Lab 4 : Dipesh Singh - 190905520

Question 1 :

Write a program for assignment problem by brute-force technique and analyze its time efficiency. Obtain the experimental result of order of growth and plot the result.

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


void swap(int *arr, int i, int j)
{
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
void nextPermute(int *cur, int num);
int findCost(int **matrix, int num, int *arr, int* cnt);
int solve(int **matrix, int num);
void reverse(int *cur, int start, int end);
int findCeil(int *cur, int first, int l, int h);


int main()
{
    int num;
    printf("Enter the number of jobs : ");
    scanf("%d", &num);
    printf("Enter the adjacency : ");
    int **matrix = (int **)calloc(num, sizeof(int *));
    for (int i = 0; i < num; i++)
    {
        matrix[i] = (int *)calloc(num, sizeof(int));
        for (int j = 0; j < num; j++)
        {
```

```
            scanf("%d", &matrix[i][j]);
        }
    }
    for (int i = 0; i < num; i++)
    {
        for (int j = 0; j < num; j++)
        {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    int count = solve(matrix, num);
    printf("The number of operations is : %d\n", count);
    return 0;
}

int findCost(int **matrix, int num, int *arr, int* cnt)
{
    int result = 0;
    for (int i = 0; i < num; i++)
    {
        (*cnt)++;
        result += matrix[i][arr[i]];
    }
    return result;
}

int fact(int num)
{
    int result = 1;
    for (int i = 1; i <= num; i++)
    {
```

```
        result *= i;
    }
    return result;
}

int solve(int **matrix, int num)
{
    int *cur = (int *)calloc(num, sizeof(int));
    int *best = (int *)calloc(num, sizeof(int));
    for (int i = 0; i < num; i++)
    {
        cur[i] = i;
    }
    int loop = fact(num);
    int min = __INT_MAX__;
    int temp, cnt = 0;
    printf("%d\n", loop);
    while (loop--)
    {
        temp = findCost(matrix, num, cur, &cnt);
        printf("The cost is : %d\n", temp);
        if (temp < min)
        {
            min = temp;
            for (int i = 0; i < num; i++)
            {
                best[i] = cur[i];
            }
        }
        nextPermute(cur, num);
    }
```

```
    printf("Minimum cost is : %d\nAnd the jobs assigned to person 0 to
 %d : ", min, num);
    for (int i = 0; i < num; i++)
    {
        printf("%d ", best[i] + 1);
    }
    printf("\n");
    return cnt;
}


void nextPermute(int *cur, int num)
{
    int i;
    for (i = num - 2; i >= 0; --i)
        if (cur[i] < cur[i + 1])
            break;
    int ceilIndex = findCeil(cur, cur[i], i + 1, num - 1);
    swap(cur, i, ceilIndex);
    reverse(cur, i + 1, num - 1);
}
void reverse(int *cur, int start, int end)
{
    while (start < end)
    {
        swap(cur, start, end);
        start++;
        end--;
    }
}
int findCeil(int *cur, int first, int l, int h)
{
    int ceilIndex = l;
```

```
    for (int i = l + 1; i <= h; i++)
        if (cur[i] > first && cur[i] < cur[ceilIndex])
            ceilIndex = i;

    return ceilIndex;
}
```

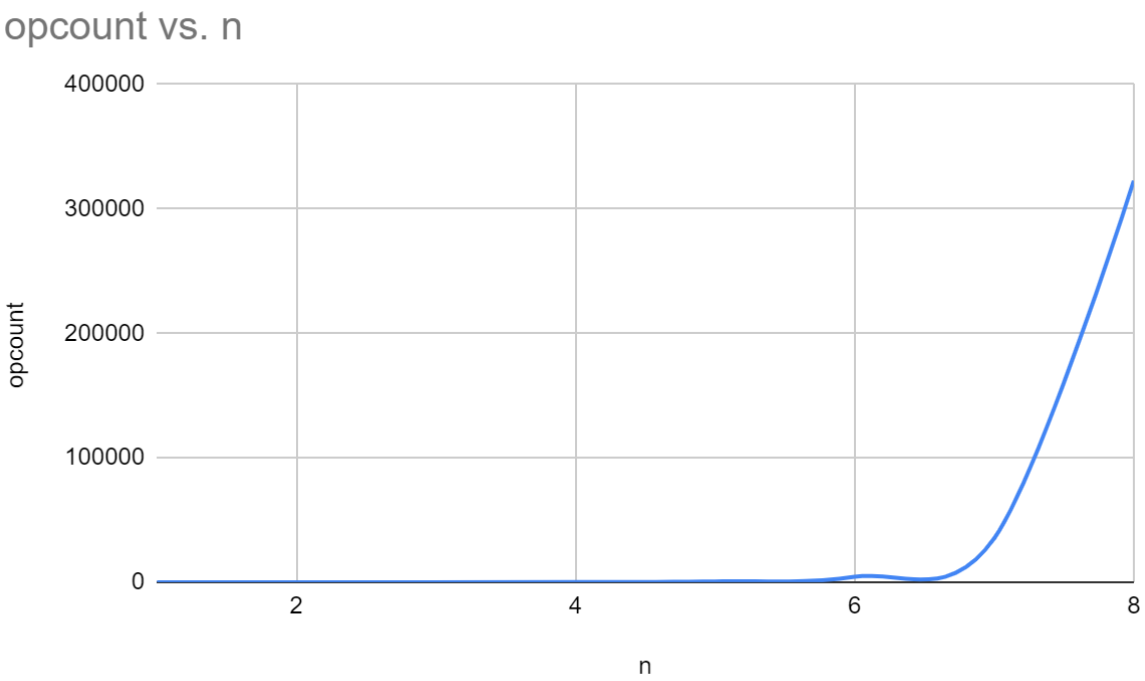
```
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab$ cd Lab\ 4
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 4$ make assignment
cc     assignment.c   -o assignment
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 4$ ./assignment
Enter the number of jobs : 4
Enter the adjacency : 10 3 8 9 7 5 4 8 6 9 2 9 8 7 10 5
10 3 8 9
7 5 4 8
6 9 2 9
8 7 10 5
The cost is : 22
The cost is : 34
The cost is : 28
The cost is : 30
The cost is : 37
The cost is : 27
The cost is : 17
The cost is : 29
The cost is : 18
The cost is : 24
The cost is : 27
The cost is : 21
The cost is : 29
The cost is : 31
The cost is : 24
The cost is : 30
The cost is : 29
The cost is : 33
The cost is : 35
The cost is : 25
The cost is : 30
The cost is : 24
The cost is : 26
The cost is : 30
Minimum cost is : 17
And the jobs assigned to person 0 to 4 : 2 1 3 4
The number of operations is : 96
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 4$
```

The basic operation of this algorithm is addition. When we find a permutation, we use the findCost function to find the cost pertaining to that particular permutation. There are n jobs hence the time complexity for addition is O(n).

Since we have n! permutations in all, the time complexity of going through all the permutations O(n!).

Hence the total time complexity of assignment problem is O(n*n!).

n	opcount
1	1
2	4
3	18
4	96
5	600
6	4320
7	35280
8	322560



Question 2 :

Write a program for depth-first search of a graph. Identify the push and pop order of vertices.

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


int isEmpty(int top)
{
    if (top == -1)
    {
        return 1;
    }
    return 0;
}


int isFull(int top, int capacity)
{
    if (top == capacity - 1)
    {
        return 1;
    }
    return 0;
}


void push(int **Stack, int *top, int *capacity, int key)
{
    if (isFull(*top, *capacity))
    {
        *Stack = (int *)realloc(*Stack, sizeof(int) * (*capacity) * 2)
;
        *capacity *= 2;
    }
```

```
    (*top)++;
    (*Stack)[*top] = key;
}

int pop(int **Stack, int *top)
{
    int temp = (*Stack)[*top];
    (*top)--;
    return temp;
}

void display(int *Stack, int top)
{
    if (isEmpty(top))
    {
    }
    else
    {
        printf("stack : ");
        int i;
        for (i = 0; i <= top; i++)
        {
            printf("%d ", *(Stack + i));
        }
        printf("\n");
    }
}

void insertEdgeM(int **matrix, int first, int second)
{
    matrix[first][second] = 1;
    matrix[second][first] = 1;
```

```
}

void displayMatrix(int **matrix, int n)
{
    for (int i = -1; i < n; i++)
    {
        if (i != -1)
            printf("%d -> ", i);
        for (int j = 0; j < n; j++)
        {
            if (i == -1)
            {
                if (j == 0)
                {
                    printf("\t");
                }
                printf("%d\t", j);
                continue;
            }
            if (j == 0)
            {
                printf("\t");
            }
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
}

void dfs(int **matrix, int num, int **Stack, int *top, int *capacity)
{
    int *visited = (int *)calloc(num, sizeof(int));
```

```
for (int i = 0; i < num; i++)
{
    visited[i] = 0;
}
char result[100];
int resultIndex = 0;
char popped[100];
int poppedIndex = 0;
push(Stack, top, capacity, 0);
printf("pushed : %d\n", 0);
char p = (char)('0' + 0);
result[resultIndex++] = p;
display(*Stack, *top);
visited[0] = 1;
int cur = *Stack[*top];
int flag, ele;
while (1)
{
    if(!(isEmpty(*top))){
        flag = 0;
        for (int i = 0; i < num; i++)
        {
            if (visited[i] == 0 && matrix[cur][i] == 1)
            {
                visited[i] = 1;
                printf("pushed : %d\n", i);
                p = (char)('0' + i);
                result[resultIndex++] = p;
                push(Stack, top, capacity, i);
                display(*Stack, *top);
                flag = 1;
                break;
```

```
        }
    }
    if (flag == 0)
    {
        ele = pop(Stack, top);
        p = (char)('0' + ele);
        popped[poppedIndex++] = p;
        printf("popped : %d\n", ele);
        display(*Stack, *top);
    }
    cur = (*Stack)[*top];
}
else{
    flag = 1;
    for (int i = 0; i < num && flag; i++){
        if(visited[i]==0){
            visited[i] = 1;
            printf("pushed : %d\n", i);
            p = (char)('0' + i);
            result[resultIndex++] = p;
            push(Stack, top, capacity, i);
            display(*Stack, *top);
            flag = 0;
            cur = (*Stack)[*top];
            break;
        }
    }
    if(flag == 1){
        break;
    }
}
```

```
    }
    while (!(isEmpty(*top)))
    {
        int rem = pop(Stack, top);
        p = (char)('0' + rem);
        popped[poppedIndex++] = p;
        printf("popped : %d\n", rem);
        display(*Stack, *top);
    }
    printf("The dfs is : ");
    for (int i = 0; i < resultIndex; i++)
    {
        printf("%c ", result[i]);
    }
    printf("\nThe traversal stack is : ");
    for (int i = 0; i < poppedIndex; i++)
    {
        printf("%c ", popped[i]);
    }
    printf("\n");
}

int main()
{
    int num = 9;
    int **matrix = (int **)calloc(num, sizeof(int *));
    for (int i = 0; i < num; i++)
    {
        matrix[i] = (int *)calloc(num, sizeof(int));
        for (int j = 0; j < num; j++)
        {
            matrix[i][j] = 0;
```

```
        }
    }
    insertEdgeM(matrix, 0, 1);
    insertEdgeM(matrix, 2, 3);
    insertEdgeM(matrix, 4, 2);
    insertEdgeM(matrix, 2, 5);
    insertEdgeM(matrix, 6, 5);
    insertEdgeM(matrix, 4, 7);
    insertEdgeM(matrix, 0, 8);
    insertEdgeM(matrix, 2, 8);
    insertEdgeM(matrix, 6, 8);
    insertEdgeM(matrix, 6, 7);
    displayMatrix(matrix, num);
    int top = -1, capacity = 2;
    int *Stack = (int *)calloc(capacity, sizeof(int));
    dfs(matrix, num, &Stack, &top, &capacity);
    return 0;
}
```

```
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 4$ ./dfs
        0       1       2       3       4       5       6       7       8
0 ->    0       1       0       0       0       0       0       0       1
1 ->    1       0       0       0       0       0       0       0       0
2 ->    0       0       0       1       1       1       0       0       1
3 ->    0       0       1       0       0       0       0       0       0
4 ->    0       0       1       0       0       0       0       1       0
5 ->    0       0       1       0       0       0       1       0       0
6 ->    0       0       0       0       0       1       0       1       1
7 ->    0       0       0       0       1       0       1       0       0
8 ->    1       0       1       0       0       0       1       0       0
pushed : 0
stack : 0
pushed : 1
stack : 0 1
popped : 1
stack : 0
pushed : 8
stack : 0 8
pushed : 2
stack : 0 8 2
pushed : 3
stack : 0 8 2 3
popped : 3
stack : 0 8 2
pushed : 4
stack : 0 8 2 4
pushed : 7
stack : 0 8 2 4 7
pushed : 6
stack : 0 8 2 4 7 6
pushed : 5
stack : 0 8 2 4 7 6 5
popped : 5
stack : 0 8 2 4 7 6
popped : 6
stack : 0 8 2 4 7
popped : 7
stack : 0 8 2 4
popped : 4
stack : 0 8 2
popped : 2
stack : 0 8
popped : 8
stack : 0
popped : 0
The dfs is : 0 1 8 2 3 4 7 6 5
The traversal stack is : 1 3 5 6 7 4 2 8 0
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 4$
```

Since the graph here is represented using an adjacency matrix, the time complexity is $O(|V^2|)$, since for each vertex v, we iterate over v vertices again to check if it is adjacent or not.

Question 3 :

Write a program for breadth-first search of a graph.

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

int isEmpty(int front, int rear)
{
    if (front == rear)
    {
        return 1;
    }
    return 0;
}

int isFull(int front, int rear, int capacity)
{
    if ((rear + 1) % capacity == front)
    {
        return 1;
    }
    return 0;
}

void display(int *Queue, int front, int rear, int capacity)
{
    if (isEmpty(front, rear))
    {
    }
    else
```

```
    {
        printf("queue : ");
        for (int i = (front + 1) % capacity; i != (rear + 1) % capacit
y; i = (i + 1) % capacity)
        {
            printf("%d ", Queue[i]);
        }
        printf("\n");
    }
}


void push(int **Queue, int *front, int *rear, int *capacity, int key)
{
    if (isFull(*front, *rear, *capacity))
    {
        int *newQueue = (int *)calloc((*capacity) * 2, sizeof(int));
        int i, j;
        for (i = 1, j = (*front + 1) % (*capacity); j != (*rear + 1) %
 (*capacity); j = (j + 1) % (*capacity), i++)
        {
            newQueue[i] = (*Queue)[j];
        }
        int *temp = *Queue;
        *Queue = newQueue;
        *capacity = *capacity * 2;
        *front = 0;
        *rear = --i;
        free(temp);
    }
    *rear = (*rear + 1) % (*capacity);
    (*Queue)[*rear] = key;
}
```

```
int pop(int **Queue, int *front, int *rear, int *capacity)
{
    int temp = (*Queue)[(*front + 1) % (*capacity)];
    *front = (*front + 1) % (*capacity);
    return temp;
}
void insertEdgeM(int **matrix, int first, int second)
{
    matrix[first][second] = 1;
    matrix[second][first] = 1;
}

void displayMatrix(int **matrix, int n)
{
    for (int i = -1; i < n; i++)
    {
        if (i != -1)
            printf("%d -> ", i);
        for (int j = 0; j < n; j++)
        {
            if (i == -1)
            {
                if (j == 0)
                {
                    printf("\t");
                }
                printf("%d\t", j);
                continue;
            }
            if (j == 0)
            {
                printf("\t");
```

```
            }
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
}


void bfs(int **matrix, int num, int **Queue, int *front, int *rear, in
t *capacity)
{
    int *visited = (int *)calloc(num, sizeof(int));
    for (int i = 0; i < num; i++)
    {
        visited[i] = 0;
    }
    char result[100];
    int resultIndex = 0;
    char popped[100];
    int poppedIndex = 0;
    push(Queue, front, rear, capacity, 0);
    printf("pushed : %d\n", 0);
    char p = (char)('0' + 0);
    result[resultIndex++] = p;
    display(*Queue, *front, *rear, *capacity);
    visited[0] = 1;
    int cur = *Queue[*front];
    int flag, ele;
    while (1)
    {
        if (!(isEmpty(*front, *rear)))
        {
            for (int i = 0; i < num; i++)
```

```
    {
        if (visited[i] == 0 && matrix[cur][i] == 1)
        {
            visited[i] = 1;
            char p = (char)('0' + i);
            result[resultIndex++] = p;
            push(Queue, front, rear, capacity, i);
            printf("pushed : %d\n", i);
            display(*Queue, *front, *rear, *capacity);
        }
    }
    ele = pop(Queue, front, rear, capacity);
    p = (char)('0' + ele);
    popped[poppedIndex++] = p;
    printf("popped : %d\n", ele);
    display(*Queue, *front, *rear, *capacity);
    cur = (*Queue)[(*front + 1) % (*capacity)];
}
else
{
    flag = 1;
    for (int i = 0; i < num && flag; i++)
    {
        if (visited[i] == 0)
        {
            visited[i] = 1;
            printf("pushed : %d\n", i);
            p = (char)('0' + i);
            result[resultIndex++] = p;
            push(Queue, front, rear, capacity, i);
            display(*Queue, *front, *rear, *capacity);
            cur = (*Queue)[(*front + 1) % (*capacity)];
```

```
                    flag = 0;
                    break;
                }
            }
            if (flag == 1)
            {
                break;
            }
        }
    }
    while (!(isEmpty(*front, *rear)))
    {
        ele = pop(Queue, front, rear, capacity);
        p = (char)('0' + ele);
        popped[poppedIndex++] = p;
        printf("popped : %d\n", ele);
        display(*Queue, *front, *rear, *capacity);
    }
    printf("The bfs is : ");
    for (int i = 0; i < resultIndex; i++)
    {
        printf("%c ", result[i]);
    }
    printf("\nThe traversal stack is : ");
    for (int i = 0; i < poppedIndex; i++)
    {
        printf("%c ", popped[i]);
    }
    printf("\n");
}

int main()
```

```
{
    int num = 9;
    int **matrix = (int **)calloc(num, sizeof(int *));
    for (int i = 0; i < num; i++)
    {
        matrix[i] = (int *)calloc(num, sizeof(int));
        for (int j = 0; j < num; j++)
        {
            matrix[i][j] = 0;
        }
    }
    insertEdgeM(matrix, 0, 1);
    insertEdgeM(matrix, 2, 3);
    insertEdgeM(matrix, 4, 2);
    insertEdgeM(matrix, 2, 5);
    insertEdgeM(matrix, 6, 5);
    insertEdgeM(matrix, 4, 7);
    insertEdgeM(matrix, 0, 8);
    insertEdgeM(matrix, 2, 8);
    insertEdgeM(matrix, 6, 8);
    insertEdgeM(matrix, 6, 7);
    displayMatrix(matrix, num);
    int front = 0, rear = 0, capacity = 2;
    int *Queue = (int *)calloc(capacity, sizeof(int));
    bfs(matrix, num, &Queue, &front, &rear, &capacity);
    return 0;
}
```

```
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 4$ ./bfs
        0       1       2       3       4       5       6       7       8
0 ->    0       1       0       0       0       0       0       0       1
1 ->    1       0       0       0       0       0       0       0       0
2 ->    0       0       0       1       1       1       0       0       1
3 ->    0       0       1       0       0       0       0       0       0
4 ->    0       0       1       0       0       0       0       1       0
5 ->    0       0       1       0       0       0       1       0       0
6 ->    0       0       0       0       0       1       0       1       1
7 ->    0       0       0       0       1       0       1       0       0
8 ->    1       0       1       0       0       0       1       0       0
pushed : 0
queue : 0
pushed : 1
queue : 0 1
pushed : 8
queue : 0 1 8
popped : 0
queue : 1 8
popped : 1
queue : 8
pushed : 2
queue : 8 2
pushed : 6
queue : 8 2 6
popped : 8
queue : 2 6
pushed : 3
queue : 2 6 3
pushed : 4
queue : 2 6 3 4
pushed : 5
queue : 2 6 3 4 5
popped : 2
queue : 6 3 4 5
pushed : 7
queue : 6 3 4 5 7
popped : 6
queue : 3 4 5 7
popped : 3
queue : 4 5 7
popped : 4
queue : 5 7
popped : 5
queue : 7
popped : 7
The bfs is : 0 1 8 2 6 3 4 5 7
The traversal stack is : 0 1 8 2 6 3 4 5 7
dops@LAPTOP-LDOMDPE4:/mnt/d/Google Drive/Work/Study Material/2nd Year/4th Semester/DAA/DAA/Lab/Lab 4$
```

Since the graph here is represented using an adjacency matrix, the time complexity is $O(|V^2|)$, since for each vertex v, we iterate over v vertices again to check if it is adjacent or not.