AQ1:

```cpp
#include <iostream>
using namespace std;

struct node {
    int data;
    node* left;
    node* right;

    node(int value) {
        data = value;
        left = right = NULL;
    }
};

node* buildTree(node* root) {
    int data;
    cout << "Enter the data (-1 for no node): ";
    cin >> data;

    if(data == -1) {
        return NULL;
    }

    root = new node(data);

    cout << "Enter data for inserting in left of " << data << endl;
    root->left = buildTree(root->left);

    cout << "Enter data for inserting in right of " << data << endl;
    root->right = buildTree(root->right);

    return root;
}

void preOrder(node* root) {
    if(root == NULL) return;
    cout << root->data << " ";
    preOrder(root->left);
    preOrder(root->right);
}

void inOrder(node* root) {
    if(root == NULL) return;
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}
```

```cpp
void postOrder(node* root) {
    if(root == NULL) return;
    postOrder(root->left);
    postOrder(root->right);
    cout << root->data << " ";
}

int main() {
    node* root = NULL;

    root = buildTree(root);

    cout << "\nPre-order Traversal: ";
    preOrder(root);

    cout << "\nIn-order Traversal: ";
    inOrder(root);

    cout << "\nPost-order Traversal: ";
    postOrder(root);

    cout << endl;
    return 0;
}
```

Output:

```
Enter the data (-1 for no node): 8
Enter data for inserting in left of 8
Enter the data (-1 for no node): 3
Enter data for inserting in left of 3
Enter the data (-1 for no node): 1
Enter data for inserting in left of 1
Enter the data (-1 for no node): -1
Enter data for inserting in right of 1
Enter the data (-1 for no node): -1
Enter data for inserting in right of 3
Enter the data (-1 for no node): 6
Enter data for inserting in left of 6
Enter the data (-1 for no node): 4
Enter data for inserting in left of 4
Enter the data (-1 for no node): -1
Enter data for inserting in right of 4
Enter the data (-1 for no node): -1
Enter data for inserting in right of 6
Enter the data (-1 for no node): 7
Enter data for inserting in left of 7
Enter the data (-1 for no node): -1
Enter data for inserting in right of 7
Enter the data (-1 for no node): -1
Enter data for inserting in right of 8
Enter the data (-1 for no node): 10
Enter data for inserting in left of 10
Enter the data (-1 for no node): -1
Enter data for inserting in right of 10
Enter the data (-1 for no node): 14
Enter data for inserting in left of 14
Enter the data (-1 for no node): 13
Enter data for inserting in left of 13
Enter the data (-1 for no node): -1
Enter data for inserting in right of 13
Enter the data (-1 for no node): -1
Enter data for inserting in right of 14
Enter the data (-1 for no node): -1

Pre-order Traversal: 8 3 1 6 4 7 10 14 13
In-order Traversal: 1 3 4 6 7 8 10 13 14
Post-order Traversal: 1 4 7 6 3 13 14 10 8
```

AQ2:

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = NULL;
    }
};

// Insert data in BST (Tree creation)
Node* insert(Node* root, int value) {
    if (root == NULL)
        return new Node(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);

    return root;
}

// (a) Search Recursive
Node* searchRecursive(Node* root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (key < root->data)
        return searchRecursive(root->left, key);
    return searchRecursive(root->right, key);
}

// (a) Search Iterative / Non-recursive
Node* searchIterative(Node* root, int key) {
    while (root != NULL) {
        if (key == root->data)
            return root;
        else if (key < root->data)
            root = root->left;
        else
            root = root->right;
    }
```

```
    return NULL;
}

// (b) Maximum element of BST
Node* maxNode(Node* root) {
    while (root && root->right)
        root = root->right;
    return root;
}

// (c) Minimum element of BST
Node* minNode(Node* root) {
    while (root && root->left)
        root = root->left;
    return root;
}

// (d) In-order Successor
Node* inorderSuccessor(Node* root, Node* target) {
    if (target->right != NULL)
        return minNode(target->right);

    Node* successor = NULL;
    while (root != NULL) {
        if (target->data < root->data) {
            successor = root;
            root = root->left;
        }
        else if (target->data > root->data)
            root = root->right;
        else
            break;
    }
    return successor;
}

// (e) In-order Predecessor
Node* inorderPredecessor(Node* root, Node* target) {
    if (target->left != NULL)
        return maxNode(target->left);

    Node* predecessor = NULL;
    while (root != NULL) {
        if (target->data > root->data) {
            predecessor = root;
            root = root->right;
        }
        else if (target->data < root->data)
```

```
            root = root->left;
        else
            break;
    }
    return predecessor;
}

// Traversal to verify Tree formation
void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = NULL;
    int n, value;

    cout << "Enter number of nodes: ";
    cin >> n;

    cout << "Enter values: ";
    for (int i = 0; i < n; i++) {
        cin >> value;
        root = insert(root, value);
    }

    cout << "\nBST (In-order Traversal): ";
    inorder(root);
    cout << endl;

    int key;
    cout << "\nEnter element to search: ";
    cin >> key;

    cout << "Recursive Search: ";
    cout << (searchRecursive(root, key) ? "Found" : "Not Found") << endl;

    cout << "Iterative Search: ";
    cout << (searchIterative(root, key) ? "Found" : "Not Found") << endl;

    cout << "\nMinimum element = " << minNode(root)->data << endl;
    cout << "Maximum element = " << maxNode(root)->data << endl;

    Node* target = searchRecursive(root, key);
    if (target != NULL) {
        Node* succ = inorderSuccessor(root, target);
```

```
        Node* pred = inorderPredecessor(root, target);

        cout << "\nIn-order Successor of " << key << ": ";
        if (succ) cout << succ->data; else cout << "None";

        cout << "\nIn-order Predecessor of " << key << ": ";
        if (pred) cout << pred->data; else cout << "None";
    }

    cout << endl;
    return 0;
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\8> cd "f:\Work\SEM3\DSA\LAB\8\" ; if ($?) { g++ AQ2.cpp -o AQ2 } ; if ($?) { .\AQ2 }
Enter number of nodes: 8
Enter values: 9
5
7
2
1
6
4
3

BST (In-order Traversal): 1 2 3 4 5 6 7 9

Enter element to search: 5
Recursive Search: Found
Iterative Search: Found

Minimum element = 1
Maximum element = 9

In-order Successor of 5: 6
In-order Predecessor of 5: 4
PS F:\Work\SEM3\DSA\LAB\8>
```

AQ3:

```
#include <iostream>
#include <algorithm>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = NULL;
    }
};
```

```cpp
// (a) INSERT (No duplicates allowed)
Node* insert(Node* root, int value) {
    if (root == NULL)
        return new Node(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);
    else
        cout << "Duplicate not allowed: " << value << endl;

    return root;
}

// Find min node (used for delete)
Node* minNode(Node* root) {
    while (root->left != NULL)
        root = root->left;
    return root;
}

// (b) DELETE element
Node* deleteNode(Node* root, int key) {
    if (root == NULL)
        return root;

    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {  // Node found

        // Case 1: No child / 1 child
        if (root->left == NULL) {
            Node* temp = root->right;
            delete root;
            return temp;
        }
        if (root->right == NULL) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        // Case 2: 2 children → replace with inorder successor
        Node* temp = minNode(root->right);
        root->data = temp->data;
```

```
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// (c) MAXIMUM depth of BST
int maxDepth(Node* root) {
    if (root == NULL)
        return 0;
    return 1 + max(maxDepth(root->left), maxDepth(root->right));
}

// (d) MINIMUM depth of BST
int minDepth(Node* root) {
    if (root == NULL)
        return 0;

    // If one child is NULL, consider the depth of the other child
    if (root->left == NULL)
        return 1 + minDepth(root->right);
    if (root->right == NULL)
        return 1 + minDepth(root->left);

    return 1 + min(minDepth(root->left), minDepth(root->right));
}

// Inorder traversal (for display)
void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = NULL;
    int choice, value;

    while (true) {
        cout << "\n------ BST MENU ------";
        cout << "\n1. Insert";
        cout << "\n2. Delete";
        cout << "\n3. Display (In-order)";
        cout << "\n4. Maximum Depth";
        cout << "\n5. Minimum Depth";
        cout << "\n6. Exit";
        cout << "\nEnter choice: ";
        cin >> choice;
```

```
        switch (choice) {
        case 1:
            cout << "Enter value to insert: ";
            cin >> value;
            root = insert(root, value);
            break;

        case 2:
            cout << "Enter value to delete: ";
            cin >> value;
            root = deleteNode(root, value);
            break;

        case 3:
            cout << "\nBST in-order: ";
            inorder(root);
            cout << endl;
            break;

        case 4:
            cout << "Maximum Depth of BST = " << maxDepth(root) << endl;
            break;

        case 5:
            cout << "Minimum Depth of BST = " << minDepth(root) << endl;
            break;

        case 6:
            return 0;

        default:
            cout << "Invalid choice!";
        }
    }
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\8> cd "f:\Work\SEM3\DSA\LAB\8\" ; if ($?) { g++ AQ3.cpp -o AQ3 } ; if ($?) { .\AQ3 }

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 1
Enter value to insert: 10

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 1
Enter value to insert: 5

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 1
Enter value to insert: 15

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 1
Enter value to insert: 12
```

```
------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 1
Enter value to insert: 18

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 1
Enter value to insert: 11

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 3

BST in-order: 5 10 11 12 15 18

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 4
Maximum Depth of BST = 4
```

```
------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 5
Minimum Depth of BST = 2

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 2
Enter value to delete: 10

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 3

BST in-order: 5 11 12 15 18

------ BST MENU ------
1. Insert
2. Delete
3. Display (In-order)
4. Maximum Depth
5. Minimum Depth
6. Exit
Enter choice: 6
PS F:\Work\SEM3\DSA\LAB\8>
```

AQ4:

```
#include <iostream>
#include <climits>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = NULL;
    }
};

Node* insertBinaryTree() {
    int value;
    cout << "Enter data (-1 for no node): ";
    cin >> value;

    if (value == -1)
        return NULL;

    Node* root = new Node(value);

    cout << "Enter left child of " << value << endl;
    root->left = insertBinaryTree();

    cout << "Enter right child of " << value << endl;
    root->right = insertBinaryTree();

    return root;
}

bool isBSTUtil(Node* root, long long &prev) {
    if (root == NULL)
        return true;

    if (!isBSTUtil(root->left, prev))
        return false;

    if (root->data <= prev)
        return false;
    prev = root->data;

    return isBSTUtil(root->right, prev);
```

```
}

bool isBST(Node* root) {
    long long prev = LLONG_MIN;
    return isBSTUtil(root, prev);
}

int main() {
    cout << "Create Binary Tree:\n";
    Node* root = insertBinaryTree();

    if (isBST(root))
        cout << "\nThe given binary tree IS a BST\n";
    else
        cout << "\nThe given binary tree is NOT a BST\n";

    return 0;
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\8> cd "f:\Work\SEM3\DSA\LAB\8\" ; if ($?) { g++ AQ4.cpp -o AQ4 } ; if ($?) { .\AQ4 }
Create Binary Tree:
Enter data (-1 for no node): 10
Enter left child of 10
Enter data (-1 for no node): 5
Enter left child of 5
Enter data (-1 for no node): -1
Enter right child of 5
Enter data (-1 for no node): 8
Enter left child of 8
Enter data (-1 for no node): -1
Enter right child of 8
Enter data (-1 for no node): -1
Enter right child of 10
Enter data (-1 for no node): 15
Enter left child of 15
Enter data (-1 for no node): -1
Enter right child of 15
Enter data (-1 for no node): -1

The given binary tree IS a BST
PS F:\Work\SEM3\DSA\LAB\8>
```

AQ5:

```cpp
#include <iostream>
using namespace std;

void heapify(int arr[], int n, int i, bool increasing) {
    int extreme = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (increasing) {
        if (left < n && arr[left] > arr[extreme])
            extreme = left;
        if (right < n && arr[right] > arr[extreme])
            extreme = right;
    } else {
        if (left < n && arr[left] < arr[extreme])
            extreme = left;
        if (right < n && arr[right] < arr[extreme])
            extreme = right;
    }

    if (extreme != i) {
        swap(arr[i], arr[extreme]);
        heapify(arr, n, extreme, increasing);
    }
}

void heapSort(int arr[], int n, bool increasing) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i, increasing);

    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0, increasing);
    }
}

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements: ";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    int choice;
    cout << "\n1. Increasing Order\n2. Decreasing Order\nEnter choice: ";
```

```
    cin >> choice;

    if (choice == 1)
        heapSort(arr, n, true);
    else
        heapSort(arr, n, false);

    cout << "\nSorted Array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\8> cd "f:\Work\SEM3\DSA\LAB\8\" ; if ($?) { g++ AQ5.cpp -o AQ5 } ; if ($?) { .\AQ5 }
Enter number of elements: 6
Enter elements: 9
4
7
2
5
1

1. Increasing Order
2. Decreasing Order
Enter choice: 1

Sorted Array: 1 2 4 5 7 9
PS F:\Work\SEM3\DSA\LAB\8>
```

AQ6:

```
#include <iostream>
using namespace std;

int heap[100];
int size = 0;

void insert(int value) {
    size++;
    heap[size] = value;
    int i = size;

    while (i > 1 && heap[i/2] < heap[i]) {
        swap(heap[i], heap[i/2]);
        i = i / 2;
    }
    cout << value << " inserted\n";
}
```

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

void deleteMax() {
    if (size == 0) {
        cout << "Priority Queue is empty\n";
        return;
    }
    cout << "Deleted max: " << heap[1] << endl;

    heap[1] = heap[size];
    size--;

    int i = 1;
    while (true) {
        int left = 2*i, right = 2*i + 1, largest = i;

        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]);
            i = largest;
        } else break;
    }
}

void display() {
    cout << "Priority Queue: ";
    for (int i = 1; i <= size; i++)
        cout << heap[i] << " ";
    cout << endl;
}

int main() {
    int choice, value;

    while (true) {
        cout << "\n1. Insert\n2. Get Max\n3. Delete Max\n4. Display\n5.
Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
```

```
                cout << "Enter value: ";
                cin >> value;
                insert(value);
                break;
            case 2:
                cout << "Maximum: " << getMax() << endl;
                break;
            case 3:
                deleteMax();
                break;
            case 4:
                display();
                break;
            case 5:
                return 0;
            default:
                cout << "Invalid choice\n";
        }
    }
}
```

Output:

```
PS F:\Work\SEM3\DSA\LAB\8> cd "f:\Work\SEM3\DSA\LAB\8\" ; if ($?) { g++ AQ6.cpp -o AQ6 } ; if ($?) { .\AQ6 }

1. Insert
2. Get Max
3. Delete Max
4. Display
5. Exit
Enter choice: 1
Enter value: 50
50 inserted

1. Insert
2. Get Max
3. Delete Max
4. Display
5. Exit
Enter choice: 1
Enter value: 20
20 inserted

1. Insert
2. Get Max
3. Delete Max
4. Display
5. Exit
Enter choice: 1
Enter value: 60
60 inserted

1. Insert
2. Get Max
3. Delete Max
4. Display
5. Exit
Enter choice: 4
Priority Queue: 60 20 50
```

```
1. Insert
2. Get Max
3. Delete Max
4. Display
5. Exit
Enter choice: 3
Deleted max: 60

1. Insert
2. Get Max
3. Delete Max
4. Display
5. Exit
Enter choice: 4
Priority Queue: 50 20

1. Insert
2. Get Max
3. Delete Max
4. Display
5. Exit
Enter choice: 5
PS F:\Work\SEM3\DSA\LAB\8>
```