Experiment:23

Binary Search Tree Implementation

Aim

Write program for Binary search implementation - creation,insertion,deletion,traversal

Program

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

struct node {
  int key;
  struct node *left, *right;
};
struct node *newNode(int item) {
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->key = item;
  temp->left = temp->right = NULL;
  return temp;
}
void inorder(struct node *root) {
  if (root != NULL) {
    inorder(root->left);
    printf("%d -> ", root->key);
    inorder(root->right);
  }
}
void preorder(struct node *root) {
  if (root != NULL) {
    printf("%d -> ", root->key);
    preorder(root->left);
    preorder(root->right);
  }
}
void postorder(struct node *root) {
  if (root != NULL) {
    preorder(root->left);
    preorder(root->right);
   printf("%d -> ", root->key);
  }
}
```

```
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
    node->right = insert(node->right, key);

  return node;
}
struct node *minValueNode(struct node *node) {
  struct node *current = node;
  while (current && current->left != NULL)
    current = current->left;
  return current;
}
struct node *deleteNode(struct node *root, int key) {
  if (root == NULL) return root;
  if (key < root->key)
    root->left = deleteNode(root->left, key);
  else if (key > root->key)
    root->right = deleteNode(root->right, key);

  else {
    if (root->left == NULL) {
      struct node *temp = root->right;
      free(root);
      return temp;
    } else if (root->right == NULL) {
      struct node *temp = root->left;
      free(root);
      return temp;
    }
    struct node *temp = minValueNode(root->right);
    root->key = temp->key;
    root->right = deleteNode(root->right, temp->key);
  }
  return root;
}
int main() {
  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("\nPreorder traversal:");
  preorder(root);

  printf("\nPostorder traversal:");
  postorder(root);

  printf("\nAfter deleting 10\n");
  root = deleteNode(root, 10);

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

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

  printf("\nPostorder traversal:");
  postorder(root);

}
```

Output

Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 10 -> 14 ->

Preorder traversal:8 -> 3 -> 1 -> 6 -> 4 -> 7 -> 10 -> 14 ->

Postorder traversal:3 -> 1 -> 6 -> 4 -> 7 -> 10 -> 14 -> 8 ->

After deleting 10

Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 14 ->

Preorder traversal:8 -> 3 -> 1 -> 6 -> 4 -> 7 -> 14 ->

Postorder traversal:3 -> 1 -> 6 -> 4 -> 7 -> 14 -> 8 ->