**1. LEETCODE –** Find the Kth smallest element in a BST

**Code:**

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     struct TreeNode \*left;

 \*     struct TreeNode \*right;

 \* };

 \*/

int kthSmallest(struct TreeNode\* root, int k) {

    // Stack for iterative traversal

    struct TreeNode\* stack[100];

    int top = -1;

    // Current node

    struct TreeNode\* curr = root;

    // Variable to keep track of visited nodes

    int count = 0;

    // Traverse the tree until the current node is NULL

    while (curr != NULL || top != -1) {

        // Move to the leftmost node

        while (curr != NULL) {

            stack[++top] = curr;

            curr = curr->left;

        }

        // Pop the top node from the stack

        curr = stack[top--];

        // Increment the count

        count++;

        // If count equals k, return the value of the current node

        if (count == k) {

            return curr->val;

        }

        // Move to the right of the current node

        curr = curr->right;

    }

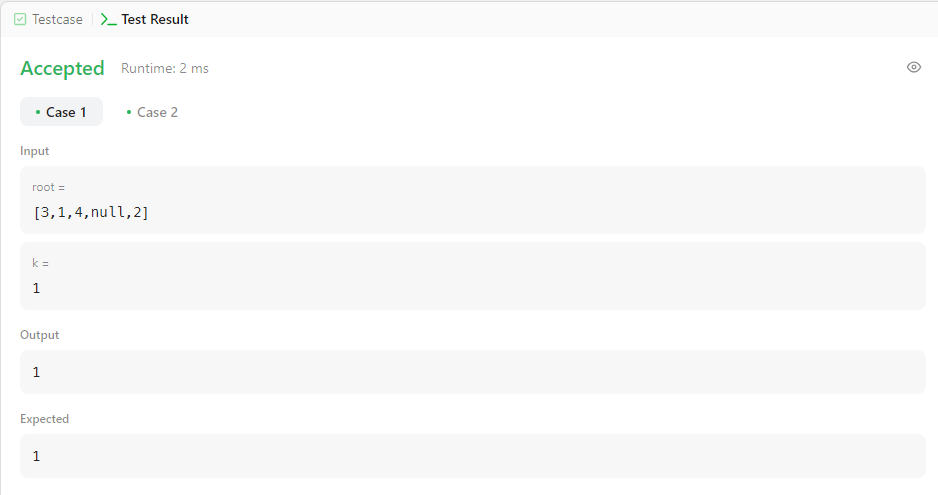
    // If k is greater than the number of nodes in the tree

    return -1; // or any other appropriate error code

}

**Output:**

****

****

**2. LEETCODE –** Minimum Absolute Difference in BST

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

// Definition for a binary tree node.

struct TreeNode {

int val;

struct TreeNode \*left;

struct TreeNode \*right;

};

// Helper function to perform in-order traversal and store values in an array

void inorderTraversal(struct TreeNode\* root, int\* arr, int\* size) {

if (root == NULL) {

return;

}

inorderTraversal(root->left, arr, size);

arr[(\*size)++] = root->val;

inorderTraversal(root->right, arr, size);

}

// Function to find the minimum absolute difference

int getMinimumDifference(struct TreeNode\* root) {

if (root == NULL) {

return 0;

}

// Array to store the values in sorted order

int arr[10000];

int size = 0;

// Perform in-order traversal to fill the array

inorderTraversal(root, arr, &size);

// Calculate the minimum difference

int minDiff = INT\_MAX;

for (int i = 1; i < size; i++) {

int diff = arr[i] - arr[i - 1];

if (diff < minDiff) {

minDiff = diff;

}

}

return minDiff;

}

// Function to create a new tree node

struct TreeNode\* newNode(int val) {

struct TreeNode\* node = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

node->val = val;

node->left = NULL;

node->right = NULL;

return node;

}

// Example usage

int main() {

struct TreeNode\* root = newNode(4);

root->left = newNode(2);

root->right = newNode(6);

root->left->left = newNode(1);

root->left->right = newNode(3);

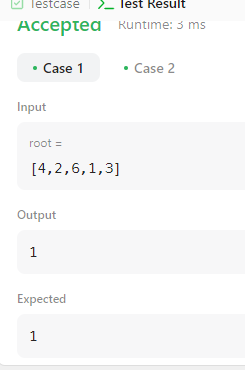
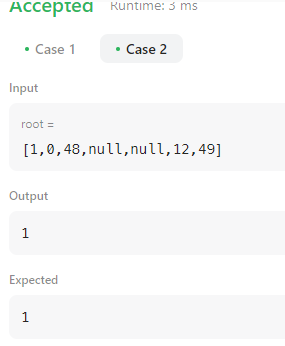
int result = getMinimumDifference(root);

printf("Minimum absolute difference: %d\n", result);

return 0;

}

**Output:**

**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]);

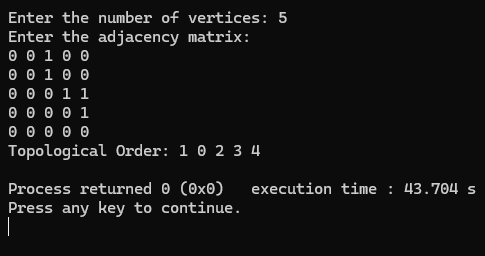
}

free(a);

return 0;

}

**Output:**

****

**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]);

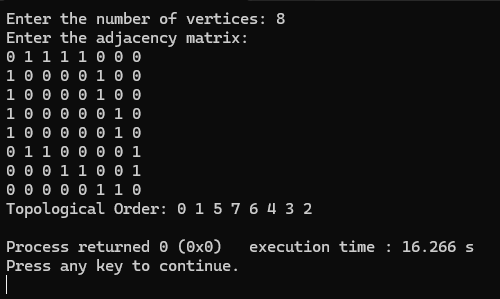
}

free(a);

return 0;

}

**Output:**

****

[**Find the Kth Largest Integer in the Array**](https://leetcode.com/problems/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);

}

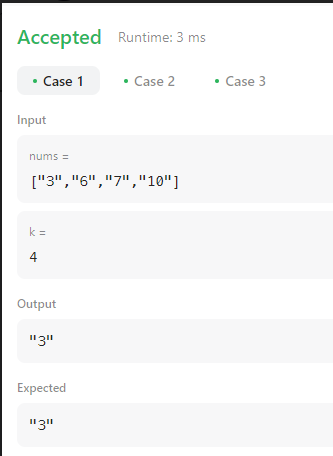
char\* kthLargestNumber(char \*\*nums, int numsSize, int k) {

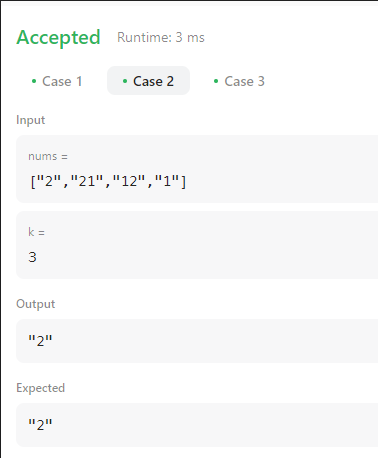
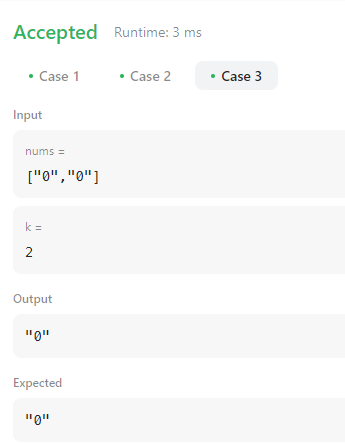
    qsort(nums, numsSize, sizeof(char\*), compare);

    return nums[k - 1];

}

**Output:**



**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);

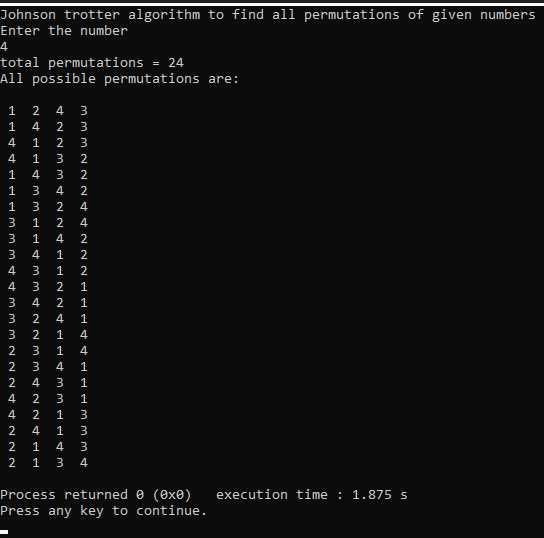
printf("\n");

}

return 0;

}

**Output:**

****

**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++)

 {

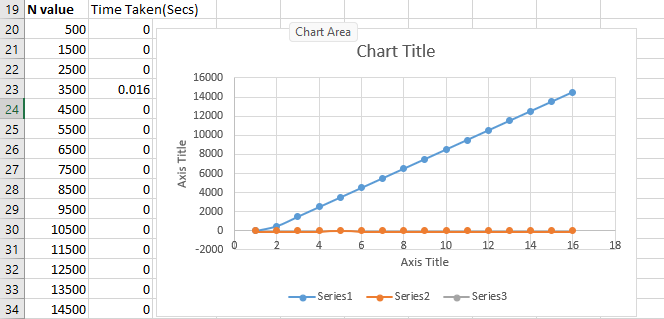
  a[i]=c[i];

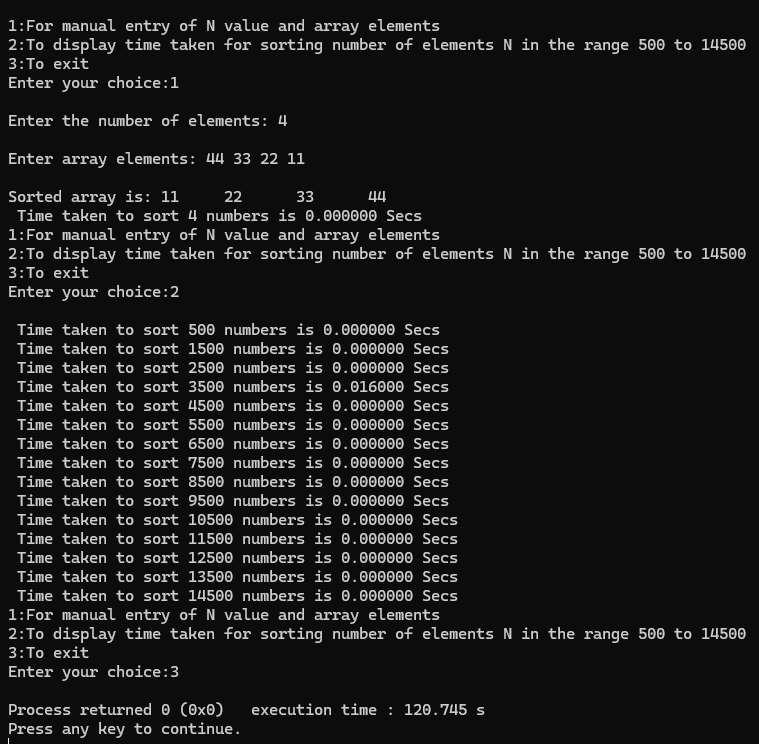
 }

}

**Output:**

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





**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]);

}

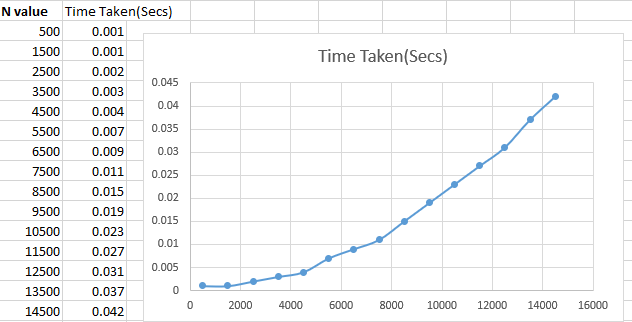
quickSort(arr, 0, n - 1);

printf("Sorted array: \n");

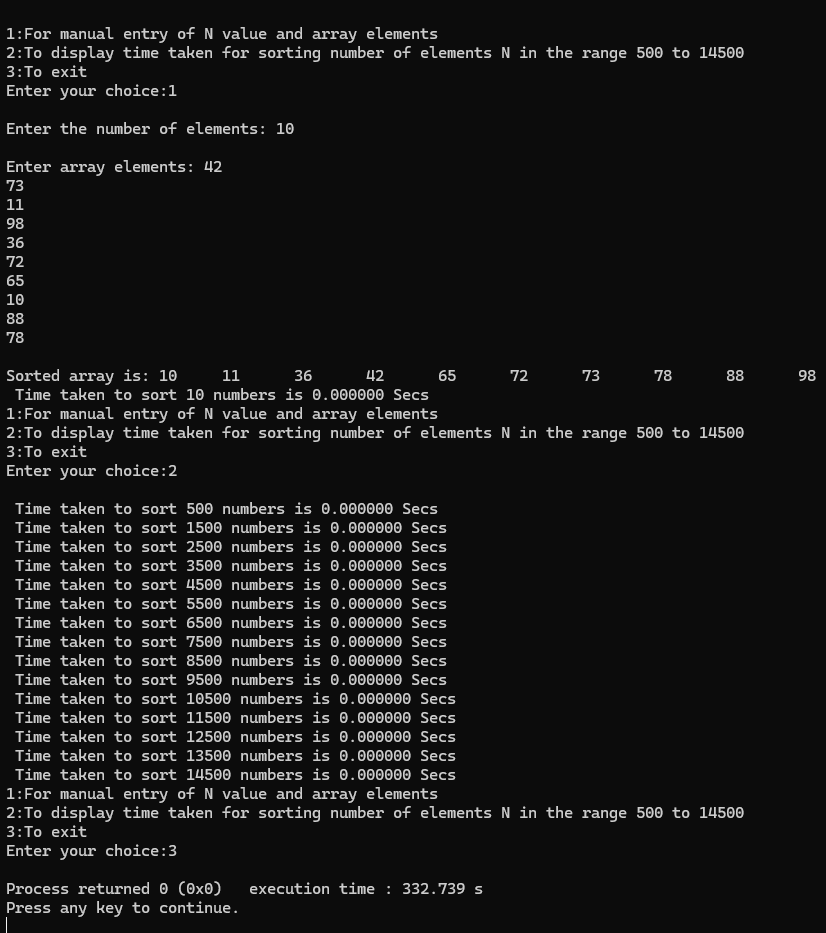
printArray(arr, n);

return 0;

}

****

**Output:**

****

**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);

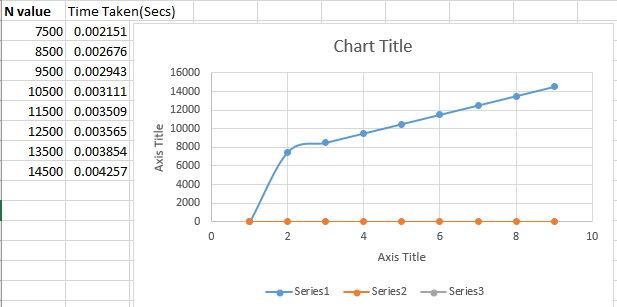
}

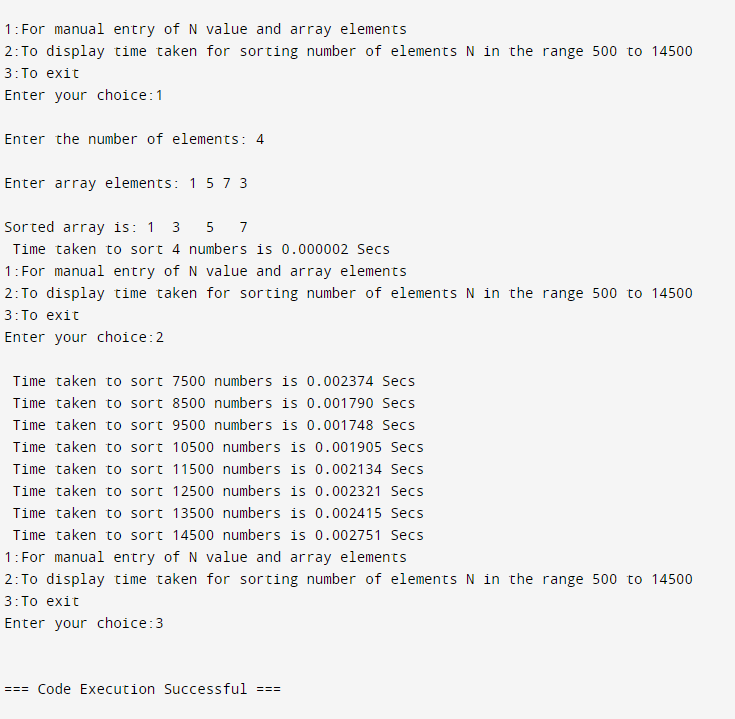
getchar();

}

}

**Output:**

****

****

**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;

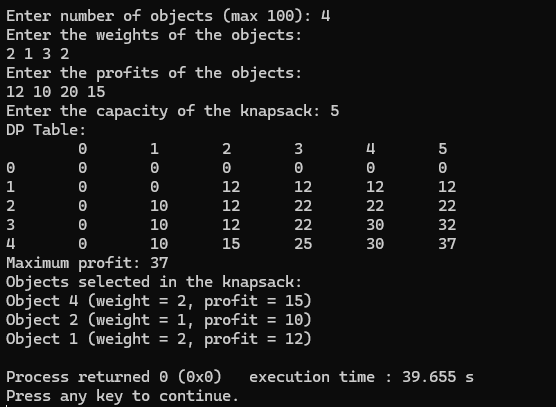
}

knapsack(n, W, weights, profits);

return 0;

}

**Output:**



**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;

}

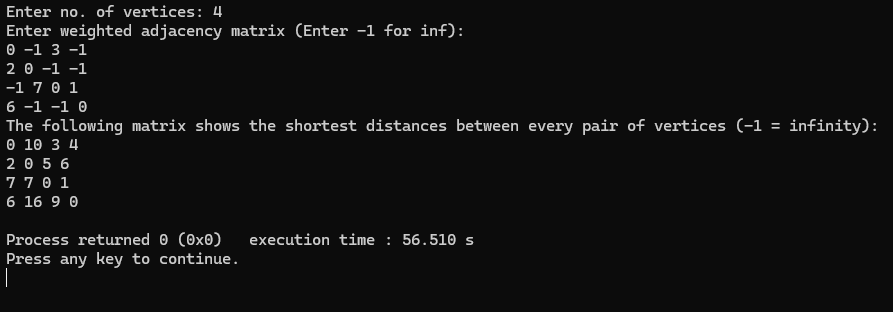
}

floydWarshall(v, graph);

return 0;

}

**Output:**



**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);

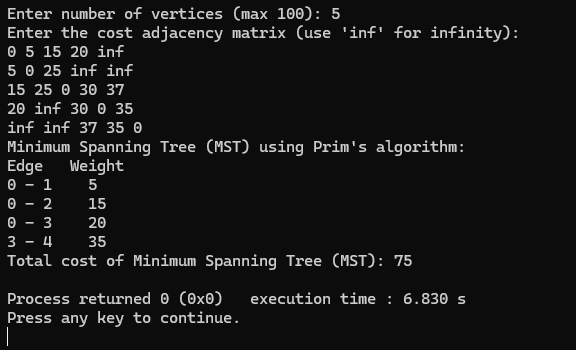
printf("Minimum Spanning Tree (MST) using Prim's algorithm:\n");

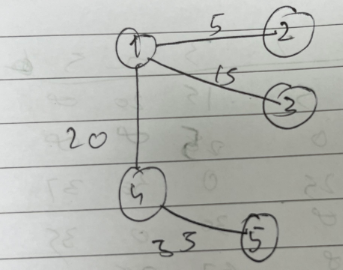
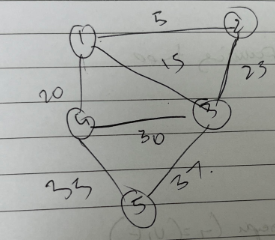
primMST(n, cost);

return 0;

}

**Output:**





**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;

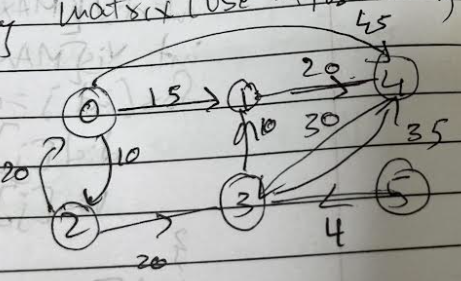
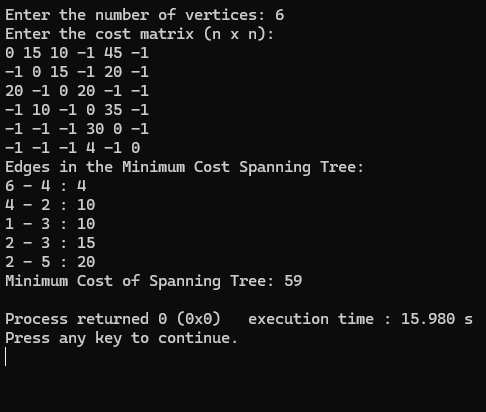
}

}

kruskals(cost\_matrix, n);

return 0;

**Output:**

}

**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);

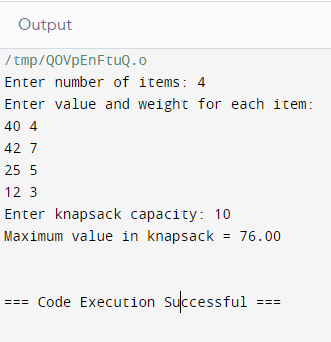
double totalValue = fractionalKnapsack(capacity, items, n);

printf("Maximum value in knapsack = %.2f\n", totalValue);

return 0;

}

**Output:**

****

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

**Code:**

#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 {

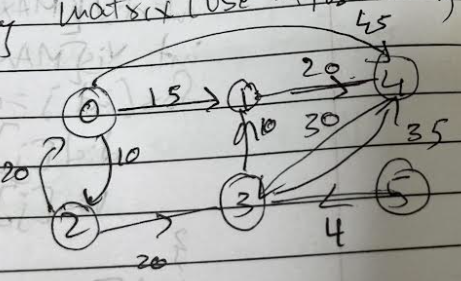
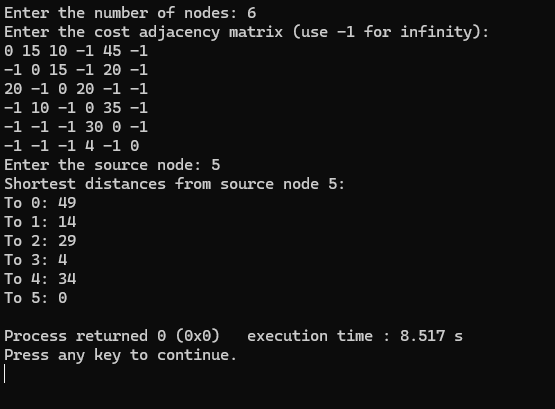
printf("To %d: %d\n", j, dist[j]);

}

}

}

**Output:**



**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:**

