B.M.S. COLLEGE OF ENGINEERING

Basavanagudi, Bengaluru- 560019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB REPORT

On

Analysis and Design of Algorithms

(23CS4PCADA)

Submitted By:

AGNEYA D A

1BM22CS024

In partial fulfilment of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

2023-24

Faculty-In-Charge

Vikranth B.M.

Assistant Professor

Department of Computer Science and Engineering

B.M.S. COLLEGE OF ENGINEERING
Basavanagudi, Bengaluru- 560019
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Lab work entitled “Analysis and Design of Algorithms (23CS4PCADA)” conducted by **Agneya D A (1BM22CS024),** who is bonafide student at **B.M.S.College of Engineering**. It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** during the academic year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of a Analysis and Design of Algorithms (23CS4PCADA) work prescribed for the said degree.

Mr. VIKRANTH. B. M
Assistant Professor
Dept of CSE
BMSCE, Bengaluru

Dr. Jyoth S Nayak
Prof & Head of Dept of
CSE
BMSCE, Bengaluru

INDEX

Sl. No.	Experiment Title	Page No.
1	Topological ordering of vertices in a given digraph.	1-4
2	Merge Sort technique	5-6
3	Quick Sort technique	7-8
4	Bottom-up Heap Sort technique	9-10
5	Knapsack problem	11-13
6	Floyd’s algorithm	14-15
7	Prim’s algorithm Kruskal’s algorithm	16-19
8	Dijkstra’s algorithm	20-21
9	Fractional Knapsack problem using Greedy technique	22-23
10	N-Queens Problem using Backtracking	24-25

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

```
//C program to implement topological sort using DFS
#include <stdio.h>

int n, a[10][10], top = 0, s[10], res[10];

void dfs_top(int, int[][10]);
void dfs(int, int, int[][10]);

void main()
{
   int i, j;
   printf("Enter number of nodes: ");
   scanf("%d", &n);
   printf("Enter adjacency matrix: ");

   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         scanf("%d", &a[i][j]);
      }
   }

   dfs_top(n, a);
   printf("Solution: ");
   for (i = n - 1; i >= 0; i--)
      printf("%d ", res[i]);
}

void dfs_top(int n, int a[][10])
{
   int i;
   for (i = 0; i < n; i++)
      s[i] = 0;
   for (i = 0; i < n; i++)
   {
      if (s[i] == 0)
         dfs(i, n, a);
   }
}

void dfs(int j, int n, int a[][10])
{
   s[j] = 1;
   int i;
```

```
    for (i = 0; i < n; i++)
    {
        if (a[j][i] == 1 && s[i] == 0)
            dfs(i, n, a);
    }

    res[top++] = j;

}
```

OUTPUT:

```
Enter the no. of nodes6
0 0 1 1 0 0
0 0 0 1 1 0
0 0 0 1 0 1
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 0
Solution: 1 4 0 2 3 5
```

```
//C program to implement topological sort using source removal method
#include <stdio.h>

int n, a[10][10], t[10], indegree[10];
int stack[10], top = -1;

void topSort(int, int[][10]);
void inDeg(int, int[][10]);

void main()
{
    int i, j;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix: ");

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }

    inDeg(n, a);
    topSort(n, a);

    printf("Result: ");
    for (i = 0; i < n; i++)
```

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

void inDeg(int n, int a[][10])
{
    int i, j;
    int sum;
    for (i = 0; i < n; i++)
    {
        sum = 0;
        for (j = 0; j < n; j++)
        {
            sum += a[j][i];
        }
        indegree[i] = sum;
    }
}

void topSort(int n, int a[][10])
{
    int i, j, k = 0, v;
    for (i = 0; i < n; i++)
    {
        if (indegree[i] == 0)
            stack[++top] = i;
    }

    while (top != -1)
    {
        v = stack[top--];
        t[k++] = v;
        for (i = 0; i < n; i++)
        {
            if (a[v][i] != 0)
            {
                indegree[i] -= 1;
                if (indegree[i] == 0)
                {
                    stack[++top] = i;
                }
            }
        }
    }
}
```

OUTPUT:

```
Enter the no. of nodes: 5
0 0 1 0 0
1 0 0 1 0
0 0 0 0 1
0 0 1 0 1
0 0 0 0 0
Solution:1 3 0 2 4
```

2. 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.

```
//C program to implement merge sort
#include <stdio.h>
#include <time.h>

int a[20], n;

void merge(int[], int, int, int);
void mergeSort(int[], int, int);

void main()
{
   clock_t start, end;
   double time_taken;
   int i;
   printf("Enter the value of n: ");
   scanf("%d", &n);
   printf("Enter the array elements: ");
   for (i = 0; i < n; i++)
      scanf("%d", &a[i]);

   start = clock();
   mergeSort(a, 0, n - 1);
   end = clock();

   printf("Sorted array: ");
   for (i = 0; i < n; i++)
   {
      printf("%d ", a[i]);
   }

   time_taken = (double)(end - start) / CLOCKS_PER_SEC;
   printf("\nTime taken = %d", time_taken);
}

void mergeSort(int a[], int low, int high)
{
   if (low < high)
   {
      int mid = (low + high) / 2;
      mergeSort(a, low, mid);
      mergeSort(a, mid + 1, high);
      merge(a, low, mid, high);
   }
}
```

```
void merge(int a[], int low, int mid, int high)
{
   int i = low, k = low, j = mid + 1;
   int c[n];
   while (i <= mid && j <= high)
   {
      if (a[i] < a[j])
      {
         c[k++] = a[i];
         i++;
      }
      else
      {
         c[k++] = a[j];
         j++;
      }
   }

   while (i <= mid)
   {
      c[k++] = a[i];
      i++;
   }

   while (j <= high)
   {
      c[k++] = a[j];
      j++;
   }

   for (i = low; i <= high; i++)
   {
      a[i] = c[i];
   }
}
```

OUTPUT:

```
Enter the value of n :10
Enter the array elements: 8 96 32 75 62 78 63 48 56 100
Sorted array: 8 32 48 56 62 63 75 78 96 100
Time taken = 0
```

3. Sort a given set of N integer elements using Quick Sort technique andcompute its time taken.

```
//C program to implement quick sort
#include <stdio.h>
#include <time.h>

int a[20], n;
void swap(int *a, int *b);
int partition(int[], int, int);
void quickSort(int[], int, int);

void main()
{
   clock_t start, end;

   double time_taken;
   int i;
   printf("Enter the value of n: ");
   scanf("%d", &n);
   printf("Enter the array elements: ");
   for (i = 0; i < n; i++)
      scanf("%d", &a[i]);

   start = clock();
   quickSort(a, 0, n - 1);
   end = clock();

   time_taken = (double)(end - start) / CLOCKS_PER_SEC;

   printf("Sorted array: ");
   for (i = 0; i < n; i++)
   {
      printf("%d ", a[i]);
   }
   printf("\nTime taken = %d", time_taken);
}

int partition(int a[], int low, int high)
{
   int i, j, pivot;
   pivot = a[low];
   i = low;
   j = high + 1;

   while (i <= j)
   {
      do
      {
          i = i + 1;
      } while (a[i] < pivot && i <= high);

      do
      {
          j = j - 1;
      } while (a[j] > pivot && j >= low);

      if (i < j)
          swap(&a[i], &a[j]);
   }
   swap(&a[low], &a[j]);
   return j;
}

void swap(int *a, int *b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}

void quickSort(int a[], int low, int high)
{
   if (low < high)
   {
      int mid = partition(a, low, high);
      quickSort(a, low, mid - 1);
      quickSort(a, mid + 1, high);
   }
}
```

OUTPUT:

```
Enter the value of n: 10
Enter the array elements: 8 96 32 75 62 78 63 48 56 100
Sorted array: 8 32 48 56 62 63 75 78 96 100
Time taken = 0
```

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

```
//C program to implement heapify
#include <stdio.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 l = 2 * i + 1;
   int r = 2 * i + 2;
   if (arr[l] > arr[largest] && l < N)
   {
      largest = l;
   }
   if (arr[r] > arr[largest] && r < N)
   {
      largest = r;
   }
   if (i != largest)
   {
      swap(&arr[i], &arr[largest]);
      heapify(arr, N, largest);
   }
}

void buildHeap(int arr[], int N)
{
   int startIndex = (N / 2) - 1;
   for (int i = startIndex; i >= 0; i--)
   {
      heapify(arr, N, i);
   }
}

void printHeap(int arr[], int N)
{
   printf("The heap is: \n");
   for (int i = 0; i < N; i++)
   {
      printf("%d ", arr[i]);
   }
```

```
    printf("\n");
}

void main()
{
    int N;
    printf("Enter number of elements: ");
    scanf("%d", &N);
    int arr[N];
    printf("Elements of the array: ");
    for (int i = 0; i < N; i++)
        scanf("%d", &arr[i]);
    buildHeap(arr, N);
    printHeap(arr, N);
}OUTPUT:
Enter number of elements: 11
Elements of the array: 1 3 5 4 6 13 10 9 8 15 17
The heap is:
17 15 13 9 6 5 10 4 8 3 1
```

5. Implement 0/1 Knapsack problem using dynamic programming.
//C program to implement knapsack problem in dynamic programming

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 10

int max(int a, int b)
{
   if (a >= b)
      return a;
   else
      return b;
}

void main()
{
   int n, m, v[MAX], w[MAX], val[MAX][MAX];
   int i, j;
   printf("Enter number of items: ");
   scanf("%d", &n);
   printf("Enter knapsack capacity: ");
   scanf("%d", &m);
   bool keep[n + 1][m + 1];
   w[0] = 0;
   v[0] = 0;

   printf("Enter each item's weight followed by value: ");
   for (i = 1; i <= n; i++)
   {
      scanf("%d%d", &w[i], &v[i]);
   }

   for (i = 0; i <= m; i++)
      val[0][i] = 0;
   keep[0][i] = false;

   for (i = 0; i <= n; i++)
   {
      for (j = 0; j <= m; j++)
      {
         if (i == 0 || j == 0)
         {
            val[i][j] = 0;
         }
         else
         {
            if (w[i] > j)
```

```
                {
                    val[i][j] = val[i - 1][j];
                    keep[i][j] = false;
                }
                else
                {
                    bool store = val[i - 1][j - w[i]] + v[i] > val[i - 1][j];
                    val[i][j] = max(val[i - 1][j - w[i]] + v[i], val[i - 1][j]);
                    keep[i][j] = store;
                }
            }
        }
    }
    for (i = 0; i <= n; i++)
    {
        for (j = 0; j <= m; j++)
        {
            printf("%d ", val[i][j]);
        }
        printf("\n");
    }

    j = m;
    int k = 0, ks[n + 1];
    for (i = n; i > 0 && j > 0; i--)
    {
        if (keep[i][j])
        {
            ks[k++] = i;
            j -= w[i];
        }
    }

    printf("Selected items: ");
    for (i = 0; i < k; i++)
    {
        printf("%d ", ks[i]);
    }

    printf("\nMax value: %d", val[n][m]);
}
```

OUTPUT:

```
Enter number of items: 4
Enter knapsack capacity: 5
Enter each item's weight followed by value: 2 12
1 10
3 20
2 15
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
Selected items: 4 2 1
Max value: 37
```

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

```
//C program to implement floyd's algorithm
#include <stdio.h>

int n, a[10][10], D[10][10];

int min(int a, int b)
{
   if (a <= b)
      return a;
   else
   {
      return b;
   }
}

void floyd(int[][10], int);

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

   floyd(a, n);
   printf("The distance matrix is: \n");
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         printf("%d ", D[i][j]);
      }
      printf("\n");
   }
}

void floyd(int a[][10], int n)
{
   int i, j, k;
```

```
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            D[i][j] = a[i][j];
        }
    }

    for (k = 0; k < n; k++)
    {
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
            {
                D[i][j] = min(D[i][j], D[i][k] + D[k][j]);
            }
        }
    }
}
```

OUTPUT:

```
Enter number of vertices: 4
Enter cost adjacency matrix:
0 99 3 99
2 0 99 99
99 6 0 1
7 99 99 0
The distance matrix is:
0 9 3 4
2 0 5 6
8 6 0 1
7 16 10 0
```

7. A. Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.

```
//C program to implement prim's algorithm
#include <stdio.h>

int n, t[10][2], cost[10][10], sum;

void prims(int cost[][10], int n);

void main()
{
   int i, j;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter cost adjacency matrix: ");
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         scanf("%d", &cost[i][j]);
      }
   }

   prims(cost, n);

   printf("Edges of the graph: ");
   for (i = 0; i < n - 1; i++)
   {
      printf("(%d , %d) ", t[i][0], t[i][1]);
   }

   printf("\nMinimal distance: %d", sum);
}

void prims(int cost[][10], int n)
{
   int i, j, k, u, v, min;
   int s[n], p[n], d[n];
   int source = 0;
   for (i = 0; i < n; i++)
   {
      d[i] = cost[source][i];
      s[i] = 0;
      p[i] = source;
   }

   s[source] = 1;
```

```
    k = 0;
    sum = 0;

    for (i = 0; i < n - 1; i++)
    {
        u = -1;
        min = 999;

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

        if (u != -1)
        {
            t[k][0] = u;
            t[k][1] = p[u];
            sum += cost[u][p[u]];
            k++;
            s[u] = 1;

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

```
OUTPUT:
Enter number of vertices: 4
Enter cost adjacency matrix: 0 1 5 2
1 0 99 99
5 99 0 3
2 99 3 0
Edges of the graph: (1 , 0) (3 , 0) (2 , 3)
Minimal distance: 6
```

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

```
//C program to implement Kruskal’s algorithm
#include <stdio.h>

int t[10][2], cost[10][10], n, sum = 0;

void kruskal(int[10][10], int);
int find(int[10], int);
void main()
{
   int i, j;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter cost adjacency matrix: ");
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         scanf("%d", &cost[i][j]);
      }
   }

   kruskal(cost, n);

   printf("Edges of the MST: ");
   for (i = 0; i < n - 1; i++)
   {
      printf("(%d , %d) ", t[i][0], t[i][1]);
   }

   printf("\nMinimal distance: %d", sum);
}

void kruskal(int cost[10][10], int n)
{
   int u, v, k = 0, count, min;
   int parent[10];
   int i, j;
   for (i = 0; i < n; i++)
   {
      parent[i] = i;
   }

   count = 0;
   while (count < n - 1)
   {
      u = -1;
      v = -1;
      min = 99;

      for (i = 0; i < n; i++)
      {
         for (j = 0; j < n; j++)
         {
             if (find(parent, i) != find(parent, j) && cost[i][j] < min)
             {
                 min = cost[i][j];
                 u = i;
                 v = j;
             }
         }
      }

      int root_u = find(parent, u);
      int root_v = find(parent, v);
      if (root_v != root_u)
      {
         parent[root_u] = root_v;
         sum += min;
         t[k][0] = u;
         t[k][1] = v;
         count++;
         k++;
      }
   }
}

int find(int parent[10], int i)
{

   while (parent[i] != i)
   {
      i = parent[i];
   }
   return i;
}
```

OUTPUT:

```
Enter number of vertices: 4
Enter cost adjacency matrix: 0 1 5 2
1 0 99 99
5 99 0 3
2 99 3 0
Edges of the MST: (0 , 1) (0 , 3) (2 , 3)
Minimal distance: 6
```

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

```
// C program to implement Dijkstra's algorithm
#include <stdio.h>
int cost[10][10], t[10][2], n, weight[10];

void djk(int[10][10], int);

void main()
{
   int i, j, s;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter cost adjacency matrix: ");
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         scanf("%d", &cost[i][j]);
      }
   }

   printf("Enter source: ");
   scanf("%d", &s);

   djk(cost, s);

   printf("Path: \n");
   for (i = 1; i < n; i++)
   {
      printf("(%d , %d) weight: %d\n", t[i][0], t[i][1], weight[i]);
   }
}

void djk(int cost[10][10], int s)
{
   int i, j, k, u, v, visited[10], min, d[10], p[10];
   for (i = 0; i < 10; i++)
   {
      d[i] = 999;
      p[i] = -1;
      visited[i] = 0;
   }

   d[s] = 0;
   visited[s] = 1;
   for (i = 0; i < n; i++)
```

```
    {
        min = 99;
        u = 0;
        for (j = 0; j < n; j++)
        {
            if (visited[j] == 0 && d[j] < min)
            {
                min = d[j];
                u = j;
            }
        }
        visited[u] = 1;

        for (v = 0; v < n; v++)
        {
            if (visited[v] == 0 && (d[u] + cost[u][v] < d[v]))
            {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
            }
        }
    }
    for (j = 0; j < n; j++)
    {
        t[j][0] = p[j];
        t[j][1] = j;
        weight[j] = d[j];
    }
}
```

OUTPUT:

```
Enter number of vertices: 4
Enter cost adjacency matrix: 0 1 5 2
1 0 99 99
5 99 0 3
2 99 3 0
Enter source:
0
Path:
(0 , 1) weight: 1
(0 , 2) weight: 5
(0 , 3) weight: 2
```

9. Implement fractional Knapsack problem using Greedy technique.

```
#include <stdio.h>
void knapsack(int n, int p[], int w[], int W)
{
   int used[n];
   for (int i = 0; i < n; ++i)
      used[i] = 0;
   int cur_w = W;
   float tot_v = 0.0;
   int i, maxi;
   while (cur_w > 0)
   {
      maxi = -1;
      for (i = 0; i < n; ++i)
         if ((used[i] == 0) &&
            ((maxi == -1) || ((float)w[i] / p[i] > (float)w[maxi] / p[maxi])))
            maxi = i;
      used[maxi] = 1;
      if (w[maxi] <= cur_w)
      {
         cur_w -= w[maxi];
         tot_v += p[maxi];
         printf("Added object %d (%d, %d) completely in the bag. Space left: %d.\n", maxi + 1,
              w[maxi], p[maxi], cur_w);
      }
      else
      {
         int taken = cur_w;
         cur_w = 0;
         tot_v += (float)taken / p[maxi] * p[maxi];
         printf("Added %d%% (%d, %d) of object %d in the bag.\n", (int)((float)taken / w[maxi] *
100), w[maxi], p[maxi], maxi + 1);
      }
   }
   printf("Filled the bag with objects worth %.2f.\n", tot_v);
}
int main()
{
   int n, W;
   printf("Enter the number of objects: ");
   scanf("%d", &n);
   int p[n], w[n];
   printf("Enter the profits of the objects: ");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &p[i]);
   }
```

```
    printf("Enter the weights of the objects: ");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &w[i]);
    }
    printf("Enter the maximum weight of the bag: ");
    scanf("%d", &W);
    knapsack(n, p, w, W);
    return 0;
}
```

OUTPUT:

```
Enter the number of objects: 7
Enter the profits of the objects: 5 10 15 7 8 9 4
Enter the weights of the objects: 1 3 5 4 1 3 2
Enter the maximum weight of the bag: 15
Added object 4 (4, 7) completely in the bag. Space left: 11.
Added object 7 (2, 4) completely in the bag. Space left: 9.
Added object 3 (5, 15) completely in the bag. Space left: 4.
Added object 6 (3, 9) completely in the bag. Space left: 1.
Added 33% (3, 10) of object 2 in the bag.
Filled the bag with objects worth 36.00.
```

10. Implement “N-Queens Problem” using Backtracking.

```
#include <stdio.h>
#include <stdbool.h>
bool place(int[], int);
void printSolution(int[], int);
void nQueens(int);
int main()
{
   int n;
   printf("Enter the number of queens: ");
   scanf("%d", &n);
   nQueens(n);
   return 0;
}
void nQueens(int n)
{
   int x[10];
   int count = 0;
   int k = 1;
   while (k != 0)
   {
      x[k] = x[k] + 1;
      while (x[k] <= n && !place(x, k))
      {
         x[k] = x[k] + 1;
      }
      if (x[k] <= n)
      {
         if (k == n)
         {
            printSolution(x, n);
            printf("Solution found\n");
            count++;
         }
         else
         {
            k++;
            x[k] = 0;
         }
      }
      else
      {
         k--;
      }
   }
   printf("Total solutions: %d\n", count);
}
```

```
bool place(int x[10], int k)
{
   int i;
   for (i = 1; i < k; i++)
   {
      if ((x[i] == x[k]) || (i - x[i] == k - x[k]) || (i + x[i] == k + x[k]))
      {
         return false;
      }
   }

   return true;
}
void printSolution(int x[10], int n)
{
   int i;
   for (i = 1; i <= n; i++)
   {
      printf("%d ", x[i]);
   }
   printf("\n");
}
```

OUTPUT:

```
Enter the number of queens: 4
2 4 1 3
Solution found
3 1 4 2
Solution found
Total solutions: 2
```