Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year:2022), B.Sc. in CSE (Day)

Course Title: Data Structure Lab

Course Code: CSE 106 Section: 221 D7

Lab Project Name: Data Structure Tree

Student Details

Name		ID
1.	Abdullha Hill Oneir	221002044

Submission Date : 03 January, 2023
Course Teacher's Name : Babe Sultana

[For Teachers use only: Don't Write Anything inside this box]

Lab Project Status

Marks: ………………………………… **Signature: ....**

Comments: .... **Date: ....**

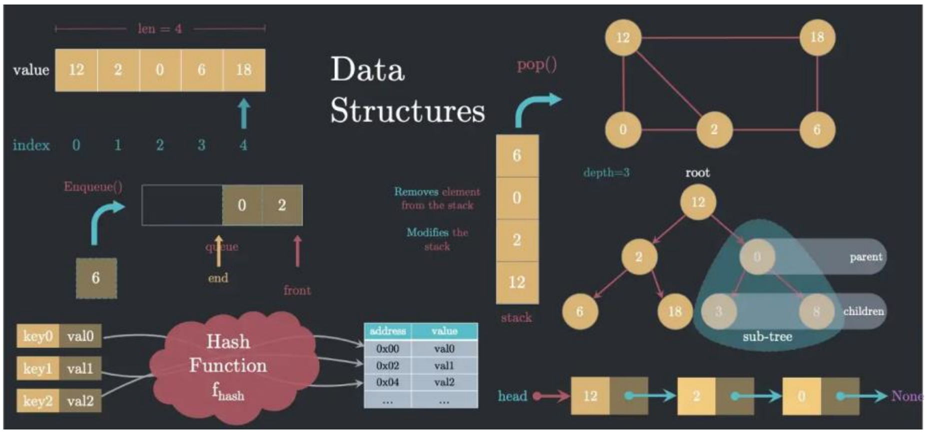
Table of Contents

Chapter 1

Introduction

1.1 Introduction

DATA STRUCTURE TREE program can be very useful in a school, college while studying the data structure. Data Structure is a specialized format of organizing, processing, retrieving and storing data and there are several basic and advanced types of data structure that is used in programs. So, this program can play an important role because this program helps the students with lot of information about data structure, how it works and the algorithms that these data structure types come with. This program comes with a lot of data structure types of programs with practical implementation. Also, it has real life code implementation using data structure.



1.2 Design Goals/Objective

Design Goals

Data Structure Tree can be called as (DST) is an educational program. This program, helps to ease the learning gap between student and the internet in all-in-one compact way. This system exists to simplify information tracking for both students and administrative persons.

Objectives

- User-Friendly Educational System where data of Data Structure types are stored in details.
- To make a system where learning can be easy and time efficient and easier to maintain.

Chapter 2

Design/Development/Implementation of the Project

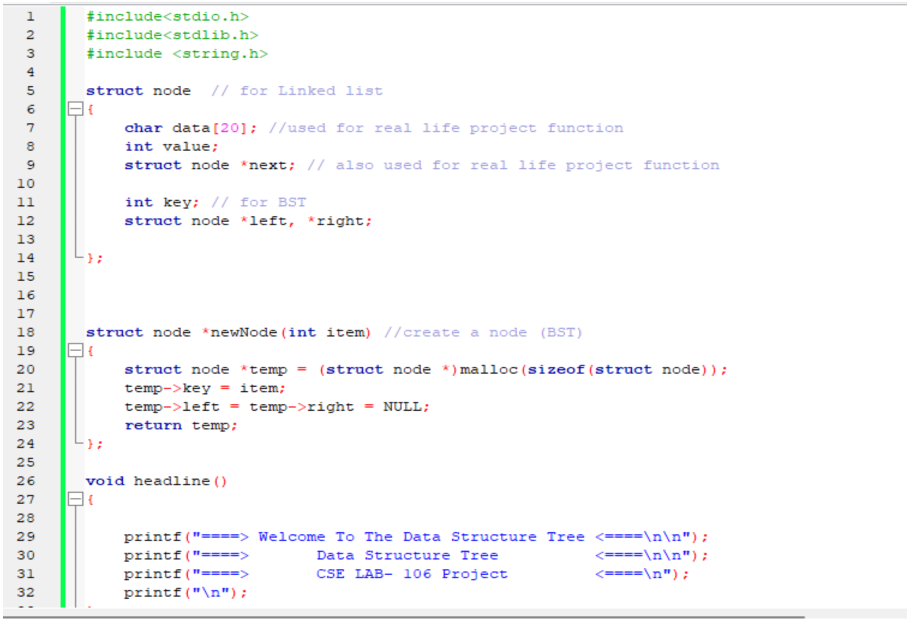
2.1 Section(Development)

For develop this system I have used

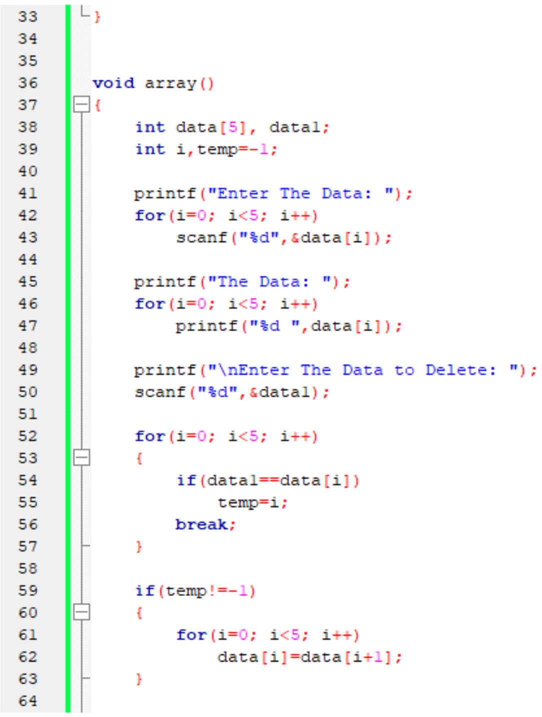
- **Language C**
- **Code Blocks**

2.2 Section (Implementation)

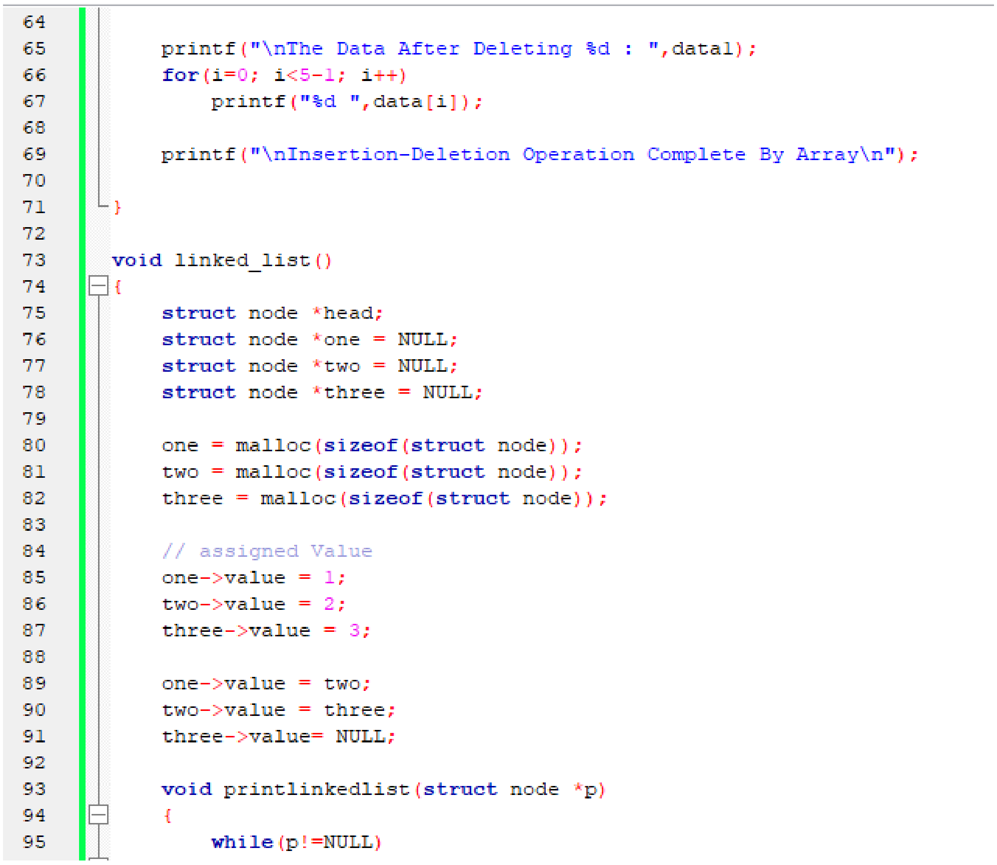
2.2.1 Source Code



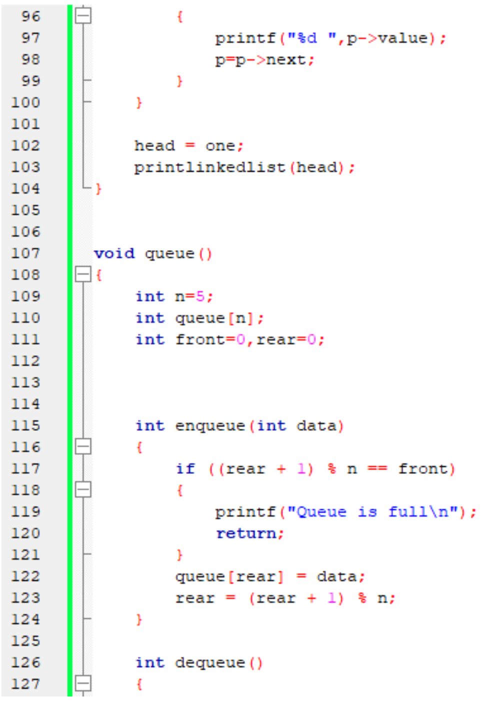
```c
#include<stdio.h>
#include<stdlib.h>
#include <string.h>

struct node  // for Linked list
{
    char data[20]; //used for real life project function
    int value;
    struct node *next; // also used for real life project function

    int key; // for BST
    struct node *left, *right;

};



struct node *newNode(int item) //create a node (BST)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
};

void headline()
{

    printf("====> Welcome To The Data Structure Tree <====\n\n");
    printf("====>       Data Structure Tree         <====\n\n");
    printf("====>       CSE LAB- 106 Project        <====\n");
    printf("\n");
```



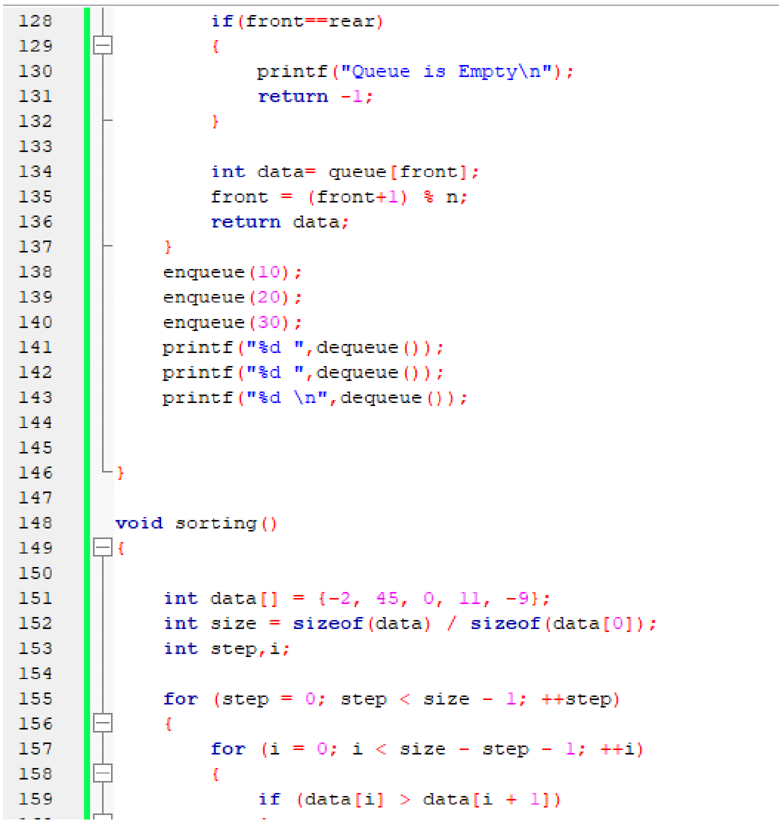
```
}


void array()
{
    int data[5], data1;
    int i,temp=-1;

    printf("Enter The Data: ");
    for(i=0; i<5; i++)
        scanf("%d",&data[i]);

    printf("The Data: ");
    for(i=0; i<5; i++)
        printf("%d ",data[i]);

    printf("\nEnter The Data to Delete: ");
    scanf("%d",&data1);

    for(i=0; i<5; i++)
    {
        if(data1==data[i])
            temp=i;
        break;
    }

    if(temp!=-1)
    {
        for(i=0; i<5; i++)
            data[i]=data[i+1];
    }

```



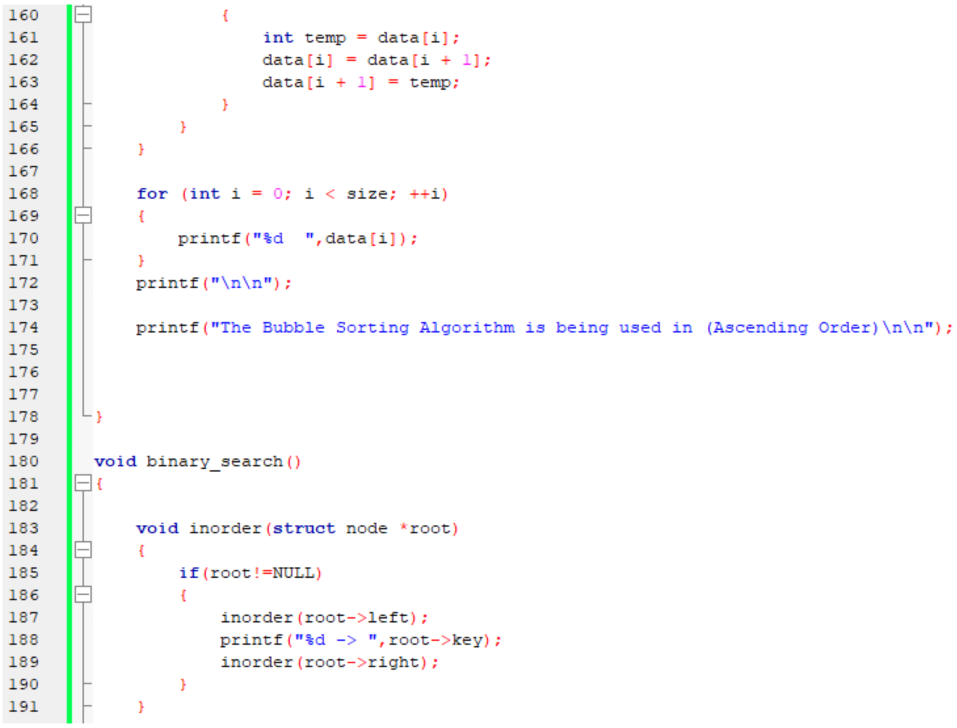
```

    printf("\nThe Data After Deleting %d : ",data1);
    for(i=0; i<5-1; i++)
        printf("%d ",data[i]);

    printf("\nInsertion-Deletion Operation Complete By Array\n");

}

void linked_list()
{
    struct node *head;
    struct node *one = NULL;
    struct node *two = NULL;
    struct node *three = NULL;

    one = malloc(sizeof(struct node));
    two = malloc(sizeof(struct node));
    three = malloc(sizeof(struct node));

    // assigned Value
    one->value = 1;
    two->value = 2;
    three->value = 3;

    one->value = two;
    two->value = three;
    three->value= NULL;

    void printlinkedlist(struct node *p)
    {
        while(p!=NULL)
```



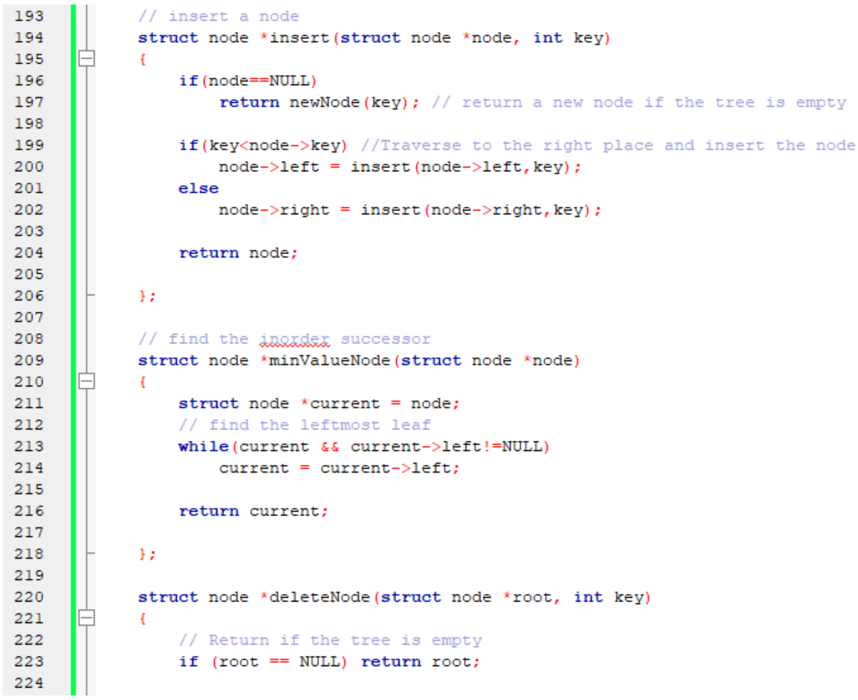
```
        {
            printf("%d ",p->value);
            p=p->next;
        }
    }

    head = one;
    printlinkedlist(head);
}


void queue()
{
    int n=5;
    int queue[n];
    int front=0,rear=0;



    int enqueue(int data)
    {
        if ((rear + 1) % n == front)
        {
            printf("Queue is full\n");
            return;
        }
        queue[rear] = data;
        rear = (rear + 1) % n;
    }

    int dequeue()
    {
```



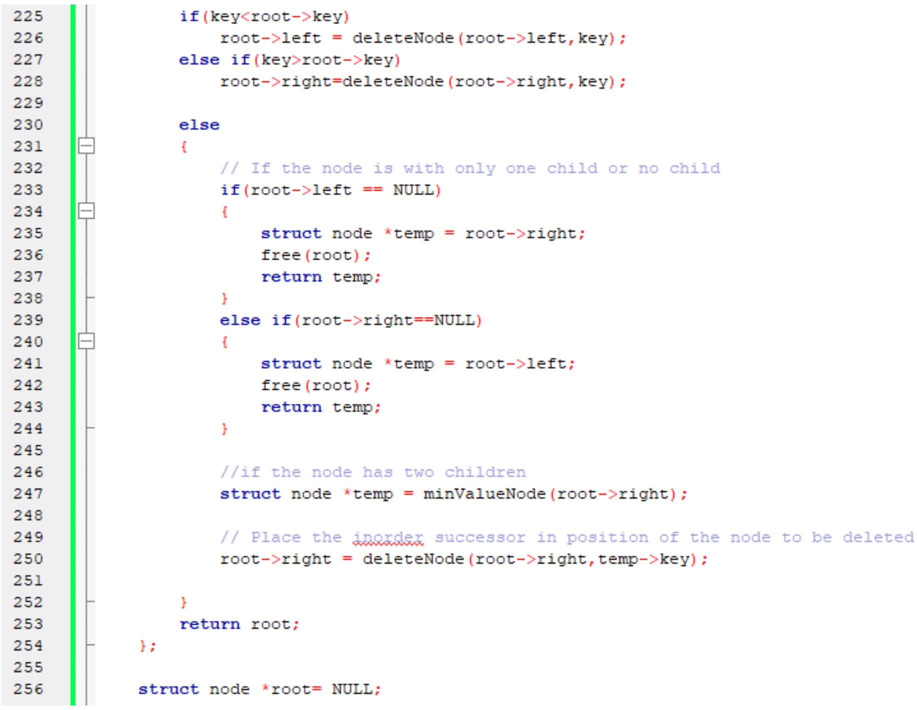
```
        if(front==rear)
        {
            printf("Queue is Empty\n");
            return -1;
        }

        int data= queue[front];
        front = (front+1) % n;
        return data;
    }
    enqueue(10);
    enqueue(20);
    enqueue(30);
    printf("%d ",dequeue());
    printf("%d ",dequeue());
    printf("%d \n",dequeue());


}

void sorting()
{

    int data[] = {-2, 45, 0, 11, -9};
    int size = sizeof(data) / sizeof(data[0]);
    int step,i;

    for (step = 0; step < size - 1; ++step)
    {
        for (i = 0; i < size - step - 1; ++i)
        {
            if (data[i] > data[i + 1])
```



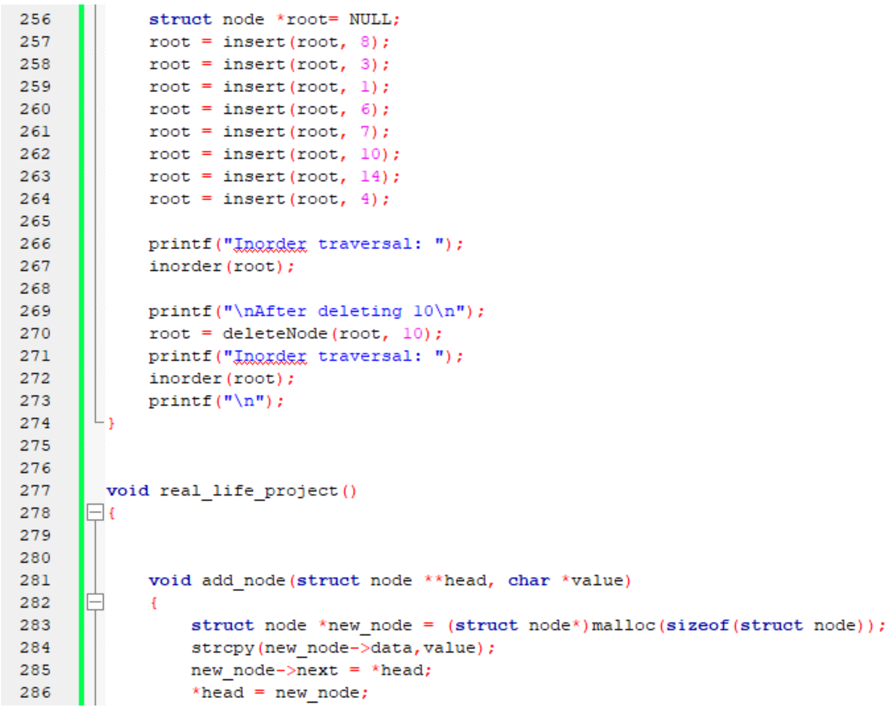
```
            {
                int temp = data[i];
                data[i] = data[i + 1];
                data[i + 1] = temp;
            }
        }
    }

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

    printf("The Bubble Sorting Algorithm is being used in (Ascending Order)\n\n");



}

void binary_search()
{

    void inorder(struct node *root)
    {
        if(root!=NULL)
        {
            inorder(root->left);
            printf("%d -> ",root->key);
            inorder(root->right);
        }
    }
```



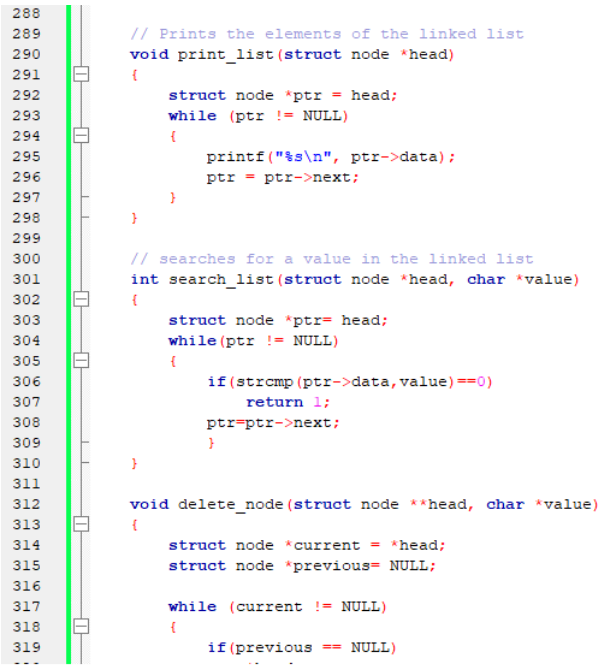
```
    // insert a node
    struct node *insert(struct node *node, int key)
    {
        if(node==NULL)
            return newNode(key); // return a new node if the tree is empty

        if(key<node->key) //Traverse to the right place and insert the node
            node->left = insert(node->left,key);
        else
            node->right = insert(node->right,key);

        return node;

    };

    // find the inorder successor
    struct node *minValueNode(struct node *node)
    {
        struct node *current = node;
        // find the leftmost leaf
        while(current && current->left!=NULL)
            current = current->left;

        return current;

    };

    struct node *deleteNode(struct node *root, int key)
    {
        // Return if the tree is empty
        if (root == NULL) return root;

```



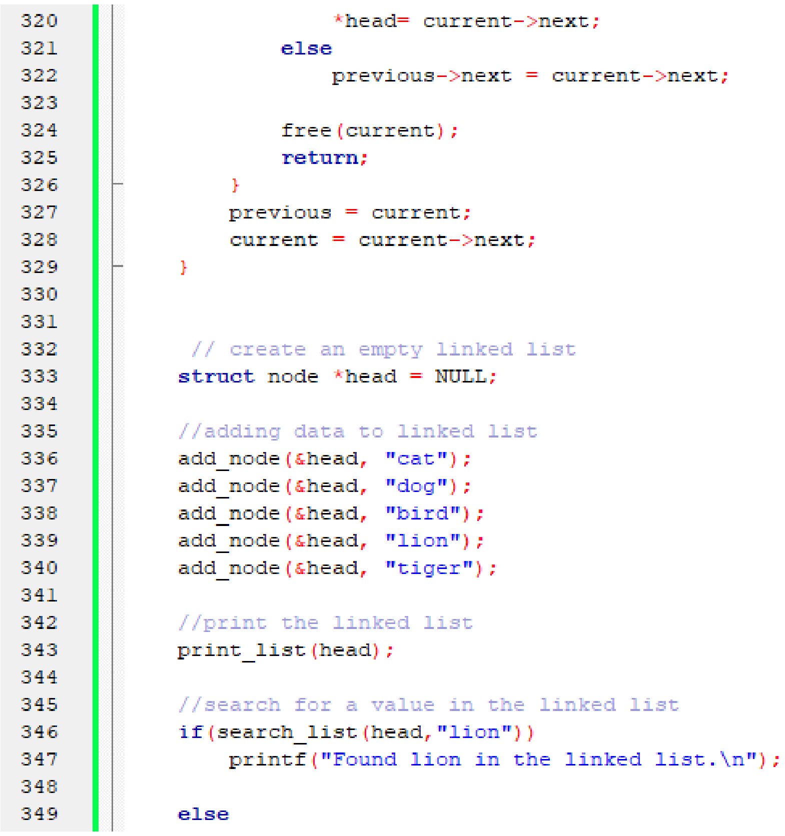
```c
        if(key<root->key)
            root->left = deleteNode(root->left,key);
        else if(key>root->key)
            root->right=deleteNode(root->right,key);

        else
        {
            // If the node is with only one child or no child
            if(root->left == NULL)
            {
                struct node *temp = root->right;
                free(root);
                return temp;
            }
            else if(root->right==NULL)
            {
                struct node *temp = root->left;
                free(root);
                return temp;
            }

            //if the node has two children
            struct node *temp = minValueNode(root->right);

            // Place the inorder successor in position of the node to be deleted
            root->right = deleteNode(root->right,temp->key);

        }
        return root;
    };

    struct node *root= NULL;
```



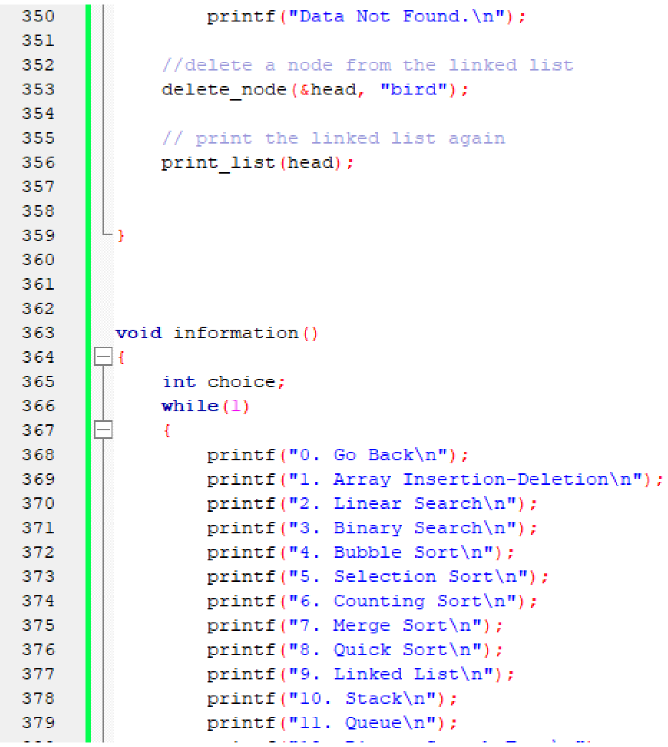
```c
    struct node *root= NULL;
    root = insert(root, 8);
    root = insert(root, 3);
    root = insert(root, 1);
    root = insert(root, 6);
    root = insert(root, 7);
    root = insert(root, 10);
    root = insert(root, 14);
    root = insert(root, 4);

    printf("Inorder traversal: ");
    inorder(root);

    printf("\nAfter deleting 10\n");
    root = deleteNode(root, 10);
    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");
}


void real_life_project()
{


    void add_node(struct node **head, char *value)
    {
        struct node *new_node = (struct node*)malloc(sizeof(struct node));
        strcpy(new_node->data,value);
        new_node->next = *head;
        *head = new_node;
```



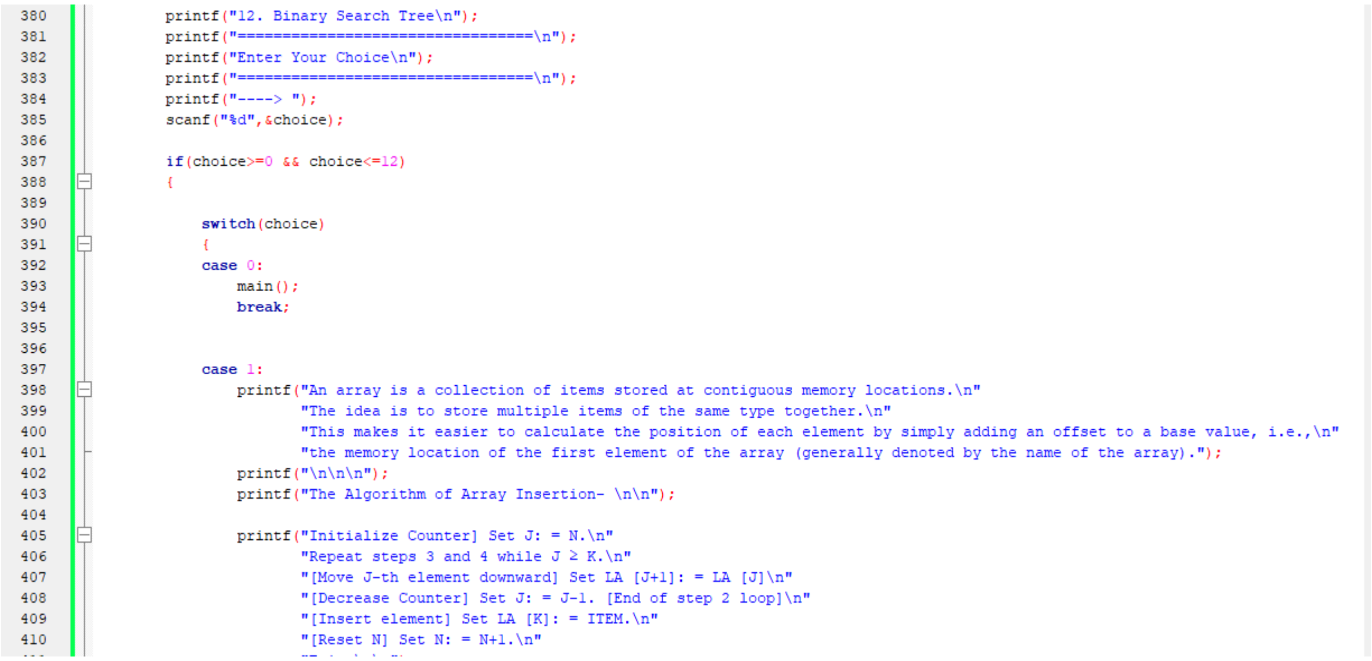
```c

// Prints the elements of the linked list
void print_list(struct node *head)
{
    struct node *ptr = head;
    while (ptr != NULL)
    {
        printf("%s\n", ptr->data);
        ptr = ptr->next;
    }
}

// searches for a value in the linked list
int search_list(struct node *head, char *value)
{
    struct node *ptr= head;
    while(ptr != NULL)
    {
        if(strcmp(ptr->data,value)==0)
            return 1;
        ptr=ptr->next;
        }
}

void delete_node(struct node **head, char *value)
{
    struct node *current = *head;
    struct node *previous= NULL;

    while (current != NULL)
    {
        if(previous == NULL)
```



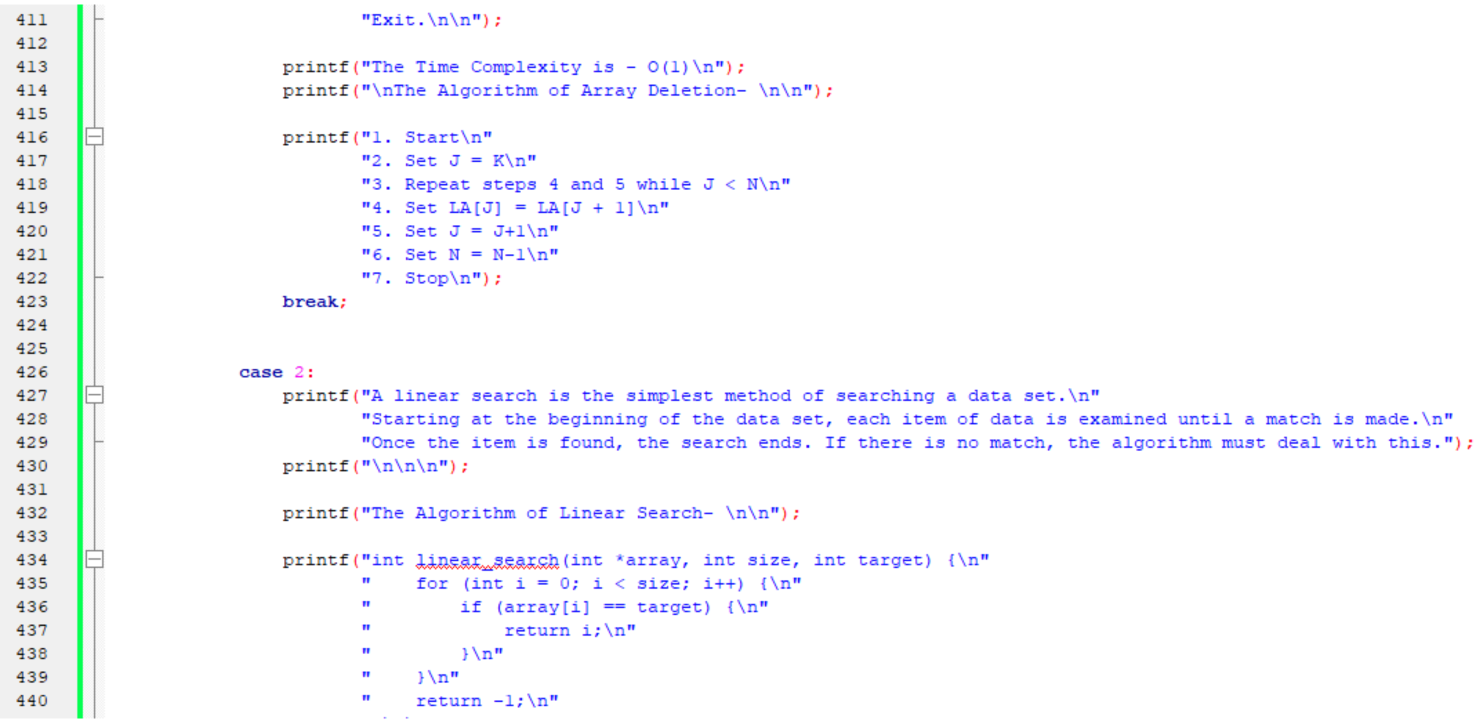
```c
                *head= current->next;
            else
                previous->next = current->next;

            free(current);
            return;
        }
        previous = current;
        current = current->next;
    }


     // create an empty linked list
    struct node *head = NULL;

    //adding data to linked list
    add_node(&head, "cat");
    add_node(&head, "dog");
    add_node(&head, "bird");
    add_node(&head, "lion");
    add_node(&head, "tiger");

    //print the linked list
    print_list(head);

    //search for a value in the linked list
    if(search_list(head,"lion"))
        printf("Found lion in the linked list.\n");

    else
```



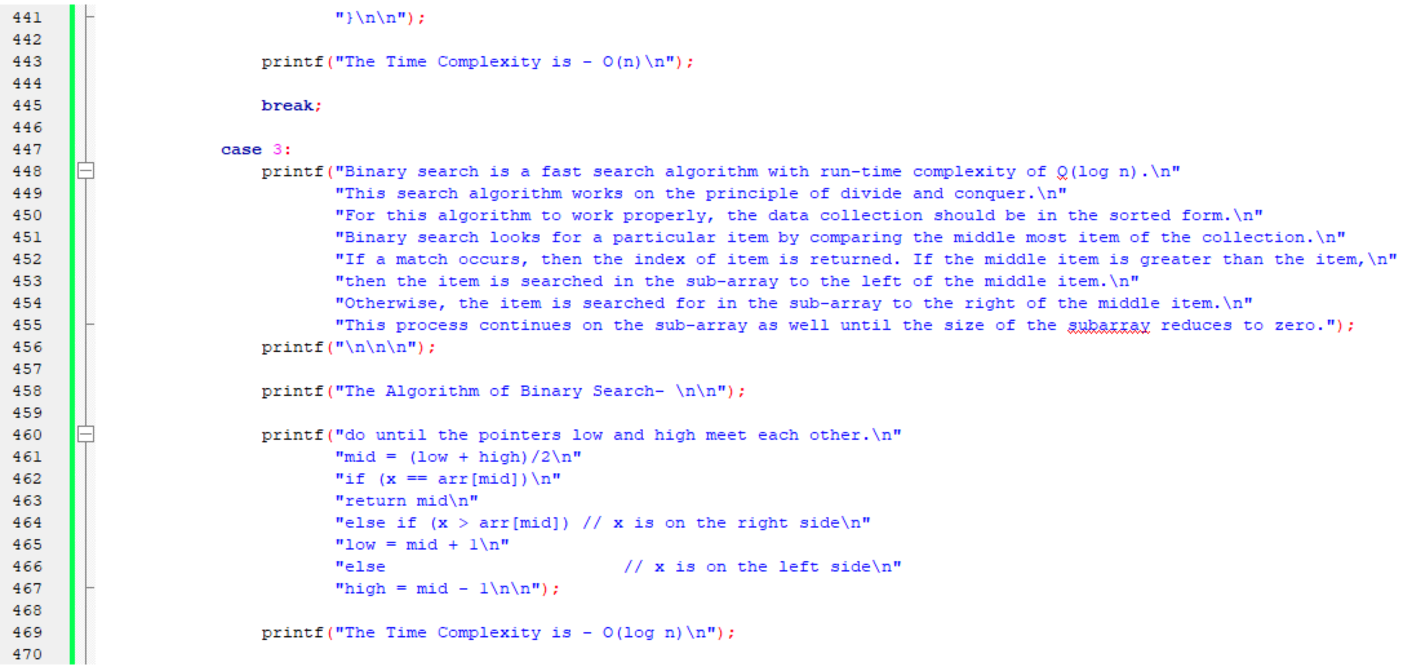
```
        printf("Data Not Found.\n");

    //delete a node from the linked list
    delete_node(&head, "bird");

    // print the linked list again
    print_list(head);


}



void information()
{
    int choice;
    while(1)
    {
        printf("0. Go Back\n");
        printf("1. Array Insertion-Deletion\n");
        printf("2. Linear Search\n");
        printf("3. Binary Search\n");
        printf("4. Bubble Sort\n");
        printf("5. Selection Sort\n");
        printf("6. Counting Sort\n");
        printf("7. Merge Sort\n");
        printf("8. Quick Sort\n");
        printf("9. Linked List\n");
        printf("10. Stack\n");
        printf("11. Queue\n");
```



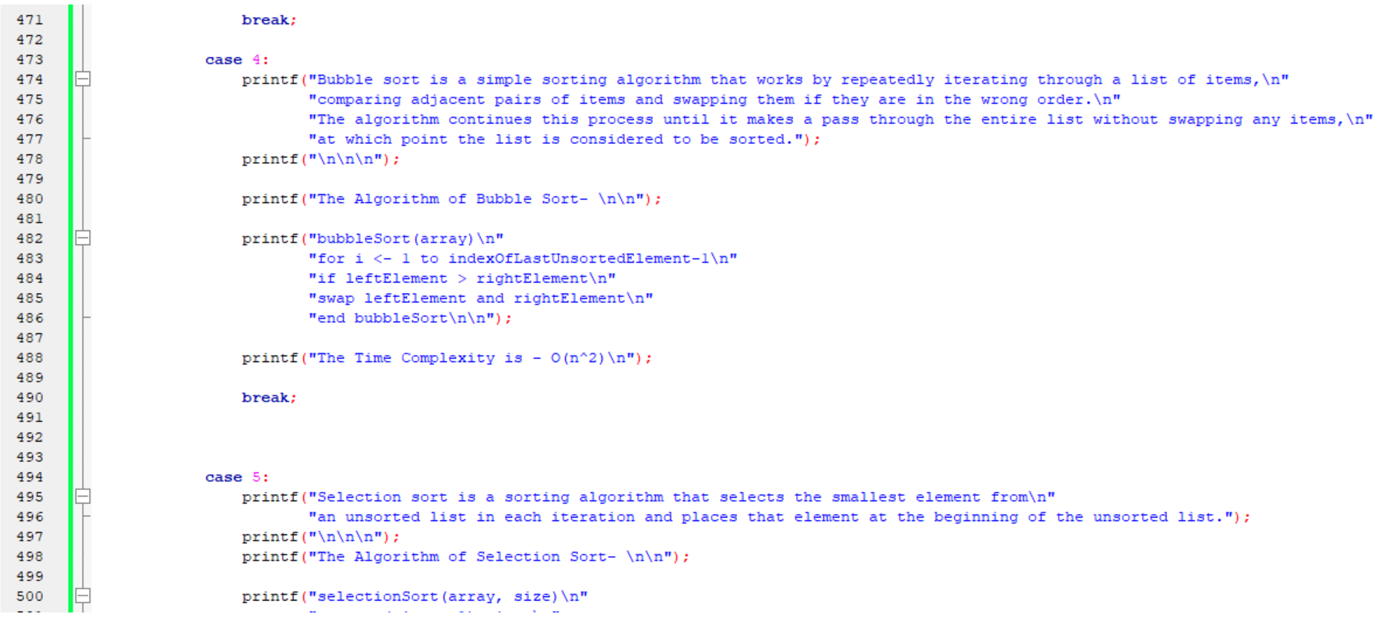
```
        printf("12. Binary Search Tree\n");
        printf("==================================\n");
        printf("Enter Your Choice\n");
        printf("==================================\n");
        printf("----> ");
        scanf("%d",&choice);

        if(choice>=0 && choice<=12)
        {

            switch(choice)
            {
            case 0:
                main();
                break;


            case 1:
                printf("An array is a collection of items stored at contiguous memory locations.\n"
                       "The idea is to store multiple items of the same type together.\n"
                       "This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e.,\n"
                       "the memory location of the first element of the array (generally denoted by the name of the array).");
                printf("\n\n\n");
                printf("The Algorithm of Array Insertion- \n\n");

                printf("Initialize Counter] Set J: = N.\n"
                       "Repeat steps 3 and 4 while J ≥ K.\n"
                       "[Move J-th element downward] Set LA [J+1]: = LA [J]\n"
                       "[Decrease Counter] Set J: = J-1. [End of step 2 loop]\n"
                       "[Insert element] Set LA [K]: = ITEM.\n"
                       "[Reset N] Set N: = N+1.\n"
```



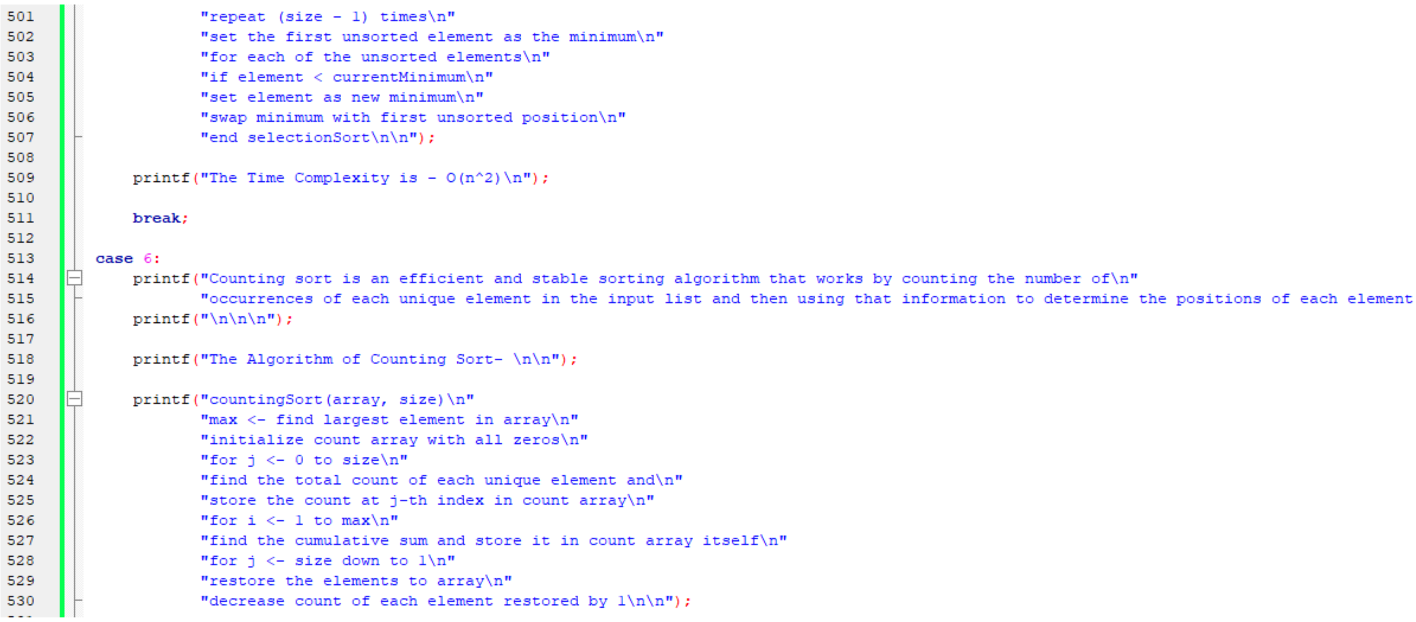
```
               "Exit.\n\n");

        printf("The Time Complexity is - O(1)\n");
        printf("\nThe Algorithm of Array Deletion- \n\n");

        printf("1. Start\n"
               "2. Set J = K\n"
               "3. Repeat steps 4 and 5 while J < N\n"
               "4. Set LA[J] = LA[J + 1]\n"
               "5. Set J = J+1\n"
               "6. Set N = N-1\n"
               "7. Stop\n");
        break;


    case 2:
        printf("A linear search is the simplest method of searching a data set.\n"
               "Starting at the beginning of the data set, each item of data is examined until a match is made.\n"
               "Once the item is found, the search ends. If there is no match, the algorithm must deal with this.");
        printf("\n\n\n");

        printf("The Algorithm of Linear Search- \n\n");

        printf("int linear_search(int *array, int size, int target) {\n"
               "    for (int i = 0; i < size; i++) {\n"
               "        if (array[i] == target) {\n"
               "            return i;\n"
               "        }\n"
               "    }\n"
               "    return -1;\n"
```



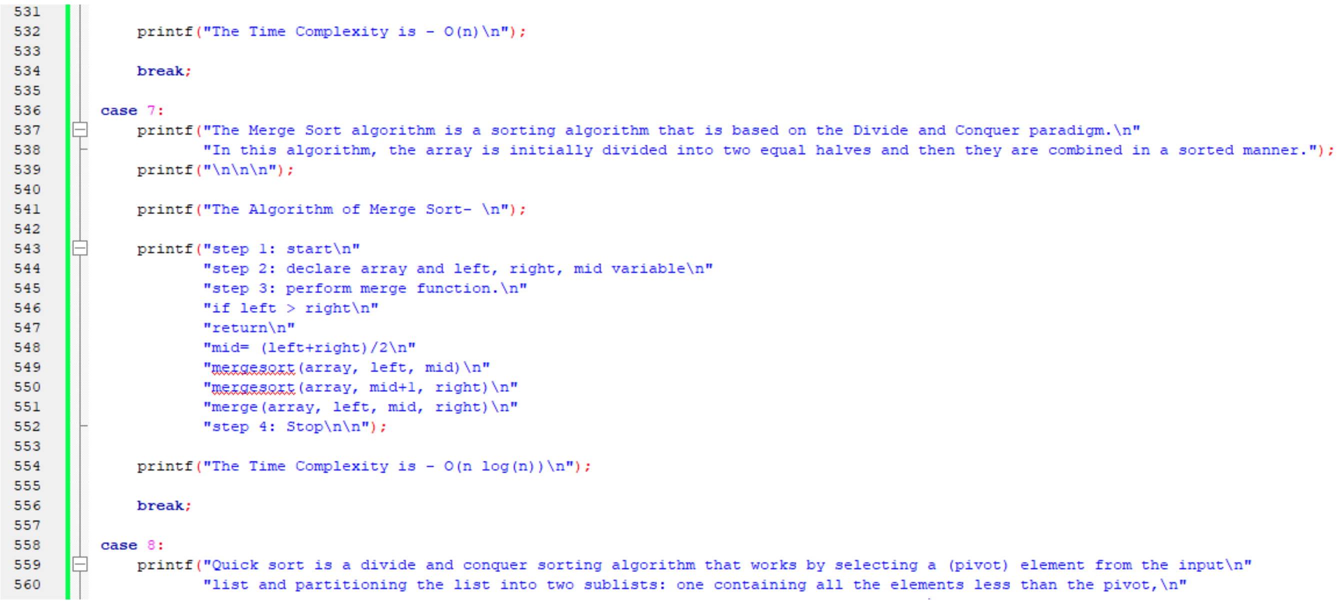
```
               "}\n\n");

        printf("The Time Complexity is - O(n)\n");

        break;

    case 3:
        printf("Binary search is a fast search algorithm with run-time complexity of O(log n).\n"
               "This search algorithm works on the principle of divide and conquer.\n"
               "For this algorithm to work properly, the data collection should be in the sorted form.\n"
               "Binary search looks for a particular item by comparing the middle most item of the collection.\n"
               "If a match occurs, then the index of item is returned. If the middle item is greater than the item,\n"
               "then the item is searched in the sub-array to the left of the middle item.\n"
               "Otherwise, the item is searched for in the sub-array to the right of the middle item.\n"
               "This process continues on the sub-array as well until the size of the subarray reduces to zero.");
        printf("\n\n\n");

        printf("The Algorithm of Binary Search- \n\n");

        printf("do until the pointers low and high meet each other.\n"
               "mid = (low + high)/2\n"
               "if (x == arr[mid])\n"
               "return mid\n"
               "else if (x > arr[mid]) // x is on the right side\n"
               "low = mid + 1\n"
               "else                       // x is on the left side\n"
               "high = mid - 1\n\n");

        printf("The Time Complexity is - O(log n)\n");

```



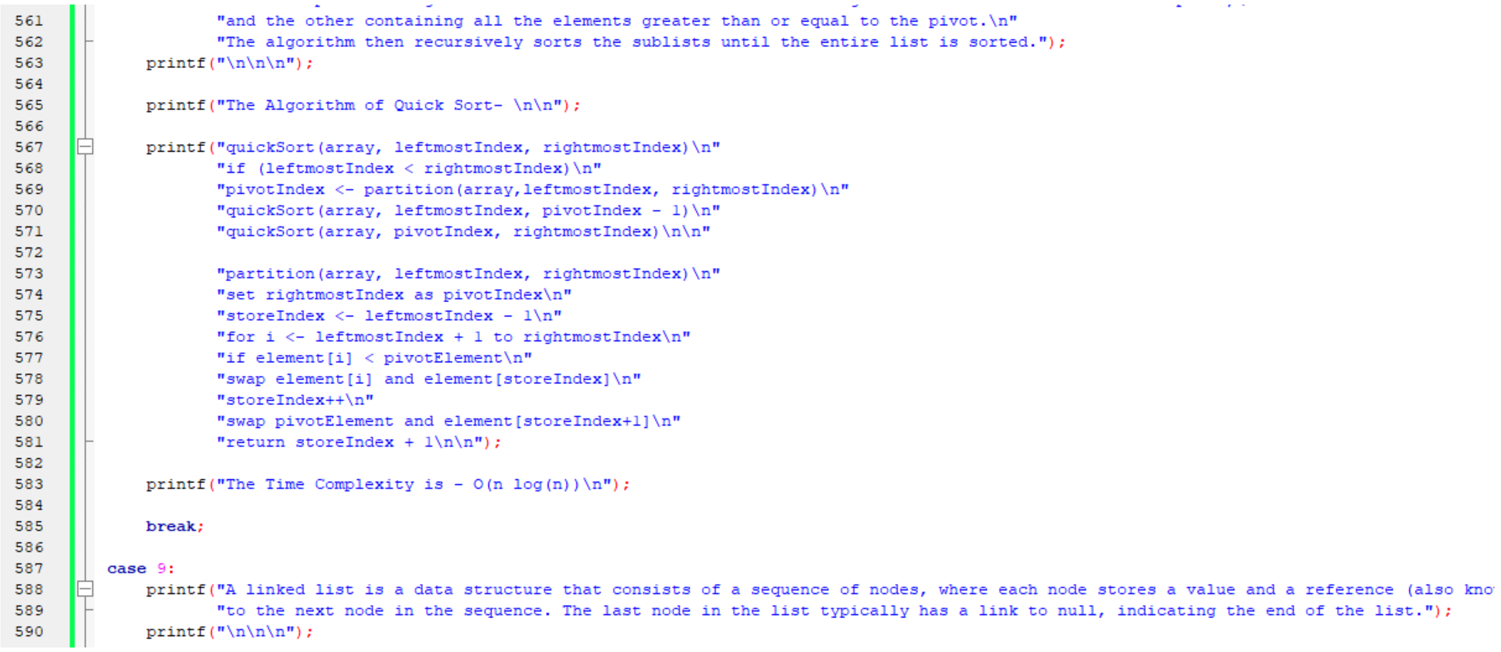
```c
            break;

        case 4:
            printf("Bubble sort is a simple sorting algorithm that works by repeatedly iterating through a list of items,\n"
                   "comparing adjacent pairs of items and swapping them if they are in the wrong order.\n"
                   "The algorithm continues this process until it makes a pass through the entire list without swapping any items,\n"
                   "at which point the list is considered to be sorted.");
            printf("\n\n\n");

            printf("The Algorithm of Bubble Sort- \n\n");

            printf("bubbleSort(array)\n"
                   "for i <- 1 to indexOfLastUnsortedElement-1\n"
                   "if leftElement > rightElement\n"
                   "swap leftElement and rightElement\n"
                   "end bubbleSort\n\n");

            printf("The Time Complexity is - O(n^2)\n");

            break;


        case 5:
            printf("Selection sort is a sorting algorithm that selects the smallest element from\n"
                   "an unsorted list in each iteration and places that element at the beginning of the unsorted list.");
            printf("\n\n\n");
            printf("The Algorithm of Selection Sort- \n\n");

            printf("selectionSort(array, size)\n"
```



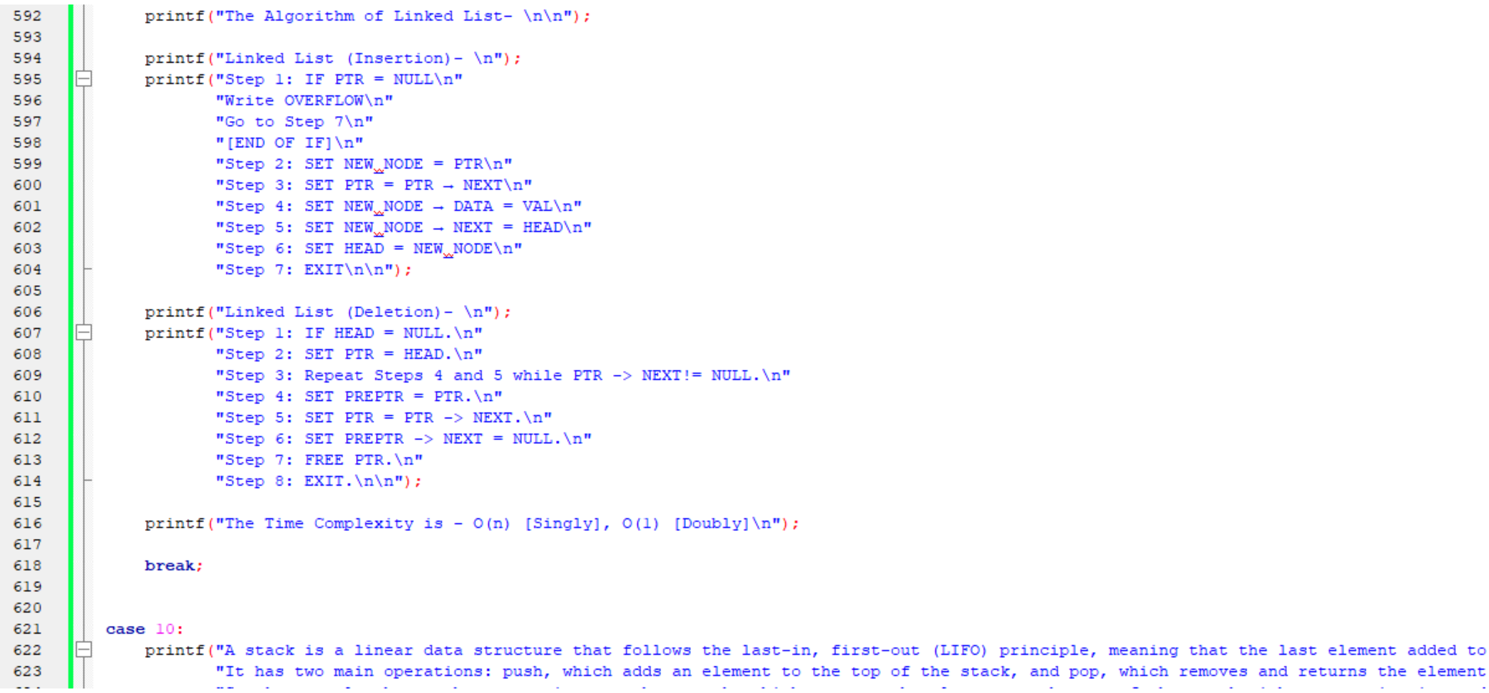
```c
                   "repeat (size - 1) times\n"
                   "set the first unsorted element as the minimum\n"
                   "for each of the unsorted elements\n"
                   "if element < currentMinimum\n"
                   "set element as new minimum\n"
                   "swap minimum with first unsorted position\n"
                   "end selectionSort\n\n");

            printf("The Time Complexity is - O(n^2)\n");

            break;

        case 6:
            printf("Counting sort is an efficient and stable sorting algorithm that works by counting the number of\n"
                   "occurrences of each unique element in the input list and then using that information to determine the positions of each element
            printf("\n\n\n");

            printf("The Algorithm of Counting Sort- \n\n");

            printf("countingSort(array, size)\n"
                   "max <- find largest element in array\n"
                   "initialize count array with all zeros\n"
                   "for j <- 0 to size\n"
                   "find the total count of each unique element and\n"
                   "store the count at j-th index in count array\n"
                   "for i <- 1 to max\n"
                   "find the cumulative sum and store it in count array itself\n"
                   "for j <- size down to 1\n"
                   "restore the elements to array\n"
                   "decrease count of each element restored by 1\n\n");
```



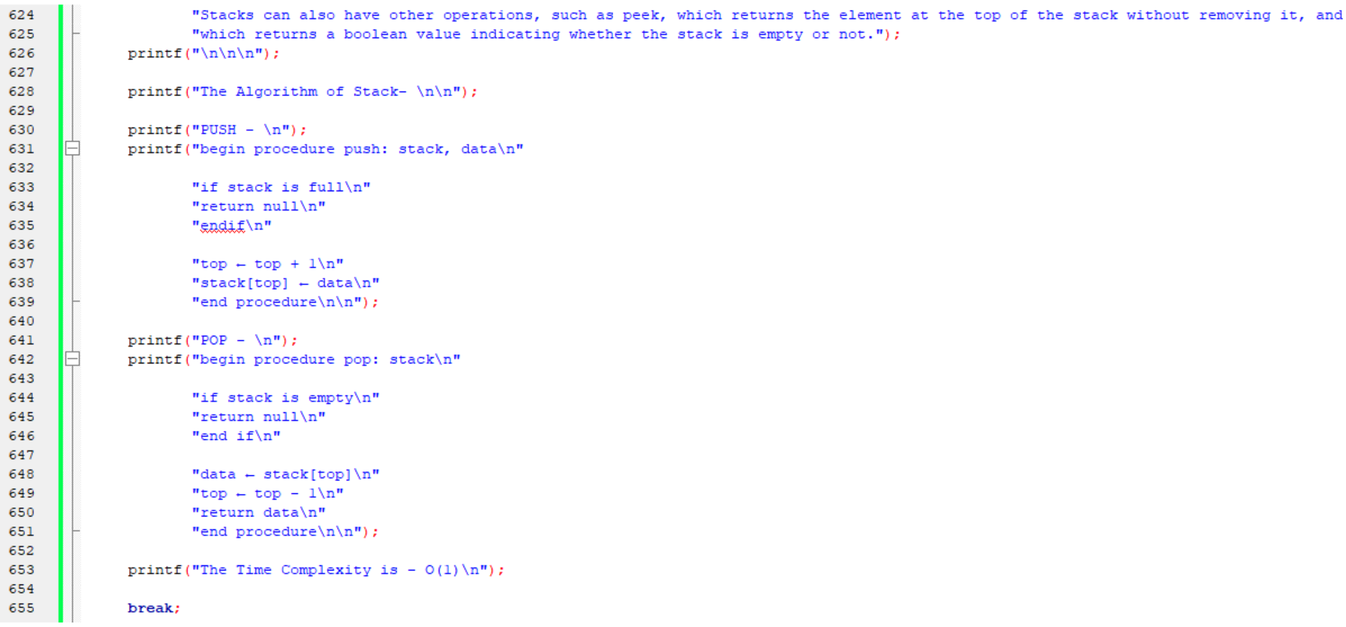
```
        printf("The Time Complexity is - O(n)\n");

        break;

    case 7:
        printf("The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm.\n"
               "In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.");
        printf("\n\n\n");

        printf("The Algorithm of Merge Sort- \n");

        printf("step 1: start\n"
               "step 2: declare array and left, right, mid variable\n"
               "step 3: perform merge function.\n"
               "if left > right\n"
               "return\n"
               "mid= (left+right)/2\n"
               "mergesort(array, left, mid)\n"
               "mergesort(array, mid+1, right)\n"
               "merge(array, left, mid, right)\n"
               "step 4: Stop\n\n");

        printf("The Time Complexity is - O(n log(n))\n");

        break;

    case 8:
        printf("Quick sort is a divide and conquer sorting algorithm that works by selecting a (pivot) element from the input\n"
               "list and partitioning the list into two sublists: one containing all the elements less than the pivot,\n"
```



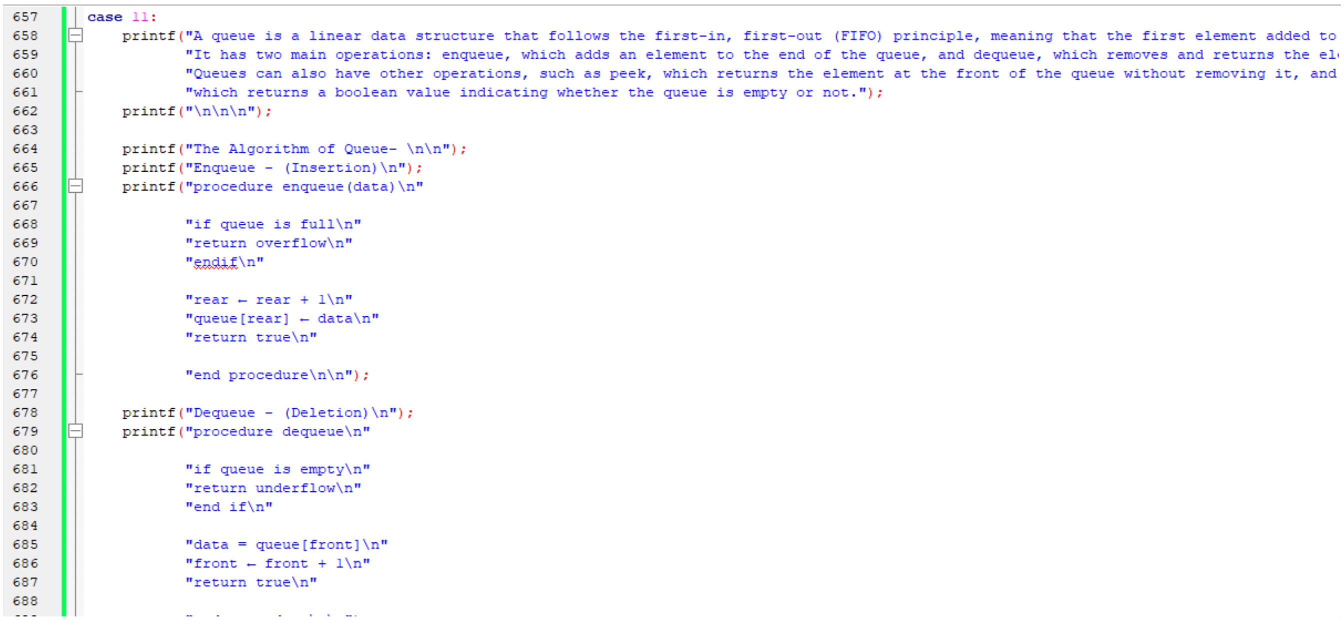
```
               "and the other containing all the elements greater than or equal to the pivot.\n"
               "The algorithm then recursively sorts the sublists until the entire list is sorted.");
        printf("\n\n\n");

        printf("The Algorithm of Quick Sort- \n\n");

        printf("quickSort(array, leftmostIndex, rightmostIndex)\n"
               "if (leftmostIndex < rightmostIndex)\n"
               "pivotIndex <- partition(array,leftmostIndex, rightmostIndex)\n"
               "quickSort(array, leftmostIndex, pivotIndex - 1)\n"
               "quickSort(array, pivotIndex, rightmostIndex)\n\n"

               "partition(array, leftmostIndex, rightmostIndex)\n"
               "set rightmostIndex as pivotIndex\n"
               "storeIndex <- leftmostIndex - 1\n"
               "for i <- leftmostIndex + 1 to rightmostIndex\n"
               "if element[i] < pivotElement\n"
               "swap element[i] and element[storeIndex]\n"
               "storeIndex++\n"
               "swap pivotElement and element[storeIndex+1]\n"
               "return storeIndex + 1\n\n");

        printf("The Time Complexity is - O(n log(n))\n");

        break;

    case 9:
        printf("A linked list is a data structure that consists of a sequence of nodes, where each node stores a value and a reference (also kno
               "to the next node in the sequence. The last node in the list typically has a link to null, indicating the end of the list.");
        printf("\n\n\n");
```



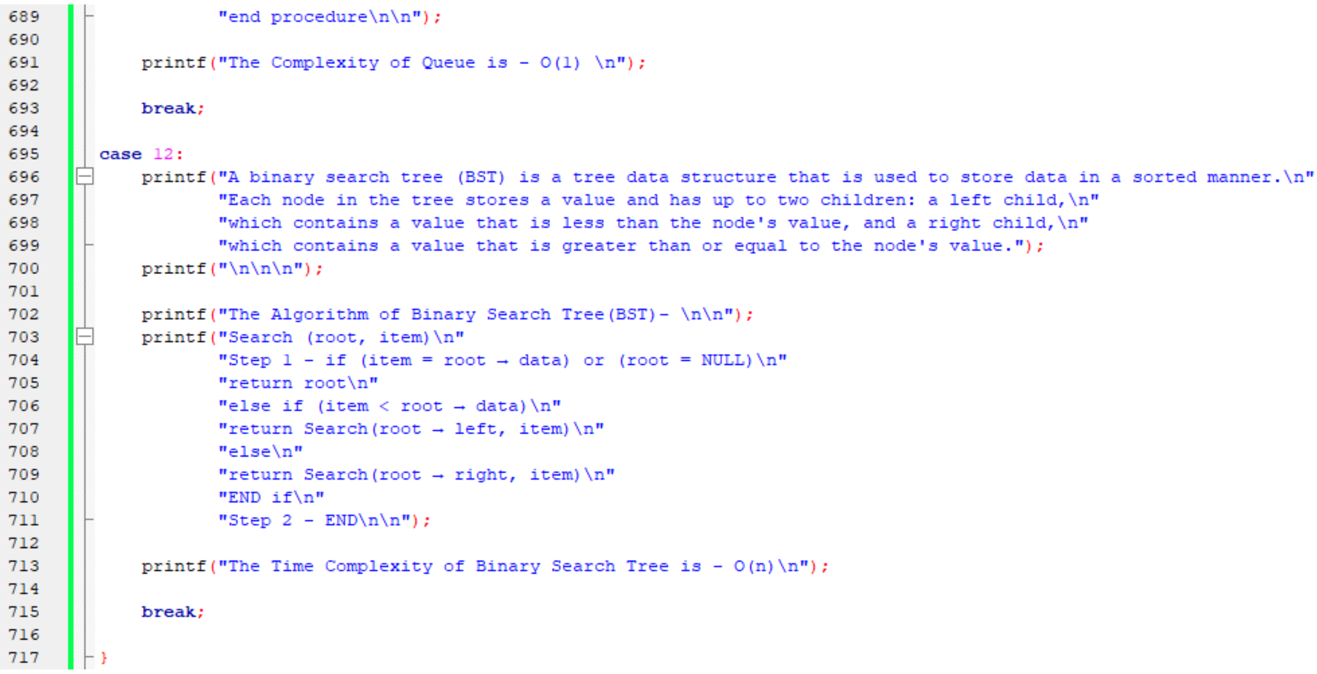
```
    printf("The Algorithm of Linked List- \n\n");

    printf("Linked List (Insertion)- \n");
    printf("Step 1: IF PTR = NULL\n"
           "Write OVERFLOW\n"
           "Go to Step 7\n"
           "[END OF IF]\n"
           "Step 2: SET NEW_NODE = PTR\n"
           "Step 3: SET PTR = PTR → NEXT\n"
           "Step 4: SET NEW_NODE → DATA = VAL\n"
           "Step 5: SET NEW_NODE → NEXT = HEAD\n"
           "Step 6: SET HEAD = NEW_NODE\n"
           "Step 7: EXIT\n\n");

    printf("Linked List (Deletion)- \n");
    printf("Step 1: IF HEAD = NULL.\n"
           "Step 2: SET PTR = HEAD.\n"
           "Step 3: Repeat Steps 4 and 5 while PTR -> NEXT!= NULL.\n"
           "Step 4: SET PREPTR = PTR.\n"
           "Step 5: SET PTR = PTR -> NEXT.\n"
           "Step 6: SET PREPTR -> NEXT = NULL.\n"
           "Step 7: FREE PTR.\n"
           "Step 8: EXIT.\n\n");

    printf("The Time Complexity is - O(n) [Singly], O(1) [Doubly]\n");

    break;


case 10:
    printf("A stack is a linear data structure that follows the last-in, first-out (LIFO) principle, meaning that the last element added to
           "It has two main operations: push, which adds an element to the top of the stack, and pop, which removes and returns the element
```



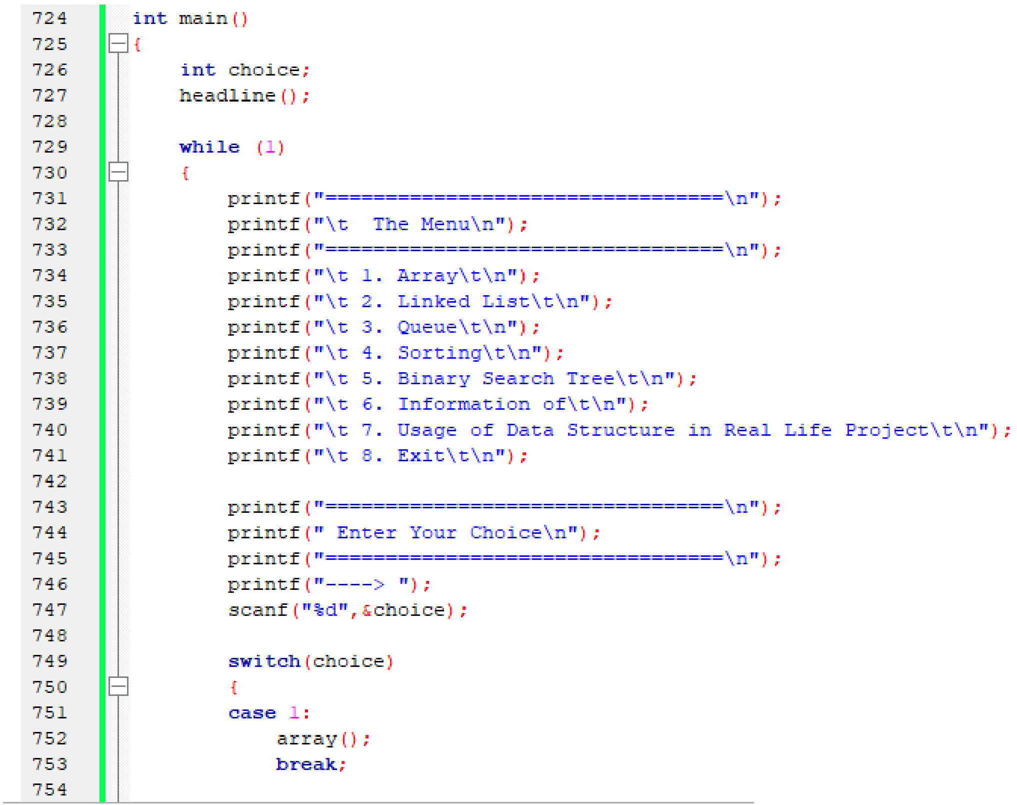
```
           "Stacks can also have other operations, such as peek, which returns the element at the top of the stack without removing it, and
           "which returns a boolean value indicating whether the stack is empty or not.");
    printf("\n\n\n");

    printf("The Algorithm of Stack- \n\n");

    printf("PUSH - \n");
    printf("begin procedure push: stack, data\n"

           "if stack is full\n"
           "return null\n"
           "endif\n"

           "top ← top + 1\n"
           "stack[top] ← data\n"
           "end procedure\n\n");

    printf("POP - \n");
    printf("begin procedure pop: stack\n"

           "if stack is empty\n"
           "return null\n"
           "end if\n"

           "data ← stack[top]\n"
           "top ← top - 1\n"
           "return data\n"
           "end procedure\n\n");

    printf("The Time Complexity is - O(1)\n");

    break;
```



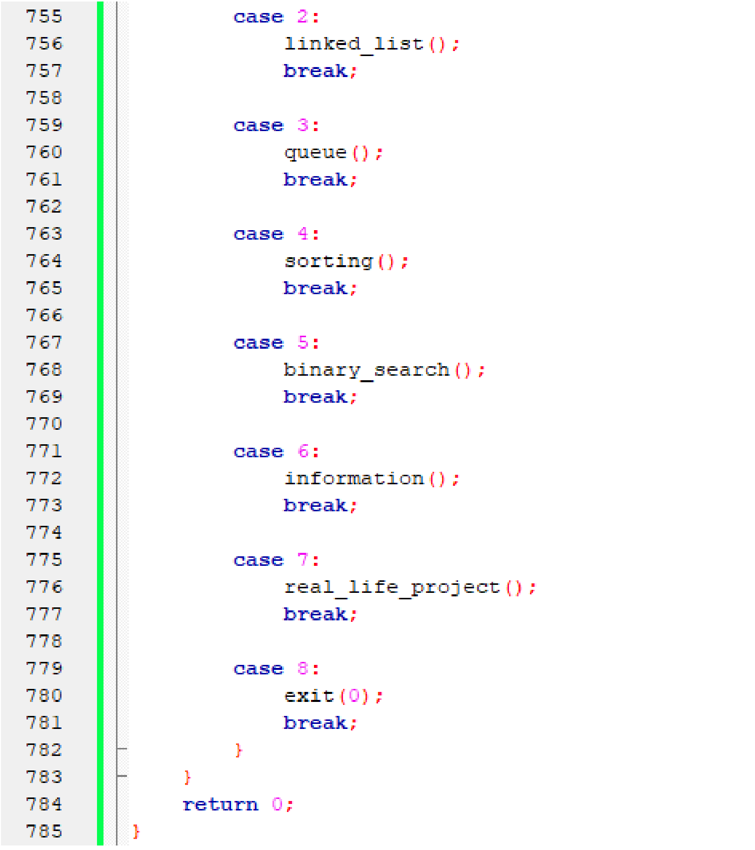
```
case 11:
    printf("A queue is a linear data structure that follows the first-in, first-out (FIFO) principle, meaning that the first element added to
           "It has two main operations: enqueue, which adds an element to the end of the queue, and dequeue, which removes and returns the el
           "Queues can also have other operations, such as peek, which returns the element at the front of the queue without removing it, and
           "which returns a boolean value indicating whether the queue is empty or not.");
    printf("\n\n\n");

    printf("The Algorithm of Queue- \n\n");
    printf("Enqueue - (Insertion)\n");
    printf("procedure enqueue(data)\n"

           "if queue is full\n"
           "return overflow\n"
           "endif\n"

           "rear ← rear + 1\n"
           "queue[rear] ← data\n"
           "return true\n"

           "end procedure\n\n");

    printf("Dequeue - (Deletion)\n");
    printf("procedure dequeue\n"

           "if queue is empty\n"
           "return underflow\n"
           "end if\n"

           "data = queue[front]\n"
           "front ← front + 1\n"
           "return true\n"

```



```
           "end procedure\n\n");

    printf("The Complexity of Queue is - O(1) \n");

    break;

case 12:
    printf("A binary search tree (BST) is a tree data structure that is used to store data in a sorted manner.\n"
           "Each node in the tree stores a value and has up to two children: a left child,\n"
           "which contains a value that is less than the node's value, and a right child,\n"
           "which contains a value that is greater than or equal to the node's value.");
    printf("\n\n\n");

    printf("The Algorithm of Binary Search Tree(BST)- \n\n");
    printf("Search (root, item)\n"
           "Step 1 - if (item = root → data) or (root = NULL)\n"
           "return root\n"
           "else if (item < root → data)\n"
           "return Search(root → left, item)\n"
           "else\n"
           "return Search(root → right, item)\n"
           "END if\n"
           "Step 2 - END\n\n");

    printf("The Time Complexity of Binary Search Tree is - O(n)\n");

    break;

}
```



```c
int main()
{
    int choice;
    headline();

    while (1)
    {
        printf("=================================\n");
        printf("\t  The Menu\n");
        printf("=================================\n");
        printf("\t 1. Array\t\n");
        printf("\t 2. Linked List\t\n");
        printf("\t 3. Queue\t\n");
        printf("\t 4. Sorting\t\n");
        printf("\t 5. Binary Search Tree\t\n");
        printf("\t 6. Information of\t\n");
        printf("\t 7. Usage of Data Structure in Real Life Project\t\n");
        printf("\t 8. Exit\t\n");

        printf("=================================\n");
        printf(" Enter Your Choice\n");
        printf("=================================\n");
        printf("----> ");
        scanf("%d",&choice);

        switch(choice)
        {
        case 1:
            array();
            break;

```



```c
        case 2:
            linked_list();
            break;

        case 3:
            queue();
            break;

        case 4:
            sorting();
            break;

        case 5:
            binary_search();
            break;

        case 6:
            information();
            break;

        case 7:
            real_life_project();
            break;

        case 8:
            exit(0);
            break;
        }
    }
    return 0;
}
```

Chapter 3

Performance Evaluation

3.1 Section (Development Tools)

For Develop this system I used C Language. Codeblocks IDE used for writing the C language

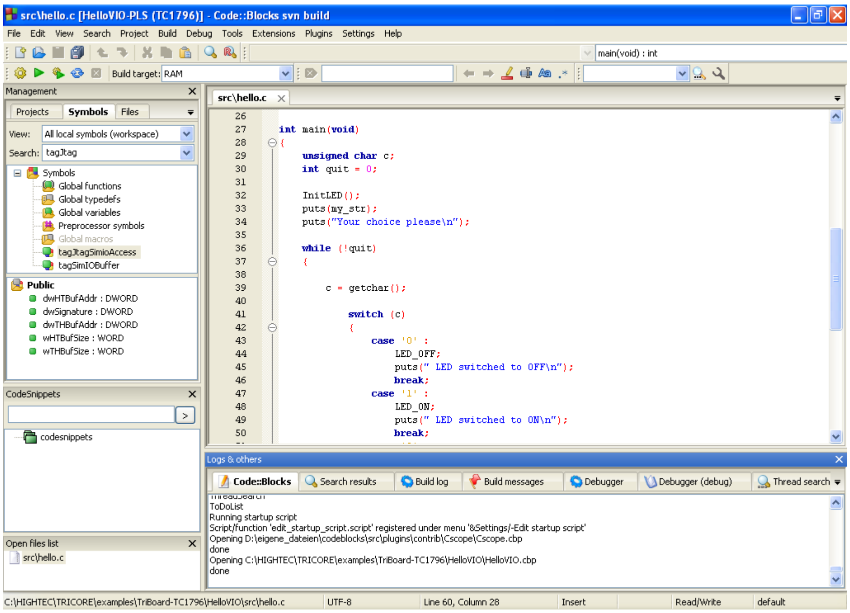
3.1.1. Introduce with C language

C is a procedural programming language with a static system that has the functionality of structured programming, recursion, and lexical variable scoping. C was created with constructs that transfer well to common hardware instructions. It has a long history of use in programs that were previously written in assembly language.

```
string sInput;
int iLength, iN;
double dblTemp;
bool again = true;

while (again) {
    iN = -1;
    again = false;
    getline(cin, sInput);
    system("cls");
    stringstream(sInput) >> dblTemp;
    iLength = sInput.length();
    if (iLength < 4) {
        again = true;
        continue;
    } else if (sInput[iLength - 3] != '.') {
        again = true;
        continue;
    } while (++iN < iLength) {
        if (isdigit(sInput[iN])) {
            continue;
        } else if (iN == (iLength - 3) ) {
            continue;
```

3.1.2. Introduce with CodeBlocks

Code::Blocks is a free, open-source cross-platform IDE that supports multiple compilers including GCC, Clang and Visual C++. It is developed in C++ using wxWidgets as the GUI toolkit. Using a plugin architecture, its capabilities and features are defined by the provided plugins



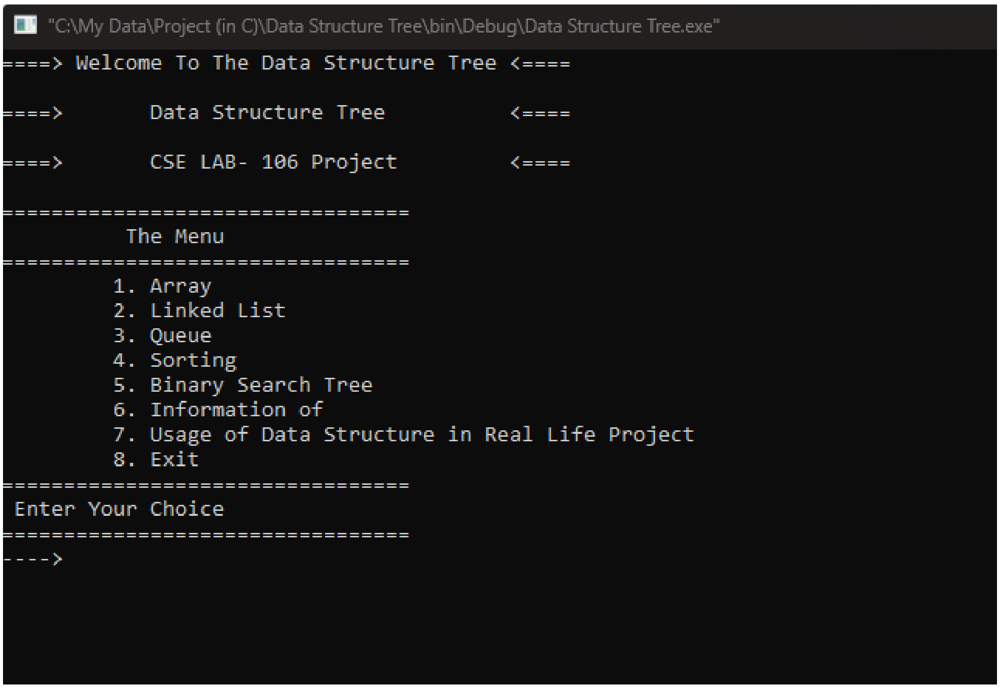
```

int main(void)
{
    unsigned char c;
    int quit = 0;

    InitLED();
    puts(my_str);
    puts("Your choice please\n");

    while (!quit)
    {

        c = getchar();

            switch (c)
            {
                case '0' :
                    LED_OFF;
                    puts(" LED switched to OFF\n");
                    break;
                case '1' :
                    LED_ON;
                    puts(" LED switched to ON\n");
                    break;
```

```
ToDoList
Running startup script
Script/function 'edit_startup_script.script' registered under menu '&Settings/-Edit startup script'
Opening D:\eigene_dateien\codeblocks\src\plugins\contrib\Cscope\Cscope.cbp
done
Opening C:\HIGHTEC\TRICORE\examples\TriBoard-TC1796\HelloVIO\HelloVIO.cbp
done
```

C:\HIGHTEC\TRICORE\examples\TriBoard-TC1796\HelloVIO\src\hello.c | UTF-8 | Line 60, Column 28 | Insert | Read/Write | default

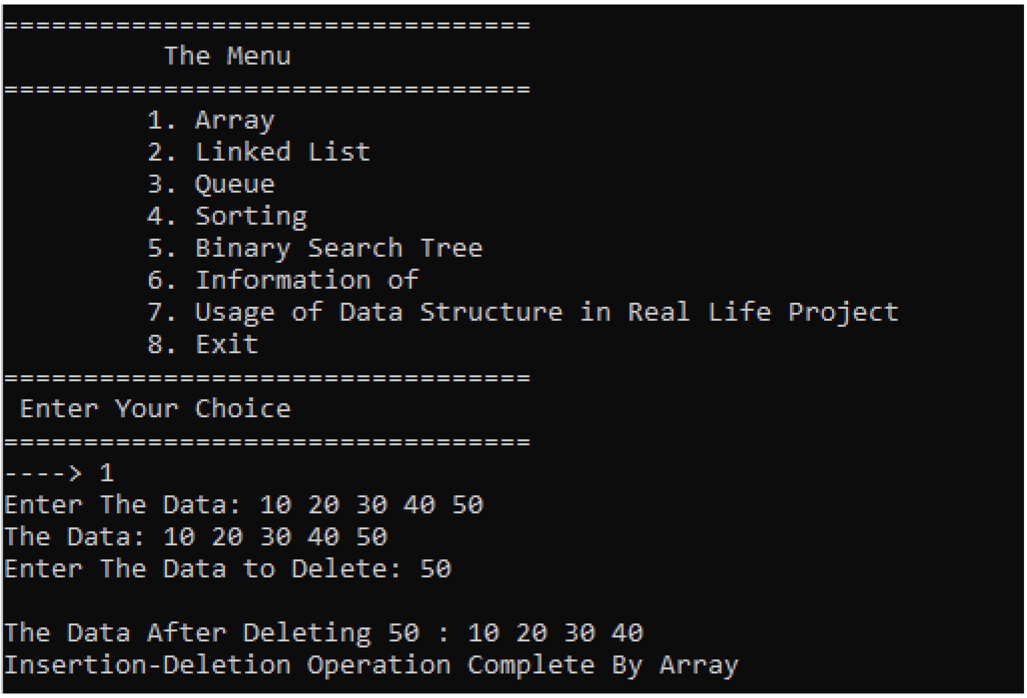
3.2 Results and Discussions

3.2.1 Results



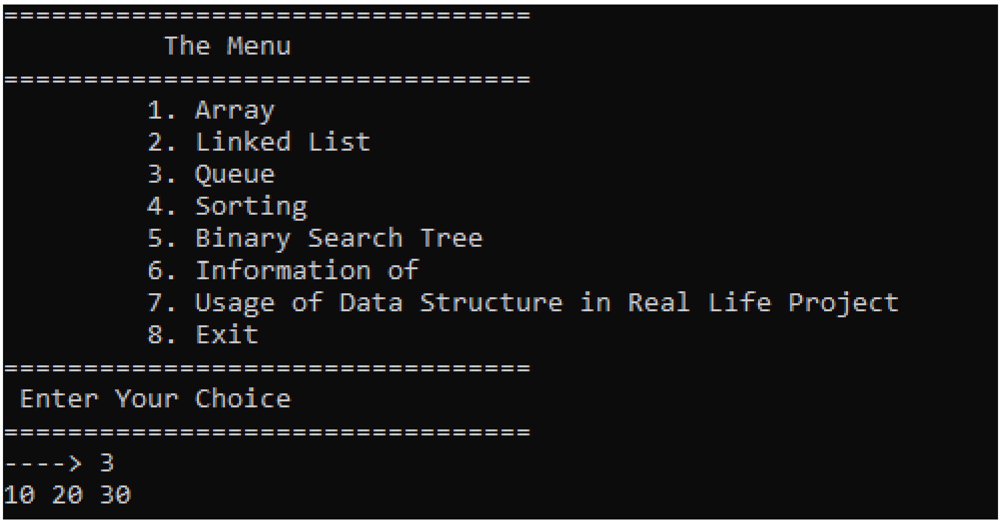
```
"C:\My Data\Project (in C)\Data Structure Tree\bin\Debug\Data Structure Tree.exe"
====> Welcome To The Data Structure Tree <====

====>       Data Structure Tree           <====

====>       CSE LAB- 106 Project          <====

=================================
          The Menu
=================================
         1. Array
         2. Linked List
         3. Queue
         4. Sorting
         5. Binary Search Tree
         6. Information of
         7. Usage of Data Structure in Real Life Project
         8. Exit
=================================
 Enter Your Choice
=================================
---->
```

Figure: Home page



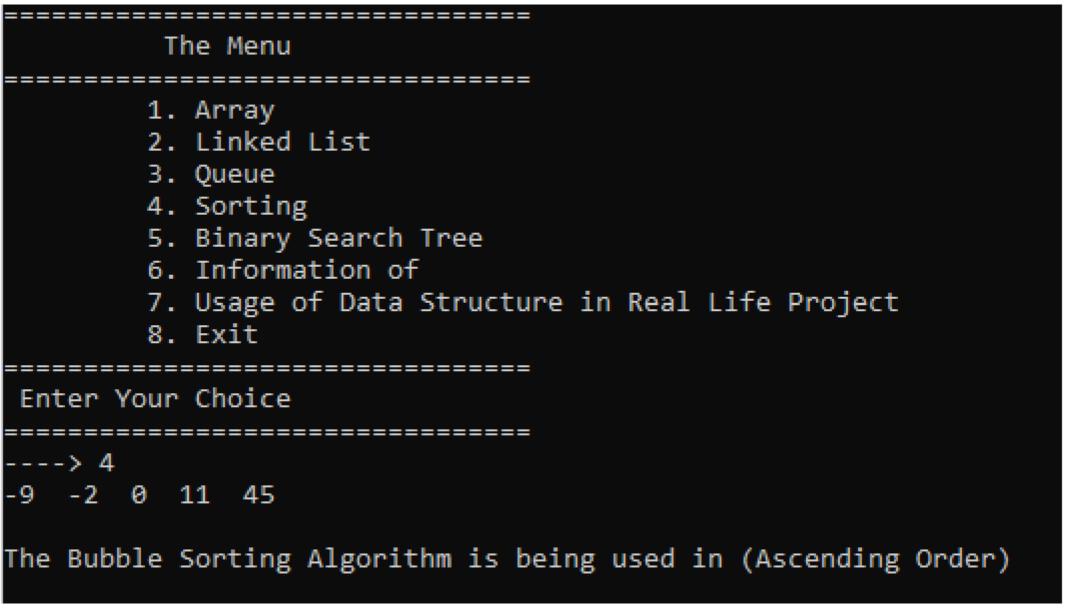
```
=================================
          The Menu
=================================
         1. Array
         2. Linked List
         3. Queue
         4. Sorting
         5. Binary Search Tree
         6. Information of
         7. Usage of Data Structure in Real Life Project
         8. Exit
=================================
 Enter Your Choice
=================================
----> 1
Enter The Data: 10 20 30 40 50
The Data: 10 20 30 40 50
Enter The Data to Delete: 50

The Data After Deleting 50 : 10 20 30 40
Insertion-Deletion Operation Complete By Array
```

Figure: Choosing Task 1



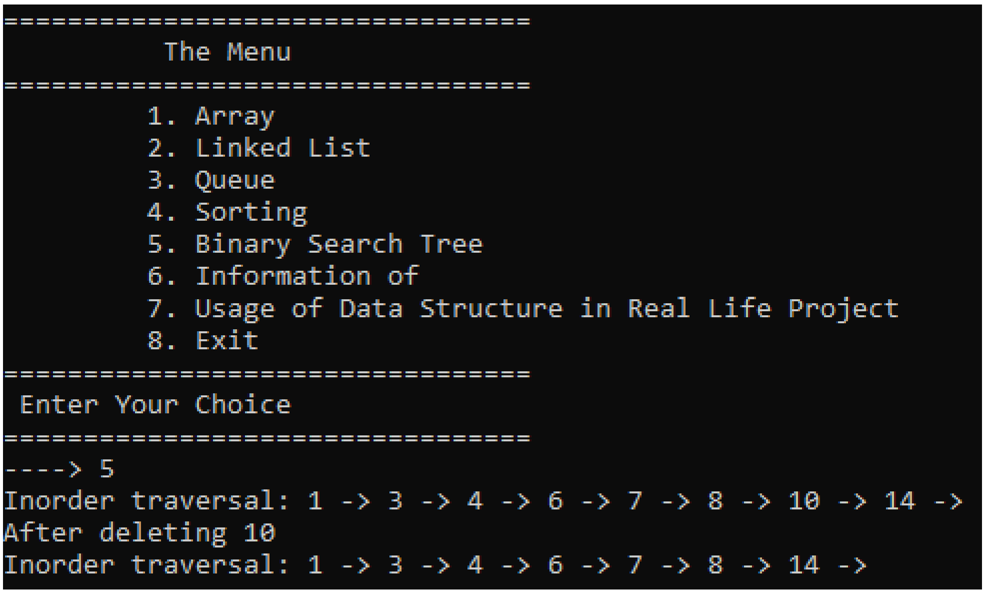
```
=================================
          The Menu
=================================
         1. Array
         2. Linked List
         3. Queue
         4. Sorting
         5. Binary Search Tree
         6. Information of
         7. Usage of Data Structure in Real Life Project
         8. Exit
=================================
 Enter Your Choice
=================================
----> 3
10 20 30
```

Figure: Queue (Task 3)



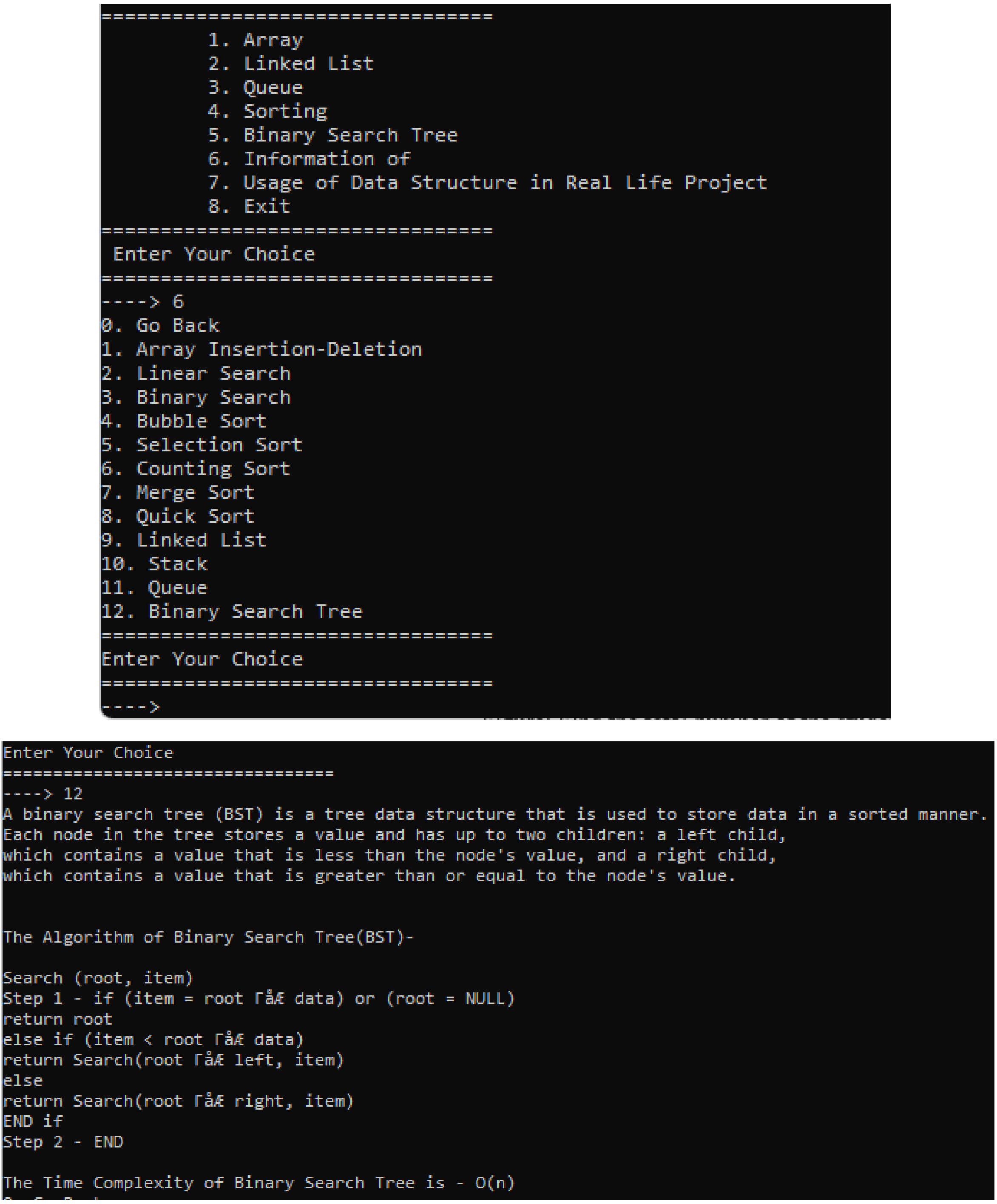
```
==================================
          The Menu
==================================
         1. Array
         2. Linked List
         3. Queue
         4. Sorting
         5. Binary Search Tree
         6. Information of
         7. Usage of Data Structure in Real Life Project
         8. Exit
==================================
 Enter Your Choice
==================================
----> 4
-9  -2  0  11  45

The Bubble Sorting Algorithm is being used in (Ascending Order)
```

Figure: Sorting (Task 4)



```
==================================
          The Menu
==================================
         1. Array
         2. Linked List
         3. Queue
         4. Sorting
         5. Binary Search Tree
         6. Information of
         7. Usage of Data Structure in Real Life Project
         8. Exit
==================================
 Enter Your Choice
==================================
----> 5
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 10 -> 14 ->
After deleting 10
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 14 ->
```

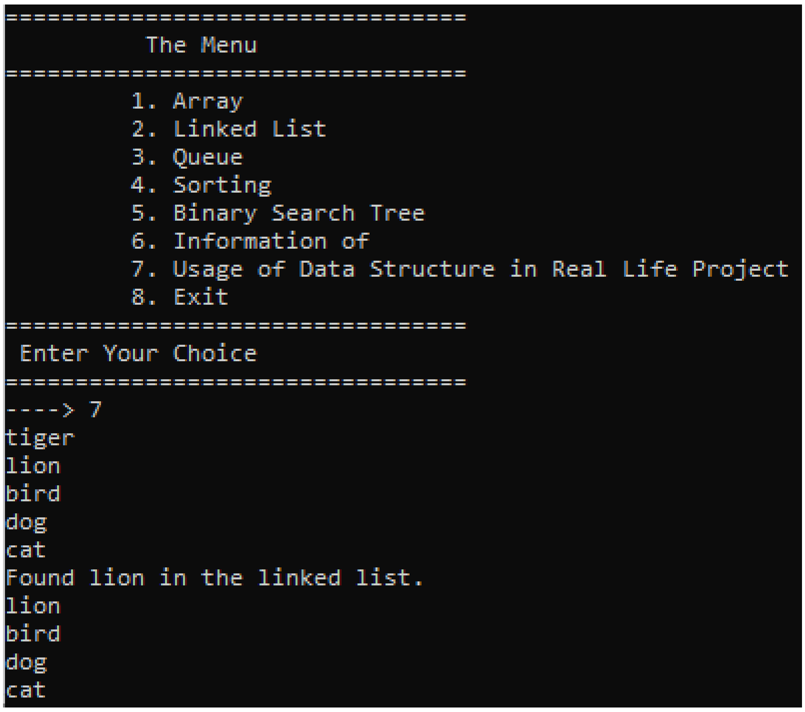
Figure: Binary Search Tree (Task 5)



```
=================================
         1. Array
         2. Linked List
         3. Queue
         4. Sorting
         5. Binary Search Tree
         6. Information of
         7. Usage of Data Structure in Real Life Project
         8. Exit
=================================
 Enter Your Choice
=================================
----> 6
0. Go Back
1. Array Insertion-Deletion
2. Linear Search
3. Binary Search
4. Bubble Sort
5. Selection Sort
6. Counting Sort
7. Merge Sort
8. Quick Sort
9. Linked List
10. Stack
11. Queue
12. Binary Search Tree
=================================
Enter Your Choice
=================================
---->
```

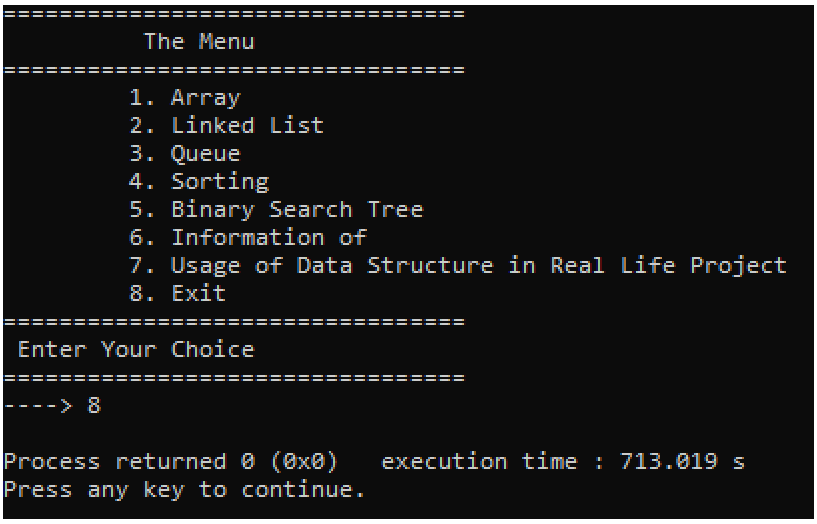
```
Enter Your Choice
=================================
----> 12
A binary search tree (BST) is a tree data structure that is used to store data in a sorted manner.
Each node in the tree stores a value and has up to two children: a left child,
which contains a value that is less than the node's value, and a right child,
which contains a value that is greater than or equal to the node's value.


The Algorithm of Binary Search Tree(BST)-

Search (root, item)
Step 1 - if (item = root ΓåÆ data) or (root = NULL)
return root
else if (item < root ΓåÆ data)
return Search(root ΓåÆ left, item)
else
return Search(root ΓåÆ right, item)
END if
Step 2 - END

The Time Complexity of Binary Search Tree is - O(n)
```

Figure: Information of (Task 6)



```
=================================
          The Menu
=================================
         1. Array
         2. Linked List
         3. Queue
         4. Sorting
         5. Binary Search Tree
         6. Information of
         7. Usage of Data Structure in Real Life Project
         8. Exit
=================================
 Enter Your Choice
=================================
----> 7
tiger
lion
bird
dog
cat
Found lion in the linked list.
lion
bird
dog
cat
```

Figure: Usage of Data Structure in Real Life Project (Task 7)



```
=================================
          The Menu
=================================
         1. Array
         2. Linked List
         3. Queue
         4. Sorting
         5. Binary Search Tree
         6. Information of
         7. Usage of Data Structure in Real Life Project
         8. Exit
=================================
 Enter Your Choice
=================================
----> 8

Process returned 0 (0x0)   execution time : 713.019 s
Press any key to continue.
```

Figure: Exit (Task 8)

3.2.2 Analysis and Outcome

After running the source code, we can see that in the console it’s running properly, and it’s running like a management system. The system is taking input and shows output to the user. This program has more than 10 data for different types of Data Structure with 7 practical implementation of these Data Structure algorithm in real life programs.

Chapter 4

Conclusion

4.1 Introduction

The project I have worked on is Data Structure Tree and its purpose was to make data input more accessible and included with lots of information. And it could take a lot of data, modify it, delete it without a problem.

4.1 Practical Implications

- User friendly interface
- easy access to data"
- less error
- Search facility

4.2 Scope of Future Work

- Adding Time Counting Capability of the functions execution time.
- More Organization.
- Graphical User Interface(GUI).

References

[1] Wikipedia.com

[2] Google.com

[3] Programiz.com