**Week :1**

**I. Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case. (Time Complexity = O(n), where n is the size of input)**

**Code:-**

#include <stdio.h>

// Function to perform linear search

int linearSearch(int arr[], int size, int key, int \*comparisons) {

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

(\*comparisons)++; // Increment comparison count

if (arr[i] == key) {

return i; // Key found, return index

}

}

return -1; // Key not found, return -1

}

int main() {

int size, key, comparisons = 0;

// Get size of the array from the user

printf("Enter the size of the array: ");

scanf("%d", &size);

int arr[size]; // Declare an array of the specified size

// Get array elements from the user

printf("Enter %d elements of the array:\n", size);

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

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

}

// Get the key to search for

printf("Enter the key to search: ");

scanf("%d", &key);

// Perform the linear search

int result = linearSearch(arr, size, key, &comparisons);

// Output the result

if (result != -1) {

printf("Element %d found at index %d. Total comparisons: %d\n", key, result, comparisons);

} else {

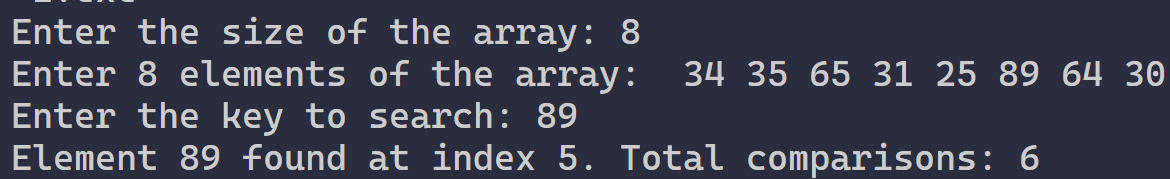
printf("Element %d not found. Total comparisons: %d\n", key, comparisons);

}

return 0;

}

**Output:-**



A screenshot of a computer code

Description automatically generated

A blue background with white text

Description automatically generated

**II. Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case. (Time Complexity = O(n logn), where n is the size of input).**

**Code:-**

#include <stdio.h>

// Function to perform binary search

int binarySearch(int arr[], int size, int key, int \*comparisons) {

int low = 0, high = size - 1;

while (low <= high) {

(\*comparisons)++; // Increment comparisons count

int mid = (low + high) / 2;

// Check if key is present at mid

if (arr[mid] == key) {

return mid; // Return the index if key is found

}

// If key is smaller than mid, it must be in the left subarray

else if (key < arr[mid]) {

high = mid - 1;

}

// If key is larger than mid, it must be in the right subarray

else {

low = mid + 1;

}

}

return -1; // Return -1 if key is not found

}

int main() {

int size, key, comparisons = 0;

// Get the size of the array from the user

printf("Enter the size of the array: ");

scanf("%d", &size);

int arr[size]; // Declare an array of the specified size

// Get sorted array elements from the user

printf("Enter %d sorted elements of the array:\n", size);

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

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

}

// Get the key to search for

printf("Enter the key to search: ");

scanf("%d", &key);

// Perform the binary search

int result = binarySearch(arr, size, key, &comparisons);

// Output the result

if (result != -1) {

printf("Element %d found at index %d. Total comparisons: %d\n", key, result, comparisons);

} else {

printf("Element %d not found. Total comparisons: %d\n", key, comparisons);

}

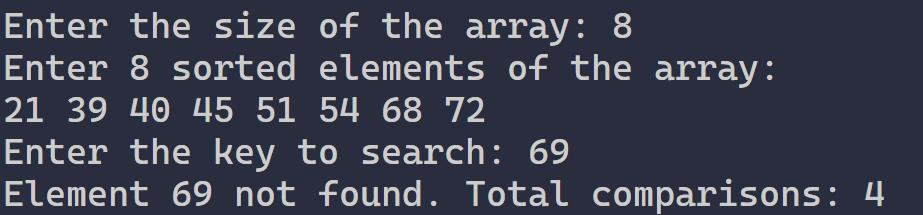
return 0;

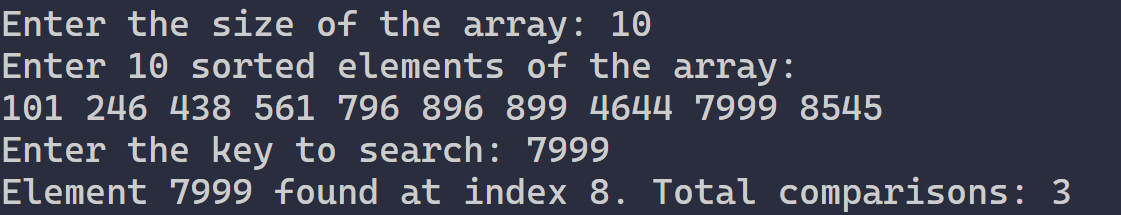
}

**Output:-**

A screenshot of a computer

Description automatically generated





**III. Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array arr[n], search at the indexes arr[0], arr[2], arr[4],.....,arr[2k] and so on. Once the interval (arr[2k] < key < arr[ 2k+1] ) is found, perform a linear search operation from the index 2k to find the element key. (Complexity < O(n), where n is the number of elements need to be scanned for searching):**

**Jump Search**

**Code:-**

#include <stdio.h>

#include <math.h>

// Function to perform Jump Search

int jumpSearch(int arr[], int n, int key, int \*comparisons) {

    int step = sqrt(n);

    int prev = 0;

    // Finding the block where the element might be present

    while (arr[(step < n ? step : n) - 1] < key) {

        (\*comparisons)++;

        prev = step;

        step += sqrt(n);

        if (prev >= n)

            return -1;

    }

    // Performing linear search within the block

    while (arr[prev] < key) {

        (\*comparisons)++;

        prev++;

        if (prev == (step < n ? step : n))  // Reached end of the block or array

            return -1;

    }

    // If element is found

    (\*comparisons)++;

    if (arr[prev] == key)

        return prev;

    return -1;  // Element not found

}

int main() {

    int T;

    printf("Enter the number of test cases: ");

    scanf("%d", &T);

    while (T--) {

        int n, key, result, comparisons = 0;

        // Input size of array

        printf("Enter the size of the array: ");

        scanf("%d", &n);

        int arr[n];

        printf("Enter the sorted array elements:\n");

        for (int i = 0; i < n; i++) {

            scanf("%d", &arr[i]);

        }

        printf("Enter the key to be searched: ");

        scanf("%d", &key);

        // Perform Jump Search

        result = jumpSearch(arr, n, key, &comparisons);

        if (result != -1) {

            printf("Present %d\n", comparisons);

        } else {

            printf("Not Present %d\n", comparisons);

        }

    }

    return 0;

}

**Output:-**

A computer screen shot of a number

Description automatically generated

**Week :2**

**I. Given a sorted array of positive integers containing few duplicate elements, design an algorithm and implement it using a program to find whether the given key element is present in the array or not. If present, then also find the number of copies of given key. (Time Complexity = O(log n))**

**Code:-**

#include <stdio.h>

// O(log n)

// Function to perform binary search for the first occurrence

int firstOccurrence(int a[], int n, int key) {

    int low = 0, high = n - 1;

    int result = -1;

    while (low <= high) {

        int mid = low + (high - low) / 2;

        if (a[mid] == key) {

            result = mid;

            high = mid - 1;  // Look for earlier occurrence

        } else if (a[mid] < key) {

            low = mid + 1;

        } else {

            high = mid - 1;

        }

    }

    return result;

}

// Function to perform binary search for the last occurrence

int lastOccurrence(int a[], int n, int key) {

    int low = 0, high = n - 1;

    int result = -1;

    while (low <= high) {

        int mid = low + (high - low) / 2;

        if (a[mid] == key) {

            result = mid;

            low = mid + 1;  // Look for later occurrence

        } else if (a[mid] < key) {

            low = mid + 1;

        } else {

            high = mid - 1;

        }

    }

    return result;

}

// Function to count frequency using binary search

int count(int a[], int n, int key) {

    int first = firstOccurrence(a, n, key);

    if (first == -1) {

        return 0;  // Key not found

    }

    int last = lastOccurrence(a, n, key);

    return last - first + 1;  // Frequency calculation

}

int main() {

    int t;

    printf("Enter Number of times to execute\n");

    scanf("%d", &t);

    for (int r = 0; r < t; r++) {

        int n;

        printf("Enter Size of the array:\n");

        scanf("%d", &n);

        int a[n];

        printf("Enter Array Elements (sorted):\n");

        for (int i = 0; i < n; i++) {

            scanf("%d", &a[i]);

        }

        int key;

        printf("Enter the element to be searched:\n");

        scanf("%d", &key);

        // Call the binary search-based count function

        printf("Frequency of %d is : %d\n", key, count(a, n, key));

    }

    return 0;

}

**Output:-**

A computer screen shot of a computer code

Description automatically generated

**II. Given a sorted array of positive integers, design an algorithm and implement it using a program to find three indices i, j, k such that arr[i] + arr[j] = arr[k].**

**Code:-**

#include <stdio.h>

void print\_Triplets(int A[], int N){

     for(int i = 0; i < N; i++){ // for first number

       for(int j = i + 1; j < N; j++){ // for second number

          for(int k = j + 1; k < N; k++){  // for third number

              if(A[i] + A[j] == A[k]){

                  printf("%d\t%d\t%d", i+1,j+1,k+1);

                  return;

                    }

             }

       }

     }

     printf("No Sequence Found");

 }

int main() {

    int A[20];

    int no\_of\_testcase,N;

    scanf("%d",&no\_of\_testcase);

    for(int i=0;i<no\_of\_testcase;i++)

        {

          scanf("%d",&N);

            for(int i=0;i<N;i++)

               scanf("%d",&A[i]);

          print\_Triplets(A, N);

        }

    return 0;

}

**Output:-**

A blue background with white text

Description automatically generated

**III. Given an array of nonnegative integers, design an algorithm and a program to count the number of pairs of integers such that their difference is equal to a given key, K.**

**Code:-**

#include <stdio.h>

// Function to count pairs with a given difference K

int countPairsWithDifference(int arr[], int n, int K) {

    int count = 0;

    // Iterate over each pair

    for (int i = 0; i < n; i++) {

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

            if (arr[j] - arr[i] == K || arr[i] - arr[j] == K) {

                count++;

            }

        }

    }

    return count;

}

int main() {

    int T;

    printf("Enter the number of test cases: ");

    scanf("%d", &T);

    while (T--) {

        int n, K;

        // Input size of array

        printf("Enter the size of the array: ");

        scanf("%d", &n);

        int arr[n];

        // Input array elements

        printf("Enter the array elements:\n");

        for (int i = 0; i < n; i++) {

            scanf("%d", &arr[i]);

        }

        // Input the key K

        printf("Enter the key difference K: ");

        scanf("%d", &K);

        // Count pairs with difference K

        int result = countPairsWithDifference(arr, n, K);

        // Output the result

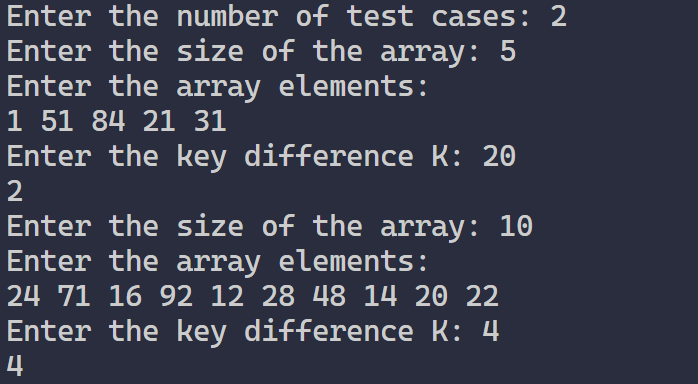
        printf("%d\n", result);

    }

    return 0;

}

**Output:-**

****

**Week :3**

**I. Given an unsorted array of integers, design an algorithm and a program to sort the array using insertion sort. Your program should be able to find number of comparisons and shifts ( shifts -total number of times the array elements are shifted from their place) required for sorting the array.**

**Code:-**

#include <stdio.h>

void insertionSort(int arr[], int n, int \*comparisons, int \*shifts) {

    \*comparisons = 0;

    \*shifts = 0;

    for (int i = 1; i < n; i++) {

        int key = arr[i];

        int j = i - 1;

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

            (\*comparisons)++;

            arr[j + 1] = arr[j];

            j--;

            (\*shifts)++;

        }

        arr[j + 1] = key;

        (\*shifts)++;

        if (j >= 0) (\*comparisons)++;

    }

}

int main() {

    int T;

    printf("Enter number of test cases:\n");

    scanf("%d", &T);

    for (int t = 0; t < T; t++) {

        int n;

        printf("Enter the size of the array:\n");

        scanf("%d", &n);

        int arr[n];

        printf("Enter the array elements:\n");

        for (int i = 0; i < n; i++) {

            scanf("%d", &arr[i]);

        }

        int comparisons = 0, shifts = 0;

        // Sort the array using insertion sort

        insertionSort(arr, n, &comparisons, &shifts);

        printf("Sorted array:\n");

        for (int i = 0; i < n; i++) {

            printf("%d ", arr[i]);

        }

        printf("\n");

        printf("comparisons = %d\n", comparisons);

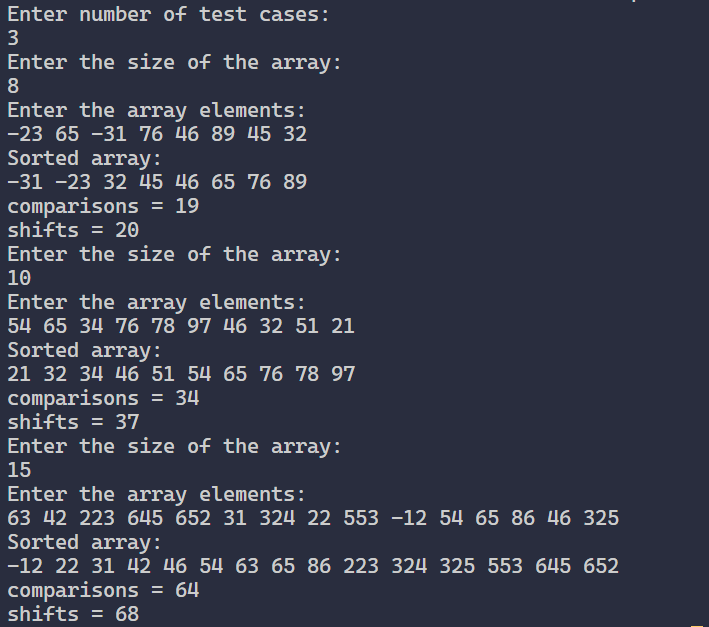
        printf("shifts = %d\n", shifts);

    }

    return 0;

}

**Output:-**



**II. Given an unsorted array of integers, design an algorithm and implement a program to sort this array using selection sort. Your program should also find number of comparisons and number of swaps required.**

**Code:-**

#include <stdio.h>

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

    int comp = 0, swap = 0, temp;

    for(int i = 0; i < n - 1; i++) {

        int min\_idx = i;

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

            comp++;

            if (arr[j] < arr[min\_idx]) {

                min\_idx = j;

            }

        }

        if (min\_idx != i) {

            temp = arr[i];

            arr[i] = arr[min\_idx];

            arr[min\_idx] = temp;

            swap++;

        }

    }

    printf("Sorted array: ");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

    printf("Comparisons: %d\n", comp);

    printf("Swaps: %d\n", swap);

}

int main() {

    int T;

    printf("Enter the number of test cases: ");

    scanf("%d", &T);

    while (T--) {

        int n;

        printf("Enter the size of the array: ");

        scanf("%d", &n);

        int arr[n];

        printf("Enter the array elements:\n");

        for (int i = 0; i < n; i++) {

            scanf("%d", &arr[i]);

        }

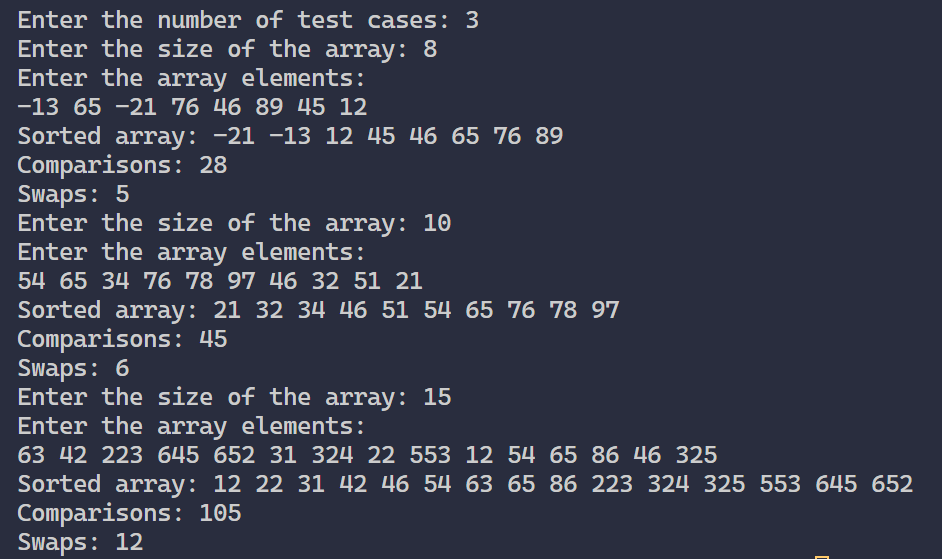
        compSwap(arr, n);

    }

    return 0;

}

**Output:-**

****

**III. Given an unsorted array of positive integers, design an algorithm and implement it using a program to find whether there are any duplicate elements in the array or not. (use sorting) (Time Complexity = O(n log n))**

**Code:-**

#include <stdio.h>

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

    int comp = 0, swap = 0, temp;

    for (int i = 0; i < n - 1; i++) {

        for (int j = 0; j < n - 1 - i; j++) {

            comp++;  // Count the comparison

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

                // Swap if elements are in the wrong order

                temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

                swap++;  // Count the swap

            }

        }

    }

    // Checking for duplicates after sorting

    int hasDuplicate = 0;

    for (int i = 1; i < n; i++) {

        comp++;  // Count comparison for checking duplicates

        if (arr[i] == arr[i - 1]) {

            hasDuplicate = 1;

            break;

        }

    }

    // Output the results

    if (hasDuplicate) {

        printf("YES\n");  // Duplicate found

    } else {

        printf("NO\n");  // No duplicates found

    }

    // Output the number of comparisons and swaps made

    printf("Comparisons: %d\n", comp);

    printf("Swaps: %d\n", swap);

}

int main() {

    int T;

    printf("Enter the number of test cases: ");

    scanf("%d", &T);

    while (T--) {

        int n;

        printf("Enter the size of the array: ");

        scanf("%d", &n);

        int arr[n];

        printf("Enter the array elements:\n");

        for (int i = 0; i < n; i++) {

            scanf("%d", &arr[i]);

        }

        // Call the duplicate function for each test case

        duplicate(arr, n);

    }

    return 0;

}

**Output:-**

**A computer screen shot of a number of array

Description automatically generated**

**Week 4:**

**I. Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by dividing the array into two subarrays and combining these subarrays after sorting each one of them. Your program should also find number of comparisons and inversions during sorting the array.**

**Code:-**

#include <stdio.h>

// Function to merge two halves of the array

int mergeAndCount(int arr[], int temp[], int left, int mid, int right, int \*compCount) {

    int i = left;    // Starting index for left subarray

    int j = mid + 1; // Starting index for right subarray

    int k = left;    // Starting index to be sorted

    int invCount = 0;  // Number of inversions

    while (i <= mid && j <= right) {

        (\*compCount)++; // Count comparisons

        if (arr[i] <= arr[j]) {

            temp[k++] = arr[i++];

        } else {

            temp[k++] = arr[j++];

            invCount += (mid - i + 1); // All remaining elements in left subarray are inversions

        }

    }

    // Copy the remaining elements of left subarray, if any

    // Copy the remaining elements of right subarray, if any

    while (j <= right) {

        temp[k++] = arr[j++];

    }

    // Copy the sorted subarray into the original array

    for (i = left; i <= right; i++) {

        arr[i] = temp[i];

    }

    return invCount;

}

// Function to implement merge sort and count comparisons and inversions

int mergeSortAndCount(int arr[], int temp[], int left, int right, int \*compCount) {

    int invCount = 0;

    if (left < right) {

        int mid = (left + right) / 2;

        // Recursively split the left subarray and count inversions

        invCount += mergeSortAndCount(arr, temp, left, mid, compCount);

        // Recursively split the right subarray and count inversions

        invCount += mergeSortAndCount(arr, temp, mid + 1, right, compCount);

        // Merge the two subarrays and count inversions during merging

        invCount += mergeAndCount(arr, temp, left, mid, right, compCount);

    }

    return invCount;

}

int main() {

    int T;

    printf("Enter the number of test cases: ");

    scanf("%d", &T);

    while (T--) {

        int n;

        // Input the size of the array

        printf("Enter the size of the array: ");

        scanf("%d", &n);

        int arr[n];

        int temp[n];

        // Input the array elements

        printf("Enter the array elements:\n");

        for (int i = 0; i < n; i++) {

            scanf("%d", &arr[i]);

        }

        // Initialize counters for comparisons and inversions

        int compCount = 0;

        int invCount = mergeSortAndCount(arr, temp, 0, n - 1, &compCount);

        // Output the sorted array

        printf("Sorted array: ");

        for (int i = 0; i < n; i++) {

            printf("%d ", arr[i]);

        }

        printf("\n");

        // Output the number of comparisons and inversions

        printf("Comparisons: %d\n", compCount);

        printf("Inversions: %d\n", invCount);

    }

    return 0;

}

**Output:-**

**A computer screen shot of a number

Description automatically generated**

**II. Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array.**

**Code:-**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to swap two elements

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

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

// Function to partition the array using a randomly chosen pivot

int partition(int arr[], int low, int high, int\* compCount, int\* swapCount) {

    // Select a random pivot

    int randomIndex = low + rand() % (high - low + 1);

    swap(&arr[randomIndex], &arr[high]);

    (\*swapCount)++; // Count the swap

    int pivot = arr[high];  // Pivot element

    int i = low - 1;        // Index of the smaller element

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

        (\*compCount)++; // Count comparisons

        if (arr[j] <= pivot) {

            i++;

            swap(&arr[i], &arr[j]); // Swap arr[i] and arr[j]

            (\*swapCount)++; // Count the swap

        }

    }

    // Move pivot to the correct position

    swap(&arr[i + 1], &arr[high]);

    (\*swapCount)++; // Count the swap

    return (i + 1); // Return the index of the pivot

}

// QuickSort algorithm with comparison and swap counting

void quickSort(int arr[], int low, int high, int\* compCount, int\* swapCount) {

    if (low < high) {

        // Partition the array and get the pivot index

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

        // Recursively sort the subarrays

        quickSort(arr, low, pi - 1, compCount, swapCount);

        quickSort(arr, pi + 1, high, compCount, swapCount);

    }

}

int main() {

    srand(time(0)); // Seed for random number generation

    int T;

    printf("Enter the number of test cases: ");

    scanf("%d", &T);

    while (T--) {

        int n;

        // Input the size of the array

        printf("Enter the size of the array: ");

        scanf("%d", &n);

        int arr[n];

        // Input the array elements

        printf("Enter the array elements:\n");

        for (int i = 0; i < n; i++) {

            scanf("%d", &arr[i]);

        }

        // Initialize counters for comparisons and swaps

        int compCount = 0;

        int swapCount = 0;

        // Apply QuickSort

        quickSort(arr, 0, n - 1, &compCount, &swapCount);

        // Output the sorted array

        printf("Sorted array: ");

        for (int i = 0; i < n; i++) {

            printf("%d ", arr[i]);

        }

        printf("\n");

        // Output the number of comparisons and swaps

        printf("Comparisons: %d\n", compCount);

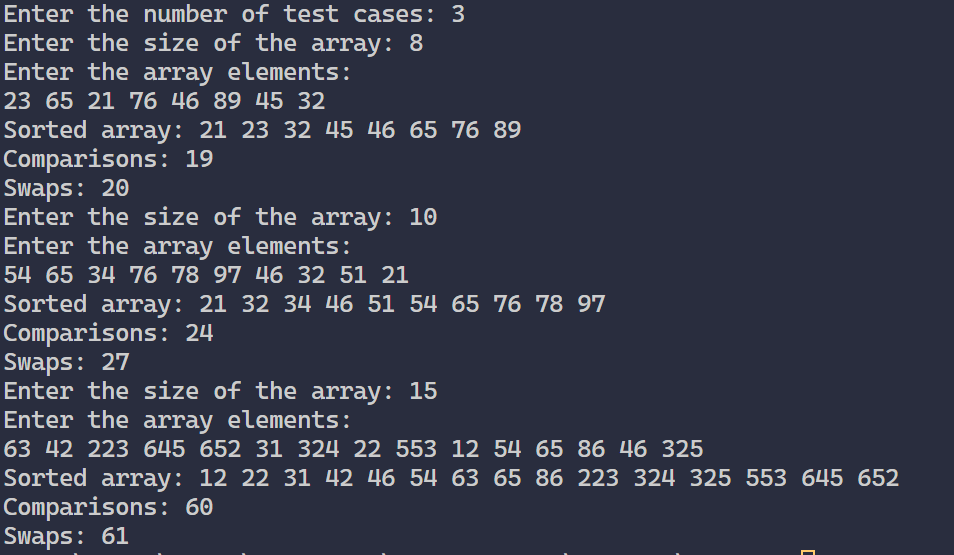
        printf("Swaps: %d\n", swapCount);

    }

    return 0;

}

**Output:-**



**III. Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))**

**Code:-**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to swap two elements

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

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

// Partition function to find the pivot

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

    int pivotValue = arr[pivotIndex];

    swap(&arr[pivotIndex], &arr[high]); // Move pivot to end

    int storeIndex = low;

    for (int i = low; i < high; i++) {

        if (arr[i] < pivotValue) {

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

            storeIndex++;

        }

    }

    // Move pivot to its final place

    swap(&arr[storeIndex], &arr[high]);

    return storeIndex; // Return the index of the pivot

}

// Quickselect algorithm to find the Kth smallest element

int quickSelect(int arr[], int low, int high, int k) {

    if (low == high) {

        return arr[low]; // Base case: only one element

    }

    // Choose a random pivot index

    int pivotIndex = low + rand() % (high - low + 1);

    pivotIndex = partition(arr, low, high, pivotIndex);

    // The pivot is in its final sorted position

    if (k == pivotIndex) {

        return arr[k]; // Found the Kth smallest

    } else if (k < pivotIndex) {

        return quickSelect(arr, low, pivotIndex - 1, k); // Search in the left partition

    } else {

        return quickSelect(arr, pivotIndex + 1, high, k); // Search in the right partition

    }

}

int main() {

    srand(time(0)); // Seed for random number generation

    int T;

    printf("Enter the number of test cases: ");

    scanf("%d", &T);

    while (T--) {

        int n;

        // Input the size of the array

        printf("Enter the size of the array: ");

        scanf("%d", &n);

        // Handle case where n is 0

        if (n <= 0) {

            printf("not present\n");

            continue;

        }

        int arr[n];

        // Input the array elements

        printf("Enter the array elements:\n");

        for (int i = 0; i < n; i++) {

            scanf("%d", &arr[i]);

        }

        // Input K value

        int k;

        printf("Enter K: ");

        scanf("%d", &k);

        // Check if K is within valid range

        if (k < 1 || k > n) {

            printf("not present\n");

            continue;

        }

        // Find the Kth smallest element (K-1 for zero-based index)

        int kthSmallest = quickSelect(arr, 0, n - 1, k - 1);

        // Output the result

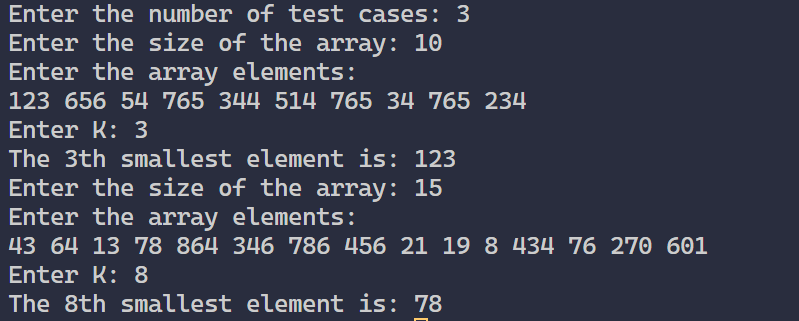
        printf("The %dth smallest element is: %d\n", k, kthSmallest);

    }

    return 0;

}

**Output:-**

****