VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

PRANEETA M REDDY(1BM22CS205)

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 **PRANEETA M REDDY(1BM22CS205),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester April-2024 to August-2024. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (23CS4PCADA)** work prescribed for the said degree.

M Lakshmi Neelima
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Course Outcome

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

Lab Program 1

Leetcode- Repeated Substring Pattern

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

Code:

```
bool repeatedSubstringPattern(char* s) {
    int n=strlen(s);
    for(int i=1;i<=(n/2);i++)
    {
        if(n%i==0)
        {
            int flag=1;
            for(int j=i;j<n;j++)
            {
                if(s[j]!=s[j%i])
                {
                    flag=0;
                    break;
                }

            }
            if(flag==1)
            {
                return true;
            }
        }
    }
    return false;
}
```

Result:

Test Case-1:

Test Case-2:

Test Case-3:

Lab Program 2

Leetcode-Kth Largest Sum in a Binary Tree

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.

Code:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
 int height(struct TreeNode* root)
 {
   if(root==NULL)
   {
      return 0;
   }
   else
   {
      int lheight=height(root->left);
      int rheight=height(root->right);
      if(lheight>rheight)
      {
         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;
}
```

Result:

Test Case-1:

Test Case-2:

Lab Program 3

Leetcode-Increasing Order Search Tree

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.

Code:

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

void inorder (struct TreeNode *root,struct TreeNode* nodes[],int *i)
{
   if(root!=NULL)
   {
      inorder(root->left,nodes,i);
      nodes[((*i)++)]=root;
      inorder(root->right,nodes,i);




   }
}

struct TreeNode* increasingBST(struct TreeNode* root) {
   int i=0;
   struct TreeNode* nodes[100];
   inorder(root,nodes,&i);

   for( int j=0;j<i-1;j++)
   {
      nodes[j]->left=NULL;
      nodes[j]->right=nodes[j+1];
   }
   nodes[i-1]->left = NULL;
   nodes[i-1]->right = NULL;
   return nodes[0];
}
```

Result:

Test Case-1:

Accepted Runtime: 3 ms

• Case 1 • Case 2

Input

root =
[5,3,6,2,4,null,8,1,null,null,null,7,9]

Output

[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

Expected

[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

Test Case-2:

Accepted Runtime: 3 ms

• Case 1 • Case 2

Input

root =
[5,1,7]

Output

[1,null,5,null,7]

Expected

[1,null,5,null,7]

Lab Program 4

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

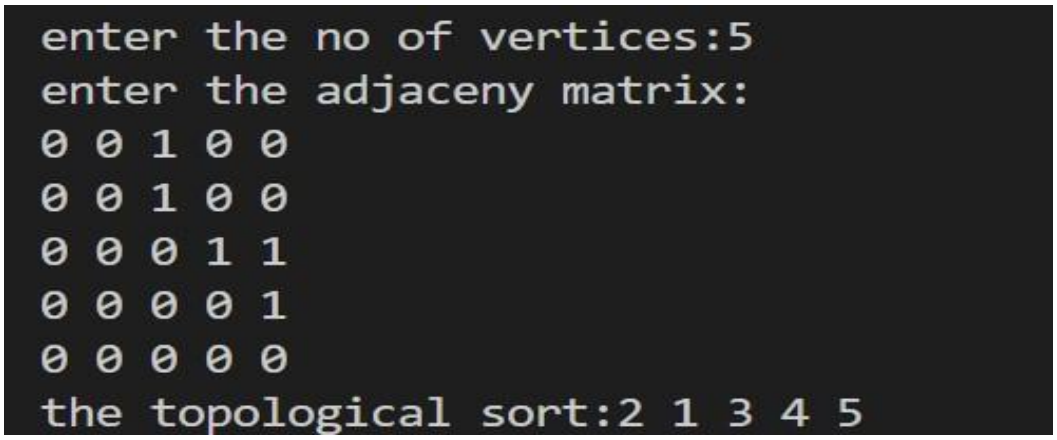
1)Source Removal Method

Code:

```
#include <stdio.h>
#define v 100
int top=-1;
void indegree(int a_matrix[v][v],int n,int in[v])
{
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<n;j++)
      {
         if(a_matrix[i][j])
         {
            in[j]++;
         }
      }
   }
}
void toposort(int a_matrix[v][v],int n)
{
   int in[v]={0};
   int topo[v];
   int k=0;

   int s[v]={0};
   indegree(a_matrix,n,in);
   for(int i=0;i<n;i++)
   {
      if(in[i]==0)
      {
        top++;
        s[top]=i;

      }
   }
   while(top!=-1)
   {
      int vertex=s[top];
      top--;
      topo[k++]=vertex;
```

```
        for(int i=0;i<n;i++)
        {
            if(a_matrix[vertex][i])
            {
                in[i]--;
                if(in[i]==0)
                {
                    top++;
                    s[top]=i;
                }
            }
        }
    }
    if(k!=n)
    {
        printf("cycle exists");
    }
    else{
    printf("the topological sort:");
    for(int i=0;i<n;i++)
    {
        printf("%d ",topo[i]+1);
    }
    }
}
int main()
{
    int a_matrix[v][v];
    int n;
    printf("enter the no of vertices:");
    scanf("%d",&n);
    printf("enter the adjaceny 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;
}
```

Result:



2)DFS

Code:

```
#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);
         }
      }

   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()
{
    int a_matrix[v][v];
    int n;

    printf("enter the no of vertices:");
    scanf("%d",&n);
    printf("enter the adjaceny 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;
}
```

Result:

```
enter the no of vertices:4
enter the adjaceny matrix:
0 0 0 0
1 0 0 0
1 0 0 1
0 1 0 0
the topological sort:2310
```

Lab Program 5

Implement Johnson Trotter algorithm to generate permutations.

Code:

```
#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 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 (int i = 0; i < num; i++) {
        printf(" %d", arr[i]);
    }
    printf("\n");
}

int factorial(int k) {
    int f = 1;
    for (int i = 1; i < k + 1; i++) {
        f = f * i;
    }
    return f;
}

int main() {
```

```
    int num = 0;
    printf("Johnson Trotter algorithm to find all permutations of given numbers \n");
    printf("Enter the number: ");
    scanf("%d", &num);

    int arr[num], d[num];
    int z = factorial(num);

    printf("Total permutations = %d\n", z);
    printf("All possible permutations are:\n");

    for (int i = 0; i < num; i++) {
        d[i] = 0;
        arr[i] = i + 1;
        printf(" %d", arr[i]);
    }
    printf("\n");

    for (int j = 1; j < z; j++) {
        permutations(arr, d, num);
    }

    return 0;
}
```

Result:

```
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
```

Lab Program 6

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.

Code:

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

void split(int[], int, int);
void combine(int[], int, int, int);

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();
            split(a, 0, n - 1);
            end = clock();
            printf("Sorted array is: ");
            for (i = 0; i < n; i++) {
               printf("%d\t", a[i]);
            }
            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();
                    split(a, 0, n - 1);
                    // 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;
}

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;
```

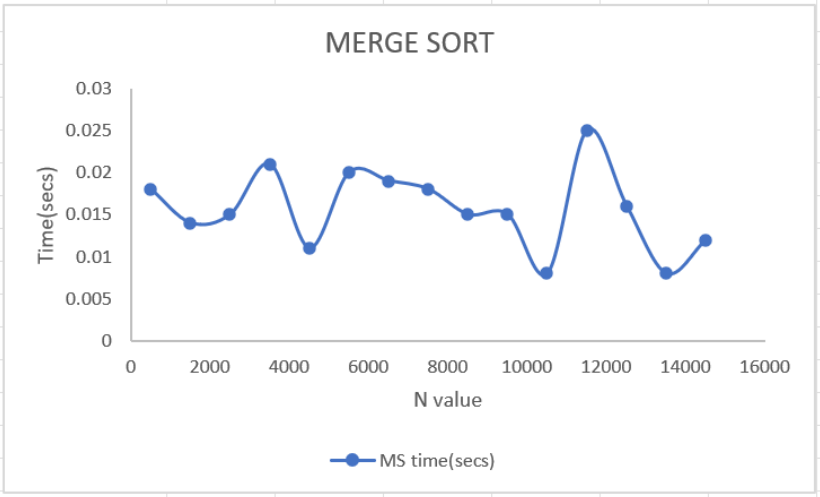
```
            ++i;
        } else {
            c[k] = a[j];
            ++k;
            ++j;
        }
    }
    if(i>mid){
    while (j <= high) {
        c[k] = a[j];
        ++k;
        ++j;
    }}
    if(j>high){
    while (i <= mid) {
        c[k] = a[i];
        ++k;
        ++i;
    }}
    for (i = low; i <= high; i++) {
        a[i] = c[i];
    }
}
```

Result:

```
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 0.018000 Secs
Time taken to sort 1500 numbers is 0.014000 Secs
Time taken to sort 2500 numbers is 0.015000 Secs
Time taken to sort 3500 numbers is 0.021000 Secs
Time taken to sort 4500 numbers is 0.011000 Secs
Time taken to sort 5500 numbers is 0.020000 Secs
Time taken to sort 6500 numbers is 0.019000 Secs
Time taken to sort 7500 numbers is 0.018000 Secs
Time taken to sort 8500 numbers is 0.015000 Secs
Time taken to sort 9500 numbers is 0.015000 Secs
Time taken to sort 10500 numbers is 0.008000 Secs
Time taken to sort 11500 numbers is 0.025000 Secs
Time taken to sort 12500 numbers is 0.016000 Secs
Time taken to sort 13500 numbers is 0.008000 Secs
Time taken to sort 14500 numbers is 0.012000 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 3
```

Graph:

N value	MS time(secs)
500	0.018
1500	0.014
2500	0.015
3500	0.021
4500	0.011
5500	0.02
6500	0.019
7500	0.018
8500	0.015
9500	0.015
10500	0.008
11500	0.025
12500	0.016
13500	0.008
14500	0.012



Lab Program 7

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

Code:

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

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

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();
            split(a, 0, n - 1);
            end = clock();
            printf("Sorted array is: ");
            for (i = 0; i < n; i++) {
               printf("%d\t", a[i]);
            }
            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();
                    split(a, 0, n - 1);
                    // 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;
}

void split(int a[], int l, int h) {
    int j;
    if (l < h) {
        j = partition(a,l,h);
        split(a, l, j);
        split(a, j+ 1, h);

    }
}

int partition(int a[], int l, int h) {
    int pivot=a[l];
    int i=l+1;
    int j=h;
    int temp,t;
    do
    {
        while(i<=h && pivot>=a[i])
        {
```

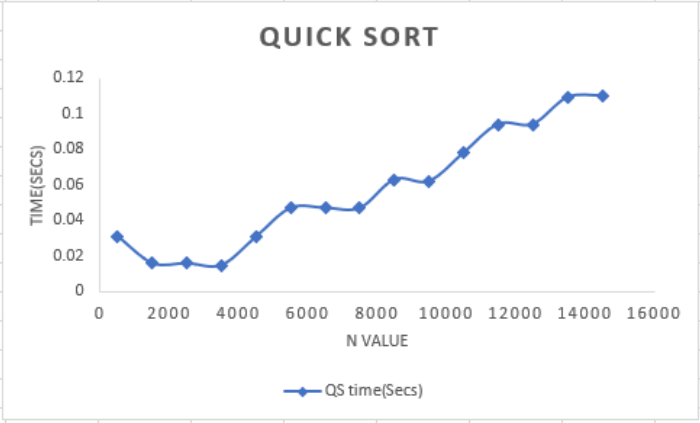
```
        i++;
      }
      while(pivot<a[j])
      {
        j--;
      }
      if(i<j)
      {
        temp=a[i];
        a[i]=a[j];
        a[j]=temp;

      }

   }while(i<j);
   t=a[l];
   a[l]=a[j];
   a[j]=t;
   return j;
}
```

Result:

```
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: 8
Enter array elements: 22 55 77 11 5 6 3 8
Sorted array is: 3      5       6       8       11      22      55      77      Time taken to sort 8 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 0.031000 Secs
Time taken to sort 1500 numbers is 0.016000 Secs
Time taken to sort 2500 numbers is 0.016000 Secs
Time taken to sort 3500 numbers is 0.015000 Secs
Time taken to sort 4500 numbers is 0.031000 Secs
Time taken to sort 5500 numbers is 0.047000 Secs
Time taken to sort 6500 numbers is 0.047000 Secs
Time taken to sort 7500 numbers is 0.047000 Secs
Time taken to sort 8500 numbers is 0.063000 Secs
Time taken to sort 9500 numbers is 0.062000 Secs
Time taken to sort 10500 numbers is 0.078000 Secs
Time taken to sort 11500 numbers is 0.094000 Secs
Time taken to sort 12500 numbers is 0.094000 Secs
Time taken to sort 13500 numbers is 0.109000 Secs
Time taken to sort 14500 numbers is 0.110000 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 3
```

Graph:

N value	QS time(Secs)
500	0.031
1500	0.016
2500	0.016
3500	0.015
4500	0.031
5500	0.047
6500	0.047
7500	0.047
8500	0.063
9500	0.062
10500	0.078
11500	0.094
12500	0.094
13500	0.109
14500	0.11



Lab Program 8

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

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.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);
   }
}

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

         case 2:
```

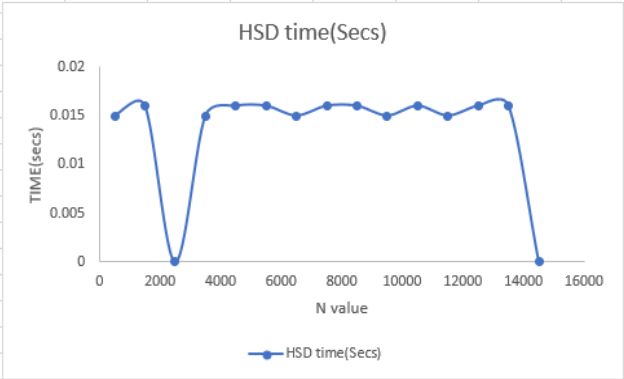
```
            n = 500;
            while (n <= 14500) {
                for (i = 0; i < n; i++) {
                    a[i] = n - i;
                }
                start = clock();
                heapSort(a, n);
                // Dummy loop to create delay
                for (j = 0; j < 50000000; 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;
}
```

Result:

```
Enter the number of elements: 5
Enter array elements: 2 7 33 12 1
Sorted array is: 1      2       7       12      33      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 0.016000 Secs
Time taken to sort 1500 numbers is 0.015000 Secs
Time taken to sort 2500 numbers is 0.016000 Secs
Time taken to sort 3500 numbers is 0.016000 Secs
Time taken to sort 4500 numbers is 0.015000 Secs
Time taken to sort 5500 numbers is 0.016000 Secs
Time taken to sort 6500 numbers is 0.016000 Secs
Time taken to sort 7500 numbers is 0.015000 Secs
Time taken to sort 8500 numbers is 0.016000 Secs
Time taken to sort 9500 numbers is 0.015000 Secs
Time taken to sort 10500 numbers is 0.016000 Secs
Time taken to sort 11500 numbers is 0.016000 Secs
Time taken to sort 12500 numbers is 0.015000 Secs
Time taken to sort 13500 numbers is 0.016000 Secs
Time taken to sort 14500 numbers is 0.016000 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 3
```

Graph:

N value	HSD time(Secs)
500	0.015
1500	0.016
2500	0
3500	0.015
4500	0.016
5500	0.016
6500	0.015
7500	0.016
8500	0.016
9500	0.015
10500	0.016
11500	0.015
12500	0.016
13500	0.016
14500	0



Lab Program 9

Implement 0/1 Knapsack problem using dynamic programming.

Code:

```
#include <stdio.h>

int max(int a, int b) {
   return (a > b) ? a : b;
}

void knapsack(int W, int wt[], int val[], int n) {
   int i, w;
   int K[n + 1][W + 1];

   for (i = 0; i <= n; i++) {
      for (w = 0; w <= W; w++) {
         if (i == 0 || w == 0) {
            K[i][w] = 0;
         } else if (wt[i - 1] <= w) {
            K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
         } else {
            K[i][w] = K[i - 1][w];
         }
      }
   }

   printf("DP Table:\n");
   for (i = 0; i <= n; i++) {
      for (w = 0; w <= W; w++) {
         printf("%4d", K[i][w]);
      }
      printf("\n");
   }

   int res = K[n][W];
   printf("\nMaximum value in Knapsack = %d\n", res);
   printf("Items included in the knapsack:\n");

   w = W;
   for (i = n; i > 0 && res > 0; i--) {
      if (res == K[i - 1][w])
         continue;
      else {
         printf("Item %d (Value: %d, Weight: %d)\n", i, val[i - 1], wt[i - 1]);
         res -= val[i - 1];
```

```
            w -= wt[i - 1];
        }
    }
}

int main() {
    int n = 4;
    int val[] = {12, 10, 20, 15};
    int wt[] = {2, 1, 3, 2};
    int W = 5;

    knapsack(W, wt, val, n);

    return 0;
}
```

Result:

```
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)
```

Lab Program 10

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

Code:

```
#include <stdio.h>
#define V 5
#define INF 99999
void printSolution(int dist[][V]);
void floydWarshall(int dist[][V])
{
   int i, j, k;
   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(dist);
}

void printSolution(int dist[][V])
{
   printf(
      "The following matrix shows the shortest distances"
      " 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");
   }
}
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}};
    floydWarshall(graph);
    return 0;
}
```

Result:

```
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
```

Lab Program 11

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

Code:

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

#define MAX 100
#define INF 9999

void prims(int n, int cost[MAX][MAX]) {
   int d[MAX], p[MAX], s[MAX];
   int source, min, sum = 0;
   int T[MAX][2], k = 0;
   min = INF;
   source = 0;
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         if (cost[i][j] != 0 && cost[i][j] < min) {
            min = cost[i][j];
            source = i;
         }
      }
   }
   for (int i = 0; i < n; i++) {
      s[i] = 0;
      d[i] = cost[source][i];
      p[i] = source;
   }
   s[source] = 1;

   for (int i = 1; i < n; i++) {
      min = INF;
      int u = -1;
      for (int j = 0; j < n; j++) {
         if (s[j] == 0 && d[j] < min) {
            min = d[j];
            u = j;
         }
      }

      T[k][0] = u;
```

```
            T[k][1] = p[u];
            k++;
            sum += cost[u][p[u]];
            s[u] = 1;


            for (int v = 0; v < n; v++) {
                if (s[v] == 0 && cost[u][v] < d[v]) {
                    d[v] = cost[u][v];
                    p[v] = u;
                }
            }
        }

        if (sum >= INF) {
            printf("Spanning tree does not exist\n");
        } else {
            printf("Spanning tree exists and MST is:\n");
            for (int i = 0; i < n - 1; i++) {
                printf("%d - %d\n", T[i][0], T[i][1]);
            }
            printf("The cost of the Minimum Spanning Tree is: %d\n", sum);
        }
    }
    int main() {
        int n;
        int cost[MAX][MAX];

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

        printf("Enter the cost adjacency matrix (use %d to represent infinity):\n", INF);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                scanf("%d", &cost[i][j]);
            }
        }
        prims(n, cost);
        return 0;
    }
```

Result:

```
Enter the number of vertices: 6
Enter the cost adjacency matrix (use 9999 to represent infinity):
0 60 10 9999 9999 9999
60 0 9999 20 40 70
10 9999 0 9999 9999 50
9999 20 99999 0 9999 80
0 40 99999 9999 0 30
9999 70 50 80 30 0
Spanning tree exists and MST is:
2 - 0
5 - 2
4 - 5
1 - 4
3 - 1
The cost of the Minimum Spanning Tree is: 150
```

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

Code:

```
#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;
}
```

Result:

```
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
```

Lab Program 12

Implement Fractional Knapsack using Greedy technique.

Code:

```
#include <stdio.h>
void greedyKnapsack(int n, float m, float p[], float w[], float x[], float *total_profit) {
    int i;
    float U;
    for (i = 1; i <= n; i++) {
        x[i] = 0.0;
    }
    U = m;
    for (i = 1; i <= n; i++) {
        if (w[i] > U) {
            break;
        }
        x[i] = 1.0;
        U = U - w[i];
        *total_profit += p[i];
    }
    if (i <= n) {
        x[i] = U / w[i];
        *total_profit += x[i] * p[i];
    }
}
int main() {
    int n, i, j;
    float m, total_profit = 0.0;

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

    float p[n + 1], w[n + 1], x[n + 1];

    printf("Enter the capacity of the knapsack: ");
    scanf("%f", &m);

    printf("Enter the profits and weights of the objects:\n");
    for (i = 1; i <= n; i++) {
        printf("Profit[%d]: ", i);
        scanf("%f", &p[i]);
        printf("Weight[%d]: ", i);
        scanf("%f", &w[i]);
```

```
    }
    for (i = 1; i < n; i++) {
        for (j = 1; j <= n - i; j++) {
            if (p[j] / w[j] < p[j + 1] / w[j + 1]) {
                float temp_p = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp_p;

                float temp_w = w[j];
                w[j] = w[j + 1];
                w[j + 1] = temp_w;
            }
        }
    }
    greedyKnapsack(n, m, p, w, x, &total_profit);
    printf("Items chosen and their fractions:\n");
    for (i = 1; i <= n; i++) {
        if (x[i] > 0) {
            printf(" %f of weight %f and profit %f is choosen\n",  x[i], w[i], p[i]);
        }
    }
    printf("Total profit: %f\n", total_profit);

    return 0;
}
```

Result:

```
Enter the number of objects: 3
Enter the capacity of the knapsack: 50
Enter the profits and weights of the objects:
Profit[1]: 10
Weight[1]: 20
Profit[2]: 60
Weight[2]: 10
Profit[3]: 120
Weight[3]: 30
Items chosen and their fractions:
 1.000000 of weight 10.000000 and profit 60.000000 is choosen
 1.000000 of weight 30.000000 and profit 120.000000 is choosen
 0.500000 of weight 20.000000 and profit 10.000000 is choosen
Total profit: 185.000000
```

Lab Program 13

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

Code:

```
#include <stdio.h>

#define INF 9999
#define N 6

void dijkstra(int cost[N][N], int src) {
   int dist[N];
   int vis[N];
   int count, min_dist, u;

   for (int i = 0; i < N; i++) {
      dist[i] = INF;
      vis[i] = 0;
   }

   dist[src] = 0;

   for (count = 0; count < N-1; count++) {
      min_dist = INF;
      for (int v = 0; v < N; v++) {
         if (!vis[v] && dist[v] <= min_dist) {
            min_dist = dist[v];
            u = v;
         }
      }

      vis[u] = 1;

      for (int v = 0; v < N; v++) {
         if (!vis[v] && cost[u][v] && dist[u] != INF && dist[u] + cost[u][v] < dist[v]) {
            dist[v] = dist[u] + cost[u][v];
         }
      }
   }

   printf("Vertex \t\t Distance from Source\n");
   for (int i = 0; i < N; i++) {
      printf("%d \t\t %d\n", i, dist[i]);
   }
}
```

```
int main() {
   int cost[N][N] = {
      {0, 15, 10, INF, 45, INF},
      {INF, 0, 15, INF, 20, INF},
      {20, INF, 0, 20, INF, INF},
      {INF, 10, INF, 0, 35, INF},
      {INF, INF, INF, 30, 0, INF},
      {INF, INF, INF, 4, INF, 0}
   };

   int src = 5;
   dijkstra(cost, src);

   return 0;
}
```

Result:

```
Vertex              Distance from Source
0                   49
1                   14
2                   29
3                   4
4                   34
5                   0
```

Lab Program 14

Implement “N-Queens Problem” using Backtracking.

Code:

```
#include <stdio.h>
#include<stdlib.h>
#define MAX 100
int x[MAX];
int place(int k, int i) {
   for (int j = 1; j < k; j++) {
      if (x[j] == i || abs(x[j] - i) == abs(j - k)) {
         return 0;
      } }
   return 1;
}
void printSolution(int n) {
   for (int i = 1; i <= n; i++) {
      printf("%d ", x[i]);
   }
   printf("\n");
}
void NQueens(int k, int n) {
   for (int i = 1; i <= n; i++) {
      if (place(k, i)) {
         x[k] = i;
         if (k == n) {
            printSolution(n);
         } else {
            NQueens(k + 1, n);
         } }}}
int main() {
   int n;
   printf("Enter the number of queens: ");
   scanf("%d", &n);
   NQueens(1, n);
   return 0;
}
```

Result:

```
Enter the number of queens: 4
2 4 1 3
3 1 4 2
```