WEEK 5

MERGE SORT,BUBBLE SORT,BINARY SEARCH TREE

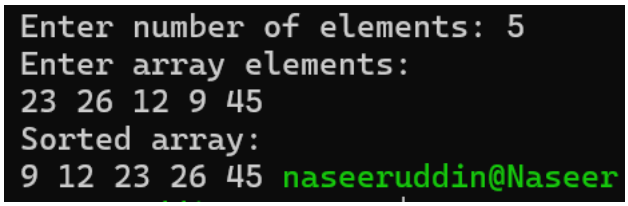
1)MERGE SORT

CODE:

```
#include <stdio.h>
void merge(int arr[], int left, int mid, int right)
{
int i, j, k;
int n1 = mid - left + 1;
int n2 = right - mid;
int L[n1], R[n2];
for (i = 0; i < n1; i++)
L[i] = arr[left + i];
for (j = 0; j < n2; j++)
R[j] = arr[mid + 1 + j];
i = 0;
j = 0;
k = left;
while (i < n1 && j < n2)
{
if (L[i] <= R[j])
arr[k++] = L[i++];
else
arr[k++] = R[j++];
}
while (i < n1)
arr[k++] = L[i++];
while (j < n2)
arr[k++] = R[j++];
}
void mergeSort(int arr[], int left, int right)
{
if (left < right)
{
int mid = (left + right) / 2;
mergeSort(arr, left, mid);
mergeSort(arr, mid + 1, right);
merge(arr, left, mid, right);
```

```
}
}
int main()
{
int arr[100], n, i;
printf("Enter number of elements: ");
scanf("%d", &n);
printf("Enter array elements:\n");
for (i = 0; i < n; i++)
scanf("%d", &arr[i]);
mergeSort(arr, 0, n - 1);
printf("Sorted array:\n");
for (i = 0; i < n; i++)
printf("%d ", arr[i]);
return 0;
}
```

OUTPUT:



2)BUBBLESORT:

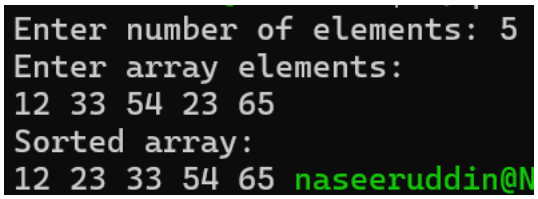
CODE:

```
#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; 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 pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
int main()
{
    int arr[100], n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter array elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
```

```
    quickSort(arr, 0, n - 1);
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT:



3) BINARY SEARCH TREE

CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int key;
    struct Node *left, *right;
};
struct Node *newNodeCreate(int value)
{
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node));
    temp->key = value;
    temp->left = temp->right = NULL;
    return temp;
}
struct Node *searchNode(struct Node *root, int target)
{
    if (root == NULL || root->key == target)
        return root;
    if (root->key < target)
        return searchNode(root->right, target);
    return searchNode(root->left, target);
}
```

```
struct Node *insertNode(struct Node *node, int value)
{
   if (node == NULL)
      return newNodeCreate(value);
   if (value < node->key)
      node->left = insertNode(node->left, value);
   else if (value > node->key)
      node->right = insertNode(node->right, value);
   return node;
}
void postOrder(struct Node *root)
{
   if (root != NULL)
   {
      postOrder(root->left);
      postOrder(root->right);
      printf(" %d ", root->key);
   }
}

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);
   }
}

struct Node *findMin(struct Node *root)
```

```
{
   if (root == NULL)
      return NULL;
   else if (root->left != NULL)
      return findMin(root->left);
   return root;
}
struct Node *delete (struct Node *root, int x)
{
   if (root == NULL)
      return NULL;
   if (x > root->key)
      root->right = delete (root->right, x);
   else if (x < root->key)
      root->left = delete (root->left, x);
   else
   {
      if (root->left == NULL && root->right == NULL)
      {
         free(root);
         return NULL;
      }
      else if (root->left == NULL || root->right == NULL)
      {
         struct Node *temp;
         if (root->left == NULL)
         {
            temp = root->right;
         }
         else
         {
            temp = root->left;
         }
         free(root);
         return temp;
      }
      else
      {
         struct Node *temp = findMin(root->right);
         root->key = temp->key;
         root->right = delete (root->right, temp->key);
```

```
        }
    }
    return root;
}

int main()
{
    struct Node *root = NULL;
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);

    if (searchNode(root, 60) != NULL)
        printf("60 found");
    else
        printf("60 not found");
    printf("\n");
    postOrder(root);
    printf("\n");
    preOrder(root);
    printf("\n");
    inOrder(root);
    printf("\n");
    struct Node *temp = delete (root, 70);
    printf("After Delete: \n");
    inOrder(root);
    return 0;
}
```

OUTPUT: