**DAY 4**

1.Write a c program to implement binary search tree?

**Program:**

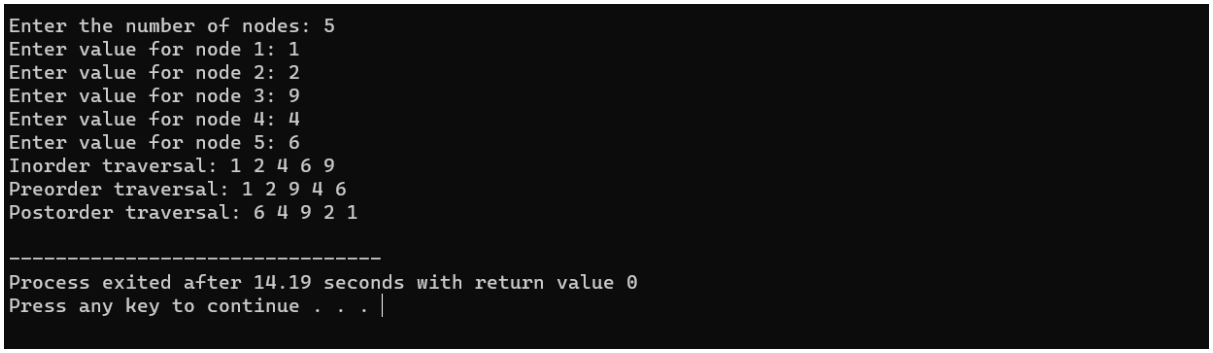
```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *left;
   struct Node *right;
};
struct Node *newNode(int data) {
   struct Node *node = (struct Node *)malloc(sizeof(struct Node));
   node->data = data;
   node->left = NULL;
   node->right = NULL;
   return node;
}
struct Node *insert(struct Node *root, int data) {
   if (root == NULL) {
      return newNode(data);
   }
   if (data < root->data) {
      root->left = insert(root->left, data);
   } else if (data > root->data) {
      root->right = insert(root->right, data);
   }
   return root;
}
void inorderTraversal(struct Node *root) {
   if (root != NULL) {
```

```
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
void preorderTraversal(struct Node *root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}
void postorderTraversal(struct Node *root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}
int main() {
    struct Node *root = NULL;

    int n, i;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("Enter the values:\n");
    for ( i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        root = insert(root, value);
```

```
    }
    printf("In-order traversal: ");
    inorderTraversal(root);
    printf("\n");
    printf("Pre-order traversal: ");
    preorderTraversal(root);
    printf("\n");
    printf("Post-order traversal: ");
    postorderTraversal(root);
    printf("\n");
    return 0;
}
```

**Output :**




2.Write a C program to implement AVL Tree?

**Program:**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int key;
    struct Node* left;
    struct Node* right;
    int height;
};
struct Node* newNode(int key) {
```

```
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = node->right = NULL;
    node->height = 1;
    return node;
}
int height(struct Node* node) {
    if (node == NULL)
        return 0;
    return node->height;
}
int max(int a, int b) {
    return (a > b) ? a : b;
}
struct Node* rightRotate(struct Node* y) {
    struct Node* x = y->left;
    struct 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;
}
struct Node* leftRotate(struct Node* x) {
    struct Node* y = x->right;
    struct 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(struct Node* node) {
    if (node == NULL)
        return 0;
    return height(node->left) - height(node->right);
}
struct Node* insert(struct Node* node, int key) {
    if (node == NULL)
        return newNode(key);
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
            return node;
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
    if (balance > 1) {
        if (key < node->left->key) {
            return rightRotate(node);
        } else {
            node->left = leftRotate(node->left);
            return rightRotate(node);
        }
    }
    if (balance < -1) {
        if (key > node->right->key) {
            return leftRotate(node);
        } else {
            node->right = rightRotate(node->right);
            return leftRotate(node);
```

```
        }
    }
    return node;
}
void inOrder(struct Node* root) {
    if (root != NULL) {
        inOrder(root->left);
        printf("%d ", root->key);
        inOrder(root->right);
    }
}
int main() {
    struct Node* root = NULL;
    int choice, key;
    while (1) {
        printf("Menu:\n");
        printf("1. Insert a key\n");
        printf("2. Print in-order traversal\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the key to insert: ");
                scanf("%d", &key);
                root = insert(root, key);
                break;
            case 2:
                printf("In-order traversal: ");
                inOrder(root);
```

```
            printf("\n");
            break;
        case 3:
            exit(0);
        default:
            printf("Invalid choice!\n");
      }
   }
   return 0;
}
```

**Output:**

```
Menu:
1. Insert a key
2. Print in-order traversal
3. Exit
Enter your choice: 1
Enter the key to insert: 5
Menu:
1. Insert a key
2. Print in-order traversal
3. Exit
Enter your choice: 1
Enter the key to insert: 5
Menu:
1. Insert a key
2. Print in-order traversal
3. Exit
Enter your choice: 2
In-order traversal: 5
```

3.Write a C program to implement hashing using linear probing?

**Program:**

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
struct HashTable {
   int table[SIZE];
   int count;
```

```
};
void initialize(struct HashTable* ht) {
    for (int i = 0; i < SIZE; i++) {
        ht->table[i] = -1;
    }
    ht->count = 0;
}
int hash(int key) {
    return key % SIZE;
}
void insert(struct HashTable* ht, int key) {
    if (ht->count == SIZE) {
        printf("Hash table is full. Cannot insert %d.\n", key);
        return;
    }
    int index = hash(key);
    while (ht->table[index] != -1) {
        index = (index + 1) % SIZE;
    }
    ht->table[index] = key;
    ht->count++;
}
int search(struct HashTable* ht, int key) {
    int index = hash(key);
    while (ht->table[index] != -1) {
        if (ht->table[index] == key) {
            return index;
        }
        index = (index + 1) % SIZE;
    }
```

```
    return -1;
}
void display(struct HashTable* ht) {
    printf("Hash Table:\n");
    for (int i = 0; i < SIZE; i++) {
        if (ht->table[i] != -1) {
            printf("Index %d: %d\n", i, ht->table[i]);
        }
    }
}
int main() {
    struct HashTable ht;
    initialize(&ht);
    int choice, key;
    do {
        printf("\nMenu:\n");
        printf("1. Insert a key\n");
        printf("2. Search for a key\n");
        printf("3. Display the hash table\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the key to insert: ");
                scanf("%d", &key);
                insert(&ht, key);
                break;
            case 2:
                printf("Enter the key to search: ");
```

```
            scanf("%d", &key);
            int index = search(&ht, key);
            if (index != -1) {
                printf("Key %d found at index %d.\n", key, index);
            } else {
                printf("Key %d not found.\n", key);
            }
            break;
        case 3:
            display(&ht);
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
      }
   } while (choice != 4);
   return 0;
}
```

**Output:**

```
Menu:
1. Insert a key
2. Search for a key
3. Display the hash table
4. Exit
Enter your choice: 1
Enter the key to insert: 56
Menu:
1. Insert a key
2. Search for a key
3. Display the hash table
4. Exit
Enter your choice: 1
Enter the key to insert: 78
Menu:
1. Insert a key
2. Search for a key
3. Display the hash table
4. Exit
Enter your choice:
3
Hash Table:
Index 6: 56
Index 8: 78
```

4.Write a C program to implement bubble sort?

**Program:**

```
#include <stdio.h>

void bubbleSort(int arr[], int n) {
   int temp;
   int swapped;
   for (int i = 0; i < n - 1; i++) {
      swapped = 0;
      for (int j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
            swapped = 1;
         }
      }
      if (swapped == 0) {
         break;
```

```
        }
    }
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    bubbleSort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

**Output:**

```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
```

5.Write a C program to implement insertion sort?

**Program:**

```
#include <stdio.h>
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
```

```
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    insertionSort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

**Output:**

```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
```

6.Write a c program to implement selection sort?

**Program:**

```
#include <stdio.h>
void selectionSort(int arr[], int n) {
   int i, j, minIndex, temp;
   for (i = 0; i < n - 1; i++) {
      minIndex = i;
      for (j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIndex]) {
            minIndex = j;
         }
      }
      temp = arr[minIndex];
      arr[minIndex] = arr[i];
      arr[i] = temp;
   }
}
int main() {
   int arr[] = {64, 34, 25, 12, 22, 11, 90};
   int n = sizeof(arr) / sizeof(arr[0]);
   printf("Original array: ");
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
```

```
    selectionSort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

**Output:**

```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
```

7.Write a c program to implement Quick sort?

**Program:**

```
#include <stdio.h>
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
```

```
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    quickSort(arr, 0, n - 1);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

**Output:**

```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
```

8.Write a C program to implement Merge sort?

**Program:**

```
#include <stdio.h>
void merge(int arr[], int l, int m, int r) {
   int i, j, k;
   int n1 = m - l + 1;
   int 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 = l;
   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++;
    }
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    mergeSort(arr, 0, n - 1);
```

```
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

**Output:**

```
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
```