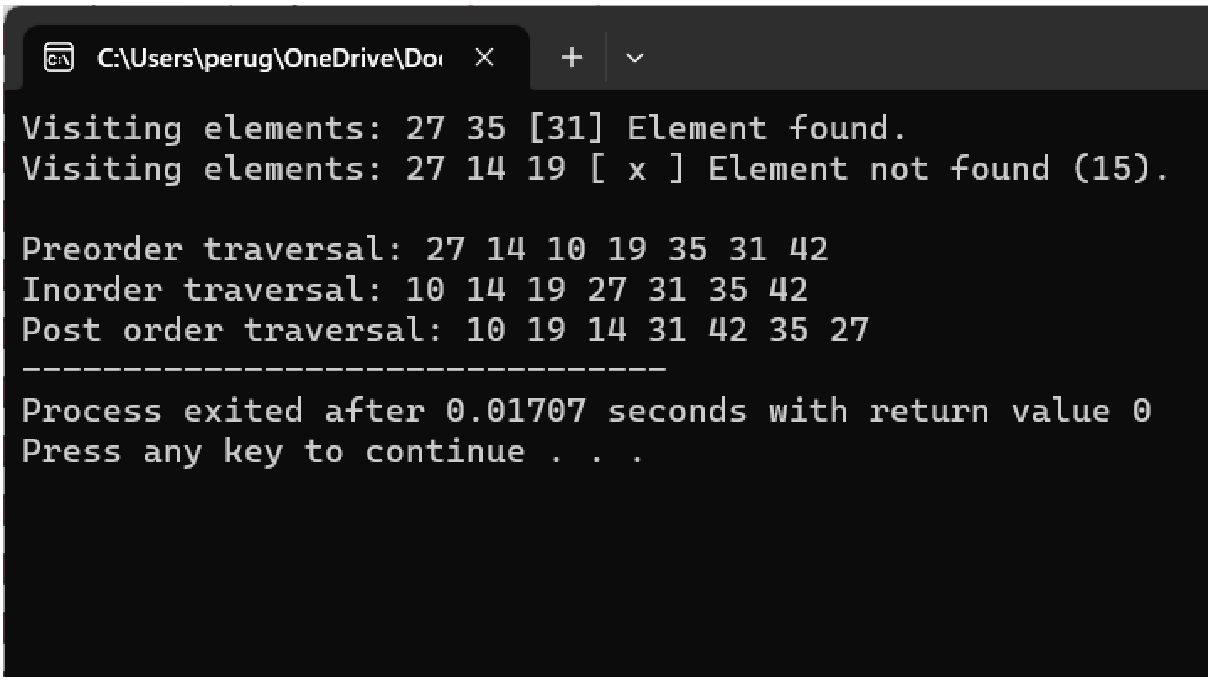
Data structure day-4

1.Write c program to implement binary tree traversal?

```
#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;
         }
      }  //go to right of the tree
      else {
         current = current->rightChild;

         //insert to the right
         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] = { 27, 14, 35, 10, 19, 31, 42 };
 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) {
      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;
}
```



```
Visiting elements: 27 35 [31] Element found.
Visiting elements: 27 14 19 [ x ] Element not found (15).

Preorder traversal: 27 14 10 19 35 31 42
Inorder traversal: 10 14 19 27 31 35 42
Post order traversal: 10 19 14 31 42 35 27
--------------------------------
Process exited after 0.01707 seconds with return value 0
Press any key to continue . . .
```

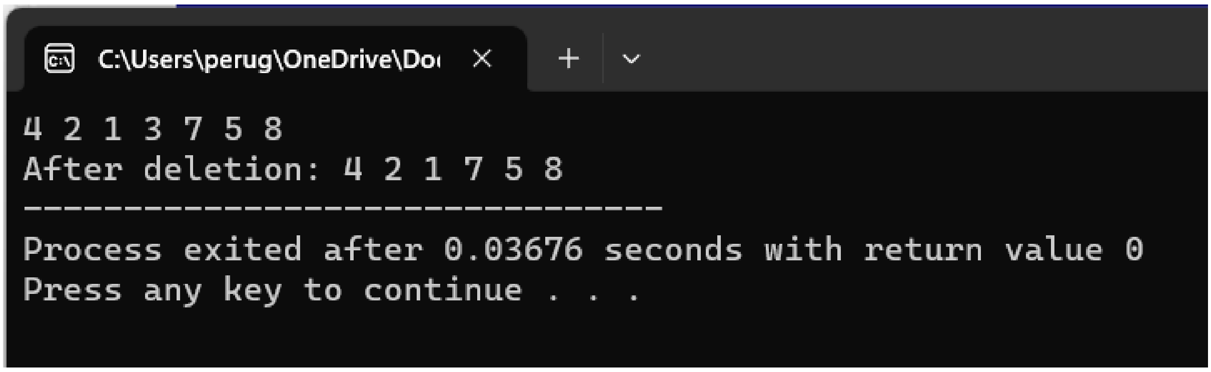
2.write c program to implement AVL tree with all rotation?

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int key;
  struct Node *left;
  struct Node *right;
  int height;
};

int max(int a, int b);
int height(struct Node *N) {
  if (N == NULL)
```

```
    return 0;
  return N->height;
}

int max(int a, int b) {
  return (a > b) ? a : b;
}
struct Node *newNode(int key) {
  struct Node *node = (struct Node *)
    malloc(sizeof(struct Node));
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}
struct Node *rightRotate(struct Node *y) {
  struct Node *x = y->left;
  struct Node *T2 = x->right;

  x->right = y;
  y->left = T2;
  y->height = max(height(y->left),
height(y->right)) + 1;
  x->height = max(height(x->left),
height(x->right)) + 1;

  return x;
}

// Left rotate
struct Node *leftRotate(struct Node *x) {
  struct Node *y = x->right;
  struct Node *T2 = y->left;

  y->left = x;
  x->right = T2;

  x->height = max(height(x->left),
height(x->right)) + 1;
  y->height = max(height(y->left),
height(y->right)) + 1;

  return y;
}
int getBalance(struct Node *N) {
  if (N == NULL)
    return 0;
  return height(N->left) - height(N->right);
}
struct Node *insertNode(struct Node *node, int key) {
  if (node == NULL)
    return (newNode(key));

  if (key < node->key)
    node->left = insertNode(node->left, key);
  else if (key > node->key)
    node->right = insertNode(node->right, key);
  else
    return node;
  node->height = 1 + max(height(node->left),
               height(node->right));

  int balance = getBalance(node);
  if (balance > 1 && key < node->left->key)
    return rightRotate(node);

  if (balance < -1 && key > node->right->key)
    return leftRotate(node);
  if (balance > 1 && key > node->left->key) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
  }

  if (balance < -1 && key < node->right->key) {
    node->right = rightRotate(node->right);
    return leftRotate(node);
  }

  return node;
}

struct Node *minValueNode(struct Node *node) {
  struct Node *current = node;

  while (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) || (root->right ==
NULL)) {
      struct Node *temp = root->left ? root->left :
root->right;

      if (temp == NULL) {
        temp = root;
        root = NULL;
      } else
        *root = *temp;
      free(temp);
    } else {
      struct Node *temp =
minValueNode(root->right);
```

```
   root->key = temp->key;

   root->right = deleteNode(root->right,
temp->key);
  }
 }

 if (root == NULL)
  return root;
 root->height = 1 + max(height(root->left),
        height(root->right));

 int balance = getBalance(root);
 if (balance > 1 && getBalance(root->left) >= 0)
  return rightRotate(root);

 if (balance > 1 && getBalance(root->left) < 0) {
  root->left = leftRotate(root->left);
  return rightRotate(root);
 }

 if (balance < -1 && getBalance(root->right) <=
0)
```

```
    return leftRotate(root);

  if (balance < -1 && getBalance(root->right) > 0)
{
    root->right = rightRotate(root->right);
    return leftRotate(root);
  }

  return root;
}
void printPreOrder(struct Node *root) {
  if (root != NULL) {
    printf("%d ", root->key);
    printPreOrder(root->left);
    printPreOrder(root->right);
  }
}
int main() {
  struct Node *root = NULL;
  root = insertNode(root, 2);
  root = insertNode(root, 1);
  root = insertNode(root, 7);
  root = insertNode(root, 4);
  root = insertNode(root, 5);
  root = insertNode(root, 3);
  root = insertNode(root, 8);
printPreOrder(root);
 root = deleteNode(root, 3);
printf("\nAfter deletion: ");
  printPreOrder(root);
return 0;
}
```



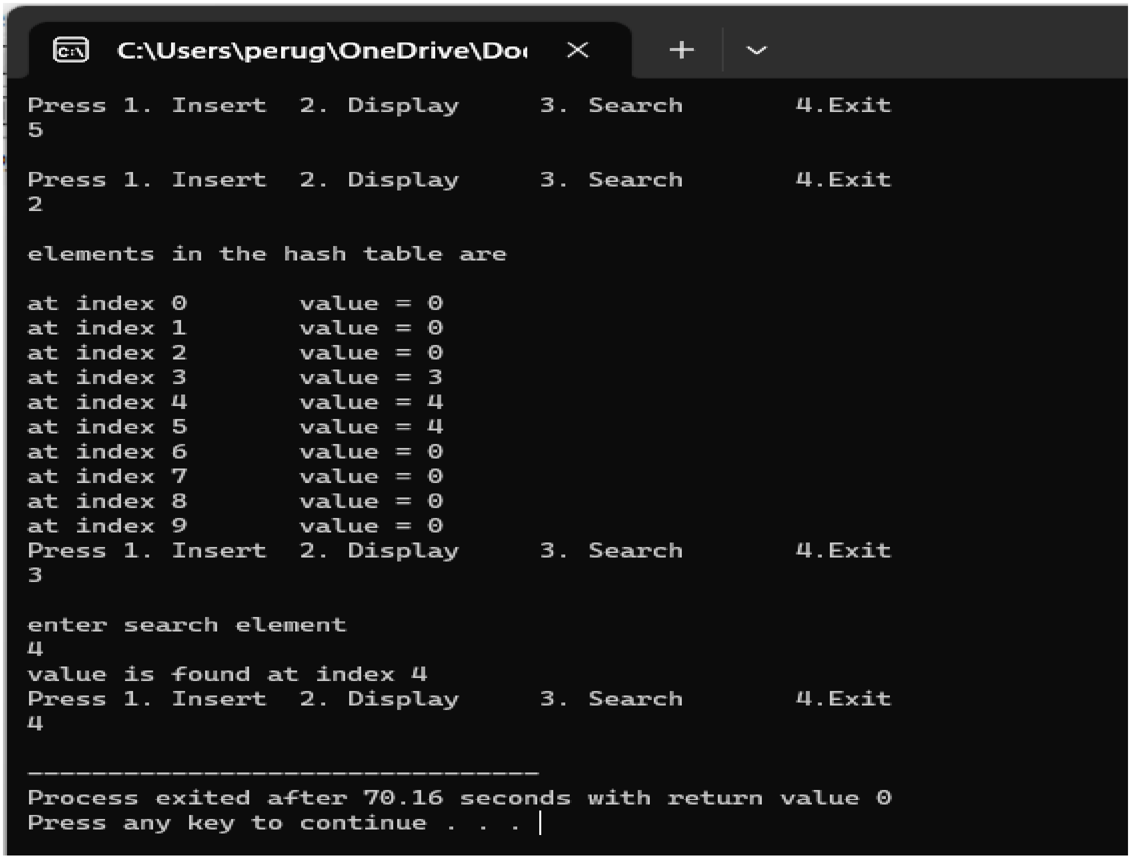
```
4 2 1 3 7 5 8
After deletion: 4 2 1 7 5 8
--------------------------------
Process exited after 0.03676 seconds with return value 0
Press any key to continue . . .
```

3.Write c program to implement hashing using linear probing technique?

```
#include <stdio.h>
 #include<stdlib.h>
 #define TABLE_SIZE 10
 int h[TABLE_SIZE]={NULL};
 void insert()
 {
  int key,index,i,flag=0,hkey;
  printf("\nenter a value to insert into hash table\n");
  scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
   {
   index=(hkey+i)%TABLE_SIZE;
   if(h[index] == NULL)
    {
     h[index]=key;
      break;
    }
   }
  if(i == TABLE_SIZE)
   printf("\nelement cannot be inserted\n");
}
void search()
{
 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
  {
  index=(hkey+i)%TABLE_SIZE;
  if(h[index]==key)
   {
   printf("value is found at index %d",index);
   break;
```

```
    }
  }
  if(i == TABLE_SIZE)
   printf("\n value is not found\n");
 }
 void display()
 {
  int i;
  printf("\nelements in the hash table are \n");
  for(i=0;i< TABLE_SIZE; i++)
  printf("\nat index %d \t value = %d",i,h[i]);
 }
  int main()
 {
   int opt,i;
   while(1)
    {
     printf("\nPress 1. Insert\t 2. Display \t3. Search
\t4.Exit \n");
     scanf("%d",&opt);
     switch(opt)
      {
        case 1:
          insert();
          break;
        case 2:
```

```
display();
       break;
     case 3:
       search();
       break;
     case 4:exit(0);
   }
  }
 return 0;
}
```

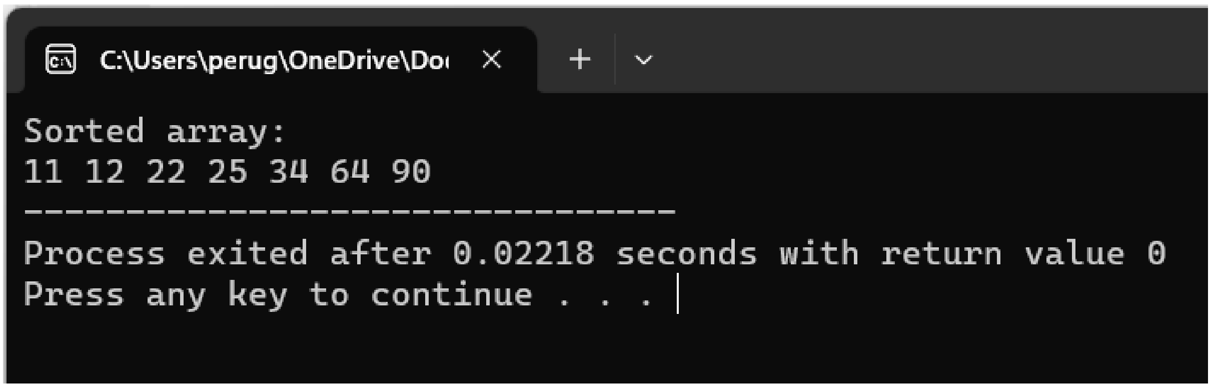


```
Press 1. Insert  2. Display     3. Search       4.Exit
5

Press 1. Insert  2. Display     3. Search       4.Exit
2

elements in the hash table are

at index 0       value = 0
at index 1       value = 0
at index 2       value = 0
at index 3       value = 3
at index 4       value = 4
at index 5       value = 4
at index 6       value = 0
at index 7       value = 0
at index 8       value = 0
at index 9       value = 0
Press 1. Insert  2. Display     3. Search       4.Exit
3

enter search element
4
value is found at index 4
Press 1. Insert  2. Display     3. Search       4.Exit
4

--------------------------------
Process exited after 70.16 seconds with return value 0
Press any key to continue . . .
```

4.write c program to implement sorting ?

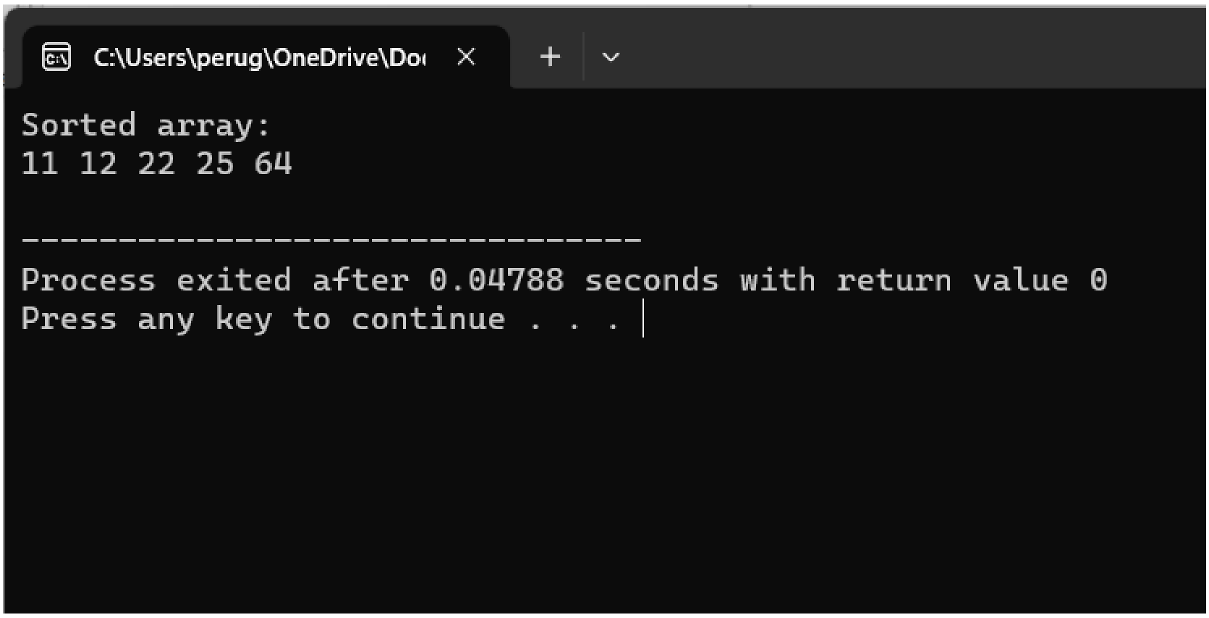
```
// Optimized implementation of Bubble sort
#include <stdbool.h>
#include <stdio.h>
void swap(int* xp, int* yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
void bubbleSort(int arr[], int n)
{
    int i, j;
    bool swapped;
    for (i = 0; i < n - 1; i++) {
        swapped = false;
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1]);
                swapped = true;
            }
        }
        if (swapped == false)
            break;
    }
}
```

```
// Function to print an array
void printArray(int arr[], int size)
{
   int i;
   for (i = 0; i < size; i++)
      printf("%d ", arr[i]);
}
int main()
{
   int arr[] = { 64, 34, 25, 12, 22, 11, 90 };
   int n = sizeof(arr) / sizeof(arr[0]);
   bubbleSort(arr, n);
   printf("Sorted array: \n");
   printArray(arr, n);
   return 0;
}
```



```
Sorted array:
11 12 22 25 34 64 90
--------------------------------
Process exited after 0.02218 seconds with return value 0
Press any key to continue . . .
```

```
// C program for implementation of selection sort
#include <stdio.h>
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;
    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;
        if(min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
}
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main()
{
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

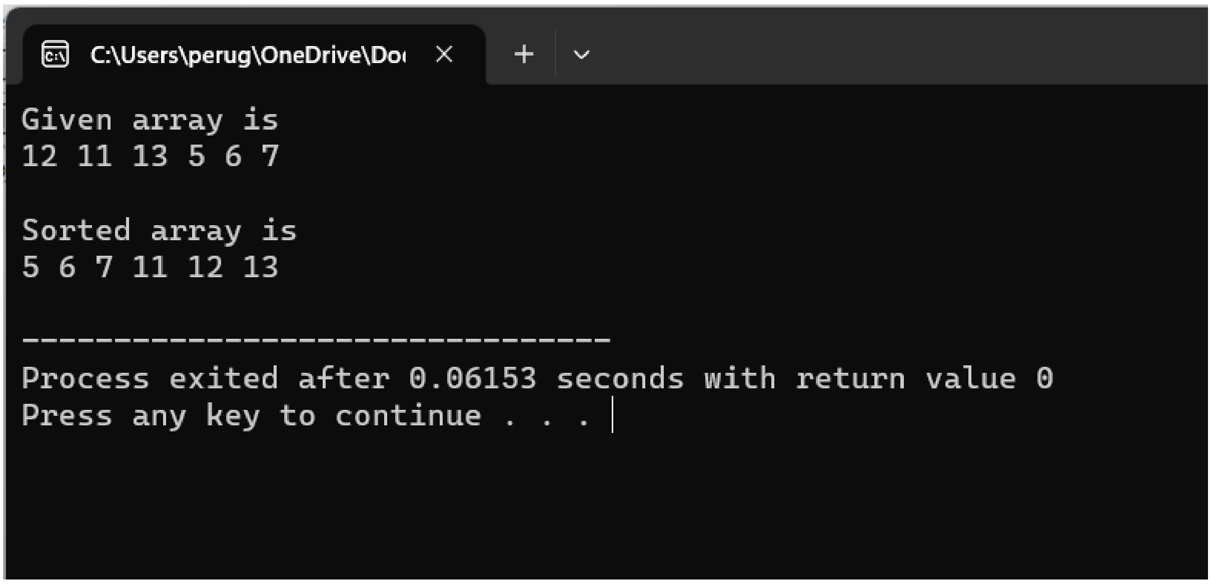


```
Sorted array:
11 12 22 25 64

--------------------------------
Process exited after 0.04788 seconds with return value 0
Press any key to continue . . .
```

```
// C program for Merge Sort
#include <stdio.h>
```

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

        merge(arr, l, m, r);
    }
}
void printArray(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
```

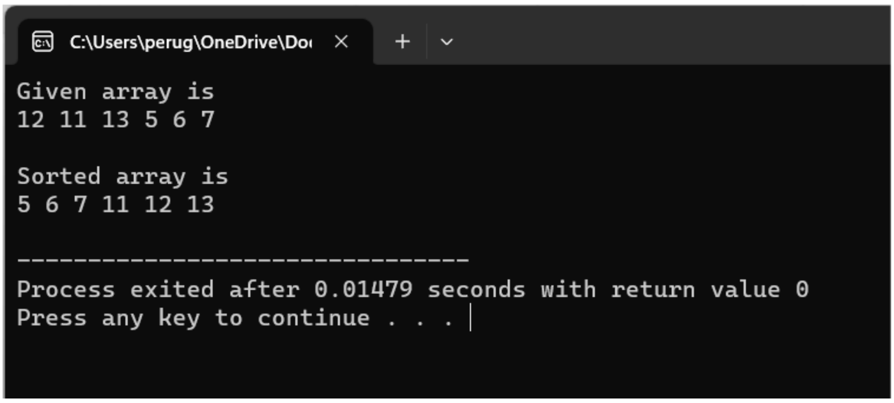
```
}
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```



```
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13

--------------------------------
Process exited after 0.06153 seconds with return value 0
Press any key to continue . . .
```

```
// C program for insertion sort
#include <math.h>
#include <stdio.h>
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
          of their current position */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main()
```

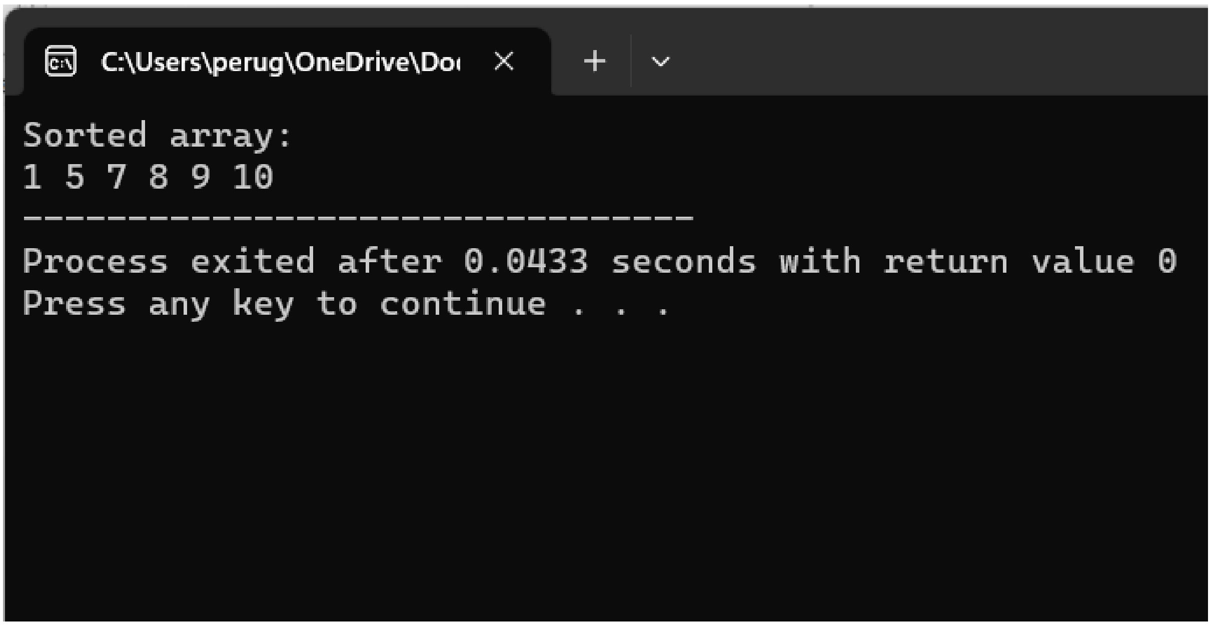
```
{
   int arr[] = { 12, 11, 13, 5, 6 };
   int n = sizeof(arr) / sizeof(arr[0]);

   insertionSort(arr, n);
   printArray(arr, n);

   return 0;
}
```



```
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13

--------------------------------
Process exited after 0.01479 seconds with return value 0
Press any key to continue . . .
```

```
// C code to implement quicksort

#include <stdio.h>
void swap(int* a, int* b)
{
   int t = *a;
   *a = *b;
   *b = t;
}
```

```
int partition(int arr[], int low, int high)
{
    // Choosing the pivot
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; 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[] = { 10, 7, 8, 9, 1, 5 };
    int N = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, N - 1);
    printf("Sorted array: \n");
    for (int i = 0; i < N; i++)
        printf("%d ", arr[i]);
    return 0;
}
```



```
Sorted array:
1 5 7 8 9 10
--------------------------------
Process exited after 0.0433 seconds with return value 0
Press any key to continue . . .
```