

Week – 1: Write a C Program to implement an AVL tree for a given set of elements which are stored in a file and perform insert and delete operations on the constructed tree. Write contents of tree into a new file using in-order.

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

```
typedef struct Node {  
    int key;  
    struct Node* left;  
    struct Node* right;  
    int height;  
} Node;
```

```
Node* createNode(int key) {  
    Node* node = (Node*)malloc(sizeof(Node));  
    node->key = key;  
    node->left = node->right = NULL;  
    node->height = 1;  
    return node;  
}
```

```
int height(Node* node) {  
    if (node == NULL)  
        return 0;  
    return node->height;  
}
```

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

```
Node* rightRotate(Node* y) {  
    Node* x = y->left;  
    Node* T2 = x->right;  
    x->right = y;  
    y->left = T2;  
    y->height = max(height(y->left), height(y->right)) + 1;  
    x->height = max(height(x->left), height(x->right)) + 1;  
    return x;  
}
```

```
Node* leftRotate(Node* x) {  
    Node* y = x->right;  
    Node* T2 = y->left;  
    y->left = x;  
    x->right = T2;  
    x->height = max(height(x->left), height(x->right)) + 1;  
    y->height = max(height(y->left), height(y->right)) + 1;  
    return y;  
}
```

```
int getBalance(Node* node) {
    if (node == NULL)
        return 0;
    return height(node->left) - height(node->right);
}

Node* insertNode(Node* node, int key) {
    if (node == NULL)
        return createNode(key);
    if (key < node->key)
        node->left = insertNode(node->left, key);
    else if (key > node->key)
        node->right = insertNode(node->right, key);
    else // Equal keys are not allowed in AVL tree
        return node;
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}

Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left != NULL)
        current = current->left;
    return current;
}

Node* deleteNode(Node* root, int key) {
    if (root == NULL)
        return root;
    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    else {
        if ((root->left == NULL) || (root->right == NULL)) {
```

```

Node* temp = root->left ? root->left : root->right;

if (temp == NULL) {
    temp = root;
    root = NULL;
} else
    *root = *temp;
free(temp);
} else {
    Node* temp = minValueNode(root->right);
    root->key = temp->key;
    root->right = deleteNode(root->right, temp->key);
}
}
if (root == NULL)
    return root;
root->height = 1 + max(height(root->left), height(root->right));
int balance = getBalance(root);
if (balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(root);
if (balance > 1 && getBalance(root->left) < 0) {
    root->left = leftRotate(root->left);
    return rightRotate(root);
}
if (balance < -1 && getBalance(root->right) <= 0)
    return leftRotate(root);
if (balance < -1 && getBalance(root->right) > 0) {
    root->right = rightRotate(root->right);
    return leftRotate(root);
}
return root;
}

void inOrderTraversal(Node* root, FILE* file) {
    if (root != NULL) {
        inOrderTraversal(root->left, file);
        fprintf(file, "%d\n", root->key);
        inOrderTraversal(root->right, file);
    }
}

int main() {
    Node* root = NULL;
    FILE* inputFile;
    inputFile = fopen("ins.txt", "r");
    if (inputFile == NULL) {
        perror("Unable to open input file");
        return 1;
    }

```

```
}
int key;
while (fscanf(inputFile, "%d", &key) != EOF) {
    root = insertNode(root, key);
}
fclose(inputFile);
inputFile = fopen("del.txt", "r");
if (inputFile == NULL) {
    perror("Unable to open input file");
    return 1;
}
while (fscanf(inputFile, "%d", &key) != EOF) {
    root = deleteNode(root, key);
}
fclose(inputFile);

FILE* outputFile = fopen("output.txt", "w");
if (outputFile == NULL) {
    perror("Unable to open output file");
    return 1;
}
inOrderTraversal(root, outputFile);
fclose(outputFile);
printf("In-order traversal written to output.txt\n");
return 0;
}
```

Sample Input and Output:

// Create a ins.txt file and add the below elements

410, 913, 655, 946, 320, 702, 874, 223, 535, 275, 628, 744, 749, 343, 136, 803, 674, 815, 610, 621, 241, 768, 687, 332, 594, 679, 462, 563, 728, 537, 750, 281, 447, 328, 821, 734, 778, 147, 314, 481, 205, 333, 841, 939, 619, 908, 726, 993, 507, 608, 49, 971, 963, 17, 92, 270, 230, 21, 430, 140, 250, 455, 150, 860, 640, 129, 570, 620, 956, 990, 63, 611, 192, 896, 519, 254, 654, 645, 93, 737, 116, 590, 36, 572, 10, 873, 937, 386, 425, 852, 323, 484, 671, 480, 407, 573, 588, 609, 336, 794

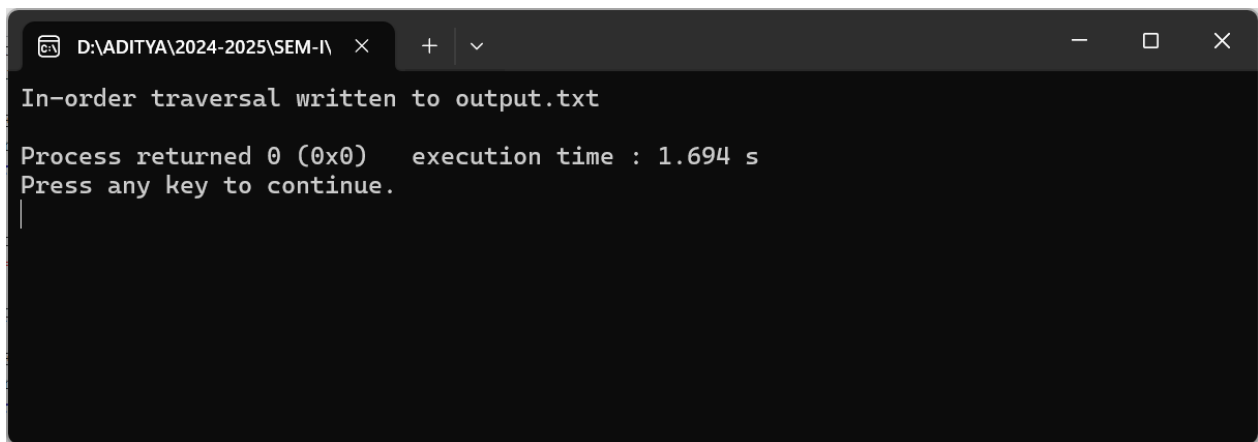
// Create a del.txt file and add the below elements

702, 874, 223, 535, 275

// Result stored in output.txt file and it having the following data

10, 17, 21, 36, 49, 63, 92, 93, 116, 129, 136, 140, 147, 150, 192, 205, 230, 241, 250, 254, 270, 281, 314, 320, 323, 328, 332, 333, 336, 343, 386, 407, 410, 425, 430, 447, 455, 462, 480, 481, 484, 507, 519, 537, 563, 570, 572, 573, 588, 590, 594, 608, 609, 610, 611, 619, 620, 621, 628, 640, 645, 654, 655, 671, 674, 679, 687, 726, 728, 734, 737, 744, 749, 750, 768, 778, 794, 803, 815, 821, 841, 852, 860, 873, 896, 908, 913, 937, 939, 946, 956, 963, 971, 990, 993

Actual Input and Output:



```
D:\ADITYA\2024-2025\SEM-I\ >
In-order traversal written to output.txt
Process returned 0 (0x0)   execution time : 1.694 s
Press any key to continue.
```

Week-2: Write a C Program to implement a B-Tree an order of 5 with a set of 100 random elements stored in array and perform searching and display operations.

```
#include <stdio.h>
#define M 5
typedef struct BTreeNode {
    int *keys;
    int t;
    struct BTreeNode **C;
    int n;
    int leaf;
} BTreeNode;
typedef struct BTree {
    BTreeNode *root;
    int t;
} BTree;

BTreeNode* createBTreeNode(int _t, int _leaf) {
    BTreeNode *newNode = (BTreeNode *)malloc(sizeof(BTreeNode));
    newNode->t = _t;
    newNode->leaf = _leaf;
    newNode->keys = (int *)malloc((2 * _t - 1) * sizeof(int));
    newNode->C = (BTreeNode **)malloc(2 * _t * sizeof(BTreeNode *));
    newNode->n = 0;
    return newNode;
}

void traverse(BTreeNode *node) {
    int i;
    for (i = 0; i < node->n; i++) {
        if (node->leaf == 0) traverse(node->C[i]);
        printf(" %d", node->keys[i]);
    }
    if (node->leaf == 0) traverse(node->C[i]);
}

BTreeNode* search(BTreeNode *node, int k) {
    int i = 0;
    while (i < node->n && k > node->keys[i]) i++;
    if (i < node->n && node->keys[i] == k) return node;
    if (node->leaf == 1) return NULL;
    return search(node->C[i], k);
}

void insert(BTree *tree, int k) {
    if (tree->root == NULL) {
        tree->root = createBTreeNode(tree->t, 1);
        tree->root->keys[0] = k;
        tree->root->n = 1;
    } else {
        if (tree->root->n == 2 * tree->t - 1) {
```

```
BTreeNode *s = createBTreeNode(tree->t, 0);
s->C[0] = tree->root;
splitChild(s, 0, tree->root);
int i = 0;
if (s->keys[0] < k) i++;
insertNonFull(s->C[i], k);
tree->root = s;
} else {
    insertNonFull(tree->root, k);
}
}
}

void insertNonFull(BTreeNode *node, int k) {
    int i = node->n - 1;
    if (node->leaf == 1) {
        while (i >= 0 && node->keys[i] > k) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = k;
        node->n = node->n + 1;
    } else {
        while (i >= 0 && node->keys[i] > k) i--;
        if (node->C[i + 1]->n == 2 * node->t - 1) {
            splitChild(node, i + 1, node->C[i + 1]);
            if (node->keys[i + 1] < k) i++;
        }
        insertNonFull(node->C[i + 1], k);
    }
}

void splitChild(BTreeNode *parent, int i, BTreeNode *y) {
    BTreeNode *z = createBTreeNode(y->t, y->leaf);
    z->n = y->t - 1;
    for (int j = 0; j < y->t - 1; j++) z->keys[j] = y->keys[j + y->t];
    if (y->leaf == 0) {
        for (int j = 0; j < y->t; j++) z->C[j] = y->C[j + y->t];
    }
    y->n = y->t - 1;
    for (int j = parent->n; j >= i + 1; j--) parent->C[j + 1] = parent->C[j];
    parent->C[i + 1] = z;
    for (int j = parent->n - 1; j >= i; j--) parent->keys[j + 1] = parent->keys[j];
    parent->keys[i] = y->keys[y->t - 1];
    parent->n = parent->n + 1;
}

int main() {
    BTree *t = (BTree *)malloc(sizeof(BTree));
    t->root = NULL;
```

Date:

```

t->t = 3;
int arr[100];
printf("Inserting 100 random elements into the B-Tree:\n");
for (int i = 0; i < 100; i++) {
    arr[i] = rand() % 5000;
    insert(t, arr[i]);
    printf("%d ", arr[i]);
}
printf("\n\nB-Tree after insertion:\n");
traverse(t->root);
int key;
printf("\n\nEnter a key to search in the B-Tree: ");
scanf("%d", &key);
if (search(t->root, key) != NULL)
    printf("\nKey %d found in the B-Tree.\n", key);
else
    printf("\nKey %d not found in the B-Tree.\n", key);
return 0;
}

```

Sample Input and Output:

Inserting 100 random elements into the B-Tree:

41 3467 1334 1500 4169 724 1478 4358 1962 4464 705 3145 3281 1827 4961 491 2995 1942 4827
 436 2391 4604 3902 153 292 2382 2421 3716 4718 4895 447 1726 4771 1538 1869 4912 667 1299
 2035 4894 3703 3811 1322 333 2673 4664 141 2711 3253 1868 547 2644 2662 2757 37 2859 3723
 4741 2529 778 2316 3035 2190 1842 288 106 4040 3942 4264 2648 2446 3805 890 1729 4370 350 6
 1101 4393 3548 4629 2623 4084 4954 3756 1840 4966 2376 3931 1308 1944 2439 4626 1323 537
 1538 1118 2082 2929 1541

B-Tree after insertion:

6 37 41 106 141 153 288 292 333 350 436 447 491 537 547 667 705 724 778 890 1101 1118 1299
 1308 1322 1323 1334 1478 1500 1538 1538 1541 1726 1729 1827 1840 1842 1868 1869 1942 1944
 1962 2035 2082 2190 2316 2376 2382 2391 2421 2439 2446 2529 2623 2644 2648 2662 2673 2711
 2757 2859 2929 2995 3035 3145 3253 3281 3467 3548 3703 3716 3723 3756 3805 3811 3902 3931
 3942 4040 4084 4169 4264 4358 4370 4393 4464 4604 4626 4629 4664 4718 4741 4771 4827 4894
 4895 4912 4954 4961 4966

Actual Input and Output:

```
D:\ADITYA\2024-2025\SEM-I\ × + ▾
Inserting 100 random elements into the B-Tree:
41 3467 1334 1500 4169 724 1478 4358 1962 4464 705 3145 3281 1827 4961 491 2995 1942 4827 436 239
1 4604 3902 153 292 2382 2421 3716 4718 4895 447 1726 4771 1538 1869 4912 667 1299 2035 4894 3703
3811 1322 333 2673 4664 141 2711 3253 1868 547 2644 2662 2757 37 2859 3723 4741 2529 778 2316 30
35 2190 1842 288 106 4040 3942 4264 2648 2446 3805 890 1729 4370 350 6 1101 4393 3548 4629 2623 4
084 4954 3756 1840 4966 2376 3931 1308 1944 2439 4626 1323 537 1538 1118 2082 2929 1541

B-Tree after insertion:
6 37 41 106 141 153 288 292 333 350 436 447 491 537 547 667 705 724 778 890 1101 1118 1299 1308
1322 1323 1334 1478 1500 1538 1538 1541 1726 1729 1827 1840 1842 1868 1869 1942 1944 1962 2035 20
82 2190 2316 2376 2382 2391 2421 2439 2446 2529 2623 2644 2648 2662 2673 2711 2757 2859 2929 2995
3035 3145 3253 3281 3467 3548 3703 3716 3723 3756 3805 3811 3902 3931 3942 4040 4084 4169 4264 4
358 4370 4393 4464 4604 4626 4629 4664 4718 4741 4771 4827 4894 4895 4912 4954 4961 4966

Enter a key to search in the B-Tree: |
```



Week-3: Write a C Program to construct Min and Max Heap using arrays, delete any element and display the content of the Heap.

// Min-Heap

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

```
#define MAX 100
```

```
int heap[MAX];
```

```
int size = 0;
```

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

```
    int temp = *a;
```

```
    *a = *b;
```

```
    *b = temp;
```

```
}
```

```
void minHeapify(int i) {
```

```
    int smallest = i;
```

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

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

```
    if (left < size && heap[left] < heap[smallest])
```

```
        smallest = left;
```

```
    if (right < size && heap[right] < heap[smallest])
```

```
        smallest = right;
```

```
    if (smallest != i) {
```

```
        swap(&heap[i], &heap[smallest]);
```

```
        minHeapify(smallest);
```

```
    }
```

```
}
```

```
void insertMinHeap(int key) {
```

```
    heap[size] = key;
```

```
    int i = size;
```

```
    size++;
```

```
    while (i != 0 && heap[(i - 1) / 2] > heap[i]) {
```

```
        swap(&heap[i], &heap[(i - 1) / 2]);
```

```
        i = (i - 1) / 2;
```

```
    }
```

```
    printf("After insertion MinHeap elements are:");
```

```
    displayHeap();
```

```
}
```

```
int extractMin() {
```

```
    if (size <= 0) {
```

```
    printf("Heap is empty!\n");
    return -1;
}
if (size == 1) {
    size--;
    return heap[0];
}
int root = heap[0];
heap[0] = heap[size - 1];
size--;
minHeapify(0);

return root;
}

void heapSort() {
    int originalSize = size;
    for (int i = 0; i < originalSize; i++) {
        printf("%d ", extractMin());
    }
    printf("\n");
    size = originalSize;
}

void displayHeap() {
    for (int i = 0; i < size; i++) {
        printf("%d ", heap[i]);
    }
    printf("\n");
}

int main() {
    int choice, element;
    while (1) {
        printf("1. Insert in Min Heap\n");
        printf("2. Delete from Min Heap\n");
        printf("3. Display Heap\n");
        printf("4. Perform Heap Sort\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter element to insert in Min Heap: ");
                scanf("%d", &element);
                insertMinHeap(element);
                break;
            case 2:
                extractMin();
                break;
            case 3:
                displayHeap();
                break;
            case 4:
                heapSort();
                break;
            case 5:
                return 0;
        }
    }
}
```

```
        break;
    case 3:
        printf("Heap elements: ");
        displayHeap();
        break;
    case 4:
        printf("Sorted elements: ");
        heapSort();
        break;
    case 5:
        exit(0);
    default:
        printf("Invalid choice! Please try again.\n");
    }
}
return 0;
}
```

Sample Input and Output:

1. Insert in Min Heap
2. Delete from Min Heap
3. Display Heap
4. Perform Heap Sort
5. Exit

Enter your choice: 1

Enter number of elements to be insert in Min Heap: 3

Enter elements to insert in Min Heap: 10 15 3

After insertion MinHeap elements are:10

After insertion MinHeap elements are:10 15

After insertion MinHeap elements are:3 15 10

1. Insert in Min Heap
2. Delete from Min Heap
3. Display Heap
4. Perform Heap Sort
5. Exit

Enter your choice: 3

Heap elements: 3 15 10

1. Insert in Min Heap
2. Delete from Min Heap
3. Display Heap
4. Perform Heap Sort
5. Exit

Enter your choice: 4

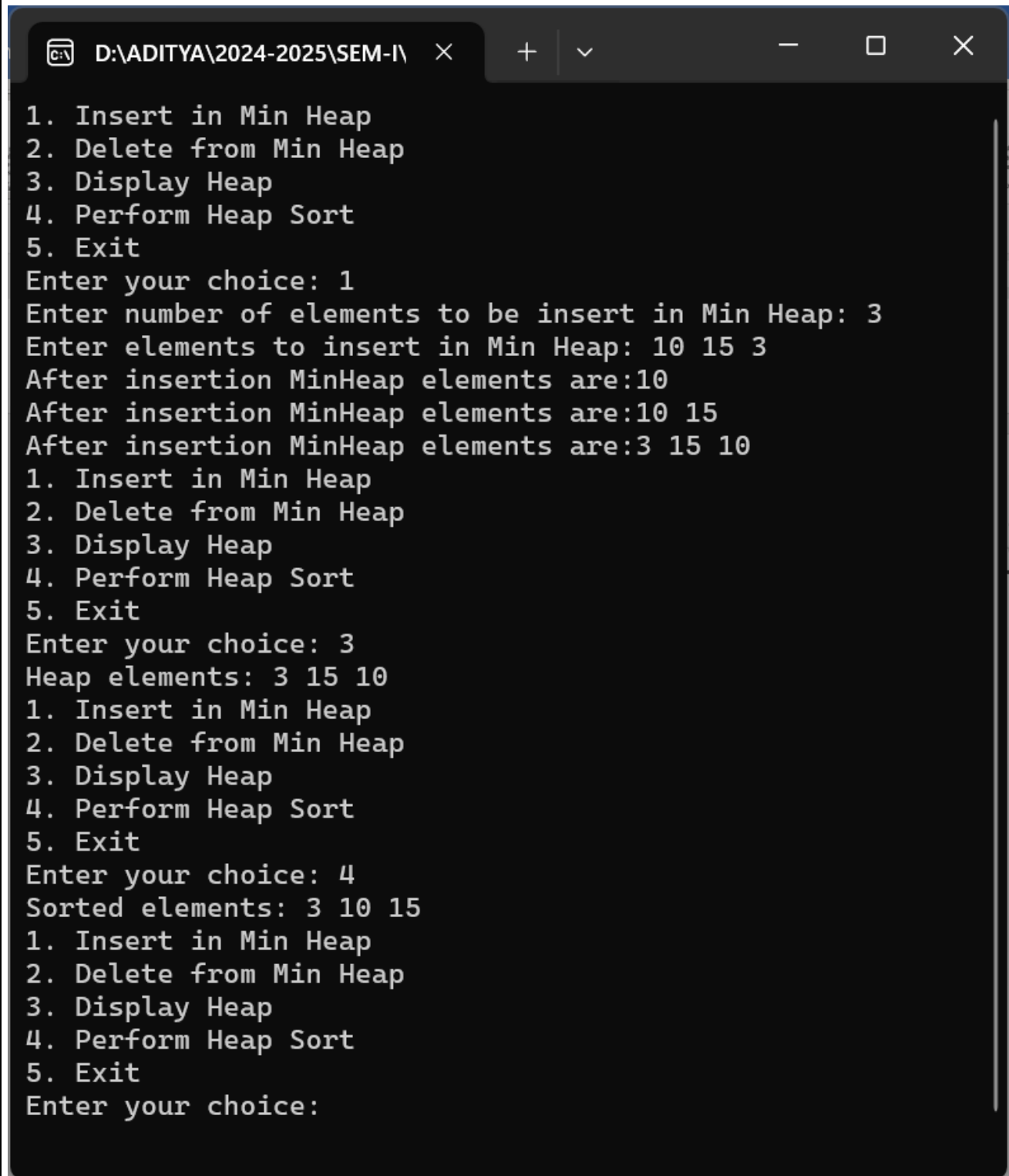
Sorted elements: 3 10 15

1. Insert in Min Heap
2. Delete from Min Heap
3. Display Heap
4. Perform Heap Sort

5. Exit

Enter your choice:5

Actual Input and Output:



```
D:\ADITYA\2024-2025\SEM-I\
1. Insert in Min Heap
2. Delete from Min Heap
3. Display Heap
4. Perform Heap Sort
5. Exit
Enter your choice: 1
Enter number of elements to be insert in Min Heap: 3
Enter elements to insert in Min Heap: 10 15 3
After insertion MinHeap elements are:10
After insertion MinHeap elements are:10 15
After insertion MinHeap elements are:3 15 10
1. Insert in Min Heap
2. Delete from Min Heap
3. Display Heap
4. Perform Heap Sort
5. Exit
Enter your choice: 3
Heap elements: 3 15 10
1. Insert in Min Heap
2. Delete from Min Heap
3. Display Heap
4. Perform Heap Sort
5. Exit
Enter your choice: 4
Sorted elements: 3 10 15
1. Insert in Min Heap
2. Delete from Min Heap
3. Display Heap
4. Perform Heap Sort
5. Exit
Enter your choice:
```

// **Max-Heap**

#include <stdio.h>

#define MAX 100

int heap[MAX];

int size = 0;

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

int temp = *a;

*a = *b;

*b = temp;

}

void maxHeapify(int i) {

int largest = i;

int left = 2 * i + 1;

int right = 2 * i + 2;

if (left < size && heap[left] > heap[largest])

largest = left;

if (right < size && heap[right] > heap[largest])

largest = right;

if (largest != i) {

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

maxHeapify(largest);

}

}

void insertMaxHeap(int key) {

heap[size] = key;

int i = size;

size++;

while (i != 0 && heap[(i - 1) / 2] < heap[i]) {

swap(&heap[i], &heap[(i - 1) / 2]);

i = (i - 1) / 2;

}

printf("After insertion MaxHeap elements are:");

displayHeap();

}

int extractMax() {

if (size <= 0) {

printf("Heap is empty!\n");

return -1;

```
}
if (size == 1) {
    size--;
    return heap[0];
}
int root = heap[0];
heap[0] = heap[size - 1];
size--;
maxHeapify(0);

return root;
}

void heapSort() {
    int originalSize = size;
    for (int i = size - 1; i >= 0; i--) {
        swap(&heap[0], &heap[i]);
        size--;
        maxHeapify(0);
    }
    size = originalSize;
    printf("Sorted elements: ");
    displayHeap();
}

void displayHeap() {
    for (int i = 0; i < size; i++)
        printf("%d ", heap[i]);
    printf("\n");
}

int main() {
    int choice, element, n, i;
    while (1) {
        printf("1. Insert in Max Heap\n");
        printf("2. Delete from Max Heap\n");
        printf("3. Display Heap\n");
        printf("4. Perform Heap Sort\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter number of elements to be insert in Min Heap: ");
                scanf("%d", &n);
                printf("Enter elements to insert in Max Heap: ");
                for(i=0; i<n; i++){
                    scanf("%d", &element);
```

```
        insertMaxHeap(element);
    }
    break;
case 2:
    extractMax();
    break;
case 3:
    printf("Heap elements: ");
    displayHeap();
    break;
case 4:
    heapSort();
    break;
case 5:
    exit(0);
default:
    printf("Invalid choice! Please try again.\n");
}
}
return 0;
}
```

Sample Input and Output:

1. Insert in Max Heap
2. Delete from Max Heap
3. Display Heap
4. Perform Heap Sort
5. Exit

Enter your choice: 1

Enter number of elements to be insert in Min Heap: 3

Enter elements to insert in Max Heap: 8 13 22

After insertion MaxHeap elements are:8

After insertion MaxHeap elements are:13 8

After insertion MaxHeap elements are:22 8 13

1. Insert in Max Heap
2. Delete from Max Heap
3. Display Heap
4. Perform Heap Sort
5. Exit

Enter your choice: 3

Heap elements: 22 8 13

1. Insert in Max Heap
2. Delete from Max Heap
3. Display Heap
4. Perform Heap Sort
5. Exit

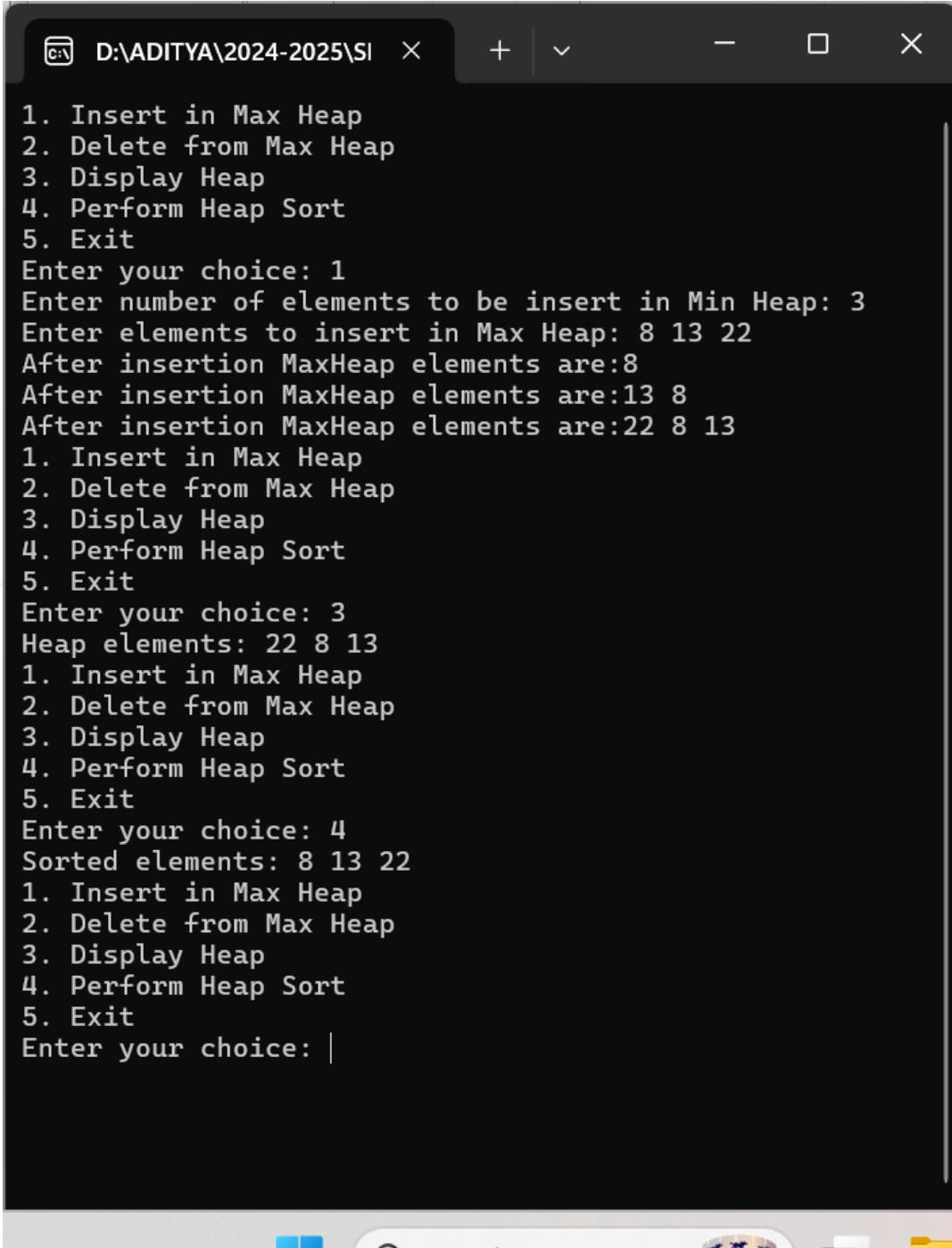
Enter your choice: 4

Sorted elements: 8 13 22

1. Insert in Max Heap
2. Delete from Max Heap
3. Display Heap
4. Perform Heap Sort
5. Exit

Enter your choice:5

Actual Input and Output:



```
D:\ADITYA\2024-2025\SI x + - □ X
1. Insert in Max Heap
2. Delete from Max Heap
3. Display Heap
4. Perform Heap Sort
5. Exit
Enter your choice: 1
Enter number of elements to be insert in Min Heap: 3
Enter elements to insert in Max Heap: 8 13 22
After insertion MaxHeap elements are:8
After insertion MaxHeap elements are:13 8
After insertion MaxHeap elements are:22 8 13
1. Insert in Max Heap
2. Delete from Max Heap
3. Display Heap
4. Perform Heap Sort
5. Exit
Enter your choice: 3
Heap elements: 22 8 13
1. Insert in Max Heap
2. Delete from Max Heap
3. Display Heap
4. Perform Heap Sort
5. Exit
Enter your choice: 4
Sorted elements: 8 13 22
1. Insert in Max Heap
2. Delete from Max Heap
3. Display Heap
4. Perform Heap Sort
5. Exit
Enter your choice: |
```

Week-4: Implement BFT and DFT for given graph using C language, when graph is represented by a) Adjacency Matrix b) Adjacency Lists

// Using Adjacency Matrix

```
#include <stdio.h>
#define MAX 100
int adjMatrix[MAX][MAX];
int visited[MAX];
int queue[MAX];
int front = -1, rear = -1;
void enqueue(int v) {
    if (rear == MAX - 1)
        printf("Queue is full\n");
    else {
        if (front == -1)
            front = 0;
        rear++;
        queue[rear] = v;
    }
}
int dequeue() {
    int v = -1;
    if (front == -1 || front > rear) {
        printf("Queue is empty\n");
    } else {
        v = queue[front];
        front++;
    }
    return v;
}
int isEmptyQueue() {
    return front == -1 || front > rear;
}
void bfs(int start, int n) {
    int i;
    printf("BFS traversal starting from vertex %d: ", start);
    enqueue(start);
    visited[start] = 1;
    while (!isEmptyQueue()) {
        int v = dequeue();
        printf("%d ", v);
        for (i = 0; i < n; i++) {
            if (adjMatrix[v][i] == 1 && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
}
```

```
}
printf("\n");
}
void dfs(int v, int n) {
    printf("%d ", v);
    visited[v] = 1;

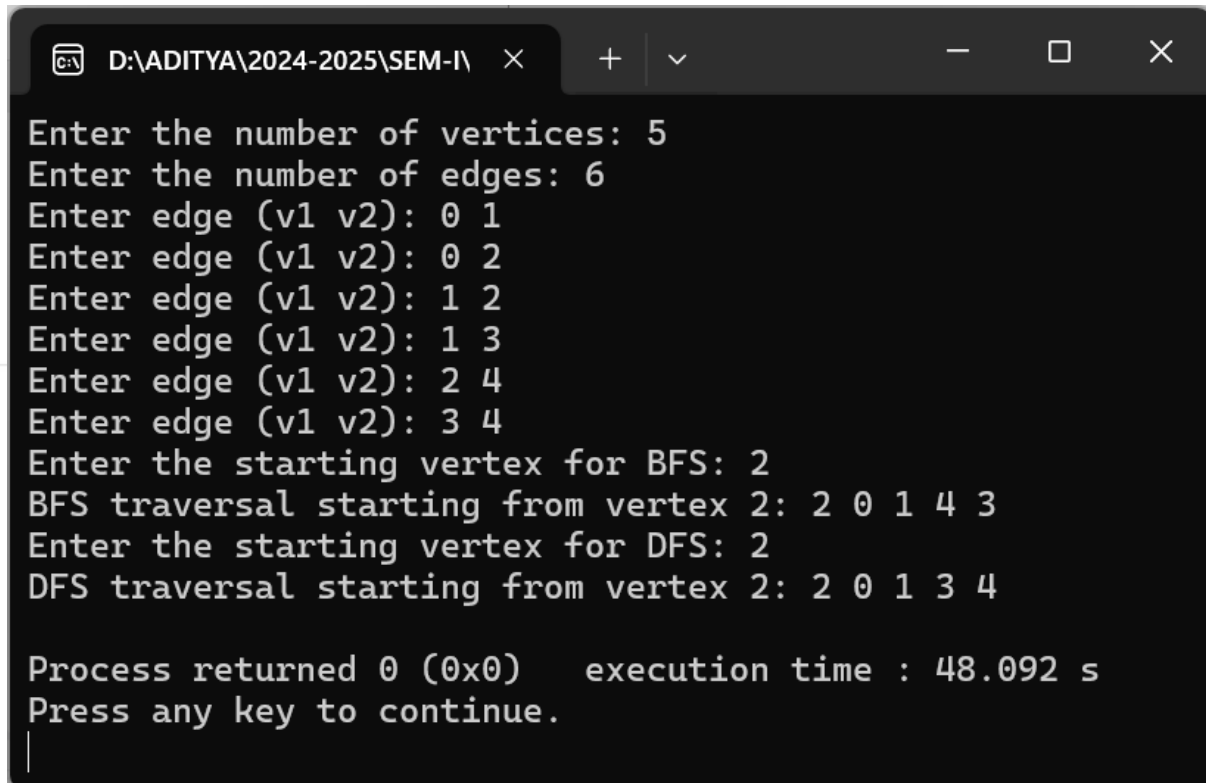
    for (int i = 0; i < n; i++) {
        if (adjMatrix[v][i] == 1 && !visited[i]) {
            dfs(i, n);
        }
    }
}
void startDFS(int start, int n) {
    printf("DFS traversal starting from vertex %d: ", start);
    dfs(start, n);
    printf("\n");
}
int main() {
    int n, e, v1, v2, start;

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

    printf("Enter the number of edges: ");
    scanf("%d", &e);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            adjMatrix[i][j] = 0;
        }
        visited[i] = 0;
    }
    for (int i = 0; i < e; i++) {
        printf("Enter edge (v1 v2): ");
        scanf("%d %d", &v1, &v2);
        adjMatrix[v1][v2] = 1;
        adjMatrix[v2][v1] = 1;
    }
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &start);
    bfs(start, n);
    for (int i = 0; i < n; i++)
        visited[i] = 0;
    printf("Enter the starting vertex for DFS: ");
    scanf("%d", &start);
    startDFS(start, n);
    return 0;
}
```

Sample Input and Output:

Enter the number of vertices: 5
Enter the number of edges: 6
Enter edge (v1 v2): 0 1
Enter edge (v1 v2): 0 2
Enter edge (v1 v2): 1 2
Enter edge (v1 v2): 1 3
Enter edge (v1 v2): 2 4
Enter edge (v1 v2): 3 4
Enter the starting vertex for BFS: 2
BFS traversal starting from vertex 2: 2 0 1 4 3
Enter the starting vertex for DFS: 2
DFS traversal starting from vertex 2: 2 0 1 3 4

Actual Input and Output:

```
D:\ADITYA\2024-2025\SEM-I\ x + v - □ ×  
Enter the number of vertices: 5  
Enter the number of edges: 6  
Enter edge (v1 v2): 0 1  
Enter edge (v1 v2): 0 2  
Enter edge (v1 v2): 1 2  
Enter edge (v1 v2): 1 3  
Enter edge (v1 v2): 2 4  
Enter edge (v1 v2): 3 4  
Enter the starting vertex for BFS: 2  
BFS traversal starting from vertex 2: 2 0 1 4 3  
Enter the starting vertex for DFS: 2  
DFS traversal starting from vertex 2: 2 0 1 3 4  
  
Process returned 0 (0x0) execution time : 48.092 s  
Press any key to continue.  
|
```

// Using Adjacency Lists

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

```
#define MAX 100
```

```
struct Node {  
    int vertex;  
    struct Node* next;  
};
```

```
struct Graph {  
    int numVertices;  
    struct Node** adjLists;  
    int* visited;  
};
```

```
struct Queue {  
    int items[MAX];  
    int front;  
    int rear;  
};
```

```
struct Node* createNode(int v) {  
    struct Node* newNode = malloc(sizeof(struct Node));  
    newNode->vertex = v;  
    newNode->next = NULL;  
    return newNode;  
}
```

```
struct Graph* createGraph(int vertices) {  
    struct Graph* graph = malloc(sizeof(struct Graph));  
    graph->numVertices = vertices;  
    graph->adjLists = malloc(vertices * sizeof(struct Node*));  
    graph->visited = malloc(vertices * sizeof(int));  
    for (int i = 0; i < vertices; i++) {  
        graph->adjLists[i] = NULL;  
        graph->visited[i] = 0;  
    }  
    return graph;  
}
```

```
void addEdge(struct Graph* graph, int src, int dest) {  
    struct Node* newNode = createNode(dest);  
    newNode->next = graph->adjLists[src];  
    graph->adjLists[src] = newNode;  
    newNode = createNode(src);  
    newNode->next = graph->adjLists[dest];  
    graph->adjLists[dest] = newNode;  
}
```

```
struct Queue* createQueue() {
    struct Queue* queue = malloc(sizeof(struct Queue));
    queue->front = -1;
    queue->rear = -1;
    return queue;
}

int isEmpty(struct Queue* queue) {
    return queue->rear == -1;
}

void enqueue(struct Queue* queue, int value) {
    if (queue->rear == MAX - 1)
        printf("Queue is full\n");
    else {
        if (queue->front == -1)
            queue->front = 0;
        queue->rear++;
        queue->items[queue->rear] = value;
    }
}

int dequeue(struct Queue* queue) {
    int item;
    if (isEmpty(queue)) {
        printf("Queue is empty\n");
        item = -1;
    } else {
        item = queue->items[queue->front];
        queue->front++;
        if (queue->front > queue->rear) {
            queue->front = queue->rear = -1;
        }
    }
    return item;
}

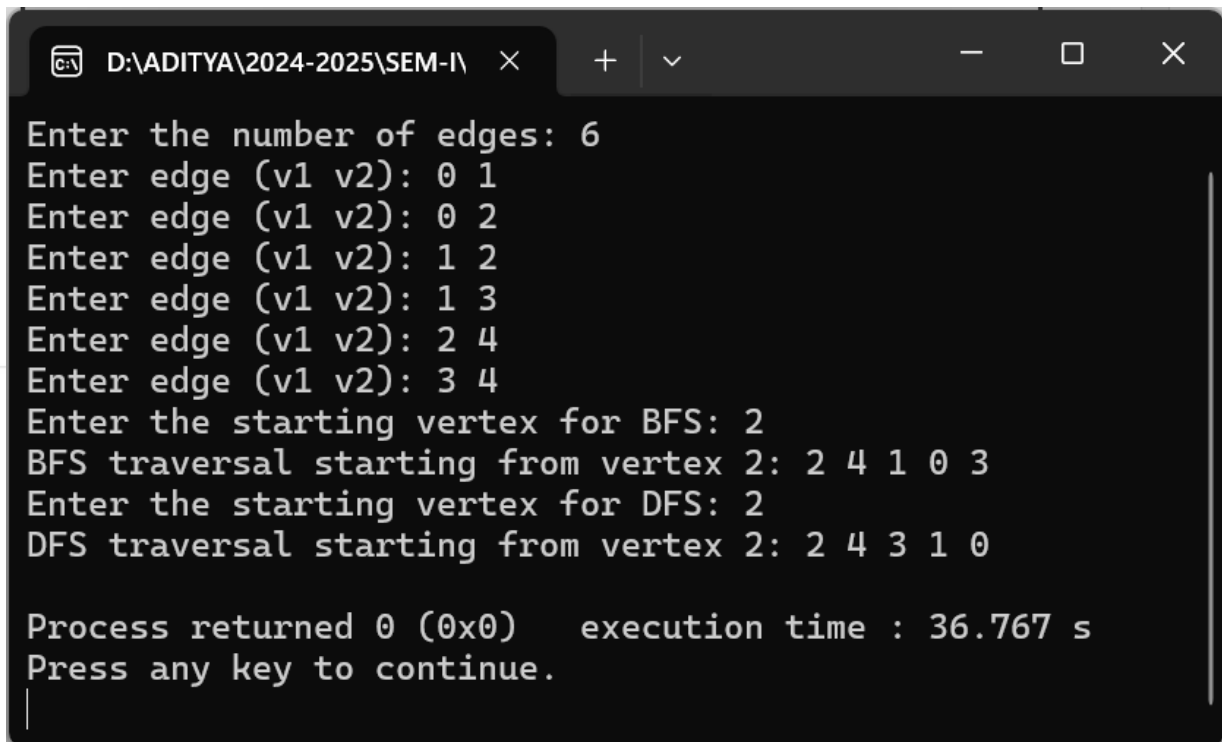
void bfs(struct Graph* graph, int startVertex) {
    struct Queue* queue = createQueue();
    graph->visited[startVertex] = 1;
    enqueue(queue, startVertex);
    printf("BFS traversal starting from vertex %d: ", startVertex);
    while (!isEmpty(queue)) {
        int currentVertex = dequeue(queue);
        printf("%d ", currentVertex);
        struct Node* temp = graph->adjLists[currentVertex];
        while (temp) {
            int adjVertex = temp->vertex;
            if (graph->visited[adjVertex] == 0) {
                graph->visited[adjVertex] = 1;
            }
            temp = temp->next;
        }
    }
}
```

```
        enqueue(queue, adjVertex);
    }
    temp = temp->next;
}
}
printf("\n");
}
void dfs(struct Graph* graph, int vertex) {
    struct Node* adjList = graph->adjLists[vertex];
    struct Node* temp = adjList;
    graph->visited[vertex] = 1;
    printf("%d ", vertex);
    while (temp != NULL) {
        int connectedVertex = temp->vertex;
        if (graph->visited[connectedVertex] == 0) {
            dfs(graph, connectedVertex);
        }
        temp = temp->next;
    }
}
void startDFS(struct Graph* graph, int startVertex) {
    printf("DFS traversal starting from vertex %d: ", startVertex);
    dfs(graph, startVertex);
    printf("\n");
}
int main() {
    int vertices, edges, v1, v2, start;
    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);
    struct Graph* graph = createGraph(vertices);
    printf("Enter the number of edges: ");
    scanf("%d", &edges);
    for (int i = 0; i < edges; i++) {
        printf("Enter edge (v1 v2): ");
        scanf("%d %d", &v1, &v2);
        addEdge(graph, v1, v2);
    }
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &start);
    bfs(graph, start);
    for (int i = 0; i < vertices; i++) {
        graph->visited[i] = 0;
    }
    printf("Enter the starting vertex for DFS: ");
    scanf("%d", &start);
    startDFS(graph, start);
    return 0;
}
```

Sample Input and Output:

Enter the number of vertices: 5
Enter the number of edges: 6
Enter edge (v1 v2): 0 1
Enter edge (v1 v2): 0 2
Enter edge (v1 v2): 1 2
Enter edge (v1 v2): 1 3
Enter edge (v1 v2): 2 4
Enter edge (v1 v2): 3 4
Enter the starting vertex for BFS: 2
BFS traversal starting from vertex 2: 2 4 1 0 3
Enter the starting vertex for DFS: 2
DFS traversal starting from vertex 2: 2 4 3 1 0

Actual Input and Output:



```
D:\ADITYA\2024-2025\SEM-I\ x + v - □ ×  
Enter the number of edges: 6  
Enter edge (v1 v2): 0 1  
Enter edge (v1 v2): 0 2  
Enter edge (v1 v2): 1 2  
Enter edge (v1 v2): 1 3  
Enter edge (v1 v2): 2 4  
Enter edge (v1 v2): 3 4  
Enter the starting vertex for BFS: 2  
BFS traversal starting from vertex 2: 2 4 1 0 3  
Enter the starting vertex for DFS: 2  
DFS traversal starting from vertex 2: 2 4 3 1 0  
  
Process returned 0 (0x0) execution time : 36.767 s  
Press any key to continue.
```


Week-5: Write a C program for finding the biconnected components in a given graph.

```
#include <stdio.h>
#define MAX 100
int adj[MAX][MAX];
int visited[MAX];
int disc[MAX];
int low[MAX];
int parent[MAX];
int stack[MAX][2];
int top = -1;
int time = 0;

void push(int u, int v) {
    stack[++top][0] = u;
    stack[top][1] = v;
}

void pop() {
    top--;
}

void printBCC(int u, int v) {
    printf("Biconnected Component: ");
    while (top != -1 && (stack[top][0] != u || stack[top][1] != v)) {
        printf("(%d, %d) ", stack[top][0], stack[top][1]);
        pop();
    }
    printf("(%d, %d)\n", stack[top][0], stack[top][1]);
    pop();
}

void biconnectedComponentsUtil(int u, int n) {
    visited[u] = 1;
    disc[u] = low[u] = ++time;
    int children = 0;
    for (int v = 0; v < n; v++) {
        if (adj[u][v]) {
            if (!visited[v]) {
                children++;
                parent[v] = u;
                push(u, v);
                biconnectedComponentsUtil(v, n);
                low[u] = (low[u] < low[v]) ? low[u] : low[v];
                if ((parent[u] == -1 && children > 1) || (parent[u] != -1 && low[v] >= disc[u])) {
                    printBCC(u, v);
                }
            } else if (v != parent[u]) {
                low[u] = (low[u] < disc[v]) ? low[u] : disc[v];
                if (disc[v] < disc[u]) {

```

```
        push(u, v);
    }
}
}
}
}
void biconnectedComponents(int n) {
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
        parent[i] = -1;
    }
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            biconnectedComponentsUtil(i, n);
            if (top != -1) {
                printf("Biconnected Component: ");
                while (top != -1) {
                    printf("(%d, %d) ", stack[top][0], stack[top][1]);
                    pop();
                }
                printf("\n");
            }
        }
    }
}
}
}
int main() {
    int n, e;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the number of edges: ");
    scanf("%d", &e);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            adj[i][j] = 0;
        }
    }
    printf("Enter the edges (u v):\n");
    for (int i = 0; i < e; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        adj[u][v] = 1;
        adj[v][u] = 1;
    }
    printf("Biconnected Components are:\n");
    biconnectedComponents(n);
    return 0;
}
```



Sample Input and Output:

Enter the number of vertices: 12

Enter the number of edges: 13

Enter the edges (u v):

0 1
0 6
1 2
1 3
2 3
2 4
3 4
5 6
5 7
5 8
7 8
8 9
10 11

Biconnected Components are:

Biconnected Component: (4, 2) (3, 4) (3, 1) (2, 3) (1, 2)

Biconnected Component: (8, 9)

Biconnected Component: (8, 5) (7, 8) (5, 7)

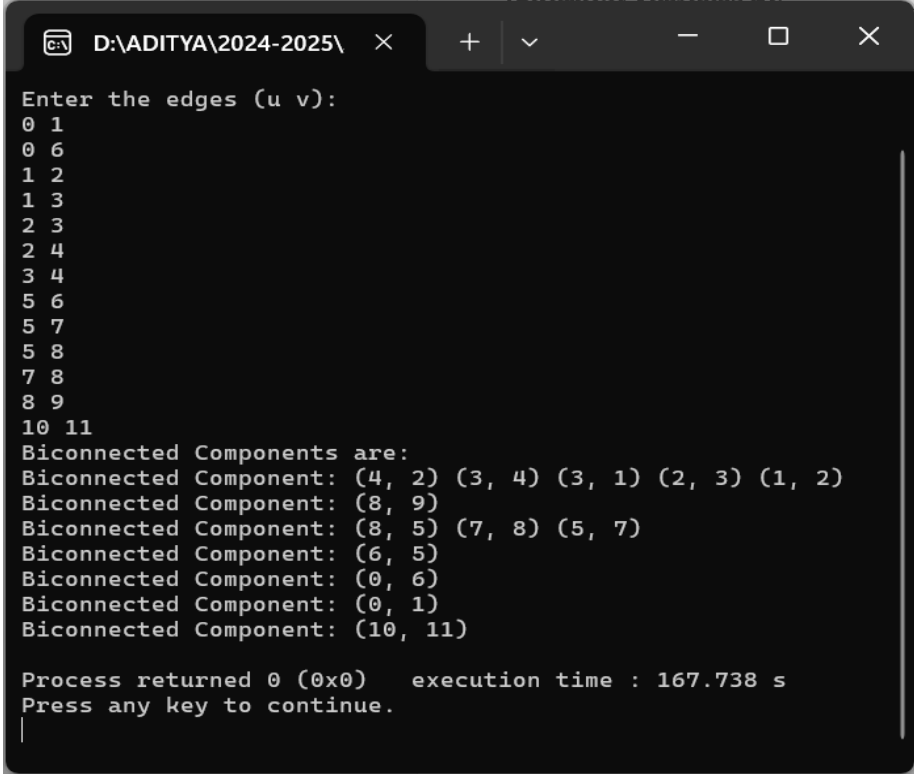
Biconnected Component: (6, 5)

Biconnected Component: (0, 6)

Biconnected Component: (0, 1)

Biconnected Component: (10, 11)

Actual Input and Output:



```
D:\ADITYA\2024-2025\
Enter the edges (u v):
0 1
0 6
1 2
1 3
2 3
2 4
3 4
5 6
5 7
5 8
7 8
8 9
10 11
Biconnected Components are:
Biconnected Component: (4, 2) (3, 4) (3, 1) (2, 3) (1, 2)
Biconnected Component: (8, 9)
Biconnected Component: (8, 5) (7, 8) (5, 7)
Biconnected Component: (6, 5)
Biconnected Component: (0, 6)
Biconnected Component: (0, 1)
Biconnected Component: (10, 11)
Process returned 0 (0x0) execution time : 167.738 s
Press any key to continue.
```

Week-6: Implement Quick sort and Merge sort using C language and observe the execution time for various input sizes (Average, Worst and Best cases).

// Quick sort

```
#include <stdio.h>
void quicksort (int [], int, int);
int partition(int [], int, int);

int main()
{
    int a[50];
    int n, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("\nEnter the %d elements to be sorted:\n", n);
    for (i = 0; i < n; i++)
    {
        printf("a[%d]=\t", i);
        scanf("%d", &a[i]);
    }
    quicksort(a, 0, n - 1);
    printf("After applying quick sort\n");
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    printf("\n");
    return 0;
}
```

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

```
int partition(int a[], int low, int high)
{
    int pvt, i, j, temp;
    if (low < high)
    {
        pvt = low;
        i = low;
```

```
j = high;
while (i < j)
{
    while (a[i] <= a[pvt] && i <= high)
    {
        i++;
    }
    while (a[j] > a[pvt] && j >= low)
    {
        j--;
    }
    if (i < j)
    {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
temp = a[j];
a[j] = a[pvt];
a[pvt] = temp;
return j;
}
```

**ADITYA UNIVERSITY****Sample Input and Output:** (Formerly Aditya Engineering College (A))

Enter the number of elements: 9
Enter the 9 elements to be sorted:
a[0]= 12
a[1]= 23
a[2]= 0
a[3]= 9
a[4]= -12
a[5]= -23
a[6]= 54
a[7]= 23
a[8]= 45
After applying quick sort
-23 -12 0 9 12 23 23 45 54

Actual Input and Output:

```
"D:\ADITYA\2024-2025\SEM-I" × + - □ ×
Enter the 9 elements to be sorted:
a[0]= 12
a[1]= 23
a[2]= 0
a[3]= 9
a[4]= -12
a[5]= -23
a[6]= 54
a[7]= 23
a[8]= 45
After applying quick sort
-23 -12 0 9 12 23 23 45 54

Process returned 0 (0x0)   execution time : 32.259 s
Press any key to continue.
```



// Merge sort

#include<stdio.h>

void divide(int arr[], int l, int r);

void merge(int arr[], int l, int m, int r);

int n;

int main()

{

int arr[10], i;

printf("Enter the number of elements to be sorted:");

scanf("%d", & n);

printf("Enter %d elements to be sorted: \n", n);

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

{

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

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

}

divide(arr, 0, n - 1);

printf("\nSorted array is:\t");

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

{

printf("%4d", arr[i]);

}

printf("\n");

return 0;

}

void divide(int arr[], int l, int r)

{

if (l < r) {

int m = (l+r) / 2;

divide(arr, l, m);

divide(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void merge(int arr[], int l, int m, int r)

{

int i, j, k, n1, n2;

n1 = m - l + 1;

n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

```
i = 0;
j = 0;
k = 1;
while (i < n1 && j < n2)
{
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = R[j];
        j++;
    }
    k++;
}
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}
}
```



ADITYA UNIVERSITY
(Formerly Aditya Engineering College (A))

Sample Input and Output:

Enter the number of elements: 9

Enter the 9 elements to be sorted:

a[0]= 12

a[1]= 23

a[2]= 0

a[3]= 9

a[4]= -12

a[5]= -23

a[6]= 54

a[7]= 23

a[8]= 45

After applying quick sort

-23 -12 0 9 12 23 23 45 54

Actual Input and Output:

```
"D:\ADITYA\2024-2025\SEM-I" × + ▾ − □ ×  
Enter the number of elements to be sorted:5  
Enter 5 elements to be sorted:  
Arr[0] =      12  
Arr[1] =      23  
Arr[2] =     -12  
Arr[3] =       0  
Arr[4] =      34  
  
Sorted array is:      -12   0  12  23  34  
  
Process returned 0 (0x0)   execution time : 11.867 s  
Press any key to continue.
```



Week-7: Compare the performance of Single Source Shortest Paths using Greedy method when the graph is represented by adjacency matrix and adjacency lists using C language.

```
#include <stdio.h>
#define infinity 999

void dijkstra(int n, int v, int cost[10][10], int dist[]) {
    int i, u, count, w, flag[10], min;
    for (i = 0; i < n; i++) {
        flag[i] = 0;
        dist[i] = cost[v][i];
    }
    dist[v] = 0;
    flag[v] = 1;
    count = 1;
    while (count < n) {
        min = infinity;
        u = -1;
        for (w = 0; w < n; w++) {
            if (!flag[w] && dist[w] < min && dist[w] != min) {
                min = dist[w];
                u = w;
            }
        }
        if (u == -1) break;
        flag[u] = 1;
        count++;
        for (w = 0; w < n; w++) {
            if (!flag[w] && (dist[u] + cost[u][w] < dist[w]) && dist[w] != min) {
                dist[w] = dist[u] + cost[u][w];
            }
        }
    }
}

int main() {
    int n, v, i, j, cost[10][10], dist[10];
    printf("Enter the number of nodes:\n");
    scanf("%d", &n);
    printf("Enter the cost matrix (use 0 for no direct path):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
            if (cost[i][j] == 0)
                cost[i][j] = infinity;
        }
    }
    printf("Enter the source node (0 to %d):\n", n - 1);
```

```
scanf("%d", &v);
if (v < 0 || v >= n) {
    printf("Invalid source node.\n");
    return 1;
}
dijkstra(n, v, cost, dist);
printf("Shortest path(s) from node %d:\n", v);
for (i = 0; i < n; i++) {
    if (i != v) {
        printf("%d -> %d, cost = %d\n", v, i, dist[i]);
    }
}
return 0;
}
```

Sample Input and Output:

Enter the number of nodes:

9

Enter the cost matrix (use 0 for no direct path):

0 4 0 0 0 0 8 0

4 0 8 0 0 0 11 0

0 8 0 7 0 4 0 2

0 0 7 0 9 14 0 0

0 0 0 9 0 10 0 0

0 0 4 14 10 0 2 0

0 0 0 0 2 0 1 6

8 11 0 0 0 0 1 0

0 0 2 0 0 0 6 7

Enter the source node (0 to 8):

0

Shortest path(s) from node 0:

0 -> 1, cost = 4

0 -> 2, cost = 12

0 -> 3, cost = 19

0 -> 4, cost = 21

0 -> 5, cost = 11

0 -> 6, cost = 9

0 -> 7, cost = 8

0 -> 8, cost = 14

Actual Input and Output:

```
D:\ADITYA\2024-2025\SEM-I\ × + ▾ — □ ×

Enter the number of nodes:
9
Enter the cost matrix (use 0 for no direct path):
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0
Enter the source node (0 to 8):
0
Shortest path(s) from node 0:
0 -> 1, cost = 4
0 -> 2, cost = 12
0 -> 3, cost = 19
0 -> 4, cost = 21
0 -> 5, cost = 11
0 -> 6, cost = 9
0 -> 7, cost = 8
0 -> 8, cost = 14

Process returned 0 (0x0)    execution time : 25.512 s
Press any key to continue.
|
```

Week-8: Write a C program to implement Job Sequencing with deadlines using Greedy method.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int deadline;
    int profit;
    char id;
} Job;

int compare(const void* a, const void* b) {
    Job* j1 = (Job*)a;
    Job* j2 = (Job*)b;
    return j2->profit - j1->profit;
}

void jobSequencing(Job jobs[], int n) {
    qsort(jobs, n, sizeof(Job), compare);
    int result[n];
    int slot[n];
    for (int i = 0; i < n; i++) {
        slot[i] = -1;
    }
    int totalProfit = 0;
    for (int i = 0; i < n; i++) {
        for (int j = jobs[i].deadline - 1; j >= 0; j--) {
            if (slot[j] == -1) {
                result[j] = i;
                slot[j] = 1;
                totalProfit += jobs[i].profit;
                break;
            }
        }
    }
    printf("Job Sequence: ");
    for (int i = 0; i < n; i++) {
        if (slot[i] != -1) {
            printf("%c ", jobs[result[i]].id);
        }
    }
    printf("\nTotal Profit: %d\n", totalProfit);
}

int main() {
    int n, i;
    printf("Enter the number of Jobs:\n");
    scanf("%d", &n);
    Job jobs[n];
    printf("Enter the Deadline, Profit and Job_label(Single Character) with one space:\n");
```

```
for(i=0;i<n;i++)
{
    scanf("%d %d %c",&jobs[i].deadline,&jobs[i].profit,&jobs[i].id);
}
jobSequencing(jobs, n);
return 0;
}
```

Sample Input and Output:

Enter the number of Jobs:

5

Enter the Deadline, Profit and Job_label(Single Character) with one space:

2 20 1

2 60 2

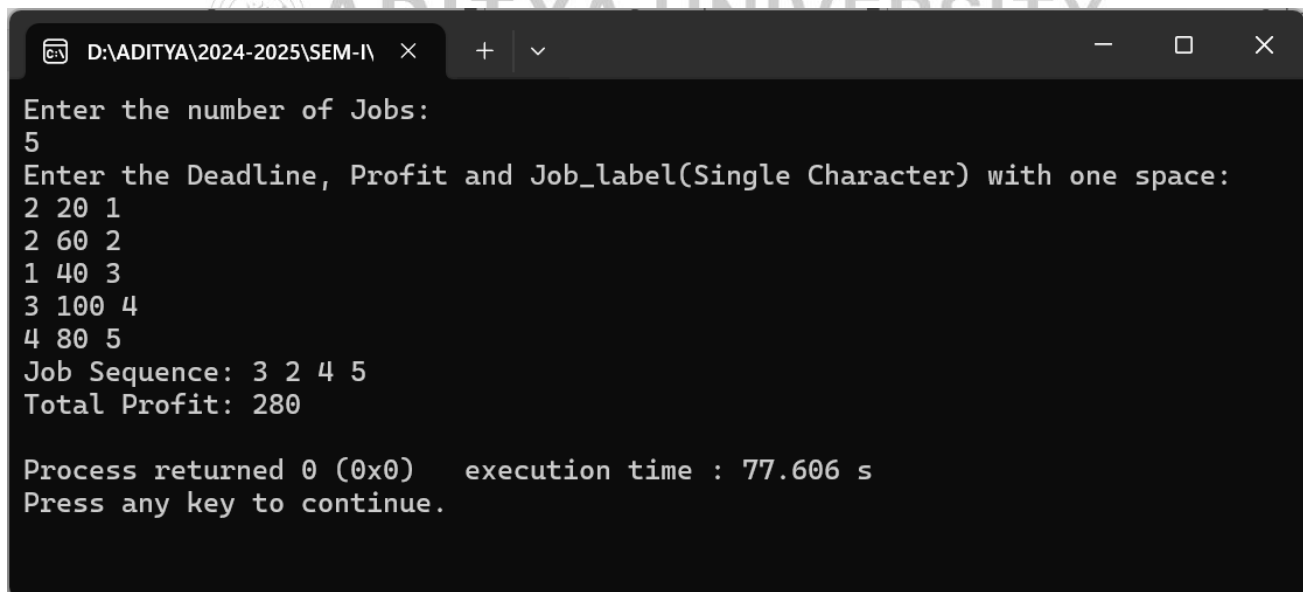
1 40 3

3 100 4

4 80 5

Job Sequence: 3 2 4 5

Total Profit: 280

Actual Input and Output:

```
D:\ADITYA\2024-2025\SEM-I\ x + v - □ ×
Enter the number of Jobs:
5
Enter the Deadline, Profit and Job_label(Single Character) with one space:
2 20 1
2 60 2
1 40 3
3 100 4
4 80 5
Job Sequence: 3 2 4 5
Total Profit: 280

Process returned 0 (0x0) execution time : 77.606 s
Press any key to continue.
```

Week-9: Write a C program to solve 0/1 Knapsack problem Using Dynamic Programming.

```
#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("Maximum profit: %d\n", K[n][W]);
    printf("Items included in the knapsack:\n");
    w = W;
    for (i = n; i > 0 && K[i][w] != 0; i--) {
        if (K[i][w] != K[i - 1][w]) {
            printf("Item %d (Weight: %d, Value: %d)\n", i, wt[i - 1], val[i - 1]);
            w -= wt[i - 1];
        }
    }
}

int main() {
    int i, n, W;
    int val[10], wt[10];
    printf("Enter the number of items: ");
    scanf("%d", &n);
    printf("Enter the weights and values of the items:\n");
    for (i = 0; i < n; i++)
        scanf("%d %d", &wt[i], &val[i]);
    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &W);
    knapsack(W, wt, val, n);
    return 0;
}
```

Sample Input and Output:

Enter the number of items: 4

Enter the weights and values of the items:

3 2

4 3

6 1

5 4

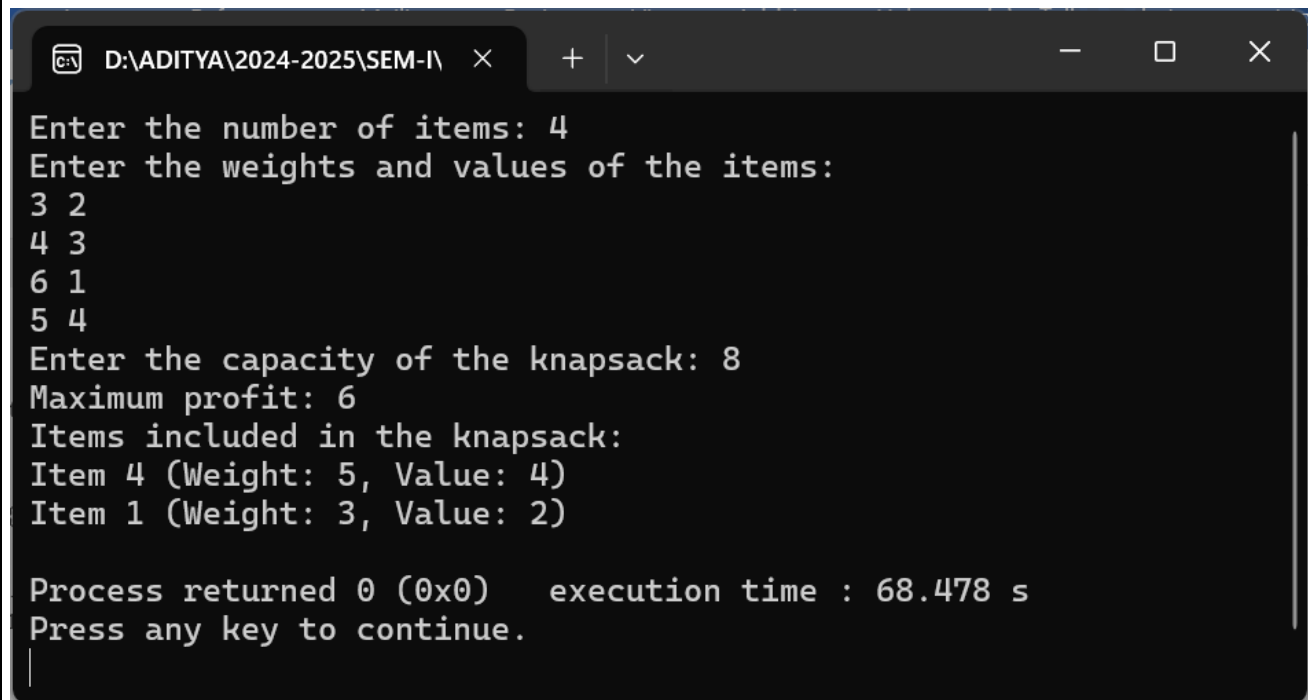
Enter the capacity of the knapsack: 8

Maximum profit: 6

Items included in the knapsack:

Item 4 (Weight: 5, Value: 4)

Item 1 (Weight: 3, Value: 2)

Actual Input and Output:

```
D:\ADITYA\2024-2025\SEM-I\ x + v - □ ×  
Enter the number of items: 4  
Enter the weights and values of the items:  
3 2  
4 3  
6 1  
5 4  
Enter the capacity of the knapsack: 8  
Maximum profit: 6  
Items included in the knapsack:  
Item 4 (Weight: 5, Value: 4)  
Item 1 (Weight: 3, Value: 2)  
  
Process returned 0 (0x0) execution time : 68.478 s  
Press any key to continue.  
|
```


Week-10: Write a C program to implement N-Queens Problem using Backtracking strategy.

```
#include<stdio.h>
#include<math.h>
int board[20],count;
void queen(int row,int n);

int main()
{
    int n,i,j;
    printf(" - N Queens Problem Using Backtracking -");
    printf("\n\nEnter number of Queens:");
    scanf("%d",&n);
    queen(1,n);
    return 0;
}

void print(int n)
{
    int i,j;
    printf("\n\nSolution %d:\n\n",++count);
    for(i=1;i<=n;++i)
        printf("\t%d",i);
    for(i=1;i<=n;++i)
    {
        printf("\n%d",i);
        for(j=1;j<=n;++j) //for nxn board
        {
            if(board[i]==j)
                printf("\tQ%d",i); //queen at i,j position
            else
                printf("\t-"); //empty slot
        }
    }
}

int place(int row,int column)
{
    int i;
    for(i=1;i<=row-1;++i)
    {
        if(board[i]==column)
            return 0;
        else
            if(abs(board[i]-column)==abs(i-row))
                return 0;
    }
    return 1; //no conflicts
}
```

```

void queen(int row,int n)
{
    int column;
    for(column=1;column<=n;++column)
    {
        if(place(row,column))
        {
            board[row]=column; //no conflicts so place queen
            if(row==n) //dead end
                print(n); //printing the board configuration
            else //try queen with next position
                queen(row+1,n);
        }
    }
}

```

Sample Input and Output:

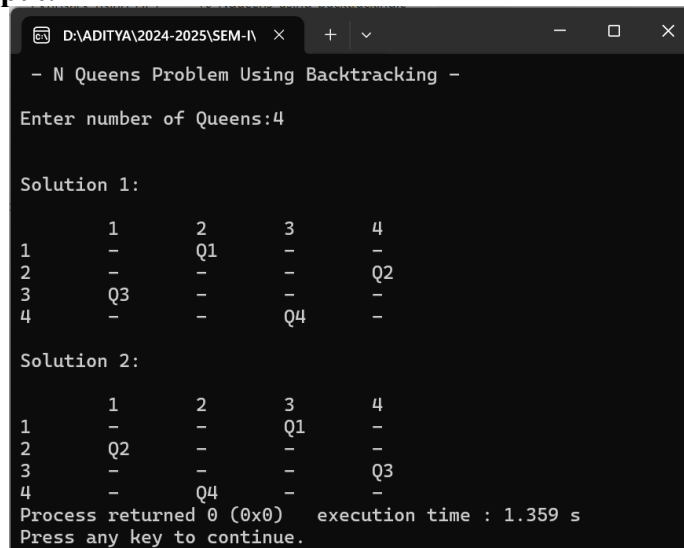
Enter number of Queens:4

Solution 1:

	1	2	3	4
1	-	Q1	-	-
2	-	-	-	Q2
3	Q3	-	-	-
4	-	-	Q4	-

Solution 2:

	1	2	3	4
1	-	-	Q1	-
2	Q2	-	-	-
3	-	-	-	Q3
4	-	Q4	-	-

Actual Input and Output:


```

D:\ADITYA\2024-2025\SEM-IV x
- N Queens Problem Using Backtracking -
Enter number of Queens:4

Solution 1:

    1    2    3    4
1    -    Q1   -    -
2    -    -    -    Q2
3    Q3   -    -    -
4    -    -    Q4   -

Solution 2:

    1    2    3    4
1    -    -    Q1   -
2    Q2   -    -    -
3    -    -    -    Q3
4    -    Q4   -    -

Process returned 0 (0x0)   execution time : 1.359 s
Press any key to continue.

```

Week-11: Write a C program to implement 0/1 Knapsack problem using Backtracking strategy

```

#include <stdio.h>
#define max 10
int w[max], i, j, p[max];
int n, m;
float unit[max];
int y[max], x[max], fp = -1, fw;
void get() {
    printf("\n Enter total number of items: ");
    scanf("%d", &n);
    printf("\n Enter the Maximum capacity of the Sack: ");
    scanf("%d", &m);
    printf("\n Enter the weight and profit of the item: ", i + 1);
    for (i = 0; i < n; i++) {
        scanf("%d %d", &w[i], &p[i]);
    }
}
void show() {
    float s = 0.0;
    printf("\n\tItem\tWeight\tCost\tUnit Profit\tSelected ");
    for (i = 0; i < n; i++)
        printf("\n\t%d\t%d\t%d\t%f\t%d", i + 1, w[i], p[i], unit[i], x[i]);
    printf("\n\n The Sack now holds following items : ");
    for (i = 0; i < n; i++)
        if (x[i] == 1) {
            printf("%d\t", i + 1);
            s += (float) p[i] * (float) x[i];
        }
    printf("\n Maximum Profit: %f\n\n", s);
}
void sort() {
    int t, t1;
    float t2;
    for (i = 0; i < n; i++)
        unit[i] = (float) p[i] / (float) w[i];
    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (unit[i] < unit[j]) {
                t2 = unit[i];
                unit[i] = unit[j];
                unit[j] = t2;
                t = p[i];
                p[i] = p[j];
                p[j] = t;
                t1 = w[i];
                w[i] = w[j];
                w[j] = t1;
            }
        }
    }
}

```

```

    }
  }
} }
float bound(float cp, float cw, int k) {
    float b = cp;
    float c = cw;
    for (i = k; i <= n; i++) {
        c = c + w[i];
        if (c < m)
            b = b + p[i];
        else
            return (b + (1 - (c - m) / (float) w[i]) * p[i]);
    }
    return b;
}
void knapsack(int k, float cp, float cw) {
    if (cw + w[k] <= m) {
        y[k] = 1;
        if (k <= n)
            knapsack(k + 1, cp + p[k], cw + w[k]);
        if (((cp + p[k]) > fp) && (k == n)) {
            fp = cp + p[k];
            fw = cw + w[k];
            for (j = 0; j <= k; j++)
                x[j] = y[j];
        }
    }
    if (bound(cp, cw, k) >= fp) {
        y[k] = 0;
        if (k <= n)
            knapsack(k + 1, cp, cw);
        if ((cp > fp) && (k == n)) {
            fp = cp;
            fw = cw;
            for (j = 0; j <= k; j++)
                x[j] = y[j];
        }
    }
}
int main() {
    get();
    printf("\n The Knapsack is arranged in the order\n");
    sort();
    knapsack(0, 0.0, 0.0);
    show();
    return 0;
}

```

Sample Input and Output:

Enter total number of items: 4
Enter the Maximum capacity of the Sack: 8
Enter the weight and profit of the item: 3 2
4 3
6 1
5 4

The Knapsack is arranged in the order

Item	Weight	Profit	Unit Profit	Selected
1	5	4	0.800000	1
2	4	3	0.750000	0
3	3	2	0.666667	1
4	6	1	0.166667	0

The Sack now holds following items: 1 3
Maximum Profit: 6.000000

Actual Input and Output:

```
D:\ADITYA\2024-2025\SEM-IV × + - □ ×

Enter total number of items: 4

Enter the Maximum capacity of the Sack: 8

Enter the weight and profit of the item: 3 2
4 3
6 1
5 4

The Knapsack is arranged in the order

      Item    Weight  Profit  Unit Profit  Selected
      1       5      4      0.800000      1
      2       4      3      0.750000      0
      3       3      2      0.666667      1
      4       6      1      0.166667      0

The Sack now holds following items : 1 3
Maximum Profit: 6.000000
```

Week-12: Write a C program to implement Travelling Salesperson problem using Branch and Bound approach.

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#define N 4
int findMinRow(int costMatrix[N][N], int row[], bool visited[], int current) {
    int min = INT_MAX;
    for (int i = 0; i < N; i++) {
        if (i != current && !visited[i] && row[i] < min) {
            min = row[i];
        }
    }
    return (min == INT_MAX) ? 0 : min;
}
```

```
int findMinCol(int costMatrix[N][N], int col[], bool visited[], int current) {
    int min = INT_MAX;
    for (int i = 0; i < N; i++) {
        if (i != current && !visited[i] && col[i] < min) {
            min = col[i];
        }
    }
    return (min == INT_MAX) ? 0 : min;
}
```

```
int reduceMatrix(int costMatrix[N][N], int current, bool visited[]) {
    int rowReduction = 0, colReduction = 0;
    for (int i = 0; i < N; i++) {
        int minRow = findMinRow(costMatrix, costMatrix[i], visited, current);
        rowReduction += minRow;
        for (int j = 0; j < N; j++) {
            if (costMatrix[i][j] != INT_MAX) {
                costMatrix[i][j] -= minRow;
            }
        }
    }
    for (int j = 0; j < N; j++) {
        int col[N];
        for (int i = 0; i < N; i++) {
            col[i] = costMatrix[i][j];
        }
        int minCol = findMinCol(costMatrix, col, visited, current);
        colReduction += minCol;
        for (int i = 0; i < N; i++) {
            if (costMatrix[i][j] != INT_MAX) {
                costMatrix[i][j] -= minCol;
            }
        }
    }
}
```

```

    }
    }
}
return rowReduction + colReduction;
}

void tspBranchAndBound(int costMatrix[N][N]) {
    bool visited[N] = {false};
    visited[0] = true; // Starting from the first city
    int current = 0;
    int totalCost = 0;
    printf("Path: 0");
    for (int count = 1; count < N; count++) {
        int bestNextCity = -1;
        int minCost = INT_MAX;
        for (int i = 0; i < N; i++) {
            if (!visited[i] && costMatrix[current][i] < minCost) {
                minCost = costMatrix[current][i];
                bestNextCity = i;
            }
        }
        totalCost += minCost;
        current = bestNextCity;
        visited[current] = true;
        printf(" -> %d", current);
    }
    totalCost += costMatrix[current][0]; // Returning to the starting point
    printf("\nTotal minimum cost: %d\n", totalCost);
}

int main() {
    int costMatrix[N][N];
    printf("Enter the cost matrix (%d x %d):\n", N, N);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (i == j) {
                costMatrix[i][j] = INT_MAX; // No self-loop
            } else {
                printf("Cost from city %d to city %d: ", i, j);
                scanf("%d", &costMatrix[i][j]);
            }
        }
    }
    printf("Solving TSP using Branch and Bound approach:\n");
    tspBranchAndBound(costMatrix);
    return 0;
}

```

Sample Input and Output:

Enter the cost matrix (4 x 4):

Cost from city 0 to city 1: 4

Cost from city 0 to city 2: 2

Cost from city 0 to city 3: 3

Cost from city 1 to city 0: 4

Cost from city 1 to city 2: 2

Cost from city 1 to city 3: 5

Cost from city 2 to city 0: 2

Cost from city 2 to city 1: 2

Cost from city 2 to city 3: 3

Cost from city 3 to city 0: 3

Cost from city 3 to city 1: 5

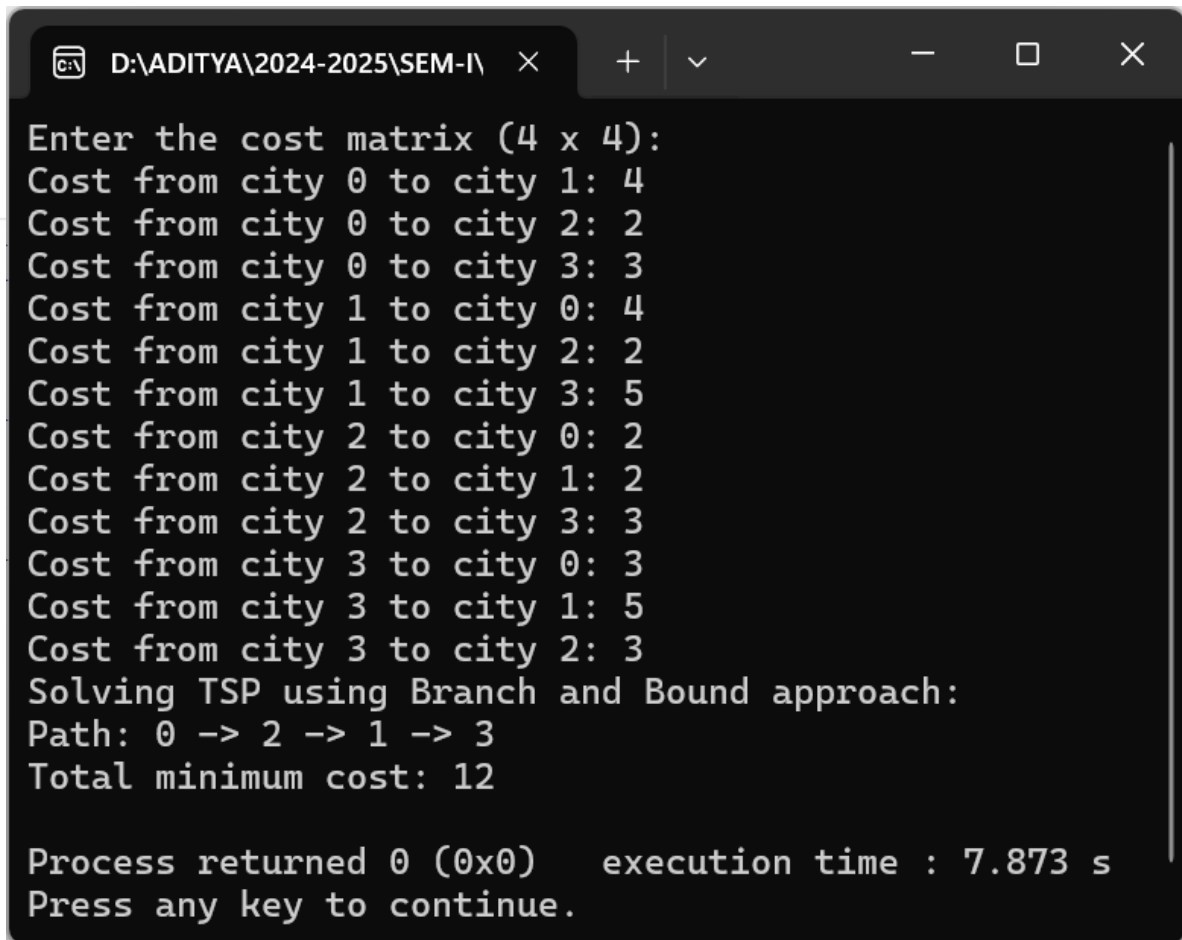
Cost from city 3 to city 2: 3

Solving TSP using Branch and Bound approach:

Path: 0 -> 2 -> 1 -> 3

Total minimum cost: 12

Actual Input and Output:



```
D:\ADITYA\2024-2025\SEM-I\ x + v - □ ×

Enter the cost matrix (4 x 4):
Cost from city 0 to city 1: 4
Cost from city 0 to city 2: 2
Cost from city 0 to city 3: 3
Cost from city 1 to city 0: 4
Cost from city 1 to city 2: 2
Cost from city 1 to city 3: 5
Cost from city 2 to city 0: 2
Cost from city 2 to city 1: 2
Cost from city 2 to city 3: 3
Cost from city 3 to city 0: 3
Cost from city 3 to city 1: 5
Cost from city 3 to city 2: 3
Solving TSP using Branch and Bound approach:
Path: 0 -> 2 -> 1 -> 3
Total minimum cost: 12

Process returned 0 (0x0)    execution time : 7.873 s
Press any key to continue.
```