<u>Assessment-3: Sorting and Searching and Tree traversal</u>

Reg.no: 20BDS0117

Name: SANJANA.SAIRAMA

Slot: L37+L38

<u>Aim/Objective</u>: Write C program for the questions using sorting, searching and tree traversal.

1) Write a C program to implement Binary Search and Linear Search.

<u>Pseudo code:</u>

```
1.Begin

2. procedure sequential_search(int array[], int size, int n)

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

           if (array[i] == n)

              print ("%d found at location %d.\n", n, i+1)

              break

           endif

        endfor

        if (i == size)

           print Not found! %d is not present in the list

        endif

end procedure

3. procedure binary_search(int array[], int size, int n)

        int i, first, last, middle

        first = 0

        last = size - 1

        middle = (first+last) / 2

        while (first <= last)

           if (array[middle] < n)
```

```
        first = middle + 1
     endif
      else if (array[middle] == n)
        print (found at location %d.\n", n, middle+1)
        break
      endelse if
      else
        last = middle - 1
        middle = (first + last) / 2
    endwhile
    if ( first > last )
      print Not found! %d is not present in the list.
    endif
End procedure
```

4. Take input from user and display the required element to be searched.

5. End

Code

```c
#include <stdio.h>
    void sequential_search(int array[], int size, int n)
    {
      int i;
      for (i = 0; i < size; i++)
      {
        if (array[i] == n)
        {
          printf("%d found at location %d.\n", n, i+1);
          break;
```

```
        }
      }
      if (i == size)
        printf("Not found! %d is not present in the list.\n", n);
    }
void binary_search(int array[], int size, int n)
    {
      int i, first, last, middle;
      first = 0;
      last = size - 1;
      middle = (first+last) / 2;

      while (first <= last) {
        if (array[middle] < n)
          first = middle + 1;
        else if (array[middle] == n) {
          printf("%d found at location %d.\n", n, middle+1);
         break;
        }
        else
          last = middle - 1;

        middle = (first + last) / 2;
      }
      if ( first > last )
        printf("Not found! %d is not present in the list.\n", n);
    }
```

```c
int main()
{
   int a[200], i, j, n, size;
   printf("Enter the size of the list:");
   scanf("%d", &size);
   printf("Enter %d Integers in ascending order\n", size);
   for (i = 0; i < size; i++)
      scanf("%d", &a[i]);
   printf("Enter value to find\n");
   scanf("%d", &n);
   printf("Sequential search\n");
   sequential_search(a, size, n);
   printf("Binary search\n");
   binary_search(a, size, n);
   return 0;
}
```

Output

"C:\Users\USER\Downloads\binary and linear search\bin\Debug\binary and linear search.exe"

```
Enter the size of the list:5
Enter 5 Integers in ascending order
34 45 56 78 89
Enter value to find
78
Sequential search
78 found at location 4.
Binary search
78 found at location 4.

Process returned 0 (0x0)   execution time : 37.874 s
Press any key to continue.
```

2) Write a C program to implement Binary Tree using Array.

<u>Pseudo code</u>

1.Begin

2.Initialize the structure and variables

3. node* insert(char c[],int n)

```
      node*tree=NULL
      if(c[n]!='\0')
             tree=(node*)malloc(sizeof(node))
            tree->left=insert(c,2*n+1)
            tree->data=c[n]
            tree->right=insert(c,2*n+2)
      endif
 return tree
end procedure
```

4. procedure inorder(node*tree)

```
      if(tree!=NULL)
             inorder(tree->left)
            printf("%c\t",tree->data)
            inorder(tree->right)
      endif
  end procedure
```

5. procedure main()

```
            node*tree=NULL
            char c[]={'A','B','C','D','E','F','\0','G','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0'}
             tree=insert(c,0)
             inorder(tree)
   end procedure
```

6.End

Code

```
#include<stdio.h>
typedef struct node
{
 struct node*left;
 struct node*right;
 char data;
}node;

node* insert(char c[],int n)
{ node*tree=NULL;
 if(c[n]!='\0')
 {
  tree=(node*)malloc(sizeof(node));
  tree->left=insert(c,2*n+1);
  tree->data=c[n];
  tree->right=insert(c,2*n+2);
 }
 return tree;
}

void inorder(node*tree)
{
 if(tree!=NULL)
 {
  inorder(tree->left);
```

```
  printf("%c\t",tree->data);
  inorder(tree->right);
 }
}

void main()
{
 node*tree=NULL;
 char c[]={'A','B','C','D','E','F','\0','G','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0'};
 tree=insert(c,0);
 inorder(tree);
}
```

<u>Output</u>

C:\Users\USER\Downloads\cdc\bin\Debug\cdc.exe

```
G       D       B       E       A       F       C
Process returned 0 (0x0)   execution time : 0.069 s
Press any key to continue.
```

3) Write a C program to implement Binary Tree using Linked List.

<u>Pseudo code</u>

1.Begin

2.Initialize the structure and variables

3. struct node* createNode(int data)

    struct node *newNode = (struct node*)malloc(sizeof(struct node))

    newNode->data = data

    newNode->left = NULL

    newNode->right = NULL

```
        return newNode
end procedure
4. struct queue
         int front, rear, size
        struct node* *arr
end procedure
5.struct queue* createQueue()
        struct queue* newQueue = (struct queue*) malloc(sizeof( struct queue ))
         newQueue->front = -1
         newQueue->rear = 0
        newQueue->size = 0
        newQueue->arr = (struct node**) malloc(100 * sizeof( struct node* ))
         return newQueue
end procedure
6. procedure enqueue(struct queue* queue, struct node *temp)
        queue->arr[queue->rear++] = temp
        queue->size++
  end procedure
7.struct node *dequeue(struct queue* queue
  queue->size--
  return queue->arr[++queue->front]
 end procedure
8.Procedure insertNode(int data)
  struct node *newNode = createNode(data)
  if(root == NULL)
    root = newNode
```

```
    return
  endif
  else
    struct queue* queue = createQueue()
    enqueue(queue, root)
    while(true)
      struct node *node = dequeue(queue)
      if(node->left != NULL && node->right != NULL)
        enqueue(queue, node->left)
        enqueue(queue, node->right)
      endif
      else
        if(node->left == NULL) {
            node->left = newNode
            enqueue(queue, node->left)
        endif
      end else
        else
          node->right = newNode
          enqueue(queue, node->right)
        end else
        break
    end else
  end while
end else
end procedure
```

9.Procedure inorderTraversal(struct node *node)

```
   if(root == NULL)
      print Tree is empty
      return
   endif
   else
      if(node->left != NULL)
         inorderTraversal(node->left)
      print("%d ", node->data)
      if(node->right != NULL)
         inorderTraversal(node->right)
      endif
  end else
   End procedure
```

10.Insert the elements and display the binary tree

11.End

Code

```
#include <stdlib.h>
#include <stdbool.h>
struct node{
   int data;
   struct node *left;
   struct node *right;
};
struct node *root = NULL;
struct node* createNode(int data){
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = data;
```

```
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}
struct queue
{
    int front, rear, size;
    struct node* *arr;
};
struct queue* createQueue()
{
    struct queue* newQueue = (struct queue*) malloc(sizeof( struct queue ));

    newQueue->front = -1;
    newQueue->rear = 0;
    newQueue->size = 0;

    newQueue->arr = (struct node**) malloc(100 * sizeof( struct node* ));

    return newQueue;
}
void enqueue(struct queue* queue, struct node *temp){
    queue->arr[queue->rear++] = temp;
    queue->size++;
}
struct node *dequeue(struct queue* queue){
```

```
    queue->size--;
    return queue->arr[++queue->front];
}
void insertNode(int data) {
    struct node *newNode = createNode(data);
    if(root == NULL){
        root = newNode;
        return;
    }
    else {
        struct queue* queue = createQueue();
        enqueue(queue, root);
        while(true) {
            struct node *node = dequeue(queue);
            if(node->left != NULL && node->right != NULL) {
                enqueue(queue, node->left);
                enqueue(queue, node->right);
            }
            else {
                if(node->left == NULL) {
                    node->left = newNode;
                    enqueue(queue, node->left);
                }
                else {
                    node->right = newNode;
                    enqueue(queue, node->right);
                }
```

```
            break;
          }
        }
      }
    }
void inorderTraversal(struct node *node) {
  if(root == NULL){
    printf("Tree is empty\n");
    return;
  }
  else {
    if(node->left != NULL)
      inorderTraversal(node->left);
    printf("%d ", node->data);
    if(node->right != NULL)
      inorderTraversal(node->right);
    }
  }
int main(){
  insertNode(11);
  printf("Binary tree after insertion: \n");
  inorderTraversal(root);
  insertNode(34);
  insertNode(30);
  printf("\nBinary tree after insertion: \n");
  inorderTraversal(root);
  insertNode(14);
```

```
    insertNode(15);

    printf("\nBinary tree after insertion: \n");

    inorderTraversal(root);

    insertNode(20);

    insertNode(40);

    printf("\nBinary tree after insertion: \n");

    inorderTraversal(root);

    return 0;

}
```

Output

"C:\Users\USER\Downloads\binary and linear search\bin\Debug\binary and l

```
Binary tree after insertion:
11
Binary tree after insertion:
34 11 30
Binary tree after insertion:
14 34 15 11 30
Binary tree after insertion:
14 34 15 11 20 30 40
Process returned 0 (0x0)   execution time : 0.084 s
Press any key to continue.
```

4) Write a C program to implement all the Tree Traversal techniques using array.

Pseudo code

1.Begin

2.Initialize the structure and variables

3. procedure insert(int data)

```
     struct node *tempNode = (struct node*) malloc(sizeof(struct node))

    struct node *current

    struct node *parent

    tempNode->data = data

    tempNode->leftChild = NULL
```

```
          tempNode->rightChild = NULL
      if(root == NULL)
         root = tempNode
      endif
   else
       current = root
       parent = NULL
      while(1)
          parent = current
          if(data < parent->data)
             current = current->leftChild
          endif
           if(current == NULL)
             parent->leftChild = tempNode
             return
           endif
        endif
        else
          current = current->rightChild
          if(current == NULL)
                    parent->rightChild = tempNode
                     return
          endif
        endelse
      endwhile
   endelse
end procedure
```

```
4. struct node* search(int data)
   struct node *current = root
   print("Visiting elements: ")
   while(current->data != data)
      if(current != NULL)
         print("%d ",current->data)
      endif
      if(current->data > data)
         current = current->leftChild
      endif
      else
         current = current->rightChild
      end else
      if(current == NULL)
         return NULL
      endif
   endwhile
   return current
end procedure
5.procedure pre_order_traversal(struct node* root)
   if(root != NULL)
      print("%d ",root->data)
      pre_order_traversal(root->leftChild)
      pre_order_traversal(root->rightChild)
   endif
end procedure
6.procedure inorder_traversal(struct node* root)
```

```
  if(root != NULL)
    inorder_traversal(root->leftChild)
    print("%d ",root->data)
    inorder_traversal(root->rightChild)
  endif
end procedure
```

7.procedure post_order_traversal(struct node* root)

```
  if(root != NULL)
    post_order_traversal(root->leftChild)
    post_order_traversal(root->rightChild)
    print("%d ", root->data)
  endif
end procedure
```

8.Taking the required values we get the required output

9.End

Code

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

struct node {
  int data;

  struct node *leftChild;
  struct node *rightChild;
};

struct node *root = NULL;
```

```
void insert(int data) {
   struct node *tempNode = (struct node*) malloc(sizeof(struct node));
   struct node *current;
   struct node *parent;

   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;



   if(root == NULL) {
      root = tempNode;
   } else {
      current = root;
      parent = NULL;

      while(1) {
         parent = current;

         if(data < parent->data) {
            current = current->leftChild;

            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
            }
```

```
      }
      else {
         current = current->rightChild;

         if(current == NULL) {
            parent->rightChild = tempNode;
            return;
         }
      }
   }
  }
}

struct node* search(int data) {
   struct node *current = root;
   printf("Visiting elements: ");

   while(current->data != data) {
      if(current != NULL)
         printf("%d ",current->data);

      if(current->data > data) {
         current = current->leftChild;
      }

      else {
         current = current->rightChild;
```

```
    }
    if(current == NULL) {
      return NULL;
    }
  }

  return current;
}

void pre_order_traversal(struct node* root) {
  if(root != NULL) {
    printf("%d ",root->data);
    pre_order_traversal(root->leftChild);
    pre_order_traversal(root->rightChild);
  }
}

void inorder_traversal(struct node* root) {
  if(root != NULL) {
    inorder_traversal(root->leftChild);
    printf("%d ",root->data);
    inorder_traversal(root->rightChild);
  }
}

void post_order_traversal(struct node* root) {
  if(root != NULL) {
```

```
    post_order_traversal(root->leftChild);
    post_order_traversal(root->rightChild);
    printf("%d ", root->data);
  }
}

int main() {
  int i;
  int array[7] = { 26, 14, 34, 10, 18, 31, 40 };

  for(i = 0; i < 7; i++)
    insert(array[i]);

  i = 31;
  struct node * temp = search(i);

  if(temp != NULL) {
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }else {
    printf("[ x ] Element not found (%d).\n", i);
  }

  i = 15;
  temp = search(i);

  if(temp != NULL) {
```

```c
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }else {
    printf("[ x ] Element not found (%d).\n", i);
  }

  printf("\nPreorder traversal: ");
  pre_order_traversal(root);

  printf("\nInorder traversal: ");
  inorder_traversal(root);

  printf("\nPost order traversal: ");
  post_order_traversal(root);

  return 0;
}
```

Output:

C:\Users\USER\Downloads\dsa\bin\Debug\dsa.exe

```
Visiting elements: 26 34 [31] Element found.
Visiting elements: 26 14 18 [ x ] Element not found (15).

Preorder traversal: 26 14 10 18 34 31 40
Inorder traversal: 10 14 18 26 31 34 40
Post order traversal: 10 18 14 31 40 34 26
Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

5) Write a C program to implement Insertion and Deletion (all) of Binary Search Tree

Pseudo code:

1.Begin

2.Initialize the structure and variables

3. struct node* createNode(int data)

```
        struct node *newNode = (struct node*)malloc(sizeof(struct node))
        newNode->data= data
        newNode->left = NULL
        newNode->right = NULL
        return newNode
 end procedure
```

4 .procedure insert(int data)

```
  struct node *newNode = createNode(data)
  if(root == NULL)
    root = newNode
    return
   endif
  else
    struct node *current = root, *parent = NULL
    while(true)
      parent = current;
      if(data < current->data)
        current = current->left
        if(current == NULL)
          parent->left = newNode
          return;
```

```
          endif
        endif
        else
          current = current->right
          if(current == NULL)
            parent->right = newNode
            return
          endif
        end else
      end while
    end else
end procedure
```

5.struct node* minNode(struct node *root)

```
  if (root->left != NULL)
    return minNode(root->left)
  endif
  else
    return root
  end else
end procedure
```

6.struct node* deleteNode(struct node *node, int value)

```
  if(node == NULL)
     return NULL
   end if
  else
    if(value < node->data)
      node->left = deleteNode(node->left, value)
```

```
    else if(value > node->data)
      node->right = deleteNode(node->right, value);
    else
      if(node->left == NULL && node->right == NULL)
        node = NULL
      else if(node->left == NULL)
        node = node->right
      end elseif
      else if(node->right == NULL)
        node = node->left
      endelse if
      else
        struct node *temp = minNode(node->right)
        node->data = temp->data
        node->right = deleteNode(node->right, temp->data)
      end else
    end else
    return node
  end else
end procedure
7.procedure inorderTraversal(struct node *node)
  if(root == NULL)
    print Tree is empty
     return
   endif
  else
    if(node->left!= NULL)
```

```
        inorderTraversal(node->left)
        print("%d ", node->data)
        endif
      if(node->right!= NULL)
       inorderTraversal(node->right)
      endif
   endelse
end procedure
```

8.Taking the required values we get the required output

9.End

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
struct node{
   int data;
   struct node *left;
   struct node *right;
};
struct node *root= NULL;
struct node* createNode(int data){
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data= data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}
```

```
void insert(int data) {
   struct node *newNode = createNode(data);
   if(root == NULL){
      root = newNode;
      return;
    }
   else {
      struct node *current = root, *parent = NULL;
      while(true) {
         parent = current;
         if(data < current->data) {
            current = current->left;
            if(current == NULL) {
               parent->left = newNode;
               return;
            }
         }
         else {
            current = current->right;
            if(current == NULL) {
               parent->right = newNode;
               return;
            }
         }
      }
   }
}
```

```
struct node* minNode(struct node *root) {
   if (root->left != NULL)
      return minNode(root->left);
   else
      return root;
}
struct node* deleteNode(struct node *node, int value) {
   if(node == NULL){
        return NULL;
    }
   else {
      if(value < node->data)
         node->left = deleteNode(node->left, value);
      else if(value > node->data)
         node->right = deleteNode(node->right, value);
      else {
         if(node->left == NULL && node->right == NULL)
            node = NULL;
         else if(node->left == NULL) {
            node = node->right;
         }
         else if(node->right == NULL) {
            node = node->left;
         }
         else {
            struct node *temp = minNode(node->right);
            node->data = temp->data;
```

```
            node->right = deleteNode(node->right, temp->data);
          }
        }
        return node;
    }
}
void inorderTraversal(struct node *node) {
    if(root == NULL){
        printf("Tree is empty\n");
          return;
     }
    else {
        if(node->left!= NULL)
            inorderTraversal(node->left);
        printf("%d ", node->data);
        if(node->right!= NULL)
          inorderTraversal(node->right);
    }
}
int main()
{
    insert(100);
    insert(111);
    insert(125);
    insert(130);
    insert(142);
    insert(150);
```

```c
    printf("The Binary search tree after insertion: \n");
    inorderTraversal(root);
    struct node *deletedNode = NULL;
    deletedNode = deleteNode(root, 142);
    printf("\nBinary search tree after deleting node 142: \n");
    inorderTraversal(root);
    deletedNode = deleteNode(root, 125);
    printf("\nBinary search tree after deleting node 125 \n");
    inorderTraversal(root);
    deletedNode = deleteNode(root, 111);
    printf("\nBinary search tree after deleting node 111: \n");
    inorderTraversal(root);
    return 0;
}
```

Output

"C:\Users\USER\Downloads\binary and linear search\bin\Debug\binary and linear search.exe"

```
The Binary search tree after insertion:
100 111 125 130 142 150
Binary search tree after deleting node 142:
100 111 125 130 150
Binary search tree after deleting node 125
100 111 130 150
Binary search tree after deleting node 111:
100 130 150
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

6) Write a C program to implement to search an element in the Binary Search Tree

<u>Pseudo code</u>

1.Begin

2.Initialize the structure and variables

3. procedure search(BST *root, int item)

```
BST *save,*ptr;
if (root == NULL)
   LOC = NULL
   PAR=NULL
endif
if (item == root -> info)
   LOC = root
    PAR = NULL
     return
endif
if (item < root->info)
   save = root
   ptr = root->left
endif
else
   save = root
   ptr = root -> right
endelse
while( ptr != NULL)
   if (ptr -> info == item)
      LOC = ptr
```

```
          PAR = save
          return
        endif
        if(item < ptr->info)
          save = ptr
          ptr = ptr->left
        endif
        else
          save = ptr
          ptr = ptr->right
        endelse
    endwhile
    LOC = NULL
    PAR = save
    return
end procedure
4.struct node* findmin(struct node*r)
        if (r == NULL)
                return NULL
        endif
        else if (r->left!=NULL)
                return findmin(r->left)
        endif
        else if (r->left == NULL)
                return r
        end elseif
 end procedure
```

```
5.struct node*insert(struct node*r, int x)
        if (r == NULL)
                r = (struct node*)malloc(sizeof(struct node))
                r->info = x
                 r->left = r->right = NULL
                 return r
        endif
        else if (x < r->info)
                r->left = insert(r->left, x)
         end elseif
        else if (x > r->info)
                r->right = insert(r->right, x)
                return r
         end elseif
    end procedure
6.struct node* del(struct node*r, int x)
        struct node *t
        if(r == NULL)
                print Element not found
        else if (x < r->info)
                r->left = del(r->left, x)
         end elseif
        else if (x > r->info)
                r->right = del(r->right, x)
        end elseif
        else if ((r->left != NULL) && (r->right != NULL))
                t = findmin(r->right)
```

```
                r->info = t->info
                r->right = del(r->right, r->info)
        end elseif
        else
                t = r
                if (r->left == NULL)
                        r = r->right
                endif
                else if (r->right == NULL)
                        r = r->left
                        free(t)
                end elseif
                return r
        end else
```

7.Taking the required values we get the required output

8.End

Code

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int info;
        struct node*left;
        struct node*right;
};
typedef struct node BST;
BST *LOC, *PAR;
```

```
void search(BST *root, int item)
{
   BST *save,*ptr;
   if (root == NULL)
   {
      LOC = NULL;
      PAR=NULL;
   }
   if (item == root -> info)
   {
   LOC = root;
   PAR = NULL;
   return;
   }
   if (item < root->info)
   {
      save = root;
      ptr = root->left;
   }
   else
   {
      save = root;
      ptr = root -> right;
   }
   while( ptr != NULL)
   {
      if (ptr -> info == item)
```

```
        {
            LOC = ptr;
            PAR = save;
            return;
        }
        if(item < ptr->info)
        {
            save = ptr;
            ptr = ptr->left;
        }
        else
        {
            save = ptr;
            ptr = ptr->right;
        }
    }
    LOC = NULL;
    PAR = save;
    return;
}
struct node* findmin(struct node*r)
{
        if (r == NULL)
                return NULL;
        else if (r->left!=NULL)
                return findmin(r->left);
        else if (r->left == NULL)
```

```
            return r;
}
struct node*insert(struct node*r, int x)
{
        if (r == NULL)
        {
        r = (struct node*)malloc(sizeof(struct node));
        r->info = x;
        r->left = r->right = NULL;
        return r;
        }
        else if (x < r->info)
        r->left = insert(r->left, x);
        else if (x > r->info)
        r->right = insert(r->right, x);
        return r;
}
struct node* del(struct node*r, int x)
{
        struct node *t;
        if(r == NULL)
            printf("\nElement not found");
        else if (x < r->info)
        r->left = del(r->left, x);
        else if (x > r->info)
        r->right = del(r->right, x);
        else if ((r->left != NULL) && (r->right != NULL))
```

```
        {
        t = findmin(r->right);
        r->info = t->info;
        r->right = del(r->right, r->info);
        }
        else
        {
        t = r;
        if (r->left == NULL)
           r = r->right;
        else if (r->right == NULL)
           r = r->left;
        free(t);
        }
        return r;
}
int main()
{
   struct node* root = NULL;
   int x, c = 1, z;
   int element;
   char ch;
   printf("\nEnter an element: ");
   scanf("%d", &x);
   root = insert(root, x);
   printf("\nDo you want to enter another element :y or n");
   scanf(" %c",&ch);
```

```
while (ch == 'y')
{
   printf("Enter an element:");
   scanf("%d", &x);
   root = insert(root,x);
   printf("\nPress y or n to insert another element: y or n: ");
   scanf(" %c", &ch);
}
while(1)
{
   printf("\n1 Insert an element ");
   printf("\n2 Delete an element");
   printf("\n3 Search for an element ");
   printf("\n4 Exit ");
   printf("\nEnter your choice: ");
   scanf("%d", &c);
   switch(c)
   {
      case 1:
         printf("\nEnter the item:");
         scanf("%d", &z);
         root = insert(root,z);
         break;
      case 2:
         printf("\nEnter the info to be deleted:");
         scanf("%d", &z);
         root = del(root, z);
```

```c
                break;
        case 3:
            printf("\nEnter element to be searched:  ");
            scanf("%d", &element);
            search(root, element);
            if(LOC != NULL)
                printf("\n%d Found in Binary Search Tree !!\n",element);
            else
                printf("\nIt is not present in Binary Search Tree\n");
            break;
        case 4:
            printf("\nExiting...");
                return 0;
        default:
            printf("Enter a valid choice: ");
        }
    }
    return 0;
}
```

Output

(a)

C:\Users\USER\Downloads\dsada\bin\Debug\dsada.exe

```
Enter an element: 4

Do you want to enter another element :y or n y
Enter an element:5

Press y or n to insert another element: y or n:  y
Enter an element:7

Press y or n to insert another element: y or n: n

1 Insert an element
2 Delete an element
3 Search for an element
4 Exit
Enter your choice: 3

Enter element to be searched:  7

7 Found in Binary Search Tree !!

1 Insert an element
2 Delete an element
3 Search for an element
4 Exit
Enter your choice: 4

Exiting...
Process returned 0 (0x0)   execution time : 37.269 s
Press any key to continue.
```

(b)

C:\Users\USER\Downloads\dsada\bin\Debug\dsada.exe

```
Enter an element: 8

Do you want to enter another element :y or n y
Enter an element:9

Press y or n to insert another element: y or n:  y
Enter an element:10

Press y or n to insert another element: y or n: n

1 Insert an element
2 Delete an element
3 Search for an element
4 Exit
Enter your choice: 3

Enter element to be searched:  2

It is not present in Binary Search Tree

1 Insert an element
2 Delete an element
3 Search for an element
4 Exit
Enter your choice: 4

Exiting...
Process returned 0 (0x0)   execution time : 24.307 s
Press any key to continue.
```