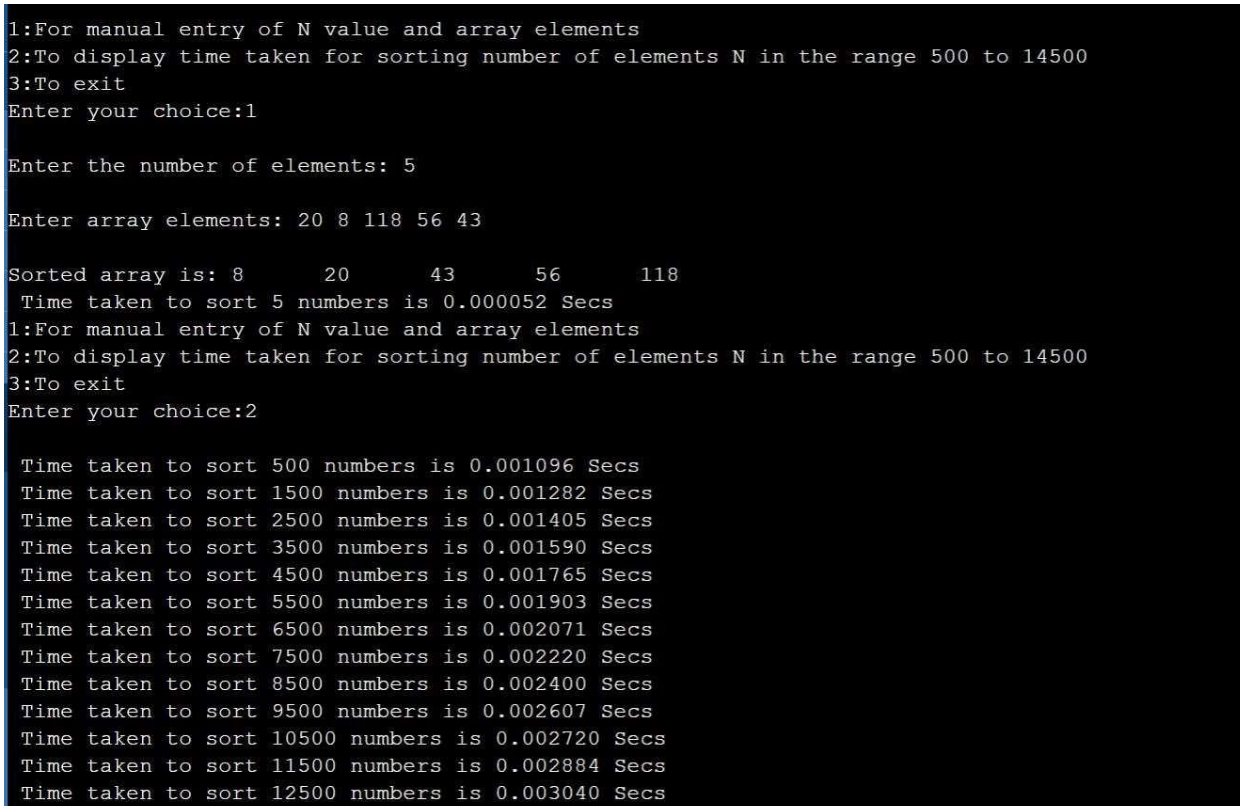
Code:

```
#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)(endstart))/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)(endstart))/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);
 }
}
```

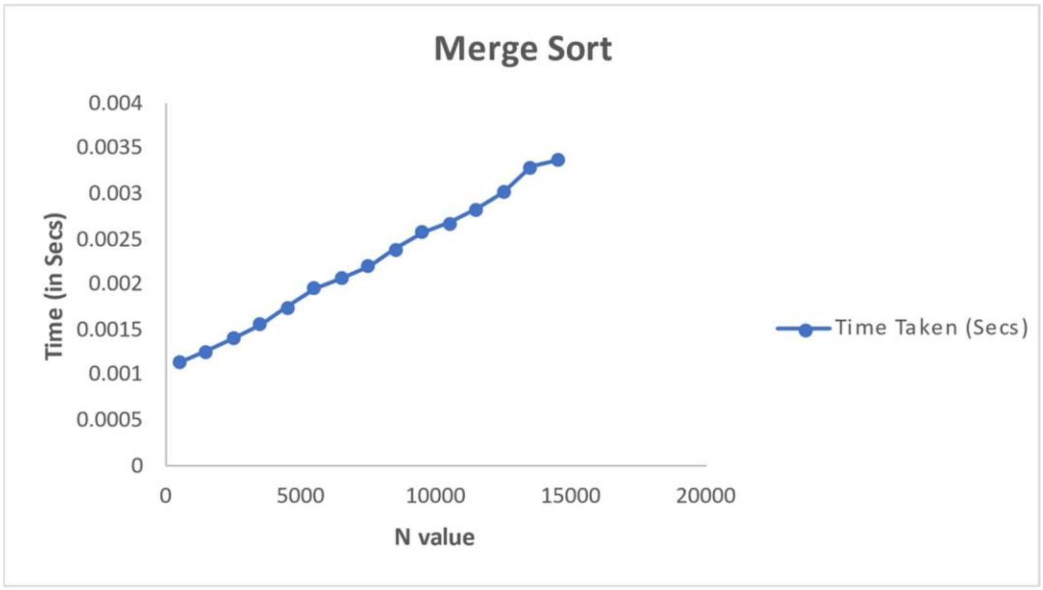
```
void combine(int a[],int low,int mid,int high)
{  int c[15000],i,j,k;
i=k=low;  j=mid+1;
while(i<=mid&&j<=high)
 {
  if(a[i]<a[j])
{
c[k]=a[i];
++k;
   ++i;   }
else   {
c[k]=a[j];
++k;
   ++j;
  }  }
if(i>mid)
 {
while(j<=high)
  {
c[k]=a[j];
++k;
   ++j;
  }  }
if(j>high)
 {
  while(i<=mid)
  {
c[k]=a[i];
++k;

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

Output:



```
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: 20 8 118 56 43

Sorted array is: 8      20      43      56      118
 Time taken to sort 5 numbers is 0.000052 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 0.001096 Secs
 Time taken to sort 1500 numbers is 0.001282 Secs
 Time taken to sort 2500 numbers is 0.001405 Secs
 Time taken to sort 3500 numbers is 0.001590 Secs
 Time taken to sort 4500 numbers is 0.001765 Secs
 Time taken to sort 5500 numbers is 0.001903 Secs
 Time taken to sort 6500 numbers is 0.002071 Secs
 Time taken to sort 7500 numbers is 0.002220 Secs
 Time taken to sort 8500 numbers is 0.002400 Secs
 Time taken to sort 9500 numbers is 0.002607 Secs
 Time taken to sort 10500 numbers is 0.002720 Secs
 Time taken to sort 11500 numbers is 0.002884 Secs
 Time taken to sort 12500 numbers is 0.003040 Secs
```

Graph:



SelectionSort

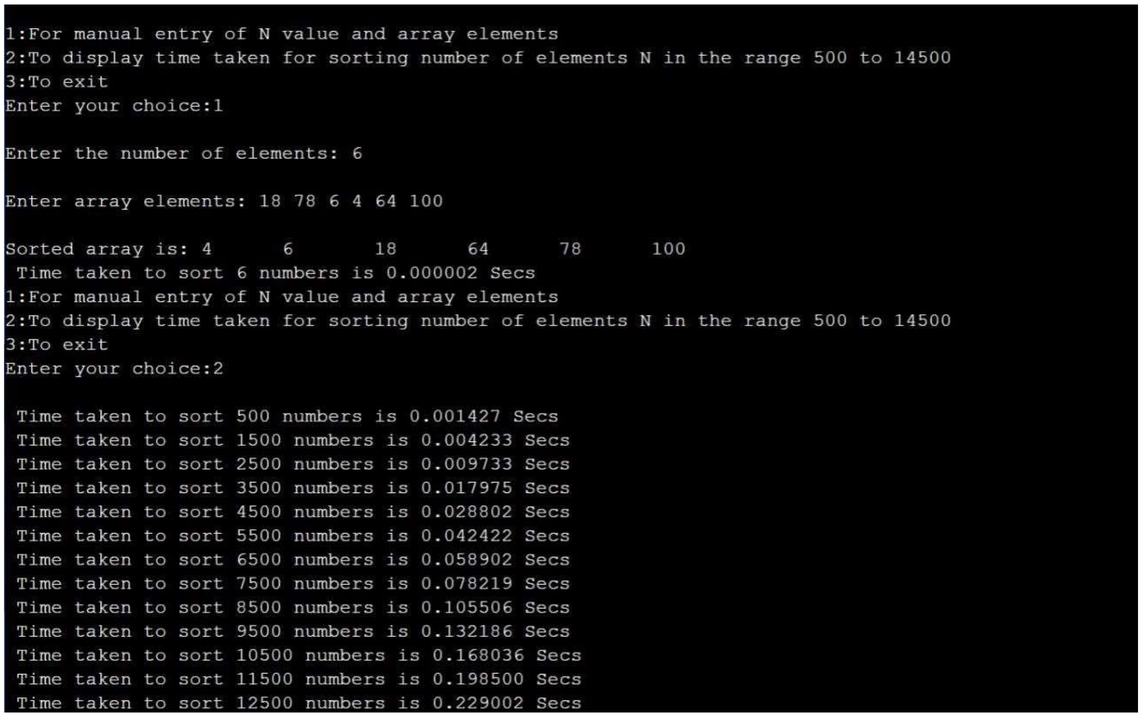
Code:

```
#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)(endstart))/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;
                       }
```

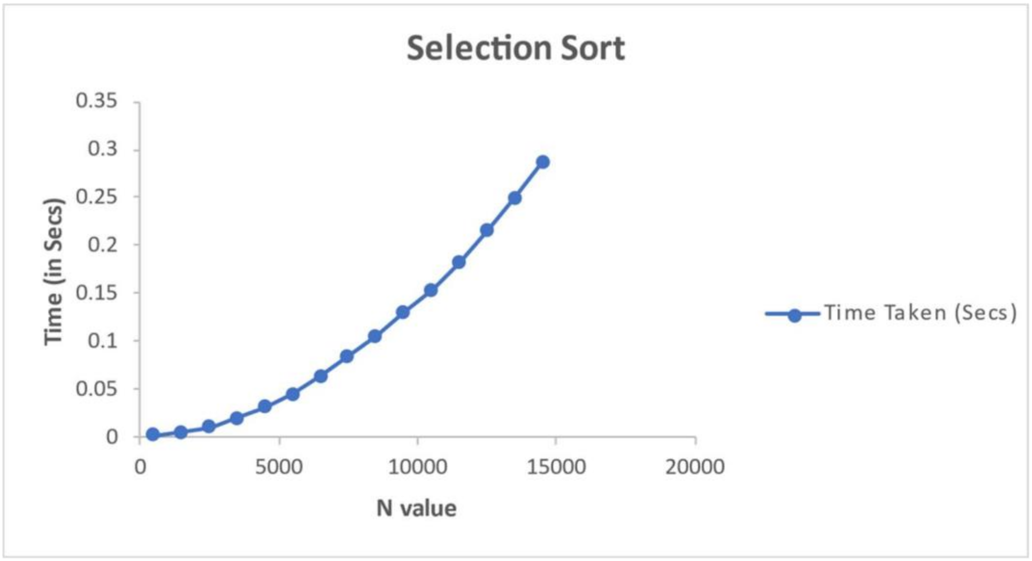
```
                    start=clock();
selsort(n,a);
          //Dummy loop to create delay
              for(j=0;j<500000;j++){ temp=38/600;}
              end=clock();
printf("\n Time taken to sort %d numbers is %f Secs",n,
(((double)(endstart))/CLOCKS_PER_SEC));                          n=n+1000;
                           }
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:



```
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: 6

Enter array elements: 18 78 6 4 64 100

Sorted array is: 4      6       18      64      78      100
 Time taken to sort 6 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 0.001427 Secs
 Time taken to sort 1500 numbers is 0.004233 Secs
 Time taken to sort 2500 numbers is 0.009733 Secs
 Time taken to sort 3500 numbers is 0.017975 Secs
 Time taken to sort 4500 numbers is 0.028802 Secs
 Time taken to sort 5500 numbers is 0.042422 Secs
 Time taken to sort 6500 numbers is 0.058902 Secs
 Time taken to sort 7500 numbers is 0.078219 Secs
 Time taken to sort 8500 numbers is 0.105506 Secs
 Time taken to sort 9500 numbers is 0.132186 Secs
 Time taken to sort 10500 numbers is 0.168036 Secs
 Time taken to sort 11500 numbers is 0.198500 Secs
 Time taken to sort 12500 numbers is 0.229002 Secs
```

Graph:



Lab Program - 7 QuickSort

Code:

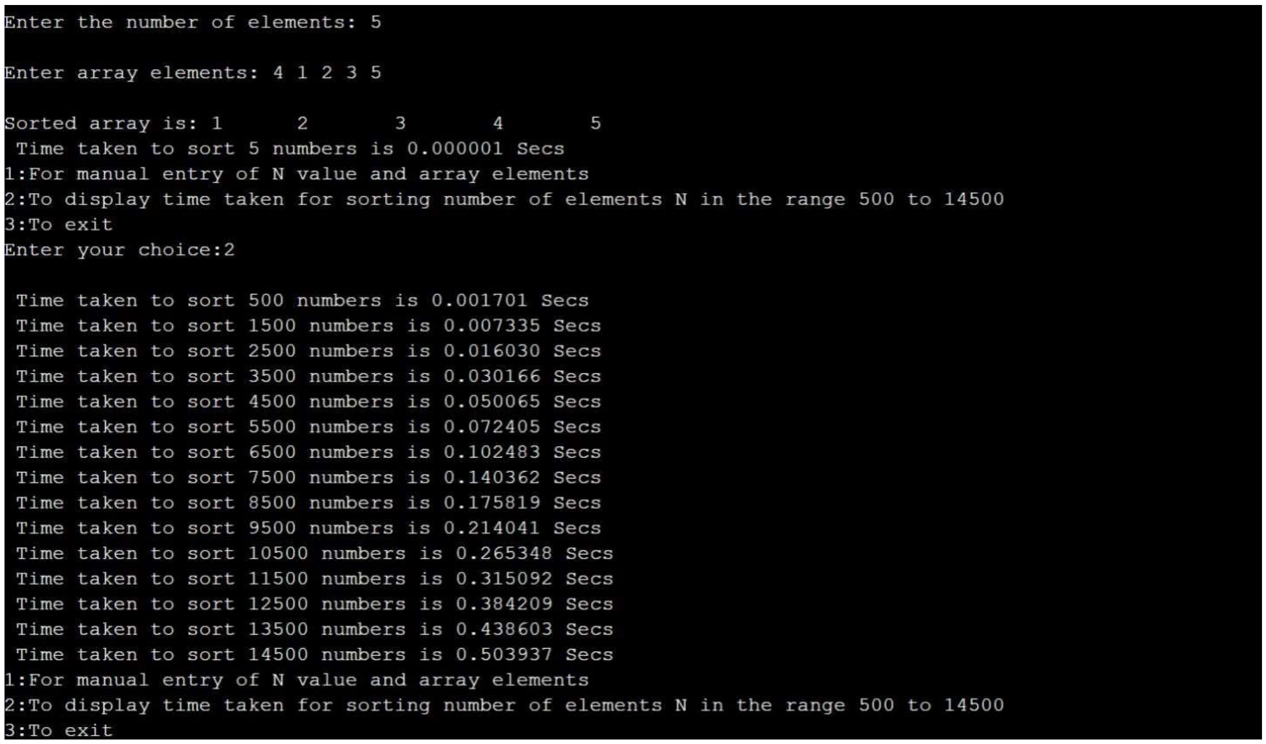
```
#include <stdio.h>

#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)(endstart))/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);
```

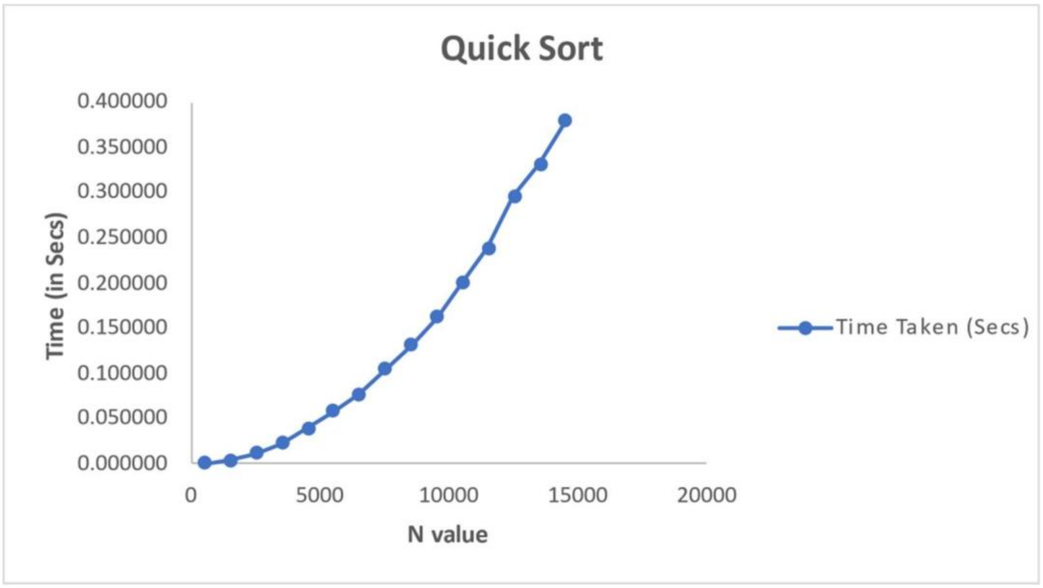
```
                    for(j=0;j<500000;j++){ temp=38/600;}
end=clock();
                    printf("\n Time taken to sort %d numbers is %f Secs",n,
(((double)(endstart))/CLOCKS_PER_SEC));
                         n=n+1000;
              }
break;               case 3:
exit(0);
       }
       getchar();
        }
}
```

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
```

Graph:



Lab Program - 8 HeapSort

Code:

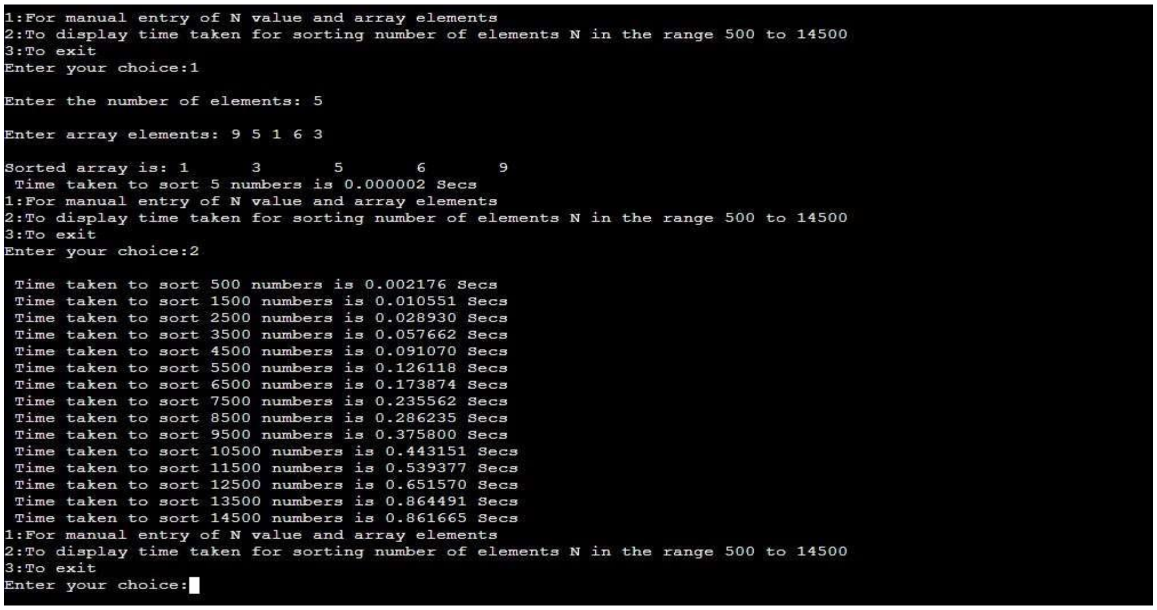
```
#include <stdio.h>

#include <time.h>


void heapify(int n , int a[])

{     int p,c,item;

for(p=(n-1)/2;p>=0;p--)

    {

item=a[p];

c=2*p+1;

while(c<=n-1)
```

```
        {
if(c+1<=n-1)
            {
if(a[c]<a[c+1])
 c++;              }
if(item<a[c])                  {
a[p]=a[c];
p=c;
c=2*p+1;
                }
                else
{
break;
                }
        }
a[p]=item;
    }
}

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

```
    }
}


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();
heapsort(n,a);
end=clock();
```

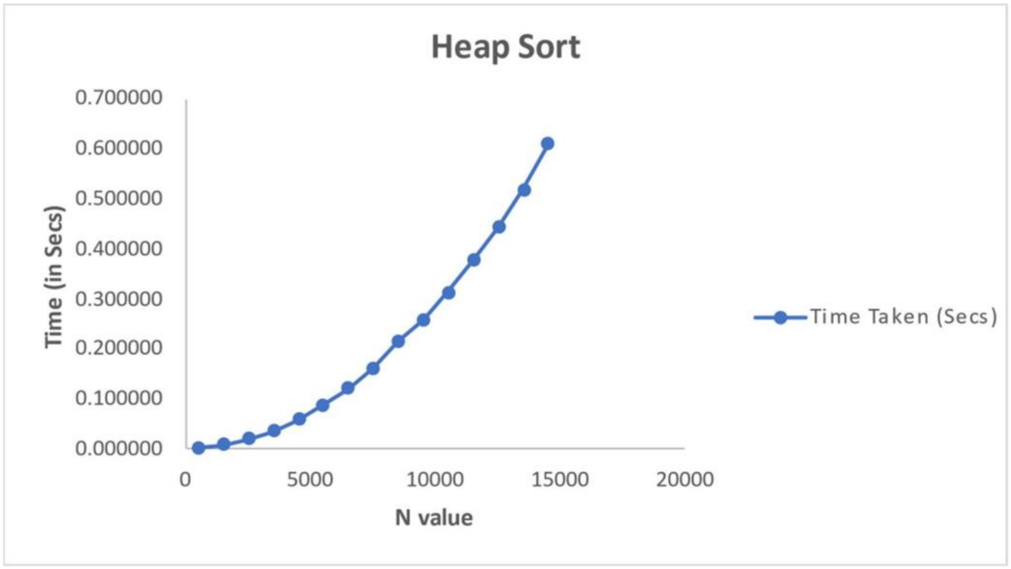
```
                printf("\nSorted array is: ");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
                printf("\n Time taken to sort %d numbers is %f Secs",n,
(((double)(endstart))/CLOCKS_PER_SEC));                  break;            case 2:
n=500;                while(n<=14500) {                    for(i=0;i<n;i++)

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

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:
```

Graph:



Lab Program -

9

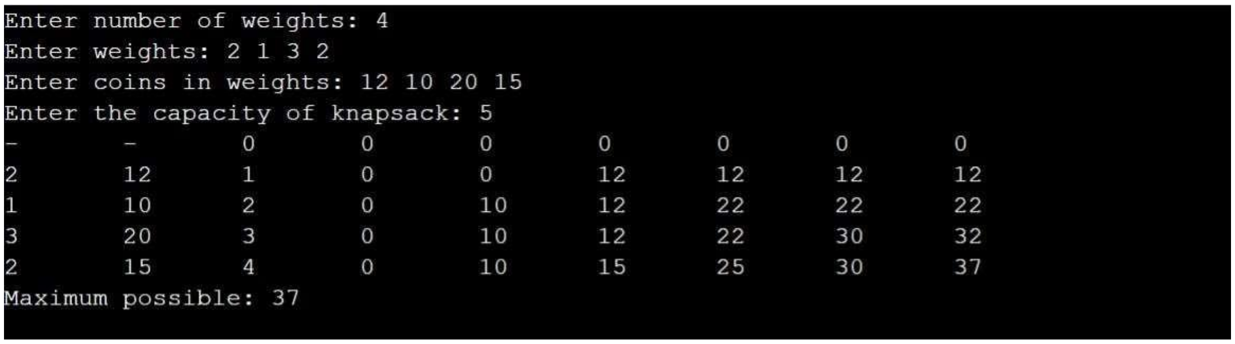
Knapsack

Code:

```
#include<stdio.h> int
max(int a, int b)
{
   return (a > b)? a : b;
}
int knapSack(int W, int wt[], int val[], int n)
{    int i,
w;
  int K[n+1][W+1];
for(i=0;i<=n;i++)
  {
      for(w=0;w<=W;w++)
      {
if(i==0||w==0)
K[i][w]=0;            else
if(wt[i-1]<=w)
              K[i][w]=max(val[i-1] + K[i-1][w-wt[i-1]],K[i-1][w]);
else
              K[i][w]=K[i-1][w];
      }
  }
  return K[n][W];
}
```

```
int main() {
    int i, n, val[20], wt[20], W;
printf("Enter number of items:");
scanf("%d",&n);
    printf("Enter value and weight of items:\n");
for(i=0;i<n;++i)
    {
    scanf("%d%d",&val[i],&wt[i]);
    }
    printf("Enter size of knapsack:");
scanf("%d",&W);
    printf("%d",knapSack(W, wt, val, n));
return 0;
}
```

Output:



```
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
```

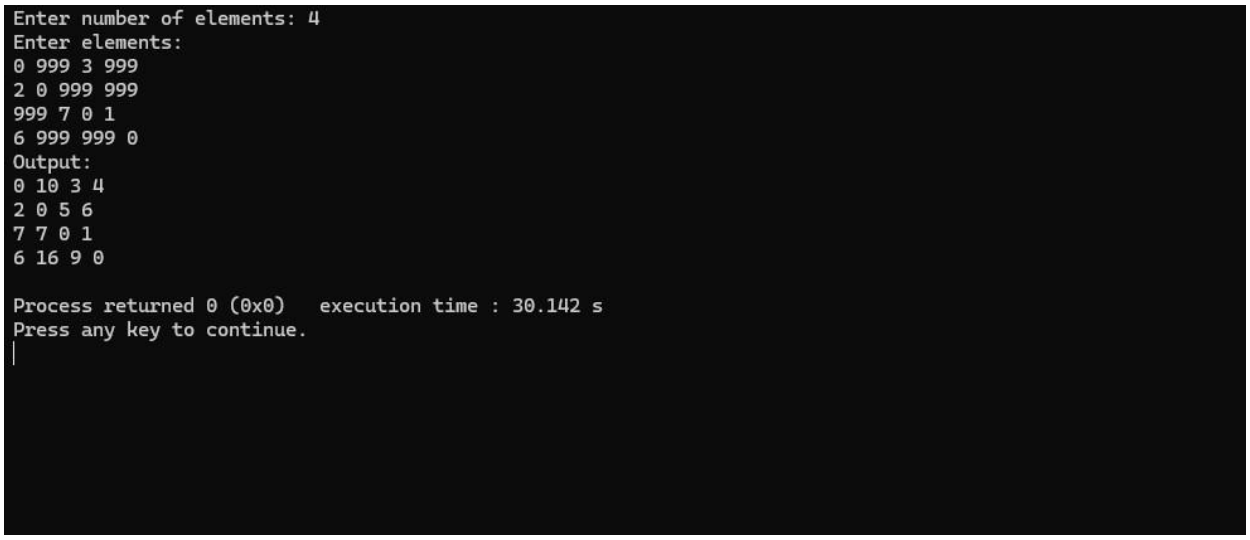
Lab Program -

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:



```
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.
```

11 Prim’s Algorithm

Code:

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

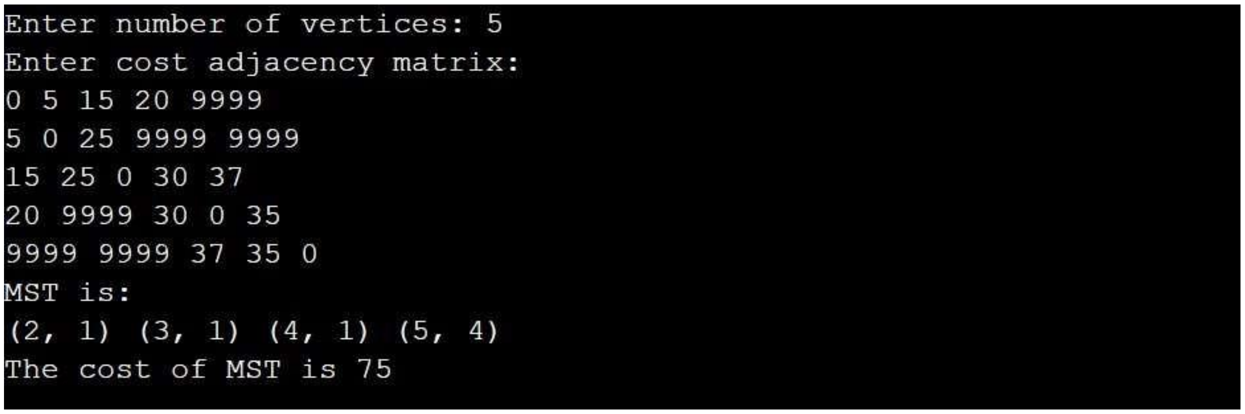
Lab Program -

```
#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);
      }
}
```

Output:



```
Enter number of vertices: 5
Enter cost adjacency matrix:
0 5 15 20 9999
5 0 25 9999 9999
15 25 0 30 37
20 9999 30 0 35
9999 9999 37 35 0
MST is:
(2, 1) (3, 1) (4, 1) (5, 4)
The cost of MST is 75
```

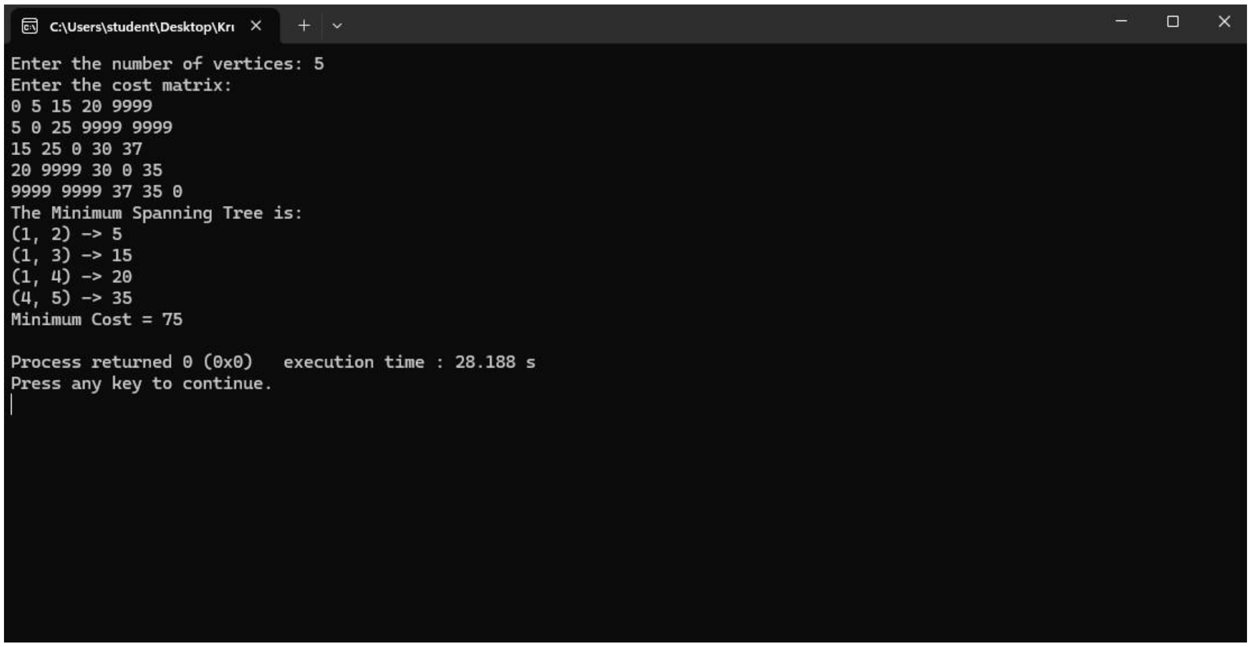
Kruskal’s Algorithm

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; // Mark as visited
    }
    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; // 9999 represents
infinity (no edge)
```

```
        }
    }
    printf("The Minimum Spanning Tree is:\n");
kruskals(c, n);     return 0;
}
```

Output:



```
C:\Users\student\Desktop\Kr
Enter the number of vertices: 5
Enter the cost matrix:
0 5 15 20 9999
5 0 25 9999 9999
15 25 0 30 37
20 9999 30 0 35
9999 9999 37 35 0
The Minimum Spanning Tree is:
(1, 2) -> 5
(1, 3) -> 15
(1, 4) -> 20
(4, 5) -> 35
Minimum Cost = 75

Process returned 0 (0x0)   execution time : 28.188 s
Press any key to continue.
```

Lab Program - 12 Fractional Knapsack using Greedy technique

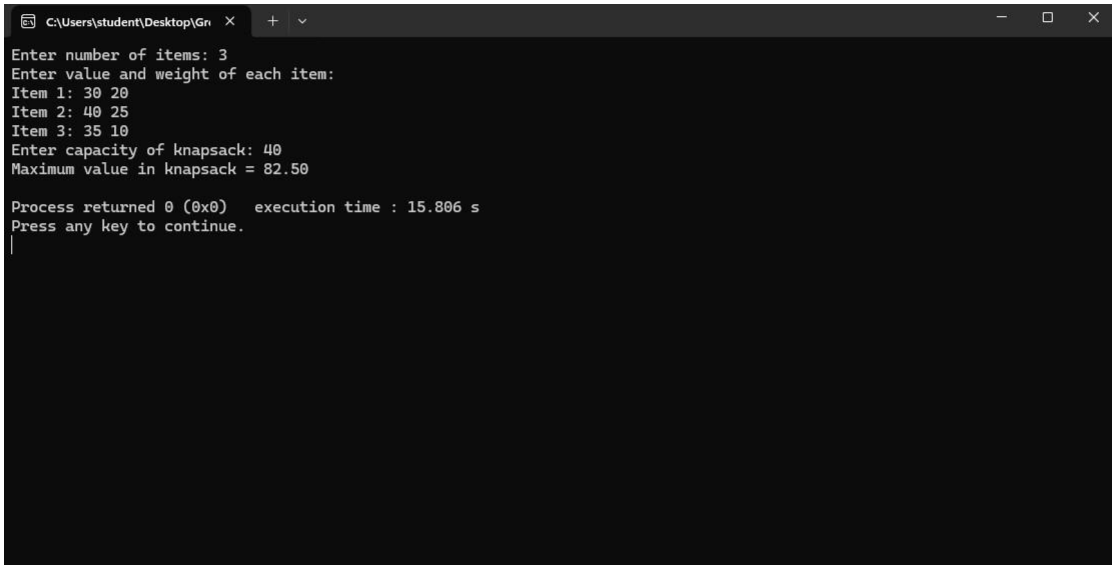
Code:

```
#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:



```
Enter number of items: 3
Enter value and weight of each item:
Item 1: 30 20
Item 2: 40 25
Item 3: 35 10
Enter capacity of knapsack: 40
Maximum value in knapsack = 82.50

Process returned 0 (0x0)   execution time : 15.806 s
Press any key to continue.
```

Lab Program - 13 Dijkstras Algorithm

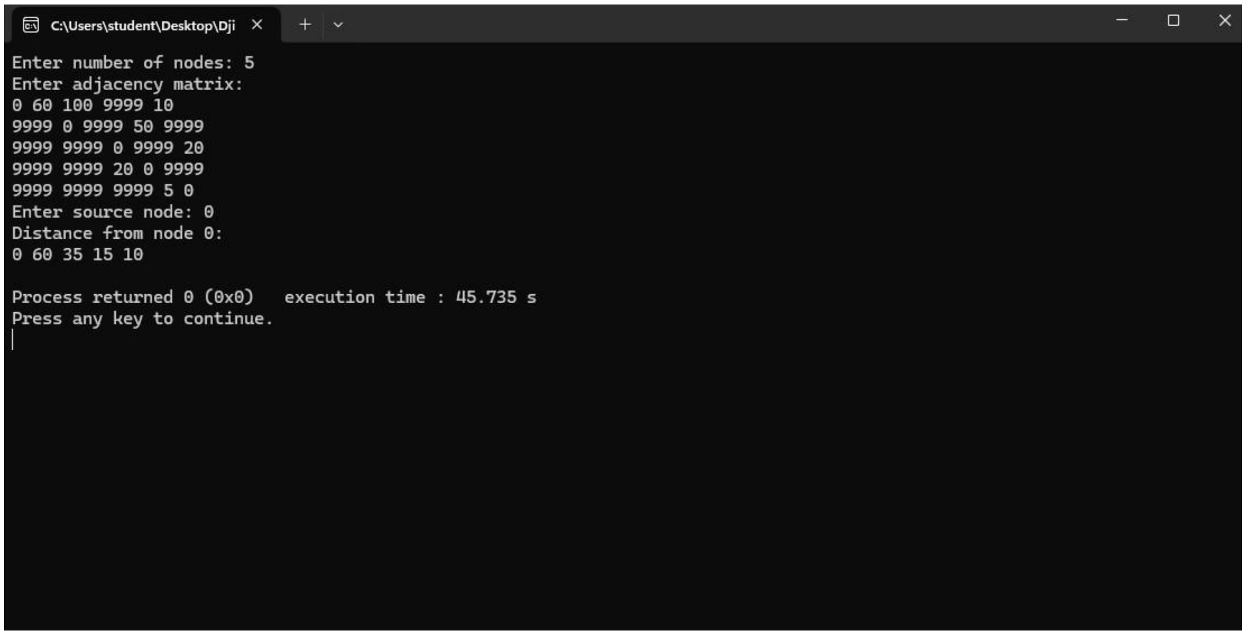
Code:

```
#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

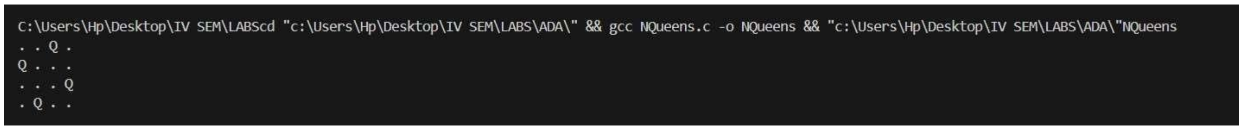
```
Enter number of nodes: 5
Enter adjacency matrix:
0 60 100 9999 10
9999 0 9999 50 9999
9999 9999 0 9999 20
9999 9999 20 0 9999
9999 9999 9999 5 0
Enter source node: 0
Distance from node 0:
0 60 35 15 10

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 . .
```