VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

POOJA M (1BM22CS195)

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 **POOJA M (1BM22CS195),** 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. Nandhini Vineeth
Designation	Associate Professor
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.

1. LEET CODE - 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:

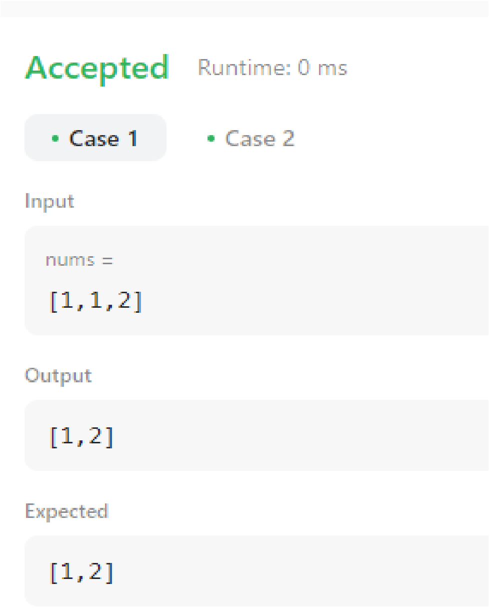
Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.

Return k.

Code:

```
int removeDuplicates(int* nums, int numsSize) {
   if (numsSize == 0) return 0;
int x = 0;
for (int y = 1; y < numsSize; y++) {
if (nums[y] != nums[x]) {
x++;
nums[x] = nums[y];
}
}
return x + 1;
}
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =

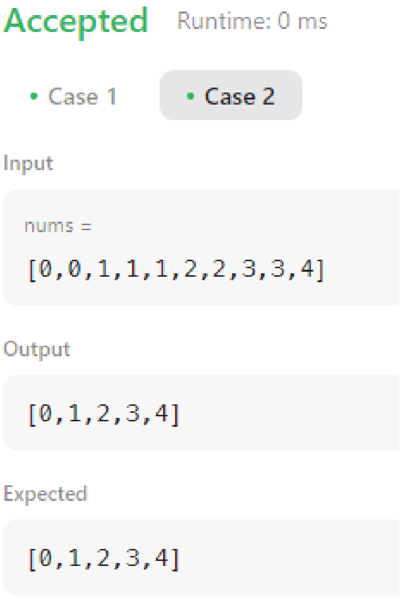
```
[1,1,2]
```

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]
```

2. LEET CODE – 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.

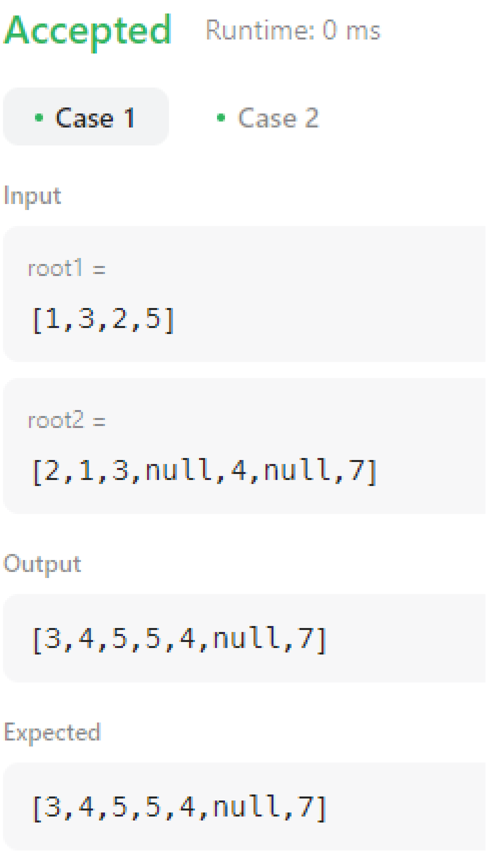
Return the merged tree.

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 == NULL) return root2;
if (root2 == NULL) return root1;
struct TreeNode* merged = (struct TreeNode*)malloc(sizeof(struct TreeNode));
merged->val = root1->val + root2->val;
merged->left = mergeTrees(root1->left, root2->left);
merged->right = mergeTrees(root1->right, root2->right);

return merged;
}
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

root1 =

```
[1,3,2,5]
```

root2 =

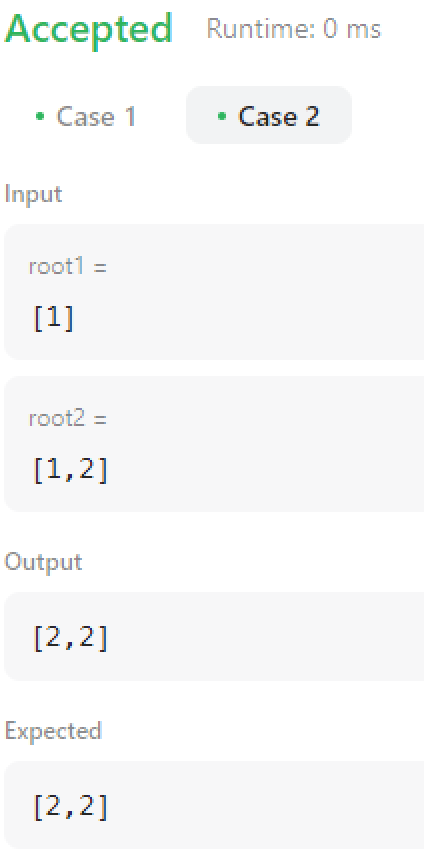
```
[2,1,3,null,4,null,7]
```

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 =

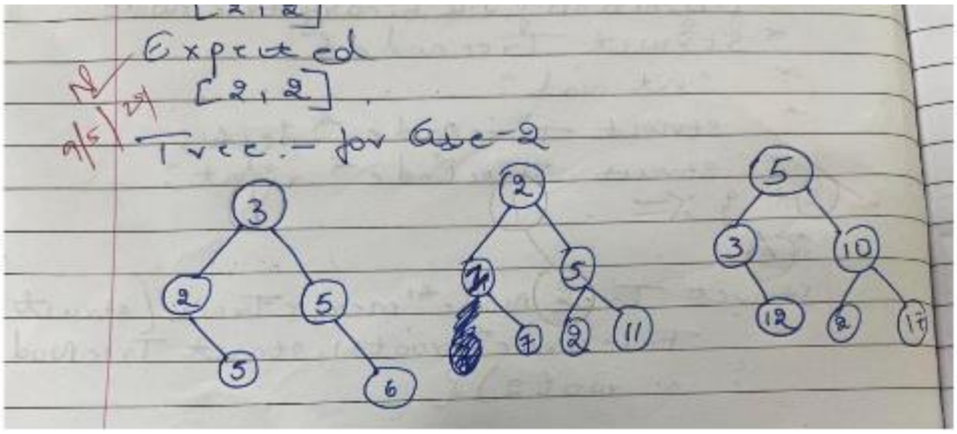
```
[1,2]
```

Output

```
[2,2]
```

Expected

```
[2,2]
```



Expected
[2,2]

Tree:- for Case-2

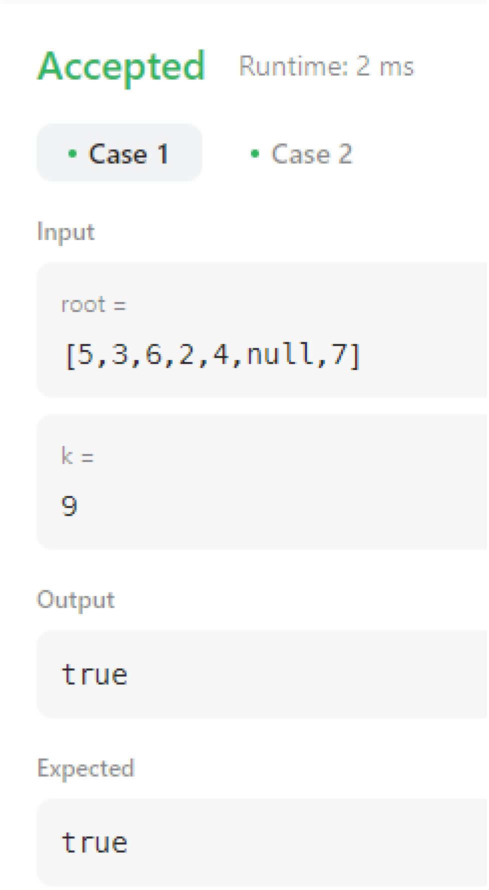
3. LEET CODE – Two Sum IV - Input is a BST

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;
 * };
 */

 bool findTarget(struct TreeNode* root, int k) {
if (root == NULL) {
return false;
}
void inOrderTraversal(struct TreeNode* root, int* arr, int* index) {
if (root == NULL) {
return;
}
inOrderTraversal(root->left, arr, index);
arr[(*index)++] = root->val;
inOrderTraversal(root->right, arr, index);
}
int numNodes = 0;
struct TreeNode* temp = root;
struct TreeNode* stack[100];
int stackSize = 0;
while (temp != NULL || stackSize > 0) {
while (temp != NULL) {
stack[stackSize++] = temp;
temp = temp->left;
}
temp = stack[--stackSize];
numNodes++;
temp = temp->right;
}
int* arr = (int*)malloc(numNodes * sizeof(int));
int index = 0;
inOrderTraversal(root, arr, &index);
int left = 0;
int right = numNodes - 1;
while (left < right) {
int sum = arr[left] + arr[right];
if (sum == k) {
free(arr);
return true;
} else if (sum < k) {
left++;
} else {
right--;
}
}
free(arr);
return false;
}
```

Output:



Accepted Runtime: 2 ms

Input

root =

```
[5,3,6,2,4,null,7]
```

k =

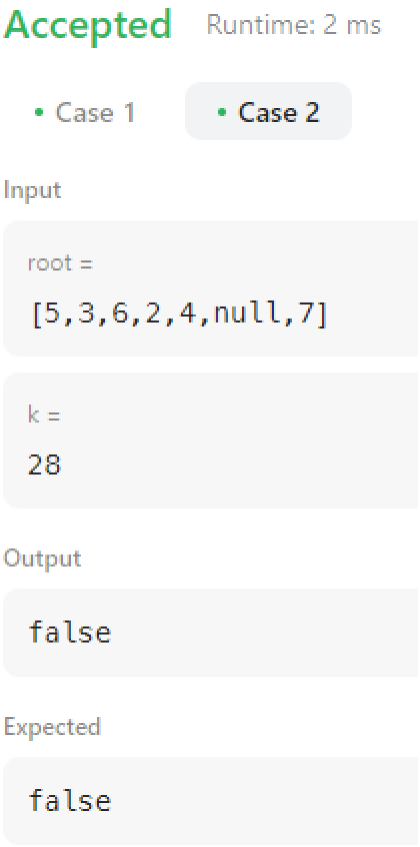
```
9
```

Output

```
true
```

Expected

```
true
```



Accepted Runtime: 2 ms

Input

root =

```
[5,3,6,2,4,null,7]
```

k =

```
28
```

Output

```
false
```

Expected

```
false
```

4. Topological Sort Algorithm Using Source Removal Method

Code:

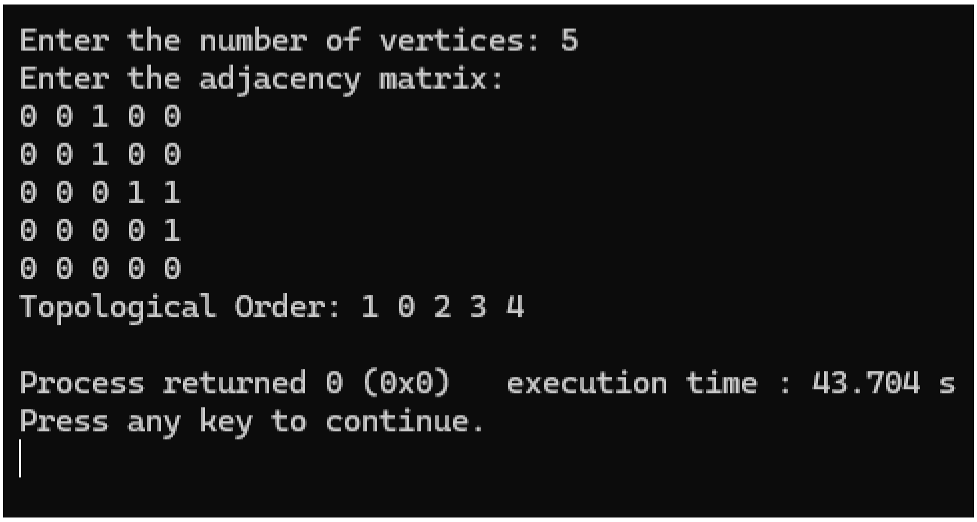
```
#include <stdio.h>

#include <stdlib.h>

void ts(int **a, int n) {

int indegree[n], s[n], top = -1, T[n], k = 0;

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

int sum = 0;
```

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

sum += a[i][j];

}

indegree[j] = sum;

}

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

if (indegree[i] == 0) {

s[++top] = i;

}

}

while (top != -1) {

int u = s[top--];

T[k++] = u;

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

if (a[u][v] == 1) {

indegree[v]--;

if (indegree[v] == 0) {

s[++top] = v;

}

}

}

}

printf("Topological Order: ");

for (int i = 0; i < k; i++) {
```

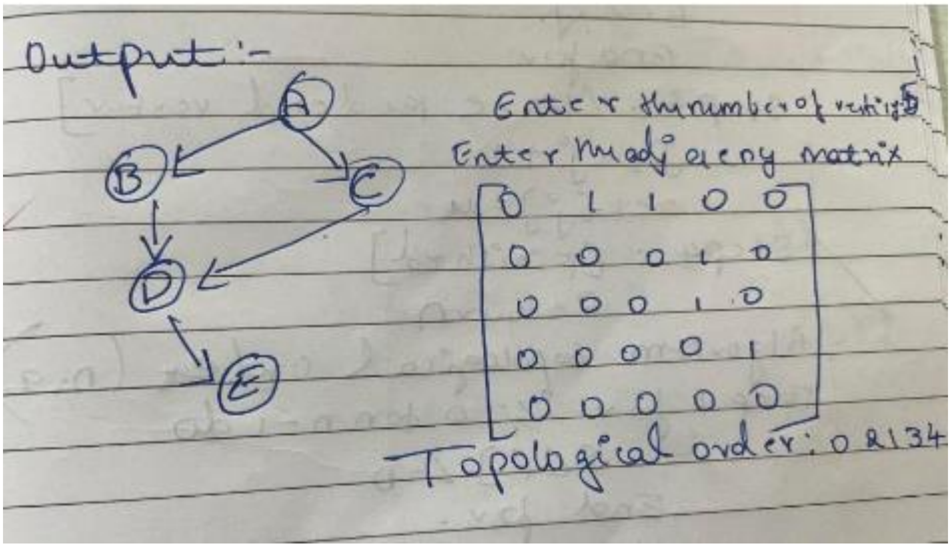
```
printf("%d ", T[i]);

}

printf("\n");

}

int main() {

int n;

printf("Enter the number of vertices: ");

scanf("%d", &n);

int **a = (int **)malloc(n * sizeof(int *));

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

a[i] = (int *)malloc(n * sizeof(int));

}

printf("Enter the adjacency matrix:\n");

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

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

scanf("%d", &a[i][j]);

}

}

ts(a, n);

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

free(a[i]);

}

free(a);

return 0;
```

}

Output:



```
Enter the number of vertices: 5
Enter the 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
Topological Order: 1 0 2 3 4

Process returned 0 (0x0)   execution time : 43.704 s
Press any key to continue.
```



Output:-

```
Enter the number of vertices: 5
Enter the adjaceny matrix
0 1 1 0 0
0 0 0 1 0
0 0 0 1 0
0 0 0 0 1
0 0 0 0 0
Topological order: 0 2134
```

Topological Sort Algorithm Using DFS

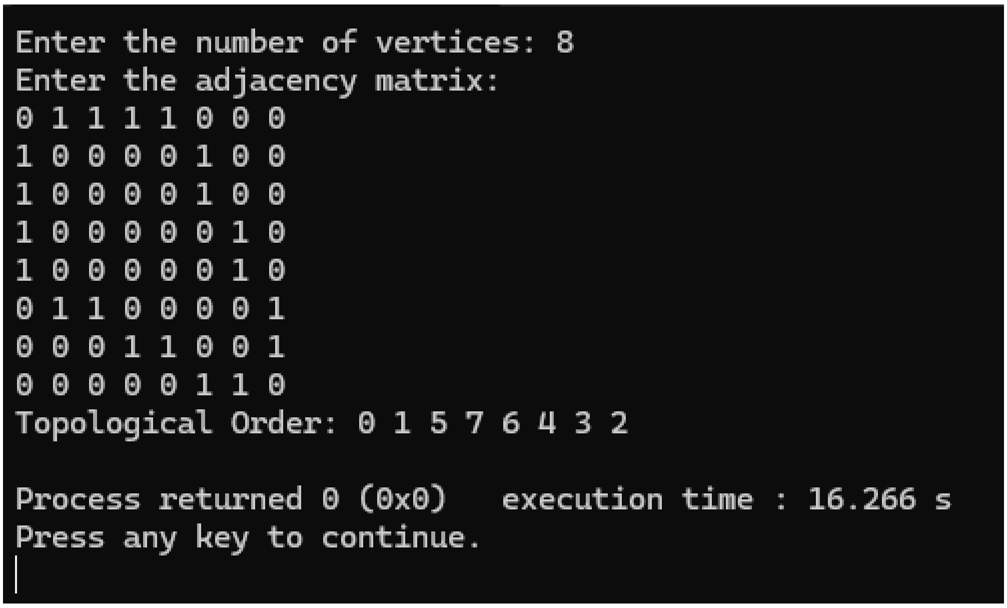
Code:

```
#include <stdio.h>

#include <stdlib.h>

void DFS(int u, int n, int **a, int *s, int *res, int *j) {

s[u] = 1;

for (int v = 0; v < n; v++) {
```

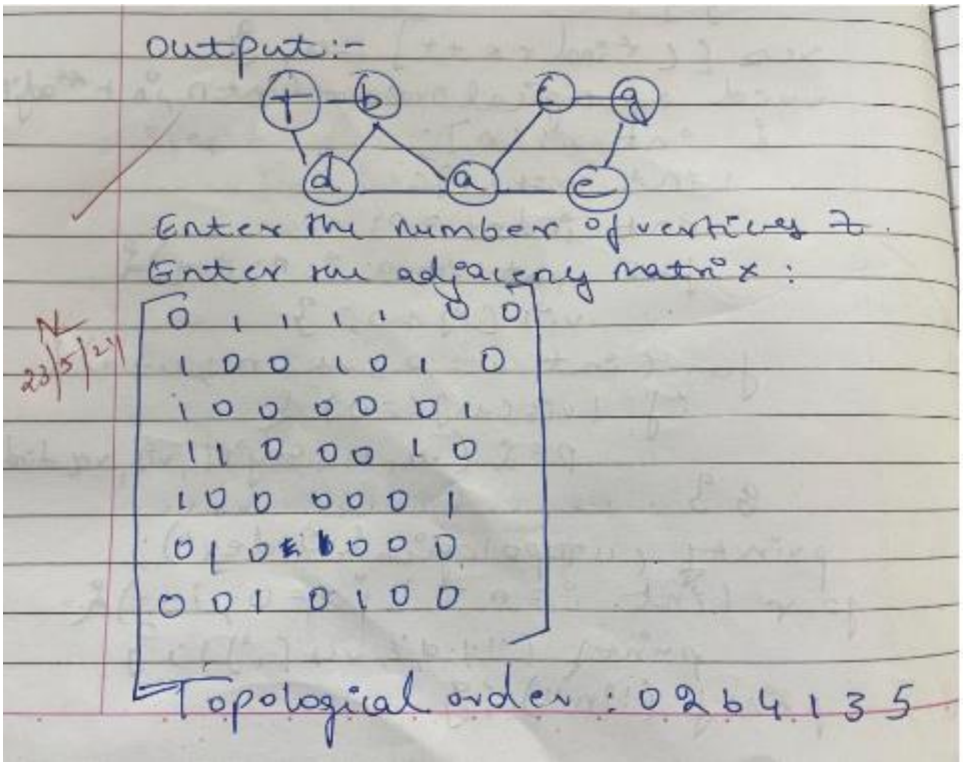
```
if (a[u][v] == 1 && s[v] == 0) {
DFS(v, n, a, s, res, j);
}
}
res[(*j)++] = u;
}
void to(int n, int **a) {
int s[n];
int res[n];
int j = 0;
for (int i = 0; i < n; i++) {
s[i] = 0;
}
for (int u = 0; u < n; u++) {
if (s[u] == 0) {
DFS(u, n, a, s, res, &j);
}
}
printf("Topological Order: ");
for (int i = n - 1; i >= 0; i--) {
printf("%d ", res[i]);
}
printf("\n");
}
```

```
int main() {

int n;

printf("Enter the number of vertices: ");

scanf("%d", &n);

int **a = (int **)malloc(n * sizeof(int *));

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

a[i] = (int *)malloc(n * sizeof(int));

}

printf("Enter the adjacency matrix:\n");

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

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

scanf("%d", &a[i][j]);

}

}

to(n, a);

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

free(a[i]);

}

free(a);

return 0;

}
```

Output:



```
Enter the number of vertices: 8
Enter the adjacency matrix:
0 1 1 1 1 0 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 0 0 0 0 1
0 0 0 1 1 0 0 1
0 0 0 0 0 1 1 0
Topological Order: 0 1 5 7 6 4 3 2

Process returned 0 (0x0)   execution time : 16.266 s
Press any key to continue.
```



Output:-

Enter the number of vertices 7.
Enter the adjacency matrix :

```
0 1 1 1 1 0 0
1 0 0 1 0 1 0
1 0 0 0 0 0 1
1 1 0 0 0 1 0
1 0 0 0 0 0 1
0 1 0 1 0 0 0
0 0 1 0 1 0 0
```

Topological order : 0 2 6 4 1 3 5

23/5/24

Find the Kth Largest Integer in the Array-LEETCODE

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.

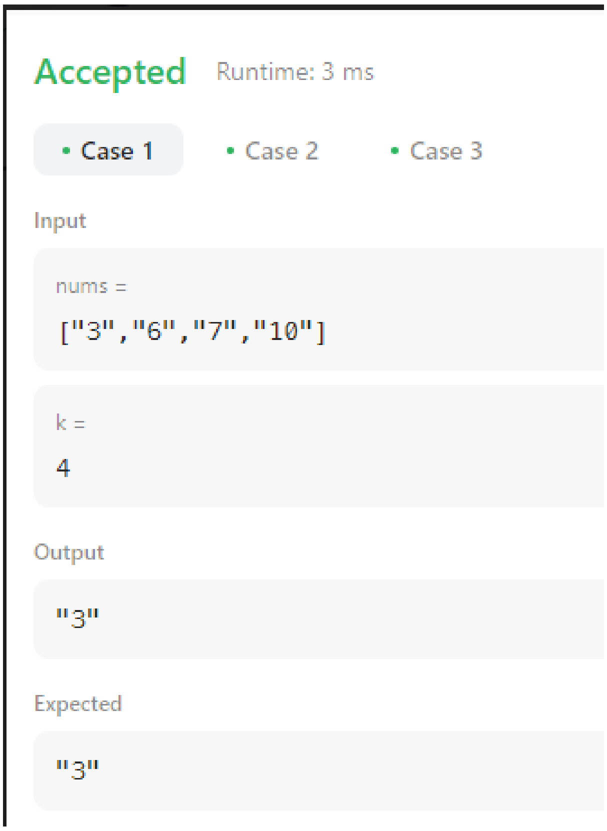
Note: Duplicate numbers should be counted distinctly. For example,
if nums is ["1","2","2"], "2" is the first largest integer, "2" is the second-largest integer, and "1"

is the third-largest integer.

CODE:

```
int compare(const void *a, const void *b) {

    const char *str1 = *(const char **)a;

    const char *str2 = *(const char **)b;

    int len1 = strlen(str1);

    int len2 = strlen(str2);

    if (len1 != len2) {

        return len2 - len1;

    }

    return strcmp(str2, str1);

}

char* kthLargestNumber(char **nums, int numsSize, int k) {

    qsort(nums, numsSize, sizeof(char*), compare);

    return nums[k - 1];

}
```

Output:



Accepted Runtime: 3 ms

Input

nums =

```
["3","6","7","10"]
```

k =

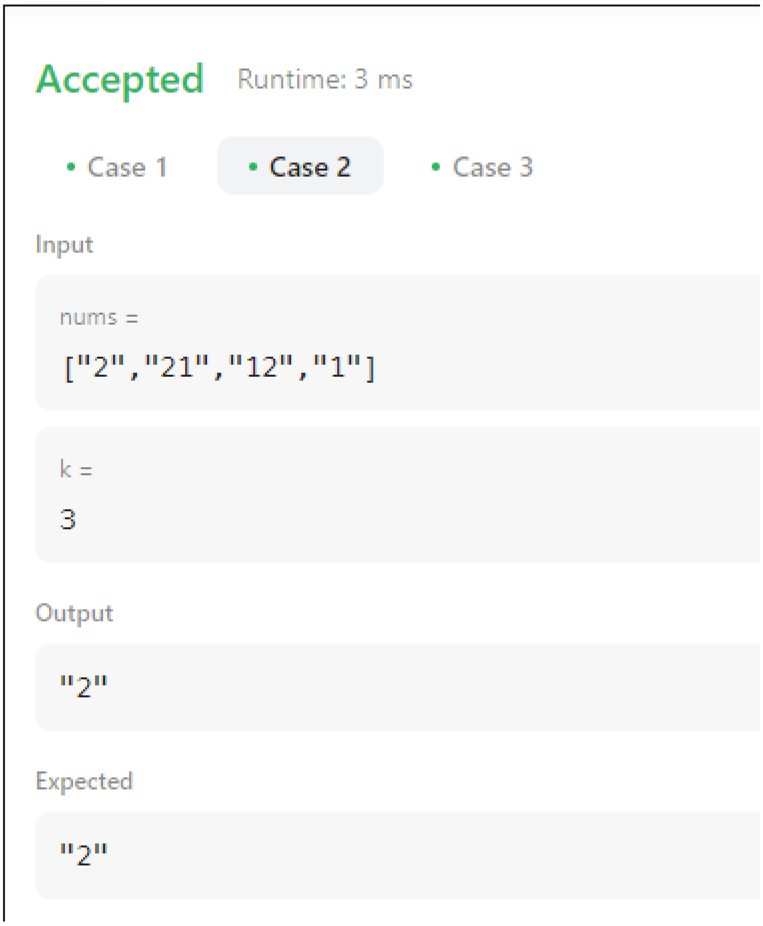
```
4
```

Output

```
"3"
```

Expected

```
"3"
```



Accepted Runtime: 3 ms

Input

```
nums =
["2","21","12","1"]
```

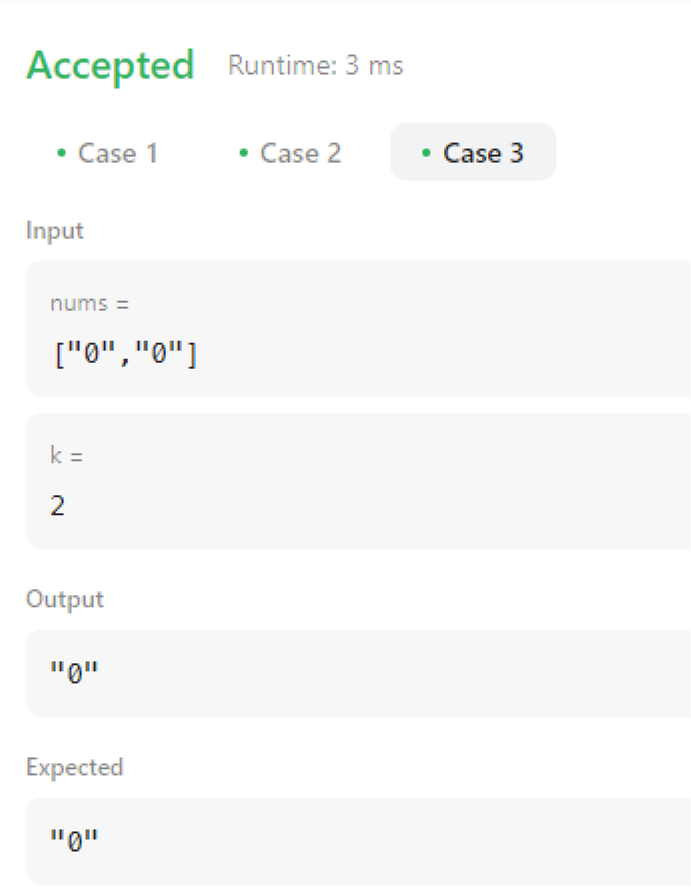
```
k =
3
```

Output

```
"2"
```

Expected

```
"2"
```



Accepted Runtime: 3 ms

Input

```
nums =
["0","0"]
```

```
k =
2
```

Output

```
"0"
```

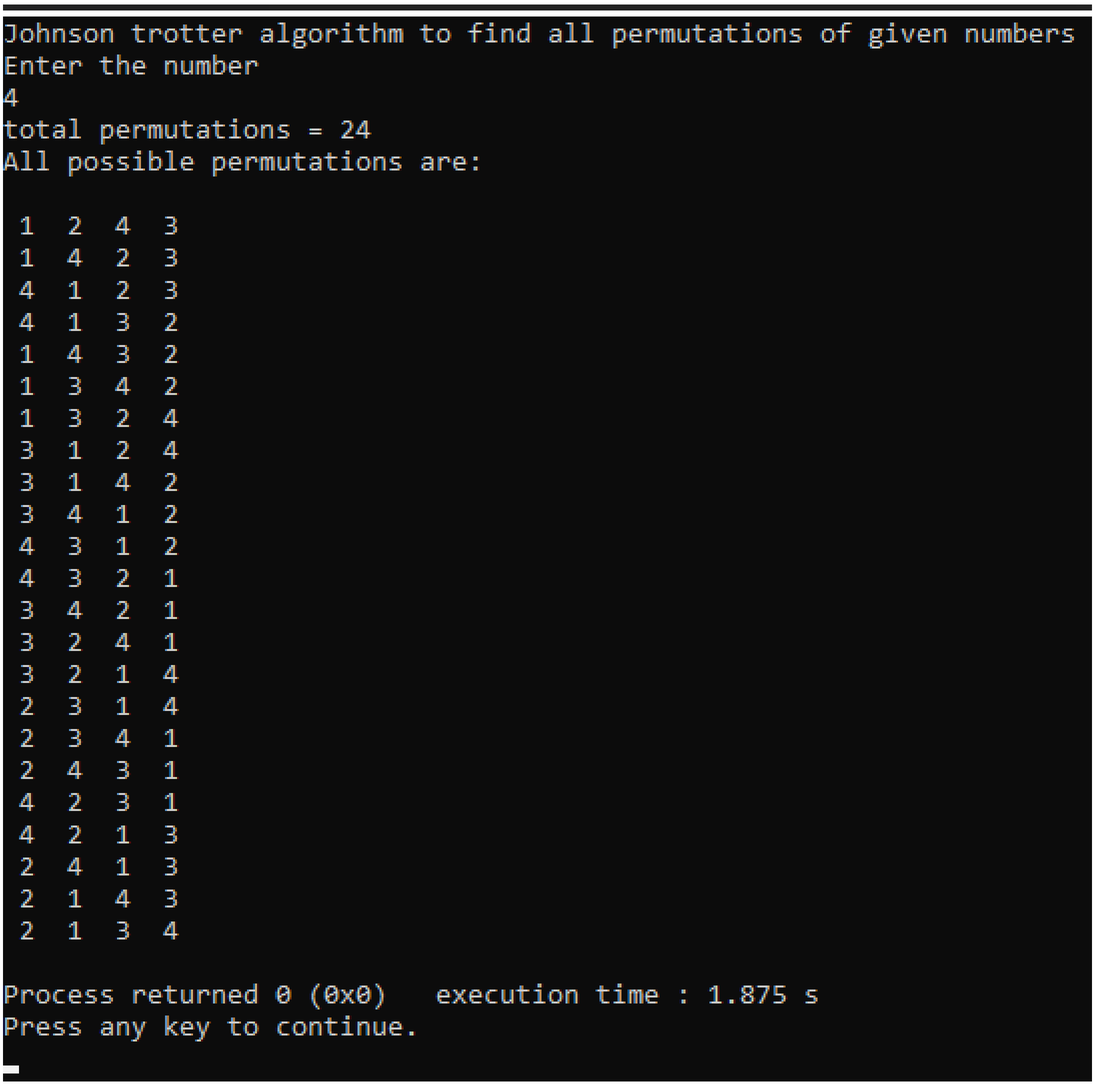
Expected

```
"0"
```

5. Johnson Trotter Algorithm

```
#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_Mobile(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_Mobile(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  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

Process returned 0 (0x0)   execution time : 1.875 s
Press any key to continue.
```

6. Merge Sort

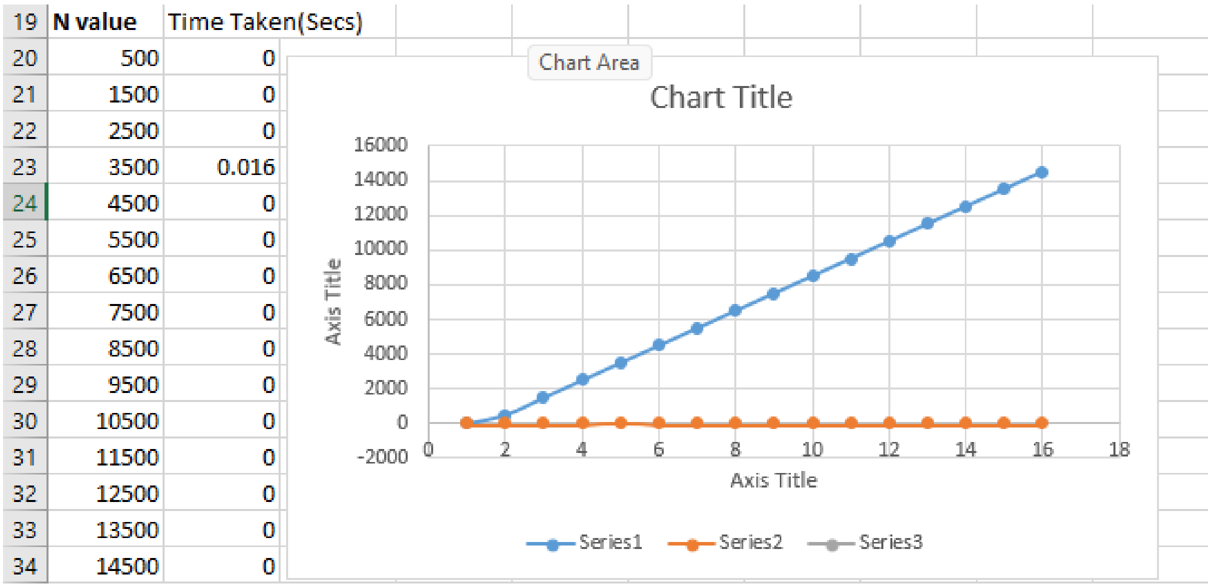
Code:

```
#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: ");
```

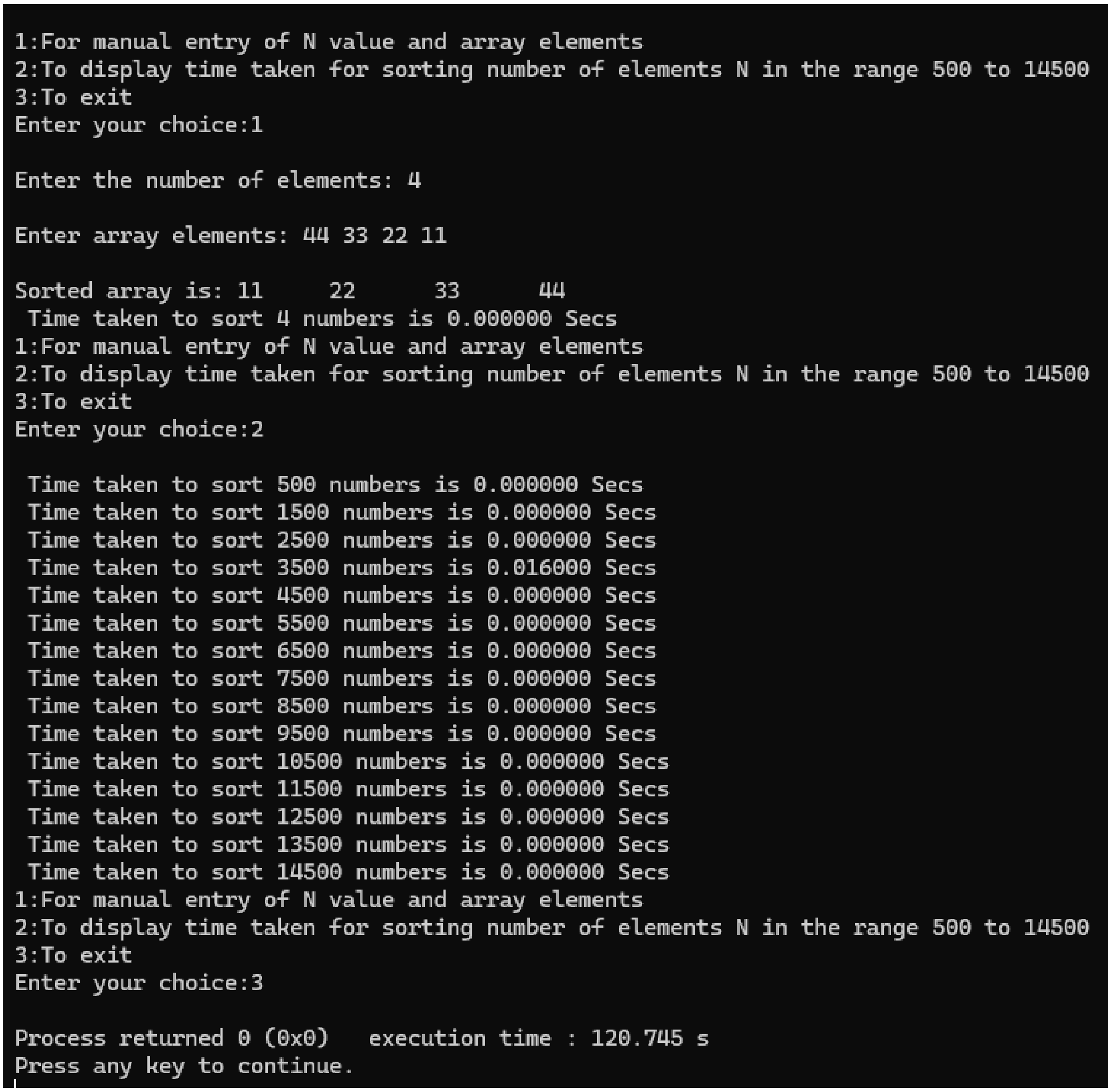
```
                    for(i=0;i<n;i++)
                    printf("%d\t",a[i]);
printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
                    break;
    case 2:
               n=500;
               while(n<=14500) {
               for(i=0;i<n;i++)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

Output:

Graph Scree shot: It can be observed from the graph below that time taken by Selection sort is more when compared to Merge sort.



19	N value	Time Taken(Secs)
20	500	0
21	1500	0
22	2500	0
23	3500	0.016
24	4500	0
25	5500	0
26	6500	0
27	7500	0
28	8500	0
29	9500	0
30	10500	0
31	11500	0
32	12500	0
33	13500	0
34	14500	0



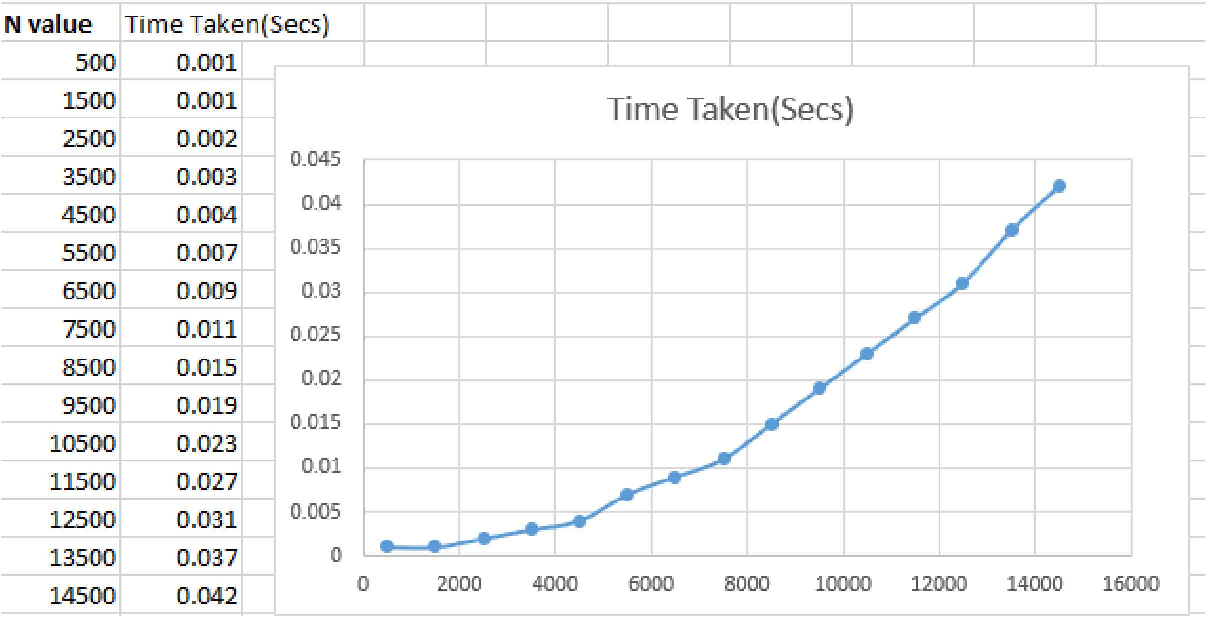
```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1

Enter the number of elements: 4

Enter array elements: 44 33 22 11

Sorted array is: 11     22      33      44
 Time taken to sort 4 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 0.000000 Secs
 Time taken to sort 1500 numbers is 0.000000 Secs
 Time taken to sort 2500 numbers is 0.000000 Secs
 Time taken to sort 3500 numbers is 0.016000 Secs
 Time taken to sort 4500 numbers is 0.000000 Secs
 Time taken to sort 5500 numbers is 0.000000 Secs
 Time taken to sort 6500 numbers is 0.000000 Secs
 Time taken to sort 7500 numbers is 0.000000 Secs
 Time taken to sort 8500 numbers is 0.000000 Secs
 Time taken to sort 9500 numbers is 0.000000 Secs
 Time taken to sort 10500 numbers is 0.000000 Secs
 Time taken to sort 11500 numbers is 0.000000 Secs
 Time taken to sort 12500 numbers is 0.000000 Secs
 Time taken to sort 13500 numbers is 0.000000 Secs
 Time taken to sort 14500 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 : 120.745 s
Press any key to continue.
```

7. Quick Sort

Sort a given set of N integer elements using quick sort technique.

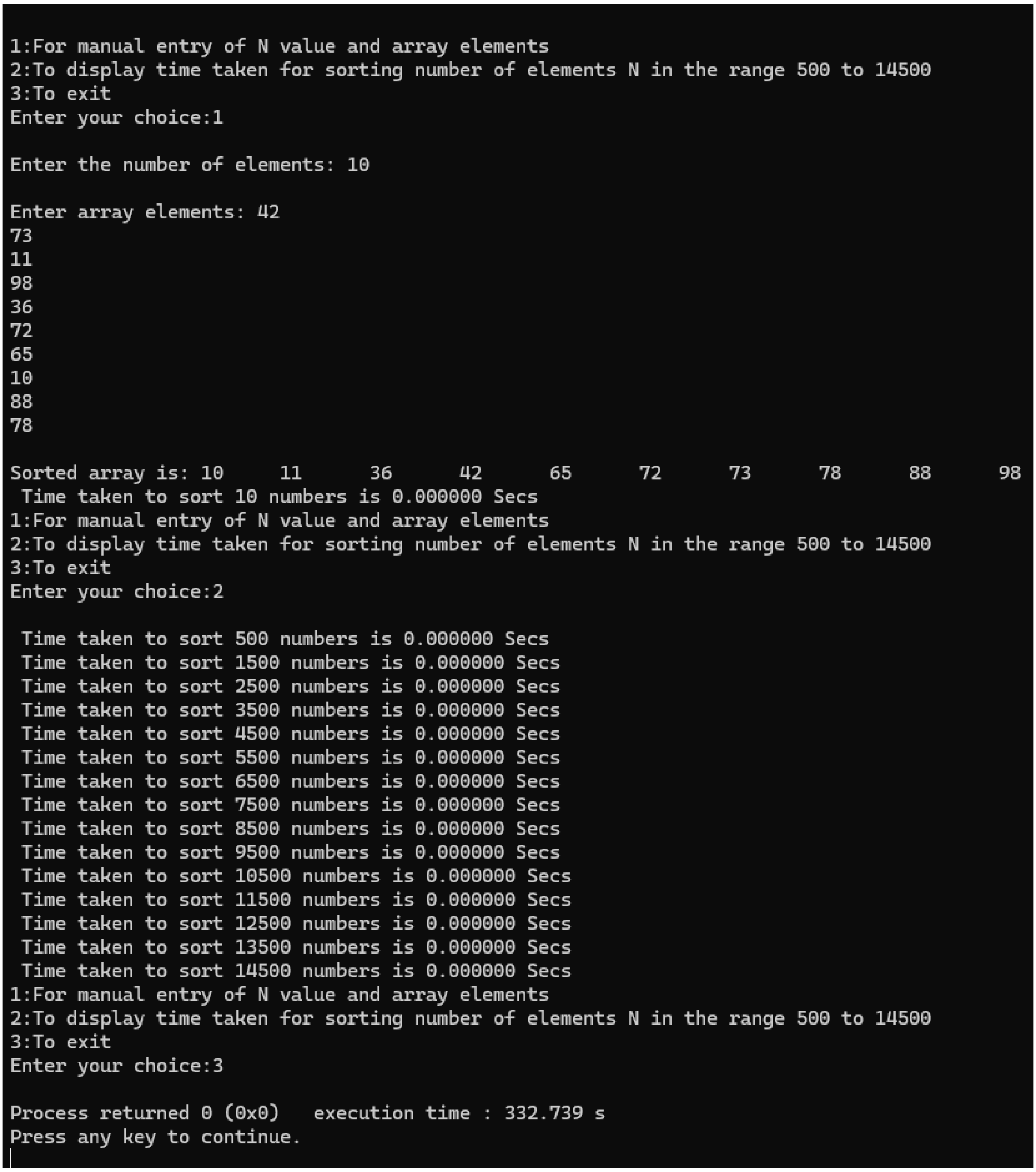
Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void swap(int *a, int *b) {
int t = *a;
*a = *b;
*b = t;
}
int partition(int arr[], int low, int high) {
int pivot = arr[high];
int i = (low - 1);
for (int j = low; j <= high - 1; j++) {
if (arr[j] < pivot) {
i++;
swap(&arr[i], &arr[j]);
}
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);
}
void quickSort(int arr[], int low, int high) {
if (low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}
void printArray(int arr[], int size) {
for (int i = 0; i < size; i++)
printf("%d ", arr[i]);
printf("\n");
}
int main() {
int n;
printf("Enter the number of elements: ");
scanf("%d", &n);
int arr[n];
printf("Enter %d elements: ", n);
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
quickSort(arr, 0, n - 1);
printf("Sorted array: \n");
printArray(arr, n);
return 0;
}
```



N value	Time Taken(Secs)
500	0.001
1500	0.001
2500	0.002
3500	0.003
4500	0.004
5500	0.007
6500	0.009
7500	0.011
8500	0.015
9500	0.019
10500	0.023
11500	0.027
12500	0.031
13500	0.037
14500	0.042

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: 10

Enter array elements: 42
73
11
98
36
72
65
10
88
78

Sorted array is: 10     11      36      42      65      72      73      78      88      98
 Time taken to sort 10 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 0.000000 Secs
 Time taken to sort 1500 numbers is 0.000000 Secs
 Time taken to sort 2500 numbers is 0.000000 Secs
 Time taken to sort 3500 numbers is 0.000000 Secs
 Time taken to sort 4500 numbers is 0.000000 Secs
 Time taken to sort 5500 numbers is 0.000000 Secs
 Time taken to sort 6500 numbers is 0.000000 Secs
 Time taken to sort 7500 numbers is 0.000000 Secs
 Time taken to sort 8500 numbers is 0.000000 Secs
 Time taken to sort 9500 numbers is 0.000000 Secs
 Time taken to sort 10500 numbers is 0.000000 Secs
 Time taken to sort 11500 numbers is 0.000000 Secs
 Time taken to sort 12500 numbers is 0.000000 Secs
 Time taken to sort 13500 numbers is 0.000000 Secs
 Time taken to sort 14500 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 : 332.739 s
Press any key to continue.
```

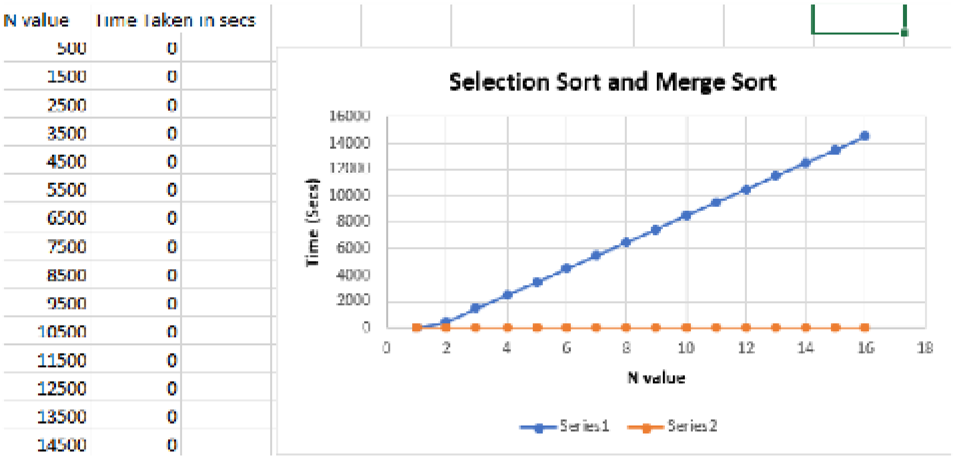
8. A given set of N integer elements using Heap Sort technique and compute its time taken.

```
#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void swap(int* a, int* b)

{

    int temp = *a;

    *a = *b;

    *b = temp;

}

void heapify(int arr[], int N, int i)

{

    int largest = i;

    int left = 2 * i + 1;

    int right = 2 * i + 2;

    if (left < N && arr[left] > arr[largest])

        largest = left;

    if (right < N && arr[right] > arr[largest])

        largest = right;

    if (largest != i) {

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

        heapify(arr, N, largest);

    }
```

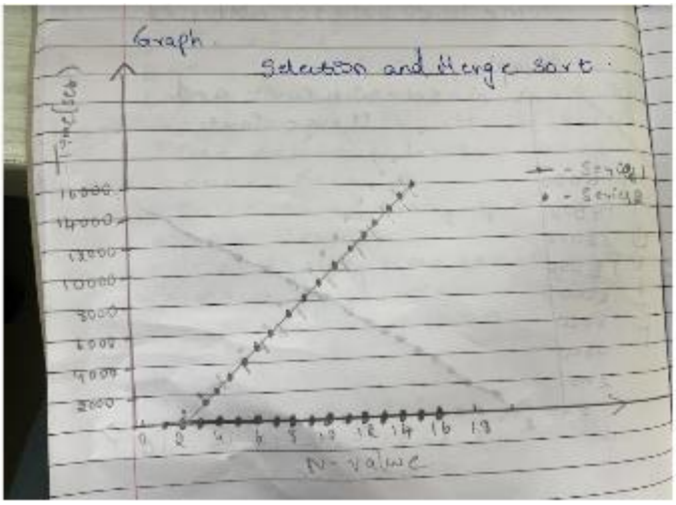
```
}
void heapSort(int arr[], int N)
{
    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);
    for (int i = N - 1; i >= 0; i--) {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}
void main(){
   int a[100000],n,i,j,ch,temp;
   clock_t start,end;
   while(1){
    printf("\n1:For 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: ");
```

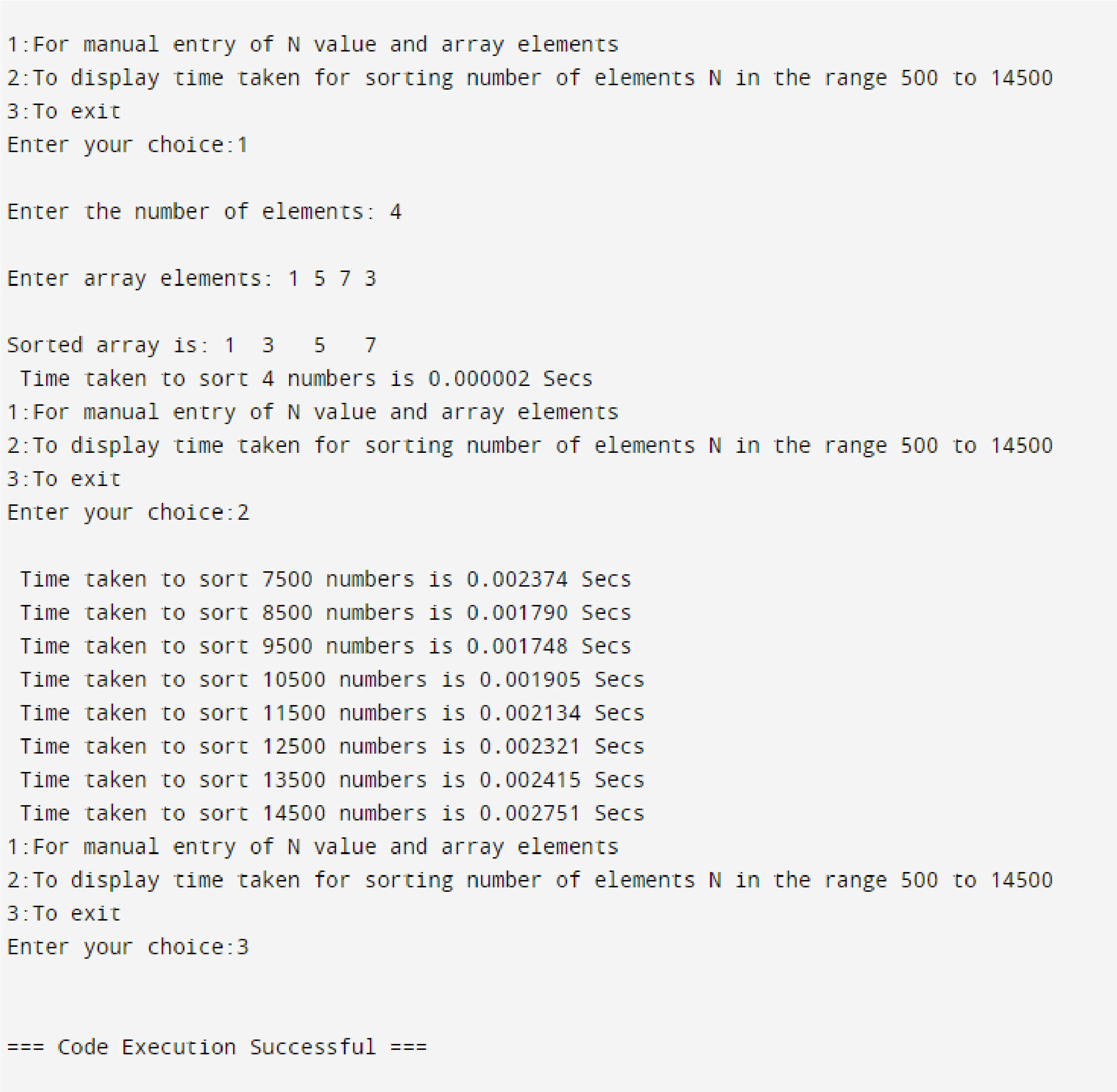
```
         for(i=0;i<n;i++){

             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=7500;

     while(n<=14500) {

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

             a[i]=n-i;

            }

            start=clock();

            heapSort(a,n);

            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:



N value	Time Taken in secs
500	0
1500	0
2500	0
3500	0
4500	0
5500	0
6500	0
7500	0
8500	0
9500	0
10500	0
11500	0
12500	0
13500	0
14500	0





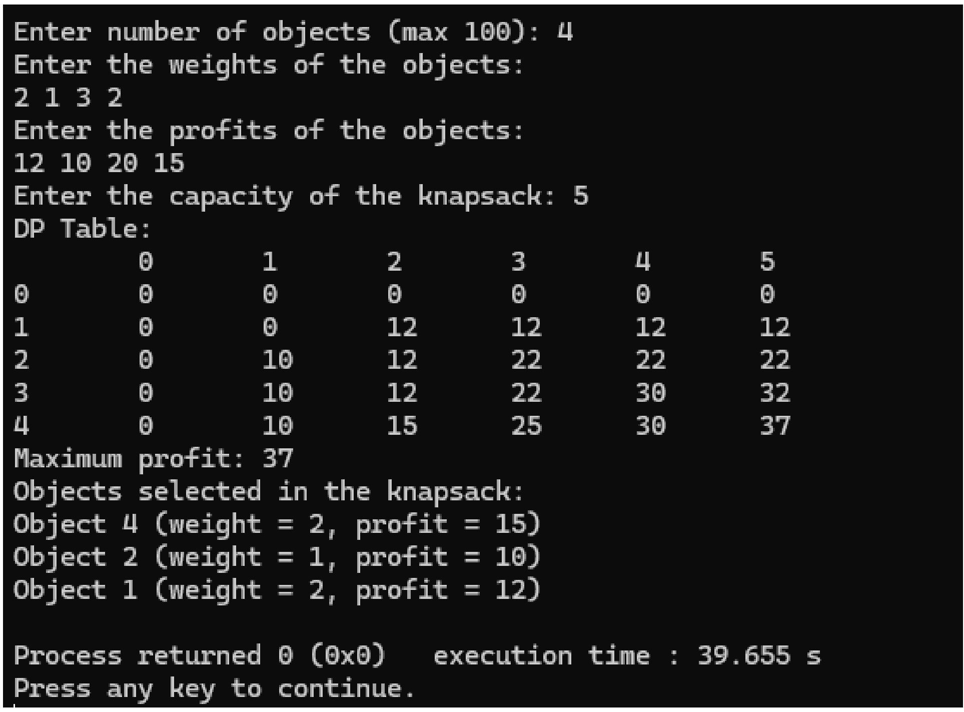
```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1

Enter the number of elements: 4

Enter array elements: 1 5 7 3

Sorted array is: 1   3   5   7
 Time taken to sort 4 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 7500 numbers is 0.002374 Secs
 Time taken to sort 8500 numbers is 0.001790 Secs
 Time taken to sort 9500 numbers is 0.001748 Secs
 Time taken to sort 10500 numbers is 0.001905 Secs
 Time taken to sort 11500 numbers is 0.002134 Secs
 Time taken to sort 12500 numbers is 0.002321 Secs
 Time taken to sort 13500 numbers is 0.002415 Secs
 Time taken to sort 14500 numbers is 0.002751 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


=== Code Execution Successful ===
```

9. Perform Knapsack problem using Dynamic programming technique using n=4 objects with associated weights and profits . Display the table values and the objects selected in the knapsack to get maximum profit.

Code:

```
#include <stdio.h>

#define MAX_OBJECTS 100

int max(int a, int b) {

    return (a > b) ? a : b;

}

void knapsack(int n, int W, int weights[], int profits[]) {

    int i, w;

    int K[MAX_OBJECTS + 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 (weights[i - 1] <= w)
            K[i][w] = max(profits[i - 1] + K[i - 1][w - weights[i - 1]], K[i - 1][w]);
        else
            K[i][w] = K[i - 1][w];
    }
}
printf("DP Table:\n");
printf("\t");
for (w = 0; w <= W; w++) {
    printf("%d\t", w);
}
printf("\n");
for (i = 0; i <= n; i++) {
    printf("%d\t", i);
    for (w = 0; w <= W; w++) {
        printf("%d\t", K[i][w]);
    }
    printf("\n");
}
int maxProfit = K[n][W];
printf("Maximum profit: %d\n", maxProfit);
printf("Objects selected in the knapsack:\n");
int res = maxProfit;
w = W;
for (i = n; i > 0 && res > 0; i--) {
        if (res == K[i - 1][w])
            continue;
        else {
            printf("Object %d (weight = %d, profit = %d)\n", i, weights[i - 1], profits[i - 1]);
            res -= profits[i - 1];
            w -= weights[i - 1];
        }
    }
}
int main() {
    int n, W;
    int weights[MAX_OBJECTS], profits[MAX_OBJECTS];
    int i;
    printf("Enter number of objects (max %d): ", MAX_OBJECTS);
    scanf("%d", &n);
    if (n <= 0 || n > MAX_OBJECTS) {
        printf("Invalid number of objects\n");
        return 1;
    }
    printf("Enter the weights of the objects:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
    }
    printf("Enter the profits of the objects:\n");
    for (i = 0; i < n; i++) {

        scanf("%d", &profits[i]);

    }

    printf("Enter the capacity of the knapsack: ");

    scanf("%d", &W);

    if (W <= 0) {

        printf("Invalid knapsack capacity\n");

        return 1;

    }

    knapsack(n, W, weights, profits);

    return 0;

}
```

Output:

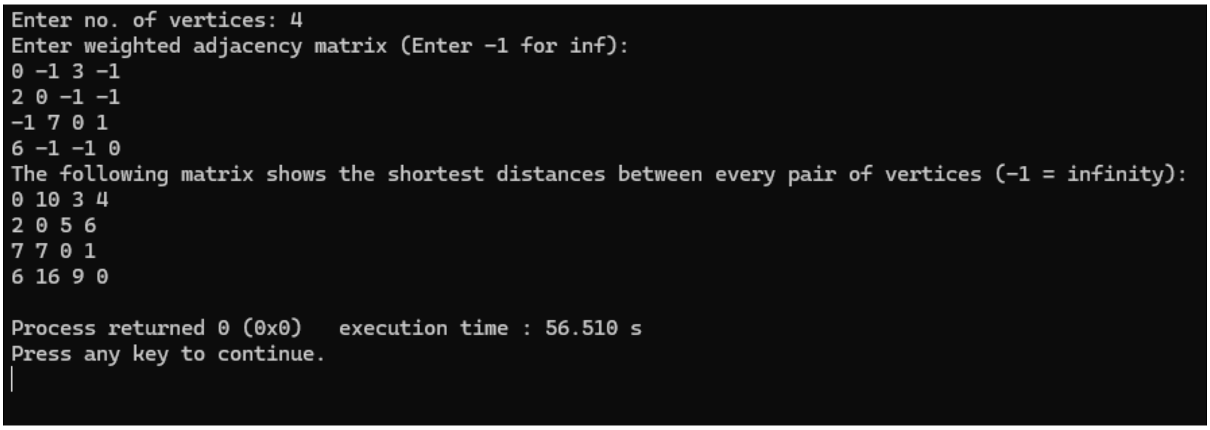


```
Enter number of objects (max 100): 4
Enter the weights of the objects:
2 1 3 2
Enter the profits of the objects:
12 10 20 15
Enter the capacity of the knapsack: 5
DP Table:
        0       1       2       3       4       5
0       0       0       0       0       0       0
1       0       0       12      12      12      12
2       0       10      12      22      22      22
3       0       10      12      22      30      32
4       0       10      15      25      30      37
Maximum profit: 37
Objects selected in the knapsack:
Object 4 (weight = 2, profit = 15)
Object 2 (weight = 1, profit = 10)
Object 1 (weight = 2, profit = 12)

Process returned 0 (0x0)   execution time : 39.655 s
Press any key to continue.
```

10. Implement All Pair Shortest paths problem using Floyd’s algorithm

```
#include <stdio.h>

#include <limits.h>

int INF = 1e5;

void printSolution(int v, int dist[v][v]) {

   printf("The following matrix shows the shortest distances between every pair of vertices (-1 = infinity):\n");

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

      for (int j = 0; j < v; j++) {

         if (dist[i][j] == INF)

            printf("-1 ");

         else

            printf("%d ", dist[i][j]);

      }

      printf("\n");

   }

}

void floydWarshall(int v, int graph[v][v]) {
    int dist[v][v], i, j, k;
    for (i = 0; i < v; i++)
        for (j = 0; j < v; j++)
            dist[i][j] = graph[i][j];
    for (k = 0; k < v; k++) {
        for (i = 0; i < v; i++) {
            for (j = 0; j < v; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    printSolution(v, dist);
}
int main() {
    int v;
    printf("Enter no. of vertices: ");
    scanf("%d", &v);
    int graph[v][v];
    printf("Enter weighted adjacency matrix (Enter -1 for inf): \n");
    for(int i = 0; i < v; i++){
        for(int j = 0; j < v; j++){
            scanf("%d", &graph[i][j]);
            if (graph[i][j] == -1) graph[i][j] = INF;
        }
    }
    floydWarshall(v, graph);
    return 0;
}
```

Output:



```
Enter no. of vertices: 4
Enter weighted adjacency matrix (Enter -1 for inf):
0 -1 3 -1
2 0 -1 -1
-1 7 0 1
6 -1 -1 0
The following matrix shows the shortest distances between every pair of vertices (-1 = infinity):
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0

Process returned 0 (0x0)   execution time : 56.510 s
Press any key to continue.
```

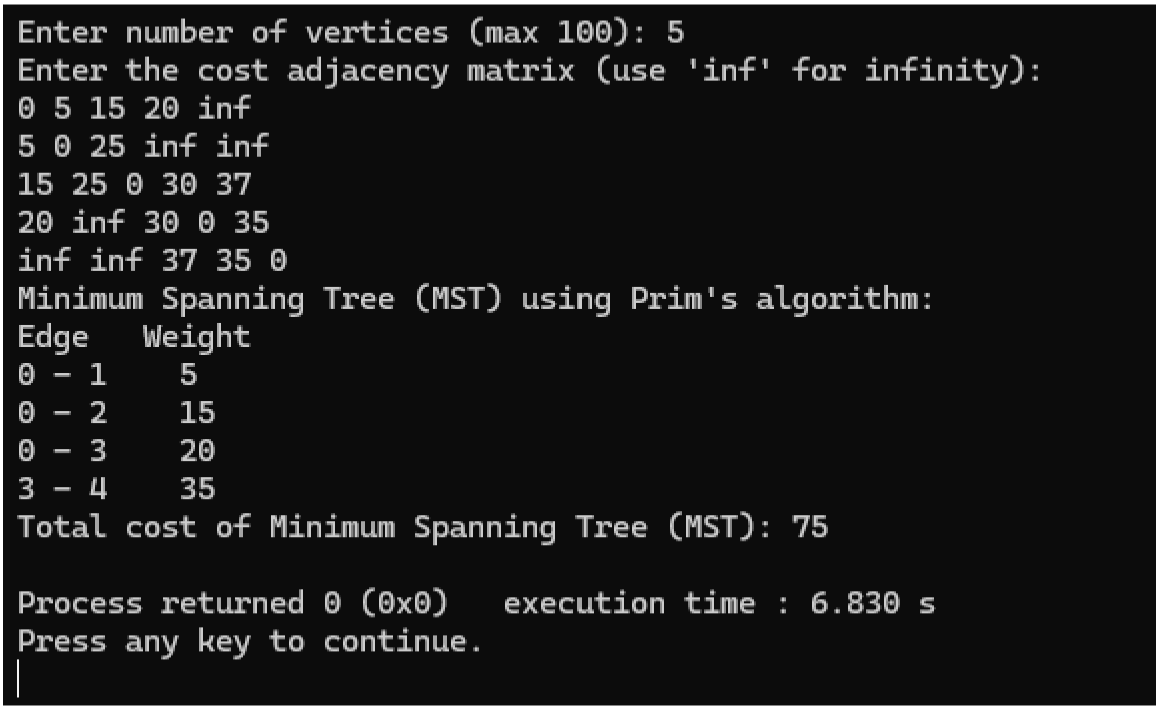
11. Pfa of the Prims algorithm pseudo code please try to convert this into C program and find the MST of a Given graph with cost adjacency matrix as input.

Code:

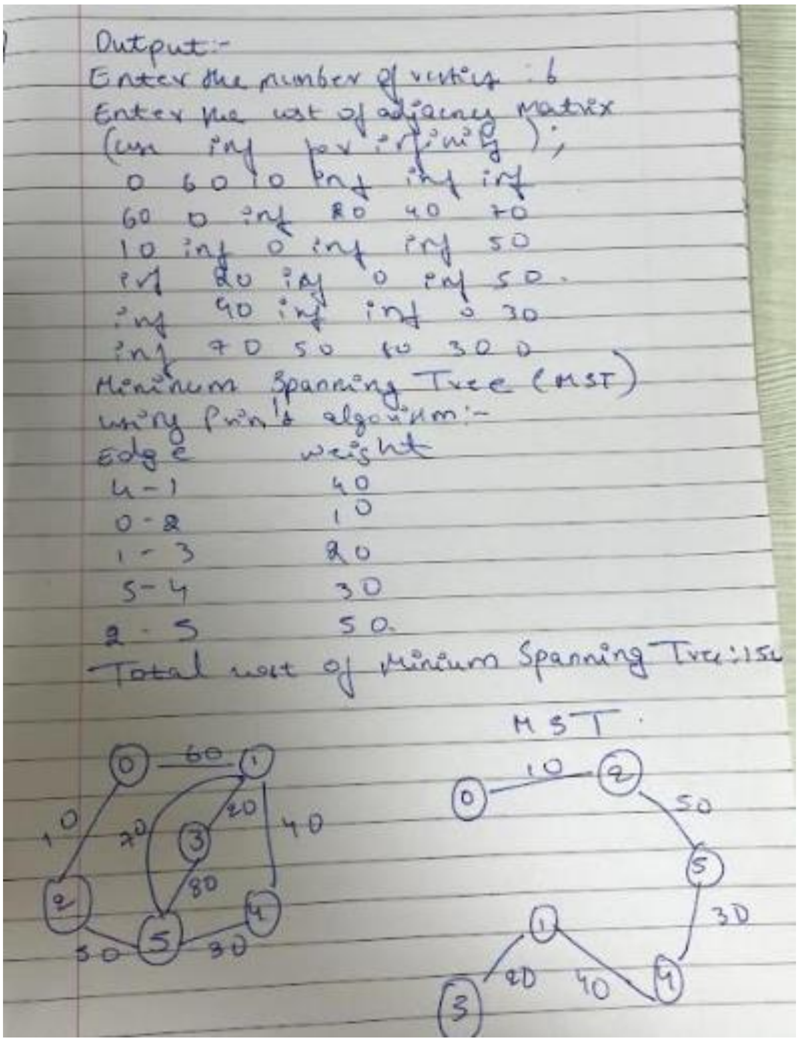
```
#include <stdio.h>

#include <string.h>

#include <limits.h>

#define MAX_VERTICES 100

#define INF INT_MAX

int minKey(int n, int d[], int s[]) {

    int min = INF, min_index;

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

        if (s[v] == 0 && d[v] < min) {

            min = d[v];

            min_index = v;

        }

    }

    return min_index;

}

int printMST(int n, int p[], int cost[MAX_VERTICES][MAX_VERTICES]) {
    int total_cost = 0;
    printf("Edge   Weight\n");
    for (int i = 1; i < n; i++) {
        printf("%d - %d    %d \n", p[i], i, cost[i][p[i]]);
        total_cost += cost[i][p[i]];
    }
    return total_cost;
}
int parseCost(int n, int cost[MAX_VERTICES][MAX_VERTICES]) {
    char input[10];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%s", input);
            if (strcmp(input, "inf") == 0) {
                cost[i][j] = INF;
            } else {
                sscanf(input, "%d", &cost[i][j]);
                if (cost[i][j] == 0 && i != j) {
                    cost[i][j] = INF;
                }
            }
        }
    }
}
```

```
void primMST(int n, int cost[MAX_VERTICES][MAX_VERTICES]) {
    int p[MAX_VERTICES];
    int d[MAX_VERTICES];
    int s[MAX_VERTICES];
    for (int i = 0; i < n; i++) {
        d[i] = INF;
        s[i] = 0;
    }
    d[0] = 0;
    p[0] = -1;
    for (int count = 0; count < n - 1; count++) {
        int u = minKey(n, d, s);
        s[u] = 1;
        for (int v = 0; v < n; v++) {
            if (cost[u][v] && s[v] == 0 && cost[u][v] < d[v]) {
                p[v] = u;
                d[v] = cost[u][v];
            }
        }
    }
    int total_cost = printMST(n, p, cost);
    printf("Total cost of Minimum Spanning Tree (MST): %d\n", total_cost);
}
int main() {
    int n;

    int cost[MAX_VERTICES][MAX_VERTICES];

    printf("Enter number of vertices (max %d): ", MAX_VERTICES);

    scanf("%d", &n);

    printf("Enter the cost adjacency matrix (use 'inf' for infinity):\n");

    parseCost(n, cost);

    printf("Minimum Spanning Tree (MST) using Prim's algorithm:\n");

    primMST(n, cost);

    return 0;

}
```

Output:



```
Enter number of vertices (max 100): 5
Enter the cost adjacency matrix (use 'inf' for infinity):
0 5 15 20 inf
5 0 25 inf inf
15 25 0 30 37
20 inf 30 0 35
inf inf 37 35 0
Minimum Spanning Tree (MST) using Prim's algorithm:
Edge   Weight
0 - 1    5
0 - 2    15
0 - 3    20
3 - 4    35
Total cost of Minimum Spanning Tree (MST): 75

Process returned 0 (0x0)   execution time : 6.830 s
Press any key to continue.
```



```
Output:-
Enter the number of vertices : 6
Enter the cost of adjacency matrix
(use inf for infinity );
 0 60 10 inf inf inf
 60 0 inf 20 40 70
 10 inf 0 inf inf 50
 inf 20 inf 0 inf 50.
 inf 40 inf inf 0 30
 inf 70 50 80 30 0
Minimum Spanning Tree (MST)
using Prim's algorithm:-
Edge        weight
 4-1          40
 0-2          10
 1-3          20
 5-4          30
 2-5          50.
Total cost of Minimum Spanning Tree:15
```

0
60
1
10
70
3
20
40
80
2
5
4
50
30
MST
0
10
2
50
5
30
1
4
3
20
40

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.

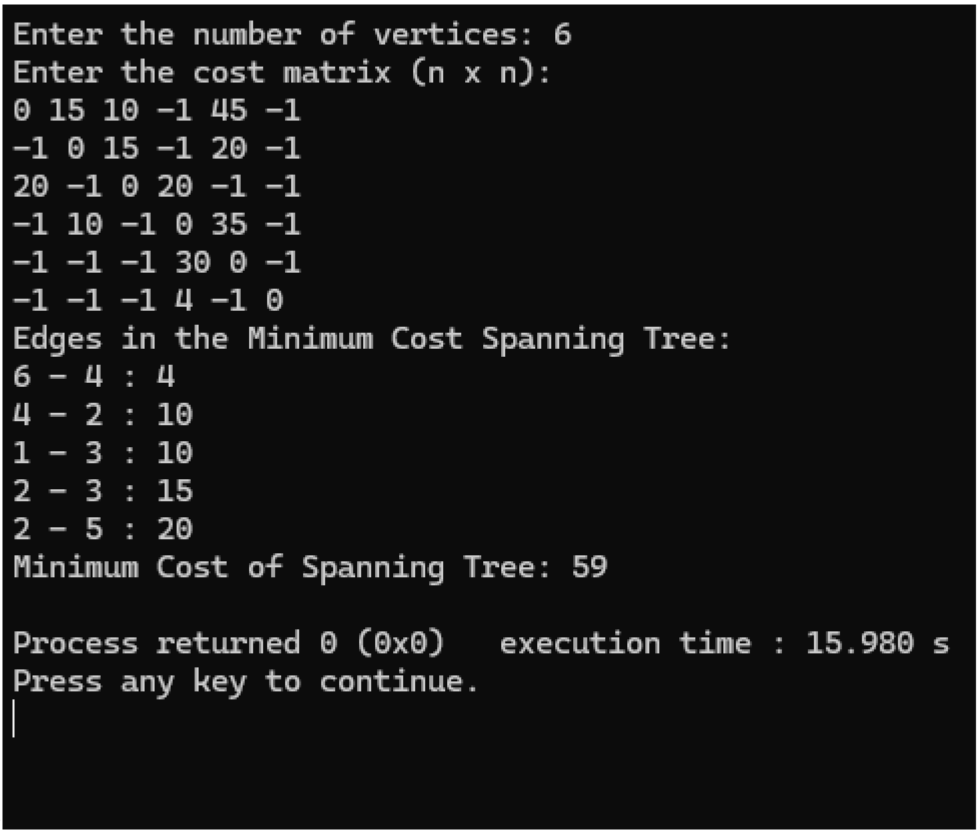
Code:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 100

#define INF 9999

struct Edge {

    int u, v, weight;

};
```

```
int compare(const void *a, const void *b) {
    struct Edge *a1 = (struct Edge *)a;
    struct Edge *b1 = (struct Edge *)b;
    return a1->weight - b1->weight;
}
int find(int parent[], int i) {
    if (parent[i] == 0)
        return i;
    return find(parent, parent[i]);
}
void unionSets(int parent[], int u, int v) {
    parent[v] = u;
}
void kruskals(int cost_matrix[][MAX], int n) {
    struct Edge edges[MAX * MAX];
    int edge_count = 0;
    int parent[MAX] = {0};
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (cost_matrix[i][j] != INF) {
                edges[edge_count++] = (struct Edge){i, j, cost_matrix[i][j]};
            }
        }
    }
    qsort(edges, edge_count, sizeof(edges[0]), compare);

    int mincost = 0;

    int ne = 0;

    printf("Edges in the Minimum Cost Spanning Tree:\n");

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

        int u = find(parent, edges[i].u);

        int v = find(parent, edges[i].v);

        if (u != v) {

            printf("%d - %d : %d\n", edges[i].u, edges[i].v, edges[i].weight);

            unionSets(parent, u, v);

            mincost += edges[i].weight;

            ne++;

        }

        if (ne == n - 1)

            break;

    }

    printf("Minimum Cost of Spanning Tree: %d\n", mincost);

}

int main() {

    int n;

    printf("Enter the number of vertices: ");

    scanf("%d", &n);

    int cost_matrix[MAX][MAX];

    printf("Enter the cost matrix (n x n):\n");
```

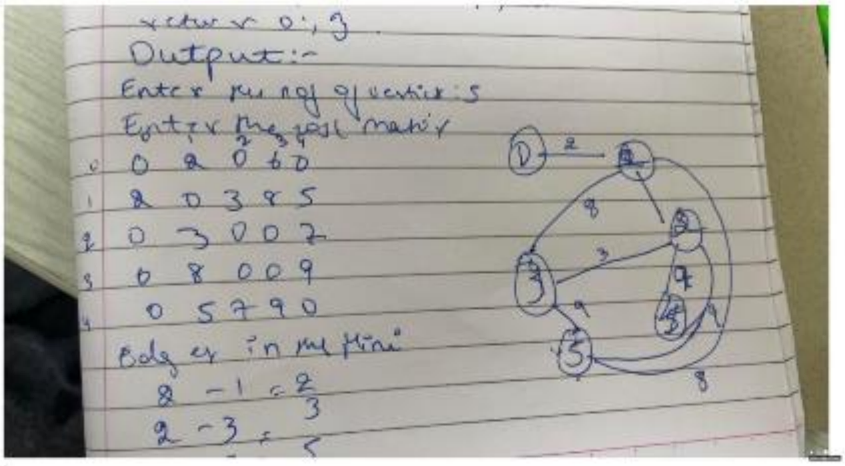
```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        scanf("%d", &cost_matrix[i][j]);
        if (cost_matrix[i][j] == 0 || cost_matrix[i][j] == -1)
            cost_matrix[i][j] = INF;
    }
}
kruskals(cost_matrix, n);
return 0;
```

Output:



```
Enter the number of vertices: 6
Enter the cost matrix (n x n):
0 15 10 -1 45 -1
-1 0 15 -1 20 -1
20 -1 0 20 -1 -1
-1 10 -1 0 35 -1
-1 -1 -1 30 0 -1
-1 -1 -1 4 -1 0
Edges in the Minimum Cost Spanning Tree:
6 - 4 : 4
4 - 2 : 10
1 - 3 : 10
2 - 3 : 15
2 - 5 : 20
Minimum Cost of Spanning Tree: 59

Process returned 0 (0x0)   execution time : 15.980 s
Press any key to continue.
```

}



return 0;
}
Output:-
Enter the no of vertices: 5
Enter the cost matrix

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

Edges in the Mini

```
2 - 1 = 2
          3
2 - 3 = 5
```

0
2
1
8
2
3
3
9
4
5
8

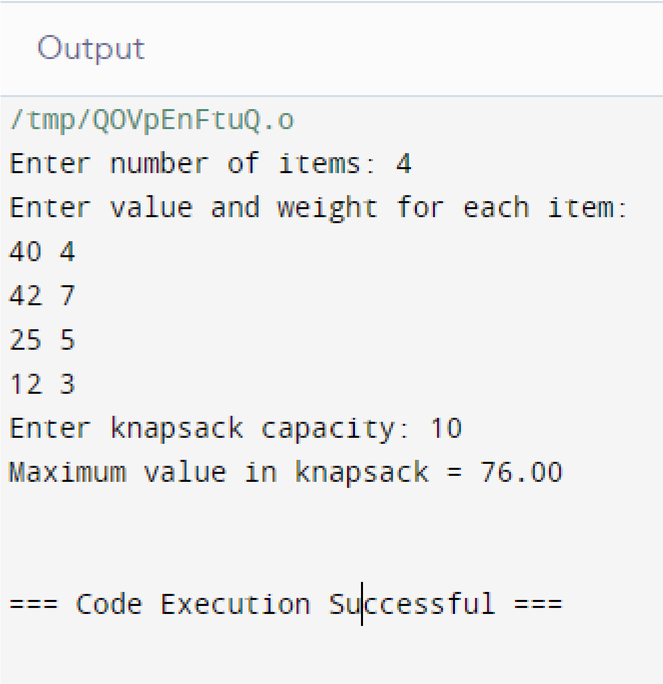
12. Implement Fractional Knapsack using Greedy technique

Code:

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

```
struct Item {
    int value;
    int weight;
    double ratio;
};
double fractionalKnapsack(int capacity, struct Item items[], int n) {
    for (int i = 0; i < n; i++) {
        items[i].ratio = (double)items[i].value / items[i].weight;
    }
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (items[j].ratio < items[j + 1].ratio) {
                struct Item temp = items[j];
                items[j] = items[j + 1];
                items[j + 1] = temp;
            }
        }
    }
    double totalValue = 0.0;
    int currentWeight = 0;
    for (int i = 0; i < n; i++) {
        if (currentWeight + items[i].weight <= capacity) {
            currentWeight += items[i].weight;
            totalValue += items[i].value;
        } else {
            int remainingCapacity = capacity - currentWeight;
            totalValue += items[i].ratio * remainingCapacity;
            break;
        }
    }
    return totalValue;
}
int main() {
    int n;
    printf("Enter number of items: ");
    scanf("%d", &n);
    struct Item items[n];
    printf("Enter value and weight for each item:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &items[i].value, &items[i].weight);
    }
    int capacity;
    printf("Enter knapsack capacity: ");
    scanf("%d", &capacity);
    double totalValue = fractionalKnapsack(capacity, items, n);
    printf("Maximum value in knapsack = %.2f\n", totalValue);
    return 0;
}
```

<u>Output:</u>



```
/tmp/QOVpEnFtuQ.o
Enter number of items: 4
Enter value and weight for each item:
40 4
42 7
25 5
12 3
Enter knapsack capacity: 10
Maximum value in knapsack = 76.00


=== Code Execution Successful ===
```

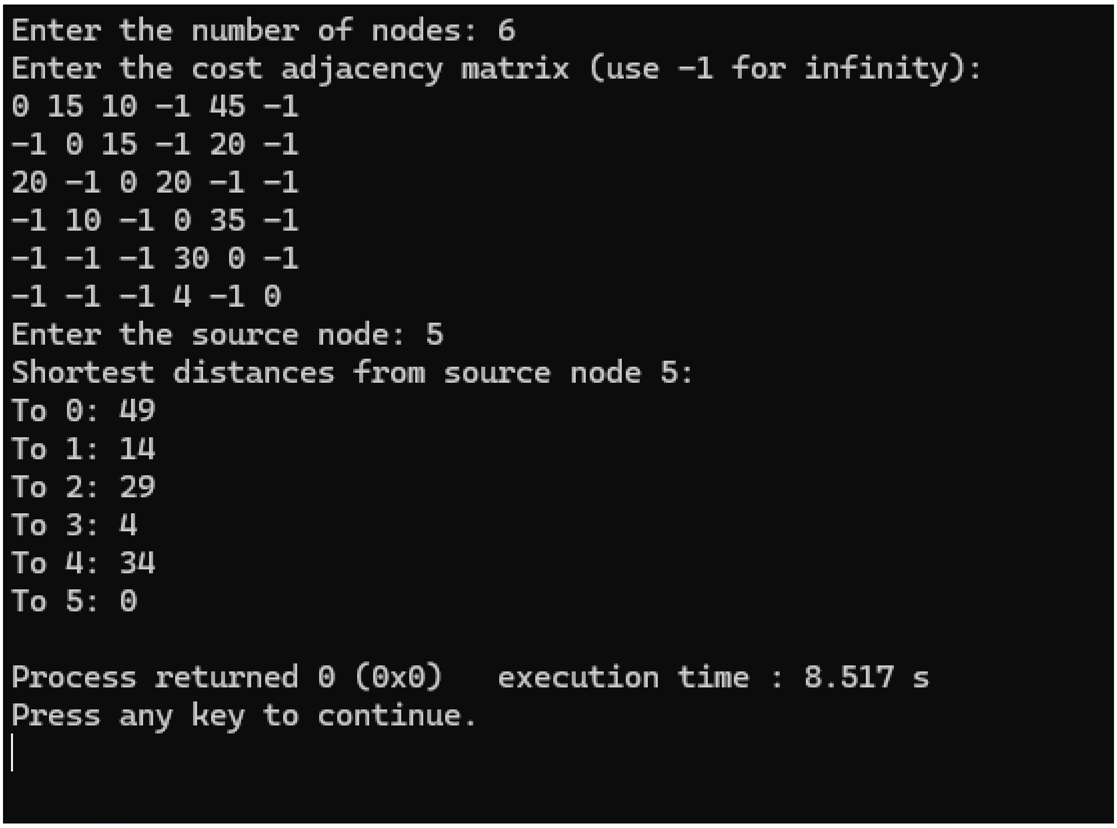
13. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

<u>Code:</u>

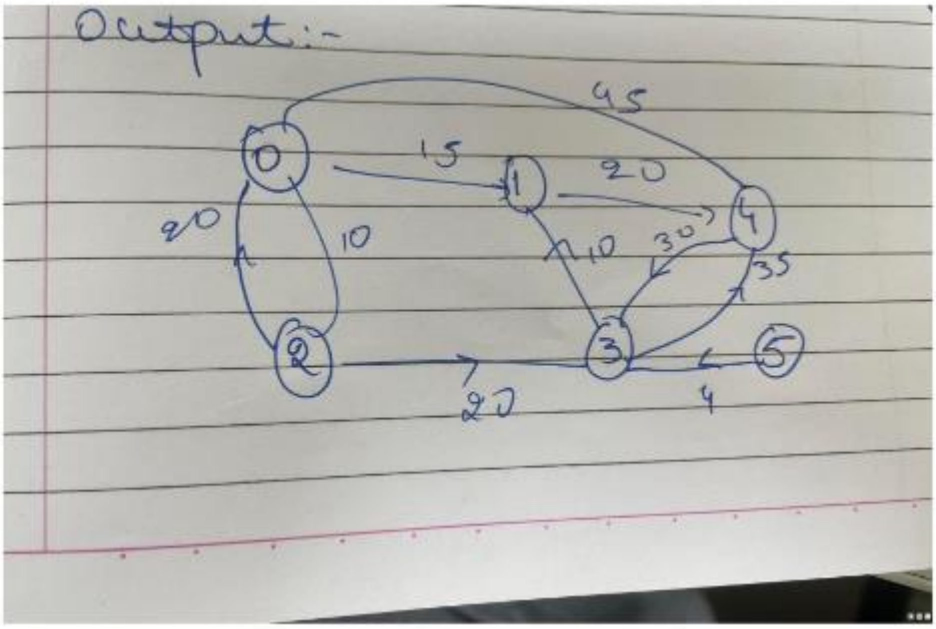
```
#include <stdio.h>

#include <stdlib.h>

#define MAX_NODES 100

#define INF 9999

void dijkstra(int n, int src, int cost[MAX_NODES][MAX_NODES]);

int main() {

    int n;

    int cost[MAX_NODES][MAX_NODES];

    int src;

    printf("Enter the number of nodes: ");

    scanf("%d", &n);

    printf("Enter the cost adjacency matrix (use -1 for infinity):\n");

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

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

            scanf("%d", &cost[i][j]);

            if (cost[i][j] == -1 && i != j) {

                cost[i][j] = INF;

            }

        }

    }

    printf("Enter the source node: ");

    scanf("%d", &src);

    dijkstra(n, src, cost);

    return 0;
```

```
}
void dijkstra(int n, int src, int cost[MAX_NODES][MAX_NODES]) {
    int dist[MAX_NODES];
    int vis[MAX_NODES];
    for (int j = 0; j < n; j++) {
        dist[j] = cost[src][j];
        vis[j] = 0;
    }
    dist[src] = 0;
    vis[src] = 1;
    int count = 1;
    while (count != n) {
        int min = INF;
        int u = -1;
        for (int j = 0; j < n; j++) {
            if (!vis[j] && dist[j] < min) {
                min = dist[j];
                u = j;
            }
        }
        if (u == -1) break;
        vis[u] = 1;
        count++;
        for (int j = 0; j < n; j++) {
            if (!vis[j] && cost[u][j] != INF && dist[u] + cost[u][j] < dist[j]) {
                dist[j] = dist[u] + cost[u][j];
            }
        }
    }
    printf("Shortest distances from source node %d:\n", src);
    for (int j = 0; j < n; j++) {
        if (dist[j] == INF) {
            printf("To %d: Infinity\n", j);
        } else {
            printf("To %d: %d\n", j, dist[j]);
        }
    }
}
```

Output:



```
Enter the number of nodes: 6
Enter the cost adjacency matrix (use -1 for infinity):
0 15 10 -1 45 -1
-1 0 15 -1 20 -1
20 -1 0 20 -1 -1
-1 10 -1 0 35 -1
-1 -1 -1 30 0 -1
-1 -1 -1 4 -1 0
Enter the source node: 5
Shortest distances from source node 5:
To 0: 49
To 1: 14
To 2: 29
To 3: 4
To 4: 34
To 5: 0

Process returned 0 (0x0)   execution time : 8.517 s
Press any key to continue.
```

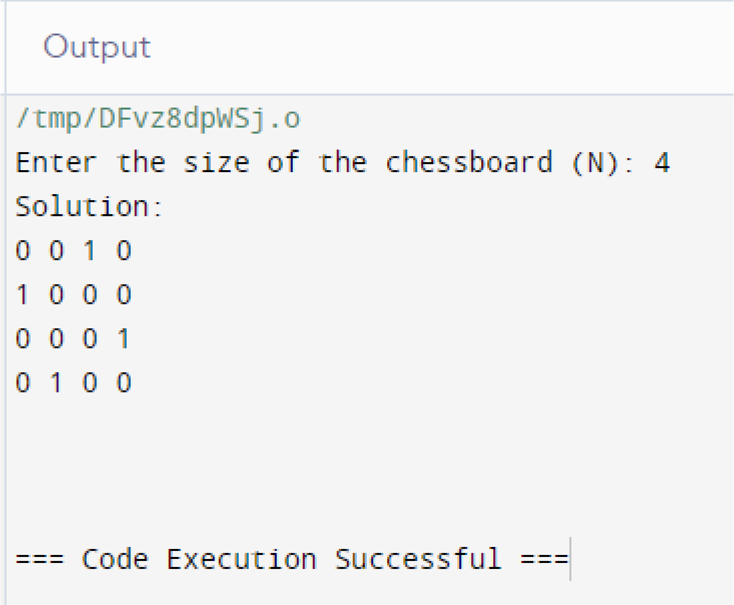


14. Implement "N-Queens Problem" using Backtracking

Code:

```
#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define N_MAX 20

int N;

int board[N_MAX][N_MAX];

void initializeBoard();

void printSolution();

bool isSafe(int row, int col);

bool solveNQueens(int col);

void initializeBoard() {

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

        for (int j = 0; j < N; j++) {

            board[i][j] = 0;

        }

    }

}

void printSolution() {

    printf("Solution:\n");

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

        for (int j = 0; j < N; j++) {

            printf("%d ", board[i][j]);

        }
        printf("\n");
    }
    printf("\n");
}
bool isSafe(int row, int col) {
    for (int i = 0; i < col; i++) {
        if (board[row][i]) {
            return false;
        }
    }
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) {
            return false;
        }
    }
    for (int i = row, j = col; i < N && j >= 0; i++, j--) {
        if (board[i][j]) {
            return false;
        }
    }
    return true;
}
bool solveNQueens(int col) {
    if (col >= N) {
        return true;
    }
    for (int i = 0; i < N; i++) {
        if (isSafe(i, col)) {
            board[i][col] = 1;
            if (solveNQueens(col + 1)) {
                return true;
            }
            board[i][col] = 0;
        }
    }
    return false;
}
int main() {
    printf("Enter the size of the chessboard (N): ");
    scanf("%d", &N);
    if (N <= 0 || N > N_MAX) {
        printf("Invalid input for N. Please enter a value between 1 and %d.\n", N_MAX);
        return 1;
    }
    initializeBoard();
    if (solveNQueens(0)) {
        printSolution();
    } else {
        printf("Solution does not exist for N = %d.\n", N);
    }
    return 0;
}
```

Output:



```
/tmp/DFvz8dpWSj.o
Enter the size of the chessboard (N): 4
Solution:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0



=== Code Execution Successful ===
```