

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Analysis and Design of Algorithms

Submitted by

Priyanshu Kumar (1BM22CS210)

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 **Priyanshu Kumar (1BM22CS210)**, who is a 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.

M Lakshmi Neelima

Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak

Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Lab Program No.	Program Details	
1	Leetcode Repeated Substring Problem.	
2	Leetcode Kth Largest Sum in a Binary Tree.	
3	Leetcode Increasing order search tree.	
4	Write a program to obtain the Topological ordering of vertices in a given digraph.	
5	Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Sort a given set of N integer elements using Merge Sort technique and compute its time taken.	
6	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	
7	Implement Johnson Trotter algorithm to generate permutations. Brute force String matching. Leetcode Find the Kth largest.	
8	Implement All Pair Shortest paths problem using Floyd's algorithm. Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	
9	Implement Knapsack using Dynamic Programming. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	

	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm	
--	---	--

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

LAB 1

Given a string *s*, check if it can be constructed by taking a substring of it and appending multiple copies of the substring together.

```
bool repeatedSubstringPattern(char *s) {  
    int len = strlen(s);  
    for (int i = 1; i <= len / 2; i++) {  
        bool flag = true;  
        for (int j = i; j < len; j += i) {  
            if (strncmp(s, s + j, i) != 0) {  
                flag = false;  
                break;  
            }  
        }  
        if (flag) return true;  
    }  
    return false;  
}
```

Submission Detail

129 / 129 test cases passed.

Status: **Accepted**

Runtime: **235 ms**

Memory Usage: **12.2 MB**

Submitted: **2 months, 2 weeks ago**

LAB 2

You are given the root of a binary tree and a positive integer k. The level sum in the tree is the sum of the values of the nodes that are on the same level. Return the kth largest level sum in the tree (not necessarily distinct). If there are fewer than k levels in the tree, return -1.

```
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

```
int treeHeight(struct TreeNode *root) {  
    if (root == NULL) return 0;  
    else {  
        int leftHeight = treeHeight(root->left);  
        int rightHeight = treeHeight(root->right);  
        return MAX(leftHeight, rightHeight) + 1;  
    }  
}
```

```
long long levelSum(struct TreeNode *root, int level) {  
    if (root == NULL) return 0;  
    if (level == 0) return root->val;  
    return levelSum(root->left, level - 1) + levelSum(root->right, level - 1);  
}
```

```
long long kthLargestLevelSum(struct TreeNode *root, int k) {  
    int height = treeHeight(root);  
    if (k > height) {  
        printf("Error\n");  
    }  
}
```

```

        return -1;
    }

    long long sumArray[height]; // Assuming the height index array is correct
    for (int i = 0; i < height; i++) {
        sumArray[i] = levelSum(root, i);
    }

    // Bubble sort to find kth largest sum
    for (int i = 0; i < height - 1; i++) {
        for (int j = 0; j < height - i - 1; j++) {
            if (sumArray[j] < sumArray[j + 1]) {
                long long temp = sumArray[j];
                sumArray[j] = sumArray[j + 1];
                sumArray[j + 1] = temp;
            }
        }
    }

    return sumArray[k - 1];
}

```

Submission Detail

129 / 129 test cases passed.

Status: **Accepted**

Runtime: **235 ms**

Memory Usage: **12.2 MB**

Submitted: **2 months, 2 weeks ago**

LAB 3

Given the root of a binary search tree, rearrange the tree in in-order so that the leftmost node in the tree is now the root of the tree, and every node has no left child and only one right child.

```
struct TreeNode* increasingBST(struct TreeNode *root) {  
    if (root == NULL) return NULL;  
    struct TreeNode *head = NULL;  
    struct TreeNode *prev = NULL;  
    inOrder(root, &head, &prev);  
    return head;  
}
```

```
void inOrder(struct TreeNode *node, struct TreeNode **head, struct TreeNode **prev) {  
    if (node == NULL) return;  
  
    inOrder(node->left, head, prev);  
  
    if (*prev == NULL) {  
        *head = node;  
    } else {  
        (*prev)->right = node;  
    }  
    node->left = NULL;  
    *prev = node;  
}
```



```
inOrder(node->right, head, prev);
if (node->left) {
    inOrder(node->left, head, prev);
}
if (*prev == NULL) {
    *head = node; // node becomes the new root if prev is still NULL
} else {
    (*prev)->right = node; // right child of prev set to current node
}
node->left = NULL; // the left child of the current node is always NULL
*prev = node; // move prev to current node

if (node->right) {
    inOrder(node->right, head, prev);
}
}
```

LAB 4

Write a program to obtain the Topological ordering of vertices in a given digraph using Source Removal method.

```
#include <stdio.h>

#define MAX_VERTICES 100

int indegree[MAX_VERTICES];
int graph[MAX_VERTICES][MAX_VERTICES];

void topologicalSort(int V) {
    int queue[MAX_VERTICES], front = 0, rear = -1;

    for (int i = 0; i < V; i++)
        if (indegree[i] == 0)
            queue[++rear] = i;

    while (front <= rear) {
        int vertex = queue[front++];

        printf("%d ", vertex);

        for (int i = 0; i < V; i++)
            if (graph[vertex][i] == 1) {
                indegree[i]--;
                if (indegree[i] == 0)
```

```

        queue[++rear] = i;
    }
}
}

int main() {
    int V, E;

    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);
    for (int i = 0; i < V; i++) {
        indegree[i] = 0;
        for (int j = 0; j < V; j++)
            indegree[i] += graph[j][i];
    }
    printf("Topological Sort: ");
    topologicalSort(V);
    return 0;
}

```

```

Enter the number of vertices: 4
Enter the adjacency matrix:
0 1 1 0
0 0 1 1
0 0 0 1
0 0 0 0
Topological Sort: 0 1 2 3

```

LAB 5

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

```
#include <stdio.h>

#include <time.h>

#include <stdlib.h> /* To recognize 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("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) /
CLOCKS_PER_SEC));
    break;

```

case 2:

```

n = 500;
while (n <= 14500) {
    for (i = 0; i < n; i++) {
        // a[i] = random(1000);
        a[i] = n - i;
    }
    start = clock();
    selsort(n, a);
    // Dummy loop to create delay
    for (j = 0; j < 500000; j++) {
        temp = 38 / 600;
    }
    end = clock();

```

```

        printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) /
CLOCKS_PER_SEC));

        n = n + 1000;

    }

    break;

case 3:

    exit(0);

}

getchar();

}

}

```

```

void selsort(int n, int a[]) {
    int i, j, t, small, pos;
    for (i = 0; i < n - 1; i++) {
        pos = i;
        small = a[i];
        for (j = i + 1; j < n; j++) {
            if (a[j] < small) {
                small = a[j];
                pos = j;
            }
        }
        t = a[i];
        a[i] = a[pos];
        a[pos] = t;
    }
}

```

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

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

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

```
#include <stdlib.h> /* To recognize 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("\nTime 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("\nTime 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];
    }
}

```

```

Enter the number of elements: 6
Enter 6 integers:
3 4 1 6 3 2
Unsorted array: 3 4 1 6 3 2
Sorted array: 1 2 3 3 4 6
Time taken to sort: 0.000000 seconds

```

LAB 6

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

```
#include <stdio.h>

#include <time.h>

#include <stdlib.h> /* To recognize exit function when compiling with gcc */

void swap(int* a, int* b);

int partition(int arr[], int low, int high);

void quicksort(int arr[], int low, int high);

void main() {
    int i, j, ch, temp;
    clock_t start, end;
    int a[110000], n;

    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 7500 to 25000");

        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("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) /
CLOCKS_PER_SEC));

break;

```

case 2:

```

n = 7500;
while (n <= 25500) {
    for (i = 0; i < n; i++) {
        // a[i] = random(1000);
        a[i] = n - i;
    }

    start = clock();
    quicksort(a, 0, n - 1);
    // Dummy loop to create delay
    for (j = 0; j < 500000; j++) {
        temp = 38 / 600;
    }
}

```

```

        end = clock();

        printf("\nTime 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 swap(int* p1, int* p2) {
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

```

```

int partition(int arr[], int low, int high) {
    // Choose the pivot
    int pivot = arr[high];

    // Index of smaller element and indicate
    // the right position of pivot found so far
    int i = (low - 1);

```

```

for (int j = low; j <= high; j++) {
    // If current element is smaller than the pivot
    if (arr[j] < pivot) {
        // Increment index of smaller element
        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 pivot = partition(arr, low, high);
        quicksort(arr, low, pivot - 1);
        quicksort(arr, pivot + 1, high);
    }
}

```

```

Enter the number of elements: 10
Enter 10 integers:
9 7 5 2 4 6 8 1 10 0
Unsorted array: 9 7 5 2 4 6 8 1 10 0
Sorted array: 0 1 2 4 5 6 7 8 9 10
Time taken to sort: 0.000000 seconds

```

LAB 7

Implement Johnson Trotter algorithm to generate permutations.

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

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

```
int flag = 0;
```

```
void 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 (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;
    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);

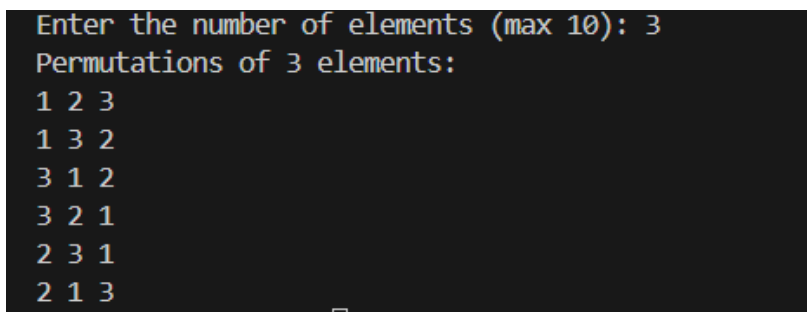
    printf("\n");

}

return 0;

}

```



```

Enter the number of elements (max 10): 3
Permutations of 3 elements:
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3

```

Brute force String matching

```

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

```

```

int main(){

    int m, n;

    printf("Enter m and n: ");

    scanf("%d%d", &m, &n);

    char s1[m], s2[n];

    fflush(stdin);

    printf("Enter string 1: ");

    scanf("%s", s1);

    printf("Enter string 2: ");

    scanf("%s", s2);

    // printf("%s %s", s1, s2);

    int i = 0, j = 0;

    while (i < n - m + 1){

        if (s1[0] == s2[i]){

            bool found = true;

            while (j < m){

                if (s1[j] != s2[i+j]) found = false;

                j++;

            }

            if (found) printf("Found at %d", i);

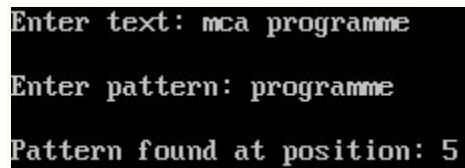
        }

        i++;

    }
}

```

```
    return 0;
}
```

A terminal window with a black background and white text. It shows three lines of input and output: 'Enter text: mca programme', 'Enter pattern: programme', and 'Pattern found at position: 5'.

```
Enter text: mca programme
Enter pattern: programme
Pattern found at position: 5
```

Leetcode Find the Kth largest

```
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];
}
```

LAB 8

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");
```

```
    }
```

```
}
```

```
// Function to implement Floyd Warshall algorithm
```

```
void floydWarshall(int v, int graph[v][v]) {
```

```
    int dist[v][v], i, j, k;
```

```
    // Initialize the solution matrix same as input graph matrix
```

```
    for (i = 0; i < v; i++)
```

```
        for (j = 0; j < v; j++)
```

```

        dist[i][j] = graph[i][j];

// Add all vertices one by one to the set of intermediate vertices.
for (k = 0; k < v; k++) {
    // Pick all vertices as source one by one
    for (i = 0; i < v; i++) {
        // Pick all vertices as destination for the above picked source
        for (j = 0; j < v; j++) {
            // If vertex k is on the shortest path from i to j, then update the value of dist[i][j]
            if (dist[i][k] + dist[k][j] < dist[i][j])
                dist[i][j] = dist[i][k] + dist[k][j];
        }
    }
}

// Print the shortest distance matrix
printSolution(v, dist);
}

```

```

int main() {
    int v;
    printf("Enter no. of vertices: ");
    scanf("%d", &v);
    int graph[v][v]; /* = { {0, 5, INF, 10}, {INF, 0, 3, INF}, {INF, INF, 0, 1}, {INF, INF, INF, 0} }; */
    printf("Enter weighted adjacency matrix (Enter -1 for inf): \n");
    for(int i = 0; i < v; i++){
        for(int j = 0; j < v; j++){

```



```

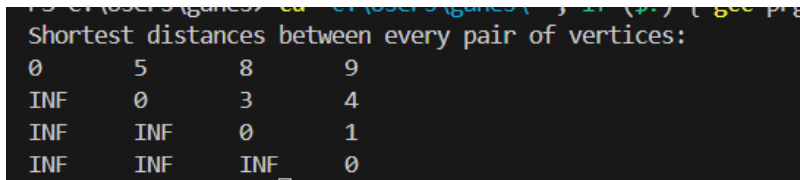
        scanf("%d", &graph[i][j]);

        if (graph[i][j] == -1) graph[i][j] = INF;
    }
}

// Print the solution
floydWarshall(v, graph);

return 0;
}

```



```

Shortest distances between every pair of vertices:
0      5      8      9
INF     0      3      4
INF    INF     0      1
INF    INF    INF     0

```

Sort 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> /* To recognise exit function when compiling with gcc*/

void swap(int* a, int* b)
{

    int temp = *a;

    *a = *b;

    *b = temp;

}

// To heapify a subtree rooted with node i
// which is an index in arr[].

```

```

// n is size of heap
void heapify(int arr[], int N, int i)
{
    // Find largest among root,
    // left child and right child

    // Initialize largest as root
    int largest = i;

    // left = 2*i + 1
    int left = 2 * i + 1;

    // right = 2*i + 2
    int right = 2 * i + 2;

    // If left child is larger than root
    if (left < N && arr[left] > arr[largest])

        largest = left;

    // If right child is larger than largest
    // so far
    if (right < N && arr[right] > arr[largest])

        largest = right;

    // Swap and continue heapifying
    // if root is not largest
    // If largest is not root

```

```

if (largest != i) {

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

    // Recursively heapify the affected
    // sub-tree
    heapify(arr, N, largest);
}
}

```

```

// Main function to do heap sort
void heapSort(int arr[], int N)
{

```

```

    // Build max heap
    for (int i = N / 2 - 1; i >= 0; i--)

```

```

        heapify(arr, N, i);

```

```

// Heap sort
for (int i = N - 1; i >= 0; i--) {

```

```

    swap(&arr[0], &arr[i]);

```

```

    // Heapify root element
    // to get highest element at
    // root again

```

```

    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<=15500) {
        for(i=0;i<n;i++){
            //a[i]=random(1000);
            a[i]=n-i;
        }
        start=clock();
        heapSort(a,n);
        //Dummy loop to create delay
        for(j=0;j<500000;j++){
            temp=38/600;
        }
        end=clock();
        printf("\n Time taken to sort %d numbers is %f Secs",n,
        (((double)(end-start))/CLOCKS_PER_SEC));
        n=n+1000;
    }
    break;
case 3:
    exit(0);
}
getchar();
}
}

```

```

Enter the number of elements: 7
Enter 7 integers:
4 2 3 5 1 6 7
Unsorted array: 4 2 3 5 1 6 7
Sorted array: 1 2 3 4 5 6 7
Time taken to sort: 0.000000 seconds

```

LAB 9

Implement Knapsack using Dynamic Programming.

```
#include <stdio.h>

// A utility function that returns the maximum of two integers
int max(int a, int b) { return (a > b) ? a : b; }

// Function to solve the Knapsack problem using Dynamic Programming
void knapsack(int W, int n, int weights[], int profits[]) {
    int K[n + 1][W + 1];

    // Build table K[][] in bottom up manner
    for (int i = 0; i <= n; i++) {
        for (int 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];
        }
    }

    // Printing the table
    printf("DP Table:\n");
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            printf("%d\t", K[i][w]);
        }
    }
}
```

```

    }

    printf("\n");
}

// Storing the result
int res = K[n][W];
printf("\nMaximum profit: %d\n", res);

// Backtracking to find the items included in the knapsack
int w = W;
printf("Selected items:\n");
for (int i = n; i > 0 && res > 0; i--) {
    if (res == K[i - 1][w])
        continue;
    else {
        // This item is included
        printf("Item %d (Weight: %d, Profit: %d)\n", i, weights[i - 1], profits[i - 1]);

        // Since this weight is included, its profit is deducted
        res -= profits[i - 1];
        w -= weights[i - 1];
    }
}

// Driver program to test the knapsack function
int main() {
    int weights[] = {2, 3, 4, 5};
    int profits[] = {3, 4, 5, 6};

```

```

int W = 5;

int n = sizeof(profits) / sizeof(profits[0]);

knapsack(W, n, weights, profits);

return 0;
}

```

```

Enter number of items: 3
Enter values and weights of items:
Enter value and weight for item 1: 60 10
Enter value and weight for item 2: 100 20
Enter value and weight for item 3: 120 30
Enter maximum weight capacity of knapsack: 50
Maximum value that can be obtained: 220

```

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

```

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 5 // Number of vertices in the graph

// Function to find the vertex with the minimum key value, from the set of vertices not yet included in
MST

int minKey(int key[], bool mstSet[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

```



```
}
```

```
// Function to print the constructed MST stored in parent[]
```

```
void printMST(int parent[], int graph[V][V]) {
```

```
    printf("Edge \tWeight\n");
```

```
    for (int i = 1; i < V; i++)
```

```
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
```

```
}
```

```
// Function to construct and print MST for a graph represented using adjacency matrix representation
```

```
void primMST(int graph[V][V]) {
```

```
    int parent[V]; // Array to store constructed MST
```

```
    int key[V]; // Key values used to pick minimum weight edge in cut
```

```
    bool mstSet[V]; // To represent set of vertices included in MST
```

```
    // Initialize all keys as INFINITE
```

```
    for (int i = 0; i < V; i++)
```

```
        key[i] = INT_MAX, mstSet[i] = false;
```

```
    // Always include first 1st vertex in MST.
```

```
    key[0] = 0; // Make key 0 so that this vertex is picked as first vertex
```

```
    parent[0] = -1; // First node is always root of MST
```

```
    // The MST will have V vertices
```

```
    for (int count = 0; count < V - 1; count++) {
```

```
        // Pick the minimum key vertex from the set of vertices not yet included in MST
```

```
        int u = minKey(key, mstSet);
```

```
        // Add the picked vertex to the MST Set
```

```

mstSet[u] = true;

// Update key value and parent index of the adjacent vertices of the picked vertex.
// Consider only those vertices which are not yet included in MST
for (int v = 0; v < V; v++)
    // graph[u][v] is non-zero only for adjacent vertices of u
    // mstSet[v] is false for vertices not yet included in MST
    // Update the key only if graph[u][v] is smaller than key[v]
    if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
        parent[v] = u, key[v] = graph[u][v];
}

// Print the constructed MST
printMST(parent, graph);
}

// Driver code
int main() {
    /* Let us create the following graph
        2   3
        (0)--(1)--(2)
        |   |   |
        6| 8/7 |5
        | /   |
        (3)----- (4)
        9      */
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },

```

```
{ 6, 8, 0, 0, 9 },  
{ 0, 5, 7, 9, 0 } };
```

```
// Print the solution
```

```
primMST(graph);
```

```
return 0;
```

```
}
```

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

LAB 10

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

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

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

```
#include <stdbool.h>
```

```
#define V 9 // Number of vertices in the graph
```

```
int minDistance(int dist[], bool sptSet[]) {
```

```
    int min = INT_MAX, min_index;
```

```
    for (int v = 0; v < V; v++)
```

```
        if (sptSet[v] == false && dist[v] <= min)
```

```
            min = dist[v], min_index = v;
```

```
    return min_index;
```

```
}
```

```
void printSolution(int dist[], int n) {
```

```
    printf("Vertex \t Distance from Source\n");
```

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

```
        printf("%d \t\t %d\n", i, dist[i]);
```

```
}
```

```
void dijkstra(int graph[V][V], int src) {
```

```
    int dist[V];
```

```
    bool sptSet[V];
```

```

for (int i = 0; i < V; i++)
    dist[i] = INT_MAX, sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {
    int u = minDistance(dist, sptSet);

    sptSet[u] = true;

    for (int v = 0; v < V; v++)
        if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
            && dist[u] + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
}

printSolution(dist, V);
}

int main() {
    int graph[V][V] = { {0, 4, 0, 0, 0, 0, 0, 8, 0},
                        {4, 0, 8, 0, 0, 0, 0, 11, 0},
                        {0, 8, 0, 7, 0, 4, 0, 0, 2},
                        {0, 0, 7, 0, 9, 14, 0, 0, 0},
                        {0, 0, 0, 9, 0, 10, 0, 0, 0},
                        {0, 0, 4, 14, 10, 0, 2, 0, 0},
                        {0, 0, 0, 0, 0, 2, 0, 1, 6},
                        {8, 11, 0, 0, 0, 0, 1, 0, 7},
                        {0, 0, 2, 0, 0, 0, 6, 7, 0} };

```

```

dijkstra(graph, 0);

return 0;
}

```

```

Enter the number of vertices: 5
Enter the adjacency matrix (enter 0 if there is no edge between two vertices):
0 10 0 0 5
10 0 1 0 2
0 1 0 4 0
0 0 4 0 3
5 2 0 3 0
Enter the source vertex: 0
Vertex    Distance from Source
0          0
1          7
2          8
3          8
4          5

```

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

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

```
#define INF 9999
```

```
#define MAX 100
```

```
void kruskals(int c[MAX][MAX], int n) {
```

```
    int parent[MAX];
```

```
    int ne = 0;
```

```
    int mincost = 0;
```

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

```
        parent[i] = 0;
```

```
}
```

```
while (ne != n - 1) {
```

```
    int min = INF;
```

```
    int u = 0, v = 0, a = 0, b = 0;
```

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

```
        for (int j = 1; j <= n; j++) {
```

```
            if (c[i][j] < min) {
```

```
                min = c[i][j];
```

```
                u = i;
```

```
                a = i;
```

```
                v = j;
```

```
                b = j;
```

```
            }
```

```
        }
```

```
    }
```

```
while (parent[u] != 0) {
```

```
    u = parent[u];
```

```
}
```

```
while (parent[v] != 0) {
```

```
    v = parent[v];
```

```
}
```

```
if (u != v) {
```

```
    printf("Edge %d-%d with cost %d\n", a, b, min);
```

```
    parent[v] = u;
```

```

        ne++;

        mincost += min;
    }

    c[a][b] = INF;
    c[b][a] = INF;
}

printf("Minimum cost of spanning tree: %d\n", mincost);
}

int main() {
    int n;
    int cost[MAX][MAX];

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

    printf("Enter the cost adjacency matrix:\n");
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            scanf("%d", &cost[i][j]);
            if (cost[i][j] == 0) {
                cost[i][j] = INF;
            }
        }
    }

    kruskals(cost, n);
}

```



```
return 0;  
}
```

```
Enter the number of vertices: 4  
Enter the number of edges: 5  
Enter the source, destination, and weight of each edge:  
0 1 10  
0 2 6  
0 3 5  
1 3 15  
1 2 6  
Edges in the Minimum Spanning Tree:  
0 -- 3 == 5  
0 -- 2 == 6  
1 -- 2 == 6
```