**Problem Statement:**

Design a menu-driven program that allows users to perform the **selection sort** algorithm on an array. The program should provide options to initialize the array, perform the selection sort operation, and display the sorted array. The user should be able to choose any of these operations from a menu and provide the required inputs.

The program should implement the following functionalities:

1. Array Initialization:

2. Selection Sort:

3. Display Sorted Array:

4. Exit:

The program should display a menu with the above options and allow the user to select an operation by entering the corresponding menu number. After executing the selected operation, the program should return to the menu and continue until the user chooses the exit option.

**Source Code:**

#include<stdio.h>

void selectionSort(int arr[], int n){

int temp =0;

int i;

int j;

for(i=0;i<n;i++){

for(j=i+1;j<n;j++){

if(arr[i] > arr[j]){

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

printf("\n");

for(int i=0; i<n; i++){

printf(" %d", arr[i]);

}

}

void main(){

printf("Yogesh Pal Parmar MCA 2B 70");

int n;

printf("\nEnter a value : ");

scanf("%d",&n);

int arr[n];

int i;

for(i=0; i<n; i++){

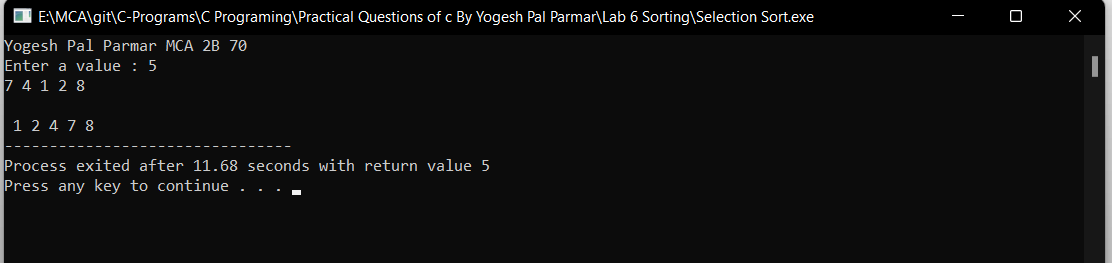
scanf("%d", &arr[i]);

}

selectionSort(arr,n);

}

**Output:**



**Problem Statement:**

Design a menu-driven program that allows users to perform the **bubble sort** algorithm on an array. The program should provide options to initialize the array, perform the bubble sort operation, and display the sorted array. The user should be able to choose any of these operations from a menu and provide the required inputs.

The program should implement the following functionalities:

1. Array Initialization:

2. Bubble Sort:

3. Display Sorted Array:

4. Exit:

The program should display a menu with the above options and allow the user to select an operation by entering the corresponding menu number. After executing the selected operation, the program should return to the menu and continue until the user chooses the exit option.

**Source Code:**

#include<stdio.h>

void bubbleSort(int arr[], int n){

int count =1;

while(count < n){

for(int i=0;i<n-count;i++){

if(arr[i] > arr[i+1]){

int temp = arr[i];

arr[i] = arr[i+1];

arr[i+1] = temp;

}

}

count++;

}

printf("\n");

for(int i=0; i<n; i++){

printf(" %d", arr[i]);

}

}

void main(){

printf("Yogesh Pal Parmar MCA 2B 70");

int n;

printf("\nEnter a value : ");

scanf("%d",&n);

int arr[n];

int i;

for(i=0; i<n; i++){

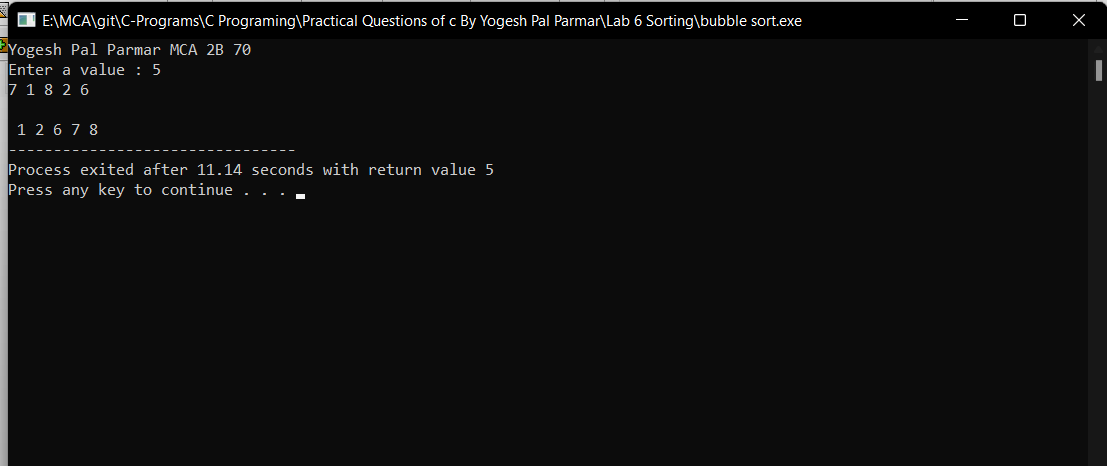
scanf("%d", &arr[i]);

}

bubbleSort(arr,n);

}

**Output:**



**Problem Statement:**

Design a menu-driven program that allows users to perform the **insertion sort** algorithm on an array. The program should provide options to initialize the array, perform the insertion sort operation, and display the sorted array. The user should be able to choose any of these operations from a menu and provide the required inputs.

The program should implement the following functionalities:

1. Array Initialization:

2. Insertion Sort:

3. Display Sorted Array:

4. Exit:

The program should display a menu with the above options and allow the user to select an operation by entering the corresponding menu number. After executing the selected operation, the program should return to the menu and continue until the user chooses the exit option.

**Source Code:**

#include<stdio.h>

void insertionSort(int arr[], int n){

for(int i=0;i<n;i++){

int current = arr[i];

int j = i-1;

while(arr[j] > current && j>=0){

arr[j+1] = arr[j];

j--;

}

arr[j+1] = current;

}

printf("\n");

for(int i=0; i<n; i++){

printf(" %d", arr[i]);

}

}

void main(){

printf("Yogesh Pal Parmar MCA 2B 70");

int n;

printf("\nEnter a value : ");

scanf("%d",&n);

int arr[n];

int i;

for(i=0; i<n; i++){

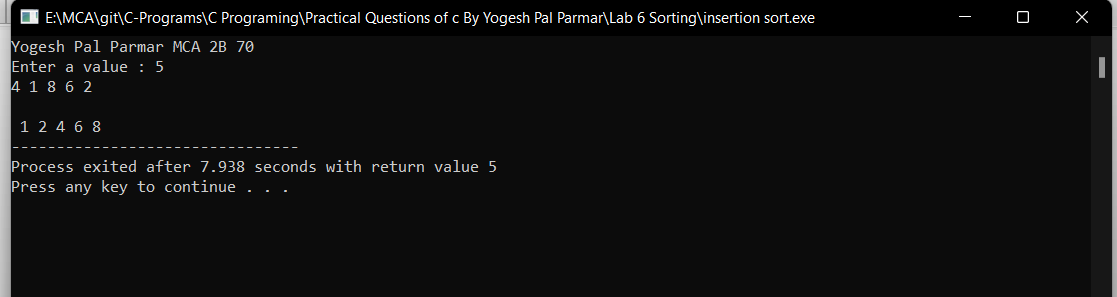
scanf("%d", &arr[i]);

}

insertionSort(arr,n);

}

**Output:**



**Problem Statement:**

Problem Statement:

Design a menu-driven program that allows users to perform the **merge sort** algorithm on an array. The program should provide options to initialize the array, perform the merge sort operation, and display the sorted array. The user should be able to choose any of these operations from a menu and provide the required inputs.

The program should implement the following functionalities:

1. Array Initialization:

2. Merge Sort:

3. Display Sorted Array:

4. Exit:

The program should display a menu with the above options and allow the user to select an operation by entering the corresponding menu number. After executing the selected operation, the program should return to the menu and continue until the user chooses the exit option.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there

are any \*/

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

/\* l is for left index and r is right index of the

sub-array of arr to be sorted \*/

void mergeSort(int arr[], int l, int r)

{

if (l < r) {

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l + (r - l) / 2;

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

/\* UTILITY FUNCTIONS \*/

/\* Function to print an array \*/

void printArray(int A[], int size)

{

int i;

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

printf("%d ", A[i]);

printf("\n");

}

/\* Driver code \*/

int main()

{

printf("Yogesh Pal Parmar MCA 2B 70\n");

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);

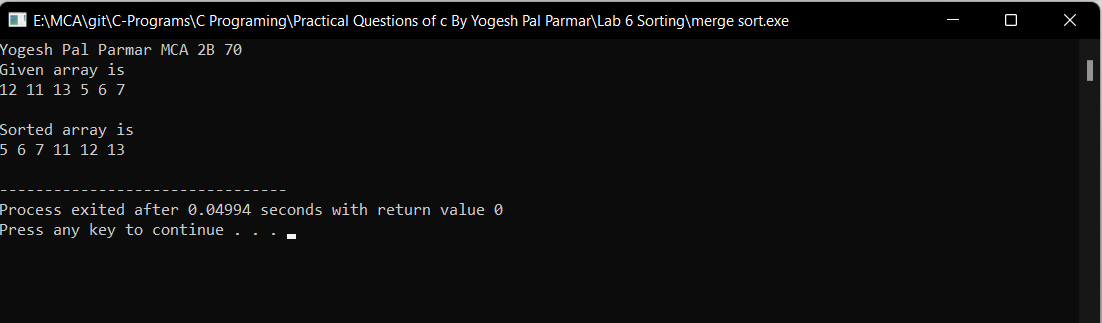
printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}

**Output:**



**Problem Statement:**

Design a menu-driven program that allows users to perform the **quick sort** algorithm on an array. The program should provide options to initialize the array, perform the quick sort operation, and display the sorted array. The user should be able to choose any of these operations from a menu and provide the required inputs.

The program should implement the following functionalities:

1. Array Initialization:

2. Quick Sort:

3. Display Sorted Array:

4. Exit:

The program should display a menu with the above options and allow the user to select an operation by entering the corresponding menu number. After executing the selected operation, the program should return to the menu and continue until the user chooses the exit option.

**Source Code:**

#include <stdio.h>

// Function to swap two elements

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

// Partition the array using the last element as the pivot

int partition(int arr[], int low, int high)

{

// Choosing the pivot

int pivot = arr[high];

// Index of smaller element and indicates

// the right position of pivot found so far

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

// If current element is smaller than the pivot

if (arr[j] < pivot) {

// Increment index of smaller element

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

// The main function that implements QuickSort

// arr[] --> Array to be sorted,

// low --> Starting index,

// high --> Ending index

void quickSort(int arr[], int low, int high)

{

if (low < high) {

// pi is partitioning index, arr[p]

// is now at right place

int pi = partition(arr, low, high);

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Driver code

int main()

{

printf("Yogesh Pal Parmar MCA 2B 70\n");

int arr[] = { 10, 7, 8, 9, 1, 5 };

int N = sizeof(arr) / sizeof(arr[0]);

printf("10, 7, 8, 9, 1, 5\n");

// Function call

quickSort(arr, 0, N - 1);

printf("Sorted array: \n");

for (int i = 0; i < N; i++)

printf("%d ", arr[i]);

return 0;

}

**Output:**

