VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

REVANTH K (1BM22CS220)

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

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 **REVANTH K (1BM22CS220),** 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   Dr. Jyothi S Nayak
Assistant Professor   Professor and Head
Department of CSE   Department of CSE
BMSCE, Bengaluru   BMSCE, Bengaluru

Index Sheet

	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.	47-52

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.

WEEK 01

LEETCODE 01 - 459 - repeated substring pattern.

```c
bool repeatedSubstringPattern(char * s){

    int i = 1;

    int l = strlen(s);

    int x = 0;

    while(i <= l/2)

    {

        x =0;

        if(l%i == 0)

        {

            while(x < l)

            {

                if(s[x%i] != s[x])

                {

                    break;

                }

                x++;

            }

            if(x == l){

                return 1;

            }

        }

        i++;

    }

    return 0;

}
```

OUTPUT :

WEEK 02

LEETCODE 02 - 2583 – Kth Largest Sum in a Binary Tree
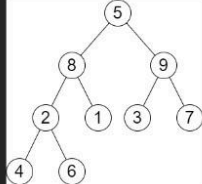
```c
int height(struct TreeNode* root)
{
  if(root==NULL)
  {
    return 0;
  }
  else
  {
    int lheight=height(root->left);
    int rheight=height(root->right);
    if(lheight>rheight)
```

```c
        {
            return lheight+1;
        }
        else
        {
            return rheight+1;
        }
    }


}
void dfs(struct TreeNode* root, int level, long long* sums) {
    if (root == NULL){
        return;
    }
    sums[level] =sums[level]+ root->val;
    if(root->left)
    {
    dfs(root->left, level + 1, sums);
    }
    if(root->right){
    dfs(root->right, level + 1, sums);
    }


}
long long kthLargestLevelSum(struct TreeNode* root, int k) {
    int h = height(root);
    if (k > h) {
```

```
    return -1;

    }

    long long* sums = (long long*)calloc(h, sizeof(long long));


    dfs(root, 0, sums);


    for (int i = 0; i < h - 1; i++) {
        for (int j = 0; j < h - i - 1; j++) {
            if (sums[j] < sums[j + 1]) {
                long long temp = sums[j];
                sums[j] = sums[j + 1];
                sums[j + 1] = temp;

            }

        }

    }



    long long largest = 0;


    largest=sums[k-1];


    free(sums);
    return largest;
}
```

OUTPUT :

WEEK 03

LEETCODE 03 - 897 - Kth largest sum in binary tree.

```c
void inorder(struct TreeNode* root,struct TreeNode** ptr,struct TreeNode** ptr1)
{
    if(root == NULL)
    {
        return;
    }
    inorder(root->left,ptr,ptr1);
    if(*ptr == NULL)
    {
        *ptr = root;
    }
    else
    {
        (*ptr1)->right=root;
```

```
    }
    *ptr1 = root;
    root -> left = NULL;
    inorder(root->right,ptr,ptr1);
}


struct TreeNode* increasingBST(struct TreeNode* root) {
    struct TreeNode* ptr =NULL;
    struct TreeNode* ptr1 =NULL;
    inorder(root,&ptr,&ptr1);
    return ptr;
}
```

OUTPUT :

WEEK 04

4a . Write program to obtain the Topological ordering of vertices in a given digraph

```c
#include<stdio.h>
#include<stdlib.h>
void topo(int a[10][10], int n);

int main()
{
   int a[10][10], n;
   printf("Enter the number of vertices: \n");
   scanf("%d", &n);
   printf("Enter the adjacency matrix of the graph: \n");
   for(int i = 0; i < n; i++)
   {
      for(int j = 0; j < n; j++)
      {
         scanf("%d", &a[i][j]);
      }
   }
   topo(a, n);
   return 0;
}

void topo(int a[10][10], int n)
{
```

```
int indegree[10];
int s[10], T[10];
int top = -1;

// Calculate indegree for each vertex
for(int i = 0; i < n; i++)
{
    int sum = 0;
    for(int j = 0; j < n; j++)
    {
        sum += a[j][i];
    }
    indegree[i] = sum;
    if(indegree[i] == 0)
    {
        top++;
        s[top] = i;
    }
}

int idx = 0;
while(top != -1)
{
    int u = s[top];
    top--;
    T[idx++] = u;
```

```c
        for(int j = 0; j < n; j++)
        {
            if(a[u][j] == 1)
            {
                indegree[j]--;
                if(indegree[j] == 0)
                {
                    top++;
                    s[top] = j;
                }
            }
        }
    }
    printf("The jobs that need to be executed in order are: \n");
    for(int i = 0; i < n; i++)
    {
        printf("%d\t", T[i]);
    }
    printf("\n");
}
```

OUTPUT :

```
Enter the number of vertices:
4
Enter the adjacency matrix of the graph:
0 0 0 0
1 0 0 0
1 0 0 1
0 1 0 0
The jobs that need to be executed in order are:
2       3       1       0

Process returned 0 (0x0)    execution time : 19.398 s
Press any key to continue.
```

4 b . Write a program to obtain the Topological ordering of vertices in a given digraph using DFS.

#include <stdio.h>

#define v 100

int j=0;

void dfs(int a_matrix[v][v],int n,int visited[],int start,int res[])

{

    visited[start]=1;

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

    {

      if(a_matrix[start][i]==1&& visited[i]==0)

      {

        dfs(a_matrix,n,visited,i,res);

      }

```c
        }
    res[j++]=start;
}
void toposort(int a_matrix,int n)
{
    int visited[v]={0};
    int res[v];
    j=0;
    for(int i=0;i<n;i++)
    {
        if(visited[i]==0)
        {
            dfs(a_matrix,n,visited,i,res);
        }
    }
    printf("the topological sort:");
    for(int i=n-1;i>=0;i--)
    {
        printf("%d",res[i]);
    }
}


int main()
```

```c
{
    int a_matrix[v][v];
    int n;

    printf("enter the no of vertices:");
    scanf("%d",&n);
    printf("enter the adjacency matrix:\n");
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            scanf("%d",&a_matrix[i][j]);
        }
    }
    toposort(a_matrix,n);
    return 0;
}
```

OUTPUT :

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

Process returned 0 (0x0)   execution time : 16.110 s
Press any key to continue.
```

WEEK 05

5. Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

#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);

int main() {

   int a[15000], n, i, j, ch, temp;

```c
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]);

            }
```

```c
        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] = n - i;

            }

            start = clock();

            split(a, 0, n-1);

            // Increase the delay by adjusting loop iteration count

            for(j = 0; j < 500000000; j++) {

                temp = 38 / 600;

            }

            end = clock();

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

            n += 1000;

        }

        break;

    case 3:

        exit(0);

        break;
```

```
        }

        getchar();

    }

    return 0;

}


void split(int a[], int low, int high) {

    int mid;

    if(low < high) {

        mid = (low + high) / 2;

        split(a, low, mid);

        split(a, mid+1, high);

        combine(a, low, mid, high);

    }

}


void combine(int a[], int low, int mid, int high) {

    int c[15000], i, j, k;

    i = k = low;

    j = mid + 1;

    while(i <= mid && j <= high) {

        if(a[i] < a[j]) {

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

```
        ++k;

        ++i;

    } else {

        c[k] = a[j];

        ++k;

        ++j;

    }

}

if(i > mid) {

    while(j <= high) {

        c[k] = a[j];

        ++k;

        ++j;

    }

}

if(j > high) {

    while(i <= mid) {

        c[k] = a[i];

        ++k;

        ++i;

    }

}

for(i = low; i <= high; i++) {
```

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

    }

}
```

OUTPUT :

```
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 1
Enter the number of elements: 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 0.091000 Secs
Time taken to sort 1500 numbers is 0.088000 Secs
Time taken to sort 2500 numbers is 0.087000 Secs
Time taken to sort 3500 numbers is 0.084000 Secs
Time taken to sort 4500 numbers is 0.093000 Secs
Time taken to sort 5500 numbers is 0.086000 Secs
Time taken to sort 6500 numbers is 0.081000 Secs
Time taken to sort 7500 numbers is 0.090000 Secs
Time taken to sort 8500 numbers is 0.080000 Secs
Time taken to sort 9500 numbers is 0.088000 Secs
Time taken to sort 10500 numbers is 0.078000 Secs
Time taken to sort 11500 numbers is 0.085000 Secs
Time taken to sort 12500 numbers is 0.088000 Secs
Time taken to sort 13500 numbers is 0.088000 Secs
Time taken to sort 14500 numbers is 0.081000 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
```

WEEK 06

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

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <stdbool.h>

```
void quickSort(int[], int, int);

int partition(int[], int, int);


void quickSort(int A[], int low, int high) {

    if(low < high) {

        int split_point = partition(A, low, high);

        quickSort(A, low, split_point -  1);

        quickSort(A, split_point + 1, high);

    }

}


int partition(int A[], int low, int high) {

    int pivot = A[low];

    int i = low;

    int j = high + 1;

    while(i <= j) {

        while(true) {

            i += 1;

            if(A[i] >= pivot) {

                break;

            }

        }

        while(true) {
```

```
            j -= 1;

            if(A[j] <= pivot) {

                break;

            }

        }


        int temp = A[i];

        A[i] = A[j];

        A[j] = temp;

    }


    int temp = A[i];

    A[i] = A[j];

    A[j] = temp;


    int temp1 = A[j];

    A[j] = A[low];

    A[low] = temp1;


    return j;

}


int main() {
```

```c
 int a[15000], n, i, j, ch;

clock_t start, end;

while(1) {

    printf("\n1: For manual entry of N value and array elements");

    printf("\n2: To display time taken for sorting number of elements N in the range 500 to
14500");

    printf("\n3: To exit");

    printf("\nEnter your choice: ");

    scanf("%d", &ch);

    switch(ch) {

        case 1:

            printf("\nEnter the number of elements: ");

            scanf("%d", &n);

            printf("\nEnter array elements: ");

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

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

            }

            start = clock();

            quickSort(a, 0, n-1);

            end = clock();

            printf("\nSorted array is: ");

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

                printf("%d\t", a[i]);

            }
```

```c
        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] = n - i;

            }

            start = clock();

            quickSort(a, 0, n-1);

            for(int j=0; j<50000000; j++) {

                float temp = 38/600;

            }

            end = clock();

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

            n += 1000;

        }

        break;

    case 3:

        exit(0);

        break;

    }
```

```
        getchar();

    }

    return 0;

}
```

OUTPUT :

```
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 1
Enter the number of elements: 5
Enter array elements: 2 4 3 1 5
Sorted array is: 1      2       3       4       5       Time taken to sort 5 numbers is 0.000000 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 2
Time taken to sort 500 numbers is 0.090000 Secs
Time taken to sort 1500 numbers is 0.080000 Secs
Time taken to sort 2500 numbers is 0.090000 Secs
Time taken to sort 3500 numbers is 0.095000 Secs
Time taken to sort 4500 numbers is 0.105000 Secs
Time taken to sort 5500 numbers is 0.122000 Secs
Time taken to sort 6500 numbers is 0.128000 Secs
Time taken to sort 7500 numbers is 0.150000 Secs
Time taken to sort 8500 numbers is 0.165000 Secs
Time taken to sort 9500 numbers is 0.183000 Secs
Time taken to sort 10500 numbers is 0.202000 Secs
Time taken to sort 11500 numbers is 0.230000 Secs
Time taken to sort 12500 numbers is 0.265000 Secs
Time taken to sort 13500 numbers is 0.276000 Secs
Time taken to sort 14500 numbers is 0.309000 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
```

WEEK 07
7 a . Implement Johnson Trotter algorithm to generate permutations

#include <stdio.h>

#include <stdlib.h>

int flag = 0;

```c
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) {
```

```c
    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;
```

```c
    for (i = 1; i < k + 1; i++) {

    f = f * i;

    }

    return f;

}


int main() {

        int num = 0;

        int i, j, 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
3
total permutations = 6
All possible permutations are:
 1  2  3
 1  3  2
 3  1  2
 3  2  1
 2  3  1
 2  1  3

Process returned 0 (0x0)   execution time : 1.745 s
Press any key to continue.
```

7 b. PATTERN MATCHING

#include <stdio.h>

#include <string.h>

// Function to find the position of the substring

int findSubstring(char *text, char *pattern) {

```c
        int n = strlen(text);

        int m = strlen(pattern);


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

        int j;

        for (j = 0; j < m; j++) {

        if (text[i + j] != pattern[j]) {

            break;

        }

        }

        if (j == m) {

        return i; // Return the starting index of the substring

        }

        }


        return -1; // Return -1 if the substring is not found

}


int main() {

        char text[100];

        char pattern[100];


    printf("Enter the text: ");
```

```c
    fgets(text, sizeof(text), stdin);

text[strcspn(text, "\n")] = '\0'; // Remove the newline character from input


printf("Enter the pattern: ");

fgets(pattern, sizeof(pattern), stdin);

pattern[strcspn(pattern, "\n")] = '\0'; // Remove the newline character from input


    int position = findSubstring(text, pattern);


    if (position != -1) {

    printf("Pattern found at index %d\n", position);

    } else {

    printf("Pattern not found in the text.\n");

    }


    return 0;

}
```

OUTPUT :

```
enter the text :
helloworld
enter the pattern :
world
pattern found at index 6

Process returned 0 (0x0)   execution time : 5.354 s
Press any key to continue.
```

LEETCODE-1985

Leetcode-4: Find kth Largest Integer in the array:

```c
int cmp(const void*a,const void*b) {

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

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

  if (strlen(str1) == strlen(str2)) {

        return strcmp(str1, str2);

    }

    return strlen(str1) - strlen(str2);

}

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

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

return nums[numsSize-k];

}
```

OUTPUT :
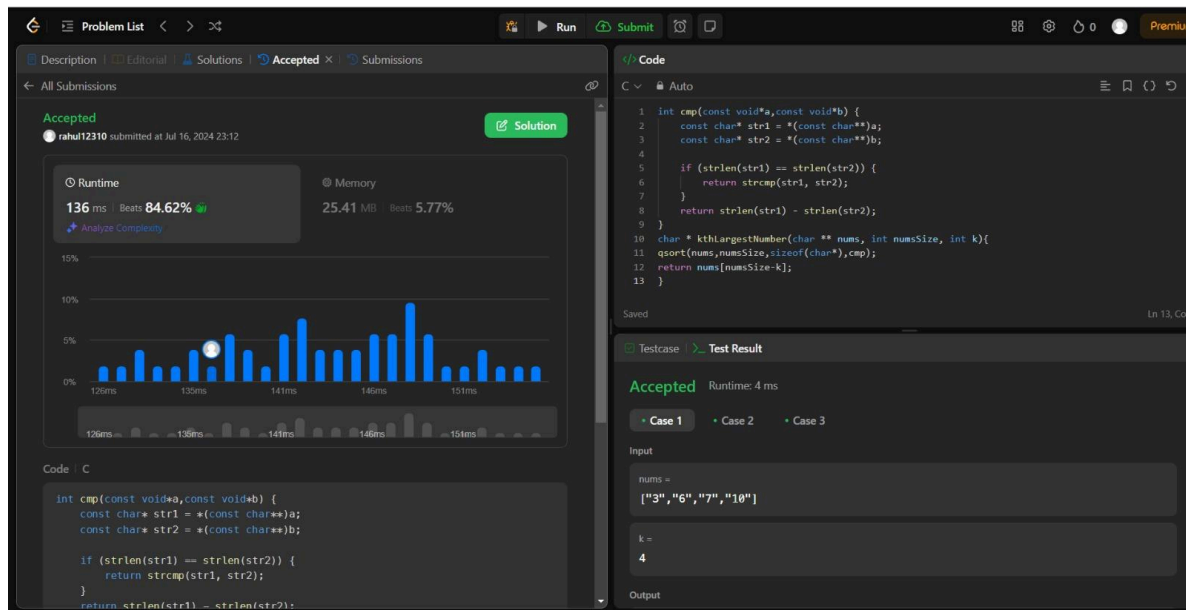
WEEK 08
8 a. Sort a given set of N integer elements using Heap Sort technique and compute its time

```c
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Heap Sort in C

// Function to swap the position of two elements

void swap(int* a, int* b)

{

    int temp = *a;

    *a = *b;
```

```c
    *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
```

```c
    // 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);

    }

}

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);

    }

}
```

```c
int main() {
    int a[15000], n, i, j, ch, temp;
    clock_t start, end;
    while (1) {
        printf("1: 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("Enter the number of elements: ");
                scanf("%d", &n);
                printf("Enter array elements: ");
                for (i = 0; i < n; i++) {
                    scanf("%d", &a[i]);
                }
                start = clock();
                heapSort(a, n);
                end = clock();
                printf("Sorted array is: ");
                for (i = 0; i < n; i++) {
                    printf("%d\t", a[i]);
```

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

    break;
case 2:

  n = 500;

    while (n <= 14500) {

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

        a[i] = n - i;

      }

      start = clock();

      heapSort(a, n);

      // Dummy loop to create delay

      for (j = 0; j < 50000000; j++) { temp = 38 / 600; }

      end = clock();

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

      n += 1000;

    }

    break
  case 3:

      exit(0);


    default:
```

```
            printf("\nInvalid choice! Please try again.\n");

      }

   }

   return 0;

}
```

OUTPUT :

```
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 1
Enter the number of elements: 5
Enter array elements: 4 3 5 2 1
Sorted array is: 1      2       3       4       5       Time taken to sort 5 numbers is 0.000000 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 2
Time taken to sort 500 numbers is 0.159000 Secs
Time taken to sort 1500 numbers is 0.113000 Secs
Time taken to sort 2500 numbers is 0.109000 Secs
Time taken to sort 3500 numbers is 0.118000 Secs
Time taken to sort 4500 numbers is 0.115000 Secs
Time taken to sort 5500 numbers is 0.131000 Secs
Time taken to sort 6500 numbers is 0.117000 Secs
Time taken to sort 7500 numbers is 0.113000 Secs
Time taken to sort 8500 numbers is 0.152000 Secs
Time taken to sort 9500 numbers is 0.125000 Secs
Time taken to sort 10500 numbers is 0.123000 Secs
Time taken to sort 11500 numbers is 0.110000 Secs
Time taken to sort 12500 numbers is 0.120000 Secs
Time taken to sort 13500 numbers is 0.136000 Secs
Time taken to sort 14500 numbers is 0.136000 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
```

8 b . Implement All Pair Shortest paths problem using Floyd's algorithm.

```c
 #include <stdio.h>

#define V 5

#define INF 99999

// A function to print the solution matrix

void printSolution(int dist[][V]);
```

```
void floydWarshall(int dist[][V])

{

    int i, j, k;

    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 (dist[i][k] + dist[k][j] < dist[i][j])

                    dist[i][j] = dist[i][k] + dist[k][j];

            }

        }

    }

    // Print the shortest distance matrix

    printSolution(dist);

}


/* A utility function to print solution */

void printSolution(int dist[][V])

{

    printf(

        "The following matrix shows the shortest distances"
```

```c
        " between every pair of vertices \n");

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

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

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

                printf("%7s", "INF");

            else

                printf("%7d", dist[i][j]);

        }

        printf("\n");

    }

}

// driver's code

int main()

{

  int graph[V][V] = { { 0, 4, INF, 5,INF },

                { INF, 0, 1, INF,6 },

                { 2,INF, 0, 3,INF },

                { INF, INF, 1, 0,2 } ,

                {1,INF,INF,4,0}};

    // Function call

    floydWarshall(graph);

    return 0;
```

}

OUTPUT :

```
The following matrix shows the shortest distances between every pair of vertices
      0      4      5      5      7
      3      0      1      4      6
      2      6      0      3      5
      3      7      1      0      2
      1      5      5      4      0

Process returned 0 (0x0)   execution time : 0.080 s
Press any key to continue.
```

WEEK 09

9 a . Implement a Knapsack problem using dynamic programming.

#include <stdio.h>

#define N 4 // Number of items

#define CAPACITY 10 // Capacity of the knapsack

// Structure to represent an item

struct Item {

   int weight;

```
    int profit;

};


// Function to find the maximum of two integers

int max(int a, int b) {

    return (a > b) ? a : b;

}


// Function to solve the 0/1 Knapsack problem using Dynamic Programming

void knapsack(struct Item items[], int n, int capacity) {

    // Create a DP table

    int dp[n + 1][capacity + 1];

    // Initialize the DP table

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

        for (int w = 0; w <= capacity; w++) {

            if (i == 0 || w == 0)

                dp[i][w] = 0;

            else if (items[i - 1].weight <= w)

                dp[i][w] = max(items[i - 1].profit + dp[i - 1][w - items[i - 1].weight], dp[i - 1][w]);

            else

                dp[i][w] = dp[i - 1][w];

        }

    }
```

```c
    // Find the maximum profit

    int maxProfit = dp[n][capacity];

    printf("Maximum Profit: %d\n", maxProfit);


    // Find the items selected

    int remainingCapacity = capacity;

    printf("Items selected:\n");

    for (int i = n; i > 0 && maxProfit > 0; i--) {

        if (maxProfit != dp[i - 1][remainingCapacity]) {

            printf("Item %d (weight: %d, profit: %d)\n", i, items[i - 1].weight, items[i - 1].profit);

            maxProfit -= items[i - 1].profit;

            remainingCapacity -= items[i - 1].weight;

        }

    }

}


int main() {

    struct Item items[N] = {

        {2, 6},  // Item 1: weight = 2, profit = 6

        {3, 5},  // Item 2: weight = 3, profit = 5

        {4, 8},  // Item 3: weight = 4, profit = 8

        {5, 9}   // Item 4: weight = 5, profit = 9

    };
```

knapsack(items, N, CAPACITY);

   return 0;

}

OUTPUT :

```
DP Table:
   0   0   0   0   0   0
   0   0  12  12  12  12
   0  10  12  22  22  22
   0  10  12  22  30  32
   0  10  15  25  30  37

Maximum value in Knapsack = 37
Items included in the knapsack:
Item 4 (Value: 15, Weight: 2)
Item 2 (Value: 10, Weight: 1)
Item 1 (Value: 12, Weight: 2)

Process returned 0 (0x0)   execution time : 0.202 s
Press any key to continue.
```


9b . Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

#include <stdio.h>

#include <stdbool.h>

```c
#include <limits.h>

#define V 5

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;

}

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]]);

}

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

{

   int parent[V];

   int key[V];

   bool mstSet[V];

// Initialize all keys as INFINITE

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

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



    key[0] = 0;

    parent[0] = -1;

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

  int u = minKey(key, mstSet);

   mstSet[u] = true;

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

  if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

            parent[v] = u, key[v] = graph[u][v];

    }

printMST(parent, graph);

}

int main()

{

    /* Let us create the following graph

        2    3

      (0)--(1)--(2)

      |  /\  |

      6| 8/  \5 |7

      |/    \|

      (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;

}
```

OUTPUT :

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

Process returned 0 (0x0)   execution time : 0.158 s
Press any key to continue.
```

WEEK 10

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

```cpp
// A C++ program for Dijkstra's single source shortest path

#include <limits.h>

#include <stdio.h>

#include <stdbool.h>

#define V 5

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   Distance from Source\n");

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

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

}

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

```c
{
    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, 11, 9, 7, 8},

                {11, 0, 5, 14, 13},

                {9, 5, 0, 12, 14},
```

{7, 14, 12, 0, 6},

{8, 13, 14, 6, 0},

};

dijkstra(graph, 0);

    return 0;

}

OUTPUT :

```
Vertex   Distance from Source
        0                           0
        1                           11
        2                           9
        3                           7
        4                           8

Process returned 0 (0x0)   execution time : 0.087 s
Press any key to continue.
```

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

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define V 5

int parent[V];

int find(int i)

```
{

    while (parent[i] != i)

        i = parent[i];

    return i;

}

void union1(int i, int j)

{

    int a = find(i);

    int b = find(j);

    parent[a] = b;

}


void kruskalMST(int cost[][V])

{

    int mincost = 0; // Cost of min MST.

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

        parent[i] = i;

    int edge_count = 0;

    while (edge_count < V - 1) {

        int min = INT_MAX, a = -1, b = -1;

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

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

                if (find(i) != find(j) && cost[i][j] < min) {
```

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

                    a = i;

                    b = j;

                }

            }

        }


        union1(a, b);

        printf("Edge %d:(%d, %d) cost:%d \n",

                edge_count++, a, b, min);

        mincost += min;

    }

    printf("\n Minimum cost= %d \n", mincost);

}

int main()

{

    int cost[][V] = {

        { INT_MAX, 2, INT_MAX, 6, INT_MAX },

        { 2, INT_MAX, 3, 8, 5 },

        { INT_MAX, 3, INT_MAX, INT_MAX, 7 },

        { 6, 8, INT_MAX, INT_MAX, 9 },

        { INT_MAX, 5, 7, 9, INT_MAX },

    };
```

```
  kruskalMST(cost);

  return 0;

}
```

OUTPUT:

```
Edge 0:(0, 1) cost:2
Edge 1:(1, 2) cost:3
Edge 2:(1, 4) cost:5
Edge 3:(0, 3) cost:6

 Minimum cost= 16

Process returned 0 (0x0)   execution time : 0.144 s
Press any key to continue.
```